TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Alejandro Guerra Manzanares – IVCM 172627

APPLICATION OF FULL MACHINE LEARNING WORKFLOW FOR MALWARE DETECTION IN ANDROID ON THE BASIS OF SYSTEM CALLS AND PERMISSIONS

Master's Thesis

Supervisor: Hayretdin Bahsi,
Ph.D.

Sven Nõmm
Ph.D.

Tallinn 2018

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Alejandro Guerra Manzanares

07.05.2018

Abstract

Mobile malware have increased attacks' frequency and sophistication in last years. Android OS is the main target of these attacks, as the most widely used smartphones OS. Traditional antivirus techniques are ineffective on detecting unknown or new malware. This situation puts Android users in a high-risky situation. Machine learning has been used in numerous researches as an alternative method to detect malicious applications overcoming antivirus problems. In this regard, machine learning classification methods use features from known data, known as predictors, to predict the malicious behaviour of unknown data. This thesis uses system calls and permissions as features to detect malicious behaviour in Android environment. Feature selection is used to select and build a Decision Tree algorithm classifier model aiming to minimize the number of predictors of two different malware datasets. Old and new malware datasets variability of predictors is analysed. Results showed that system calls provide greater discriminatory power than permissions and that hybrid approach (combination of both features) possess slightly greater discriminatory power than system calls alone. Feature selection provided similar detection accuracy (97%) using 212 system calls than with 22 of them, allowing to reduce and minimize the number of predictors used to build the model. Furthermore, it was also possible to build a classifier with a single feature and obtain over 86% accuracy in cross-dataset testing. Analysis of features showed that predictive variables have changed in malware over years, with new malware becoming more like legitimate applications and thus reducing variables discriminatory power with a few exceptions. Nevertheless, it is still possible to build a classifier to detect old and new malware without the need of using mixed datasets as a training set. A single dataset (old or new malware) could be used, depending on the number of features used, to build the model. This research shows that depending on detection objectives and requirements, different features and dataset should be used in order to accomplish them with optimum malware detection performance.

This thesis is written in English and is 77 pages long, including 6 chapters, 31 figures and 13 tables.

Annotatsioon

Täieliku masinõppe töövoo rakendamine pahavara tuvastamiseks süsteemikutsete ja pääsuõiguste alusel

Antud töö on pühendatud Android opsüsteemi pahavara analüüsimiseks ja tuvastamiseks. Viimaste aastate jooksul on muutunud nii pahavara keerukus ja selle abil korraldatavate rünnakute sagedus. Seetõttu hüppeliselt suurenes risk Android opsüsteemi kasutatavate nuttiseadmetele. Pahavara keerukuse tõttu paljud pööravad pilku masinõpe meetodite poole. Pahavara käitumist kirjeldavate parameetrite alusel arvutatakse tunnused, mille alusel treenitakse masinõpe klassifikaatoreid. Töö eripära koosneb kahest komponendist. Esimene on tunnuste arvutamine, mille tulemuseks on väiksem tunnuste hulk, kui avaldatud paljudes publikatsioonides. Nimelt 212 tunnusest on saadud hulk mis koosneb vaid 22st. Selle saavutuse alusel võib treenida lihtsamaid klassifikaatoreid. Teiseks eripära komponendiks on vana ja uue pahavara võrdlus, mis kirjeldab kuidas muutuvad pahavara omadused võrreldes standartsete rakendustega. Töö käigus oli demonstreeritud, et uus pahavara muutub sarnaseks tava rakendustega. Töö tulemus: oli välja treenitud mitu klassifikaatorit mille täpsus on 97% ümbruses. Samuti oli demonstreeritud, et vaid ühe tunnusega on võimalik treenida klassifikaator, mille täpsus on 86% ringis. Töö näitab et tunnuste valik ja treenimis andmed tuleb valida vastavalt tuvastatavale objektile ja lubatud keerukusele.

Lõputöö on kirjutatud Inglise keeles ning sisaldab teksti 77 leheküljel, 6 peatükki, 31 joonist, 13 tabelit.

Table of contents

1 Introduction 7
2 Background Information 10
2.1 APK and Android Malware 10
2.2 Machine Learning 12
2.3 Related Work 13
2.3.1 Static malware analysis and detection 13
2.3.2 Dynamic malware analysis and detection 17
2.3.3 Hybrid malware analysis and detection 24
3 Methodology 26
3.1 Phase 1. Data Acquisition 26
3.1.1 Dataset 26
3.1.2 Android Emulation 27
3.1.3 Application's feature: system calls 27
3.1.4 Application's feature: Permissions 28
3.1.5 Feature extraction 29
3.1.6 System calls and permissions' collection 30
3.1.7 Statistical hypothesis testing 31
3.2 Phase 2. Data pre-processing 38
3.2.1 Feature selection 38
3.3 Phase 3. Classification, Training and Validation 51
3.3.1 Malware detection: binary classification problem 51
3.3.2 Malware detection: performance 52
4 Malware detection – practical implementation 55
4.1 Classification algorithms 55
4.1.1 Decision Tree Algorithm 58
5 Malware Detection Model Validation 61
5.1 Selected system calls 61
5.2 Selected permissions 62

5.3 Combination of features: permissions and system calls 64
5.4 Old vs. New Malware discrimination 65
5.5 Cross-dataset malware detection validation 67
5.5.1 Dynamic approach: System calls 68
5.5.2 Static approach: Permissions 69
5.5.3 Hybrid approach: system calls and permissions 70
5.6 Mixed malware detection validation 71
5.6.1 System call features 72
5.6.2 Permission features 72
5.6.3 Hybrid approach 72
5.7 Decision Tree graphs 74
5.7.1 System calls 74
5.7.2 Permissions 77
5.7.3 Hybrid trees 80
6 Conclusions 83
References 85
Appendix 1 – System call's statistics 94
Appendix 2 – System calls' Welch's test 100
Appendix 3 – Permissions' statistics 107
Appendix 4 – Permissions' Chi Square Test 112
Appendix 5 – System calls' Fisher Score values 117
Appendix 6 – Permissions' Gini Index values 123
Appendix 7 – System call's model validation 127
Appendix 8 – Permissions' model validation 130
Appendix 9 – Hybrid model validation 132
Appendix 10 – Malware discrimination model validation 134
Appendix 11 – Cross-dataset malware model validation 137
Appendix 12 – Mixed malware detection validation 144

1 Introduction

Smartphones allow in their reduced size to perform thousands of activities that before had to be implemented with computers or physically, saving time, money and effort to the user. Mobile applications have promoted the use of mobile use and navigation. In 2017, 52,7% of global internet traffic was originated from mobile devices and it is expected that by 2018 will reach 80% [1]. In 2017, 91.3% of social media users used their mobile devices for their social media related activities and 90% of mobile device's time of use is spent in apps [2]. This growing trend is exploited by malware authors to propagate their creations worldwide. According to McAfee [3], "2018 could be the year of mobile malware", an emerging threat statement based on their 16 million malware infections detected in the third quarter of 2017 alone, twice the figure in 2016. This spike was also confirmed by Kaspersky [3][4], which detected an 80% increase in malware attacks and sophistication. McAfee report also states that Google Play store has been target of malware campaigns almost since its inception and it is still under siege [3]. Android Grabos was the latest malware campaign detected in Google Play, which affected 144 apps stored in Google Play, infecting 17.5 million smartphones in 2017 before they were removed [3]. Notwithstanding that Android's owner, Google, is increasing new versions' security features, they are still ineffective to protect or detect even the most common malware [3][5]. Furthermore, antivirus software for mobile devices has been proved to be inefficient detecting malware applications [6]. These limitations come not only from malware obfuscation techniques but also from Android OS itself which uses a filesystem-based sandbox to ensure that each installed application have only access to its own data and not to other apps or user data, unless it is explicitly permitted by the user. This directly affects Android antivirus software as they are not capable of list other directories' contents [6].

Android OS is the most widespread mobile operating system worldwide, it is run by most of smartphone producers, as it is a highly customizable and open source, based on Linux kernel. Over 87% of global smartphone devices use some kind of Android flavor [7] but only 1% of them use the latest version, Android 8.0, that includes enhanced

security capabilities [8]. Android OS users download applications via App markets. The official market is Google Play store, but there are many other third-party markets like *AndAppStore*, *GetJar*, *Handango*, etc. that attract users by providing pay applications for free. The main issue of application markets is security, especially in third-party stores, which are less controlled. Apps' stores are used by attackers to spread and infect users' devices with malware, especially trojan versions of widely-used and popular applications. The estimated 4.5 billion of Android mobile users worldwide in 2017 are posing as an enticing target for malware creators, that have evolved frequency and sophistication of attacks. According to McAfee [3], in 2010, best malware campaign could earn up to $300,000. Nowadays, it could potentially bring a revenue up to $2 million, being able to reach the billion-dollar figure by 2020 [3].

Android users are becoming more vulnerable and exposed to risks, posing unwillingly as enticing targets to cybercriminals, involved in an extremely threatening situation. Consequently, there is an important need of improvement in Android malware detection in order to mitigate this fast-growing critical risk scenario for Android users.

Traditional malware detection approaches based on signatures fail to address new malware detection and can be easily bypassed by old malware with obfuscation or other stealth techniques [6]. New methods should be implemented to overcome these limitations. Machine learning is a growing and emergent field in computer science that involves the ability of a computer to learn from experience without the need of being explicitly programmed to perform each action. Machine learning has been firstly tested in the computer security field, and lately in mobile security with promising results, it can help to improve malware detection mechanisms in mobile devices [9][10][11].

Machine learning models use data features of known data objects to make computers learn and, based on that, make predictions about unknown data objects. Relevant features about data objects should be selected from whole data features and prioritized to achieve maximum prediction accuracy and avoid information redundancy.

Present research problem statement is resumed in the following points:

- Perform feature selection for malware detection in Android system.

- Minimize number of predictors.
- Analyze the variability of selected predictors in old and new malware datasets.

This thesis aims to apply standard machine learning acquisition workflow to address malware discrimination problem in Android, by selecting potentially discriminatory features using a hybrid features approach, acquiring static and dynamic data such as Android permissions and system calls from applications. Hybrid features are the most comprehensive as they analyze the phenomena from various perspectives [12]. Android permissions are the most static features used to analyze malware [12], they are explicitly declared in AndroidManifest.xml file. As Android uses Linux Kernel, permissions are the first obstacle to attackers. System calls are the most dynamic features used to analyze malware [12]. System calls are used by applications to perform and request specific tasks since they are not authorized to interact directly with Android operating system [12]. After reviewed the state of the art of the application of machine learning to the malware detection in mobile field, an Android sandboxed testing environment was deployed using the conjunction of Linux O.S. and an Android emulator to gather both static and dynamic features from applications. Feature selection was applied to find the variables candidates of possessing higher discriminative power, which is a low-applied technique in mobile malware detection and also testing the cumulative power of hybrid features. Machine learning models were trained with different dataset combinations, from the whole dataset composed by 1000 legitimate, 1000 old malware and 1000 new malware and cross-validated, which supposes a novelty in model validation, using both old and new malware samples. Accuracy is shown as performance's metric and evaluation. Our main contribution to this field is two-fold: application of feature selection in detail for the considered features (system calls and permissions) and perform detailed analysis for old and new datasets.

This document is structured as follows: section 2 deals with thesis' background information and related work; section 3 focus on present research methodology; chapter 4 talks about this thesis' practical implementation of machine learning workflow; chapter 5 shows validation metrics and main results while chapter 6 addresses the conclusions that can be drawn from this research.

2 Background Information

2.1 APK and Android Malware

Android applications are installed in mobile devices by using an installation file called Android Application Package or APK. APK files are a type of archive files, like zip packages or java jar files, that consists in the following main components:

- AndroidManifest.xml → Application's meta-data XML file. It includes information related with application's descriptions, package information and security permissions. Security permissions is the access control that Android uses in order to provide the app access to system data and features that it needs. Permissions should be declared before they can use system data and features and depending on how sensitive the data is, they will require explicit user approve the request or be automatically granted [13].
- Classes.dex → File that contains the source code of an Android application written in Java programming language compiled into *.dex* format (Dalvik Executable). Dalvik is Android platform's virtual machine that interprets and executes *.dex* files, which are optimized for efficient storage and memory-mappable execution. [14].
- Resources.arsc → binary XML file that contains precompiled application resources.
- Resources folder (res/) → Folder that includes not pre-compiled resources that application needs on runtime such as pictures, layout, use of a database and data stored in the database, etc. [12].
- Assets (assets/) → optional folder that contains application assets that can be retrieved by *AssetManager*.
- Libraries (lib/) → optional folder that contains compiled code that is specific for different processors such as arm, mips, x86, etc.
- META-INF → folder that contains MANIFEST.MF file, APK signature, etc.

APK files zip all these files and folders into *.apk file which is used by Android to install the application. This *.apk file can, with appropriate tools, be used to collect interesting information for analyzing the application with static malware analysis procedures, that is, without running or installing the application in any device. Dynamic malware analysis requires installing and running the application in a device.

Android malware can pose multiple faces and threats. From the annoying but harmless adware to sophisticated malware being able to hijack the mobile device and access personal data [12]. In last years, there has been an important increase of profit-motivated malware, mainly based in the form of premium messages sent by applications without the will and awareness of the device user [12][15]. In relation with the social engineering method used for its installation, Android malware can be mainly categorized as repackaging, update attack and drive-by-download. But they are not mutually exclusive, as malware can use different techniques to entice the user for download [15]. Repackaging consists in the addition of malicious payload in a legitimate (non-malware) and popular application. Malware authors download most-downloaded popular applications from legitimate sources, disassemble them and attach malicious payload to it, reassemble everything together and then submit them to official or alternative Android markets. When the user downloads and installs this legitimate-looking application it also installs malware in the device without notice. Over 80% of Android malware is a repackaged application [15]. Update attack is a step-forward in malware sophistication. Instead of attaching the whole payload into the application code, it only encloses an update component that will download the entire malicious payload at app's runtime. This makes malware detection more difficult as static scanning will not detect malicious payloads [15]. Malicious payload is included in the "updated" app and not in the original one, which makes malware detection not efficient in first instance. Finally, drive-by-download is a traditional social-engineering technique applied to a new field, mobile devices. It consists in entice the user to download interesting apps [15], which will perform other actions than the expected. As stated before, Android malware shows evolving sophistication and quick development that make them harder to detect [15]. Mobile anti-virus software can barely detect over 70% of malware, and in some cases 20% of them. Development of new anti-mobile-malware solutions is an imminent need [15].

2.2 Machine Learning

Machine learning claims to make computers *learn from experience*. "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E" [16]. As an example, Arthur Lee Samuel, a pioneer in Artificial Intelligence field and the one who coined the term "machine learning" in 1959, created one of the first successful AI self-learning programs. Such a computer program that learns to play checkers (T) "might improve its performance *as measured by its ability to win checkers games to win* at the class of tasks involving *playing checkers games*, through experience *obtained by playing games against itself* [16]. Since that, machine learning applications have shown a huge expansión. Now, machine learning is everywhere. Spam filters, credit card fraud detection, picture's face recognition, search engines, recommendation systems, data mining and handwriting recognition are just an small example where machine learning is being applied nowadays. Most of machine learning problems and algorithms fall in one of the following broad types:

- *Supervised learning*: where an outcome variable (dependent variable, y) is predicted based on a set of predictors (independent variable, x). More concretely, it involves the machine learning task of inferrinf a function from labeled training data [17]. Labeled (also known as target or outcome) data is used to train a computer (learn from known labeled samples) that will be used predict the label of new unknown samples of data. Here, training examples or instances are used to train the model, in relation to specific attributes, also known as variables or features. Label of new data will be inferred from new data attributes according with learnt ones. According to the type of predicted variable, supervised machine learning algorithms can be classified as:
    - Classification: when a categorical data is predicted (e.g.: spam email).
    - Regression: when a numerical measurement is predicted (e.g.: price of housing).

- *Unsupervised learning*: where the expected outcome is to find the hidden structure of unlabeled data trying to characterize it. Here, contrary to supervised learning, there are no labels or target and the goal is to find and describe patterns

and associations among attributes [17]. Most used unsupervised learning algorithms are known as:

  - *Clustering*: where instances are grouped based on sharing certain common characteristics that make them similar.
  - *Outlier detection*: where the goal is to find data points that deviate markedly from other members of the same sample.

- *Semi-supervised learning*: is the combination of supervised and unsupervised learning. It is a technique to learn patterns in the form of a function based on labeled and unlabeled training examples [17]. It is mainly used when it takes important efforts to collect labeled data, needing to use unlabeled training examples to learn a target function.

- *Reinforcement learning*: tries to find optimal actions in a concrete situation so as to maximize a numerical reward which does not come inmeditaly with the actions but delayed in time. [17]. Actions that yield highest reward are not provided, they must be discovered by trying them (test-error).

2.3 Related Work

Machine learning models are pretended to overcome the limitations of the signature-based methods to discriminate unknown malware from benign applications, a critical detection that threatens Android users. Applications static and dynamic features have been used in recent times to train, test and validate machine learning models with promising results [18]. Static features are easy to extract and do not require to install or run the application but are prone to be lured by obfuscation methods. On the other hand, dynamic features provide more comprehensive information but they require to install and run the application in a rooted device and are more difficult to extract. Hybrid approaches provide the best set of information as they combine the advantages of both approaches but they also sum up the complexity of extracting both types of features and their combination.

2.3.1 Static malware analysis and detection

Static malware analysis involve the use of application's static features which can be collected without actually running the application, directly from *.apk file or with a

little manipulation. Android permissions are the most used static feature in static malware analysis research papers. Android, running Linux kernel, implements an important part of Linux security architecture.

A list of requested permissions is presented to the user before installation (Figure 1), that will be performed once permissions are granted by the user [12]. There are 147 official Android permissions in Android 8.1 [19], also called as Android Oreo (API 27), that are categorized into three protection level groups: normal, signature and dangerous [20]. Researchers used permissions in different ways, usually combined with other static features, as malware detection is easier to bypass by malware if only one feature set is used [21].

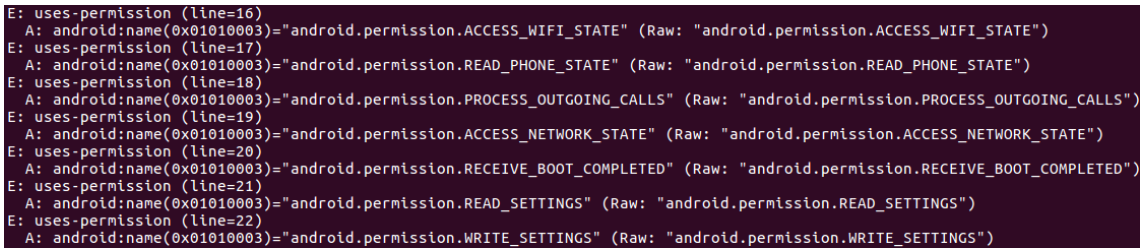


Figure 1. Permission declaration inside AndroidManifest.xml

Arp et al. [9] used a combination of static features such as requested permissions, hardware components, app components and filtered intents from application's manifest, and restricted API calls, used permissions, suspicious API calls and network addresses from disassembled code. They were embedded into a joint vector space and applied linear Support Vector Machines (SVM) algorithm to discriminate malware/bening applications. 93.9% malware detection accuracy was achieved with false positive rate of 1% with full Drebin dataset (123,453 benign applications and 5,560 malware samples) and 95.9% with specific MalGenome Project dataset [15] which is embedded in full Drebin dataset.

Peiravian and Zhu [22] combined requested permissions and API calls as features to characterize malware, trained and tested three machine learning algorithms (Support Vector Machines, Decision Tree and Bagging) with the collected features in 3 different sets (only permissions, only API calls and combination of both). Their dataset was composed by 610 malware samples and 1250 benign apk files. Best results were

achieved using Bagging classifier combining both features in each of the tests (AUC 0.991).

Nezhadkamali et al. [21] used a combination of permissions, API functions and intents. They used feature selection to find a subset of relevant features from their dataset. Their dataset was composed by 1260 malware applications from well-known MalGenome [15] dataset and 498 benign applications. They used various machine learning techniques including SVM, Random Forest and Decision Tree algorithm. Best results were achieved using information gain and Random Forest algorithm (98.6% accuracy).

Liu and Liu [23] used *pairs* of used and requested permissions by applications in order to classify malware/benign applications. They used the frequency of occurrence of two permissions as a pair, stating that it can reflect the app's potential malicious activities. Their dataset was compound by 28,548 benign apps downloaded from AppChina and 1,536 malicious apps, including MalGenome [15] and other samples collected by the authors in a security company in China. Trained and tested Decision Tree algorithm 3 times: one with requested permissions, another one with requested permissions pairs and last one with used permission pairs. First two classifiers are first layer and last one is second layer. An application categorized as malware in any of the layers was categorized as malware and not analyzed in following layer. They achieved an accuracy of 0.985 in first layer and 0.986 in the whole process (first and second layer).

APK Auditor [24] is a permission-based Android malware assessment system. It consists in three main components: an Android client, a signature database and a central server that communicates with both and handles the analysis process. Dataset was conformed by 6909 malware samples (from Contagio [25], Drebin and MalGenome datasets) and 1853 benign applications downloaded from Google Play Store. They trained and evaluated logistic regression algorithm. Overall accuracy of the system was 88.28%.

DroidMat [26] extracts requested permissions, intent messages passing and deployment of components from AndroidManifest.xml. Additionally, they extract API calls for each component. Dataset consists on 238 malware applications from Contagio dataset and 1500 benign applications downloaded from Google Play. Applied K-means, EM

clustering algorithms and k-Nearest Neighbors (kNN, with k=1) and Naïve Bayes as classifiers. Best results were achieved with combination of k-means and kNN (0.9787 accuracy).

Aung and Zaw [27] used permissions as a unique feature to detect malware. They performed feature selection using information gain, used k-Means as a clustering algorithm and Random Forests, J48 Decision Tree and CART as classifiers. Their dataset was composed by 500 samples. Best results were achieved using Random Forests algorithm (91,75% accuracy).

PUMA [28] used permissions as a unique feature to detect malware. They extracted them from the application's manifest, processed it and trained different machine learning algorithms (SimpleLogistic, NaiveBayes, BayesNet, SMO PolyKernel and NormalizedPolyKernel, IBK, J48, RandomTree and RandomForest). Validated their findings with k-fold cross validation (k=10). Their dataset was composed by 249 malware applications from VirusTotal and 1811 legitimate applications from Play Store. Best results were achieved with Random Forests with 50 trees (86.41% accuracy).

Droid Detective [29] uses permission combinations to detect malware. It groups permission combination profiles that are requested frequently by malwares but rarely by benign applications and generates rule sets for identifying malware. Their dataset is composed by 1260 malware samples from MalGenome Project and 741 benign applications collected fom Google Play. Best results are achieved when using combination of 6 permissions (87,53% accuracy).

Varma P. et al [30] used permission as a unique feature to detect malware. Their dataset is composed by 1999 legitimate applications downloaded from Google Play between 2015-2016 and 1259 MalGenome Project malware samples. They trained and tested Naïve Bayes, Decision Tree J48, Random Forest, Multi-class classifier and Multilayer perceptron algorithms. Best results were achieved with multi-class classifier (99,9% accuracy).

2.3.2 Dynamic malware analysis and detection

Dynamic malware analysis involves the collection of information about the application at runtime, so it requires to install and monitor the application to collect desired data. Dynamic features can be defined as application's behavior in interaction with the operating system or network connection [12]. System calls are the most used behavioral feature in dynamic malware analysis research papers. Android, which runs Linux kernel, has more than 200 system calls available [31] that are used by applications to request services that they are not allow to perform directly from the operating system's kernel. System calls, also called kernel calls, provide functionalities such as network, file or process related operations [32]. As shown in Figure 2, when an application running on user space requests a service from the operating system (i.e. open a file) using wrapper functions (such as open() and not invoking the system call directly), the request is interpreted by *glibc* library and CPU switches from user mode to kernel mode in order to execute the appropiate kernel function by looking into system call table. Kernel, which has enough privileges to perform the task, understands the petition and makes the request to the hardware. When the concrete task is performed, the result is sent back to user space following the inverse order. As all requests go through upper layers to Kernel before they are executed in hardware via system call interface, monitoring and capturing system calls passing through system call interface provides a good source of information about the behaviour of the application [32]. This task is usally performed using *strace*, a debugging and diagnostic utility for Linux. This tool is used to monitor interactions between Linux kernel and running processes, providing information about system calls, signals and changes of process state [33]. *Strace* tool uses at the same time a kernel system call *ptrace*, that allow process tracing [34].

In Android OS this process is slightly different and system calls are created by information flowing through a multi-layered architecture, as can be seen in Figure 3. When application on top architecture layer makes a request, that is transformed to the corresponding service in the application framework. Next, Android runtime receives the request from the service and executes it in the Dalvik VM. Upon execution, request is transformed into a collection of library calls, which result in multiple system calls to the Linux kernel. This generated sequence of system calls is low-level equivalent of the

high-level request. Flow of information goes to the opposite direction in a similar fashion [35].

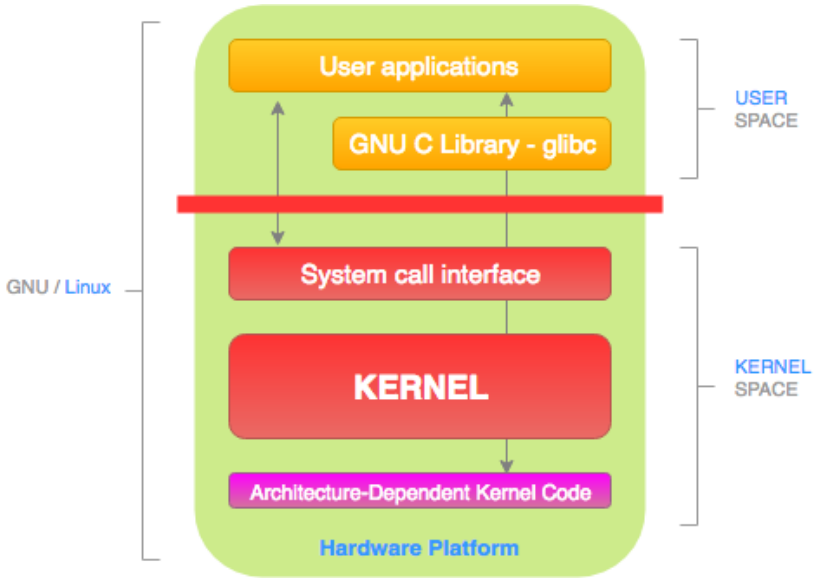


Figure 2. Linux User and Kernel space [32].

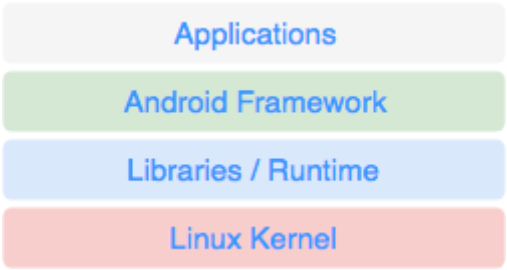


Figure 3. Abstraction layers of the Android architecture [35]

Crowdroid [32] uses system calls to detect anoumalosly behaving applications in the form of trojan horses. It uses crowdsourcing to obtain the traces of application's behavior. A client, server and database architecture is used in order to process and gather information. Crowdroid client, installed on user device, acquires data, by running strace and sending the log file via internet to the server/database. The server/database receives the log file, extracts system calls names and creates a system call feature frequency vector (each element represents a count for the specific system call requested). This system call feature vector is used as input of k-means clustering algorithm, in order to create the normality model and detect anomalous behavior. Dataset is composed by self-written malware and two malware applications from VirusTotal (Steamy WIndow and Monkey Jump 2) and legitimate versions of them.

Detection rates are 100% in self-written malware and Steamy Window, 85% in the case of Monkey Jump 2.

CSCdroid [36] uses strace to collect system calls produced by 1000 Monkey tool pseudo user events in order to detect malware. This system calls are categorized using different methods and Markov chain is used after to construct feature vectors that are the input of SVM classifier algorithm. Dataset is composed by 1189 benign applications and 1227 malicious applications from MalGenome dataset. Precision and *True Positive Ratio* are over 0.95 in all categorization methods used.

MALINE [35] uses system call acquisition and process to discriminate malware applications. It uses strace to collect system calls in a sandboxed environment using strace tool. Strace log is then processed creating two feature vectors: one representing system call frequency and another one representing system call dependency (pairs, using dependency graphs). The number of system call was controlled by the number Monkey events inserted (ranging from 1 to 5000). These vectors are used as input for training, tested and cross-validated with different machine learning classifier algorithms such as SVM, Random Forest, LASSO and Ridge regression. Dataset is composed by legitimate applications downlodaded from Google Play and Drebin dataset as malware. Best results were achieved with random forest algorithms in all cases (with accuracy results over 90%).

Maestre Vidal et al. [37] used sequence alignment and system calls to discriminate malware/benign applications during the boot process of the application. They used strace tool to monitor program's activity and assigned each system call a symbol. They extract sequences of system calls and applied sequence alignment methods to them. The analysis of the monitored data is driven by sequence alignment processes, which compare the received sequences with a collection of samples of legitimate application executions. The more completeness this dataset presents, the greater precision the system offers. Malware dataset were 5130 samples from MalGenome and Drebin and 570 benign applications. Best results were achieved considering samples of the first 2000 system calls gathered when initializing the applications within the sandbox. True Positive Ratio was 98.61% with False Positive Ratio of 6.88%.

Xiao et al. [38] collected applications' system calls during the execution of 1000 Monkey testing tool injected events. Information gathered is processed to create two vectors: frequency vector (each element represents the frequency of the corresponding system call) and co-ocurrence matrix vector (co-ocurrence matrix is created from the system call sequence, normalized and transformed into a vector). Both vectors are used as input for Adaptative Regularization of Weight Vectors (AROW), kNN, Logistic Regression, Naïve Bayes, Decision Tree, Random Forest and SVM classification algorithms. Dataset is conformed by 1227 MalGenome malware dataset samples and 1189 legitimate applications from Google Play. When using co-ocurrence matrix vector, TPR values for all machine learning algorithms achieved values over 0.95. On the other side, Random Forest was the only algorithm that achieved TPR over 0.95 regarding frequency vector input.

SCSdroid [39] focus on the use of system calls to discriminate malicious repackaged applications (MRA's) without the need of the original application. SCSdroid (System Call Sequence Droid) uses thread-grained system call sequences during runtime rather than process-grained sequences. SCSdroid authors advocate that if an MRA can be camouflaged as a benign application, its malicious behavior would still appear in the thread-grained system call sequences. A thread-grained system call sequence is the system calls recorded for a thread, while a process-grained system call sequence is the system calls recorded for a process. That is, the thread-grained system call sequences mean that the system calls produced by the process and the child-threads forked from it are independently recorded, while process-grained sequence means that the systemcalls produced by the process and all its child-threads are recorded together. Authors state that since malicious behavior always happens in a single thread, not across multiple threads, it is difficult to identify the malicious behavior if the system calls from the process and different threads are mixed together. SCSdroid first captures the system call sequence of each thread at executing MRAs, using strace tool attached to main Android process called *Zygote*, and then extracts the common subsequences (using Longest Common Substring algorithm), which are the common parts of these captured system call sequences. These extracted common subsequences can be only regarded as possibly malicious behavior of MRAs because they may also exist in benign applications. After that, Bayes Theorem is adopted to filter these non-discriminating common subsequences and then find the common subsequences which indicates the truly

malicious behavior presenting in the MRAs. The key concept of SCSdroid is that MRAs belonging to the same family, i.e., a group of MRAs which embed the same malicious codes into benign applications, will have common malicious behavior. Dataset is compound of 8 families of MRA's from different malware sources and 400 benign applications downloaded from Google and third-party markets, tested with VirusTotal [40]. Overall detection accuracy was 95.97%.

Afonso et al. [41] combined system calls and Android API function calls in order to discriminate malware from legitimate applications. They used MonkeyRunner tool to create some pseudo random events and recorded Android API function calls with APIMonitor [42] software and system calls with strace tool. They focused their analysis on frequency of 74 Android API function calls and 90 system calls and created input vectors to feed several machine learning algorithms (RandomForest, J.48, SimpleLogistic, NaiveBayes, BayesNet Search algorithm, SMO Kernel and IBk). Results were 10-fold cross-validated. Dataset is composed by 4552 malware samples from MalGenome and other sources and 3831 benign applications from AndroidPIT market. Best results were achieved with Random Forest with 100 trees (95.96% accuracy).

Wahanggara and Prayudi [43] focused on the use of system calls and *Support Vector Machines* algorithms to address the problem of malware detection. Dataset was formed by 150 applications. Malware came from Contagio and legitimate applications were downloaded from Google Play Store, with restrictions of 4 out of 5 starts rating and mínimum of 100,000 users. Strace tool was used to gather applications' sytem calls three times for each application. Data was used to train and test SVM algorithm with different kernels: radials basis function (RBF) kernel and polynomial kernel. 5-fold cross validation was performed. Best results were achieved with polinomial kernel, reaching 90% of accuracy.

Andromaly [44] is a behavioral Android malware detection framework that uses applications' behavioral traits (system calls included) at different levels in order to discriminate between malware and legitimate applications. The basis of the malware detection process consists of real-time, monitoring, collection, preprocessing and analysis of various system metrics, such as CPU consumption, number of sent packets

through the Wi-Fi, number of running processes and battery level. All data gathered is selected and reduced by applying feature selection in order to apply different machine learning algorithms more efficiently: k-Means, Logistic regression, Histograms, Decision Tree, Bayesian Networks and Naïve Bayes. Dataset was composed of 44 applications (20 benign games, 20 benign tools and 4 specific malicious applications). Best results were achieved with Naïve Bayes and Decision Tree algorithms, over 87% of accuracy.

Da et al. [45] focused their study on 23 selected system calls related with user rights in categorized samples of applications (categorization according Google Play Store). They collected frequency of each system call and normalized it. Then applied them as input vector of Random Forests algorithm. Dataset was composed by 67 benign samples downloaded from Google Play Store and 51 malware samples from Contagio dataset. They performed 10-fold cross validation. Best results were achieved with Random Forest 200 trees (98.03% of accuracy).

Multi-Level Anomaly Detector for Android Malware or *MADAM* [46] monitors device actions, its interaction with the user and the running apps, by retrieving five groups of features at four different levels of abstraction, namely the kernel-level (system calls), application level, user-level and package-level. Depending on the features, it applies signature-based approach detection or anomaly-based. Regarding to system calls, it collects type and amount of the system calls issued, focusing on file operations and network system calls (11 system calls) and detecting anomalous behaviour. Detection used two parallel kNN classifiers and a behavioral signature-based detector. Malware dataset was formed by 2784 applications from MalGenome, Contagio and VirusShare. Benign dataset was formed by 9,804 applications downloaded from Play Store, tested with VirusTotal. They achieved an accuracy of 96.9%.

Deep4MalDroid [47] focuses on the use of system call graphs to detect malware in Android. This different approach uses a novel method called Component Traversal that allows the automated execution of code routines of each Android app as completely as possible (used to overcome the limitations of ADT Monkey pseudo injected events). Then they extract Linux kernel system calls (using strace) and created weighted directed graphs (system call pairs sequences that also take into account the frequency of each

system call). These are used as inputs on a deep learning algorithm (Stacked AutoEncoders architecture), used as a classifier. They also tested this approach with other machine learning algorithms such as SVM, Artificial Neural Network, Naïve Bayes and Decision Tree. 10-fold cross validation was performed. Dataset is composed by 3000 applications from Comodo Cloud Security Center: 1500 are malware and 1500 legitimate applications. Best results were achieved with Deep learning, composed by 3 layers with 200 neurons each (93,68%).

Singh and Hofmann [48] monitored system call behaviour of 278 legitimate applications downloaded from Google Play and 216 malicious applications from Contagio dataset. System calls' frequencies vector was used to feed seven machine learning classifiers (Decision Trees, K-nearest Neighbors, Random Forest, Gradient Boosted Trees, Support Vector Machine, Neural Network and Deep Learning.). They performed two experiments: one with 337 system calls vector and other with selected system calls based on three feature selection methods (Chi-square, information gain and correlation). 10-fold cross-validation was performed. Best results were achieved with SVM algorithm using correlation (97.16%).

Ferrante et al. [49] combined memory usage, CPU and system calls to find malicous sub-traces leading to discriminate malware from legitimate applications. In the learning phase, they used clustering (KMeans++ algorithm) and then applied Random Forest algorithm as a classifier. Dataset was formed by 1709 benign applications downloaded from Google Play and 1523 malicious samples from Drebin dataset. Only selected system calls were analyzed, from a prior study [49]. Best results were achieved with Random Forest 50-500 trees (67% of accuracy).

Canfora et al. [50] focused on the collection system calls sequences generated by applications when pseudo random events were injected (using Monkey) during 60 seconds. They also tested frequencies of system calls issued during that specific amount of time (not sequences). SVM was trained and tested. Dataset is conformed by 1000 legitimate applications from Google Play and 1000 malware applications from Drebin. Alternatively, they also tested permissions alone as a malware detection method. Best results were obtained by using long sequences of system calls, achieving 97% of accuracy.

2.3.3 Hybrid malware analysis and detection

Hybrid malware analysis combine the use of both static and dynamic features as a group, used together to detect malware. Although it requires more complex processing of the malware samples, they are the most comprehensive features as they analyze both application installation file and application behavior at runtime [12]. This approach is less profuse in scientific research than prior approaches.

Xiao et al. [51] used permissions (static), system calls (dynamic) and control flow graphs (static) to detect malware. They tested different combinations of the features: each one independently, combination of all features and also combination of the two static features. They used collected data vectors as input to AROW and SVM algorithms. Dataset 1188 benign applications downloaded from Google Play and 1179 malicious applications from MalGenome Project. Best results were achived using combination of three features with AROW algorithm (TPR of 0.9905 with FPR of 0.0156).

MARVIN [52] is a system that learns to distinguish malicious from benign apps based on a set of known malware and goodware. It assigns malice scores to unknown apps in a range from 0 (benign) to 10 (malicious). It collects dynamic and static analysis, network-level behavior and meta-information, like author fingerprints and application lifetime. Core of the system uses one of two machine learning options: linear classifier or SVM. They perform feature selection in order to reduce the number of features. As a result of the learning phase, MARVIN computes malice scores of given application samples. Dataset is composed by 124189 applications (10% of them labeled as malware coming from MalGenome, Contagio and VirusTotal). MARVIN achieved 98.24% of malicious apps with less than 0.04% false positives (using SVM algorithm).

Scientific research in Android malware detection is, as can be stated from this literature review, profuse and varied in methods. Nevertheless, there are two main points missing:

- There is no feature selection or analysis in most of the studies. Only a tiny amount of studies perform it and they do not provide detailed analysis on how they select, analyze or choose appropiate features.

- There is no comparison between old and new malware. Most malware datasets used to test and train are relatively old, with samples from 2010 to 2012 in most cases. They do not test cross-malware testing, with new samples of malware.

This present research will focus on this two main lacking points detected on literature review.

3 Methodology

This master' thesis was developed in three phases: data acquisition, data pre-processing, data processing and model validation. Briefly, data collection involves the collection and gathering of all possible features from dataset while in pre-processing part features are filtered using feature selection methods to select best features, eliminate redundancy and avoid over-fitting. Finally, machine learning algorithms were trained and tested with input vectors created from dataset, 70% of dataset was used as training data and 30% as test data, to validate the model.

3.1 Phase 1. Data Acquisition

3.1.1 Dataset

Dataset used in this thesis was downloaded from different sources, according to its characteristics and usage. Dataset is composed by 3000 Android apps split as follows:

- 1000 benign apps were downloaded randomly from APKMirror [53].
- 1000 malware apps were selected randomly from Drebin malware dataset [9].
- 1000 malware apps were selected randomly from VirusTotal Academic malware samples dataset [54].

Altough APKMirror provides signatures and file hashes in order to verify and ensure that all their applications are trusted and legitimate applications, all benign applications were tested with VirusTotal malware scanner in order to verifiy that they were not detected as malware by VirusTotal scanner. All downloaded samples supported x86 architecture and were top-popular Android applications released between 2017 and 2018.

Drebin malware dataset [9] is composed by 5560 applications from 179 different malware families. These samples were collected between August 2010 and October

2012. From 5560 applications, random applications were chosen that supported x86 CPU architecture. This malware dataset will labelled as *old malware dataset*.

VirusTotal Academic malware samples is composed by 10,908 applications. These samples were detected and collected by VirusTotal between 2017 (3212 samples) and 2018 (7696 samples). From whole dataset, random applications were chosen that supported x86 CPU architecture. This malware dataset will labelled as *new malware dataset*.

3.1.2 Android Emulation

Each sample of the dataset was installed, executed, monitored, logged and uninstalled using an Android emulator software running over a Linux environment. For this experiment, an Ubuntu 16.04 LTS (Xenial Xerus) 64-bits OS was installed on an Intel Core i5-2450M CPU @ 2.50Ghz x 4 with 6 GB of RAM. Android emulation tool used was GenyMotion 2.11 (which uses Virtualbox as virtualization tool). Android Studio 3.0.1 was installed in the environment, in order to use Android SDK Tools with GenyMotion software. Android SDK Tools are a complete set of development and debugging tools for Android. The only restriction that GenyMotion imposed to applications is CPU architecture: x86 support is needed in order to run the application in GenyMotion software (ARM and other architectures are not supported). Altough Android Studio 3.0.1 includes an emulation tool, Android Virtual Device Manager, it is not rooted by default, slower and with more lag than GenyMotion (which is rooted by default and lag-free). GenyMotion x86 architecture is great for performance but do not allow to test ARM applications, as there is no option to change architecture [55]. Nevertheless, most of Android applications have x86 architecture version additonally to default ARM architecture. All applications from whole dataset were installed and tested in an emulated Samsung Galaxy S8 device running Android 7.0 (codename *Nougat*, API 24), as can be seen in the screenshot below.

3.1.3 Application's feature: system calls

System calls or kernel calls are used by applications to request services from the operating system that it directly cannot achieve. Available system calls are listed

numerically in a table (syscall_table) inside the kernel, linking a specific number with each specific system call. Although not all system calls provided by Linux kernel are supported on all architectures or platforms or some of them are deprecated, there are always more than 200 calls available [56]. These more than 200 system calls can be used to perform different tasks and get desired output by applications and operating system itself [57]. Linux architecture provides a layer over the kernel that provides essential *core libraries* to perform such actions via API calls. They include system calls for process management, time operations, system handling, filesystem etc. The best-known and most used library is GNU C library, also known as *libc* or *glibc* library [58]. Nevertheless, although Android operating system is built on the Linux kernel, it is not Linux as it does not support glibc and does not include the full set of standard Linux utilities among other major differences [59]. Android OS uses standard Linux Kernel with a patch of added "kernel enhancements" in order to provide some Android specific features such as power management and inter-process communication (IPC) binder [59]. As Android does not provide *libc* support, it uses a customised and optimized libc implementation for embedded use, known as *bionic* [59], as native library. Faster and smaller than *glibc*. It uses other native libraries such as SSL, SQLite, WebKit, etc. to perform needed tasks.

For this thesis purpose, Android 7.0 device was emulated and system calls were collected using *strace* tool. Strace tool can be used to trace system calls by attaching the tool to a running process. The output of strace is a log file that provides the system calls performed by the process during its execution (while *strace* was attached to it). Strace uses at the same time a system call to perform this action, *ptrace* [60]. Android 7.0 *Nougat* is built on Linux Kernel 4.1. For this thesis purpose, first 2000 boot up application system calls where collected and analyzed. Only Bionic x86 Nougat implementation system calls were collected, summing up 212 available system calls [61]. After collection, frequency vector (count of each system call) was created and used as input to feed a machine learning classifier model.

3.1.4 Application's feature: Permissions

Permissions are used in Android OS as a privacy resource to protect Android users from unwanted actions. Applications must request permission to access sensitive user data

(e.g.: contacts and SMS) and also for certain system features (e.g.: camera and internet). Depending on the feature requested, Android grants permission automatically or prompts the user to approve the request [20]. Since Android 6.0, permissions are given by users to applications at runtime, not before installation. Permissions are divided in two categories: normal and dangerous. While a normal permission does not threatens user privacy directly, permission is automatically accepted by the system without the user awareness; dangerous permissions are those that could lead to the access of sensitive data from the user, requiring the explicit acceptance of them by the user. Permissions are located and explicitly defined in manifest file (AndroidManifest.xml), requiring dangerous permissions the user explicit acceptance when running the application. The user can accept or deny the permission request without stopping the application, which will run with limited capabilities.

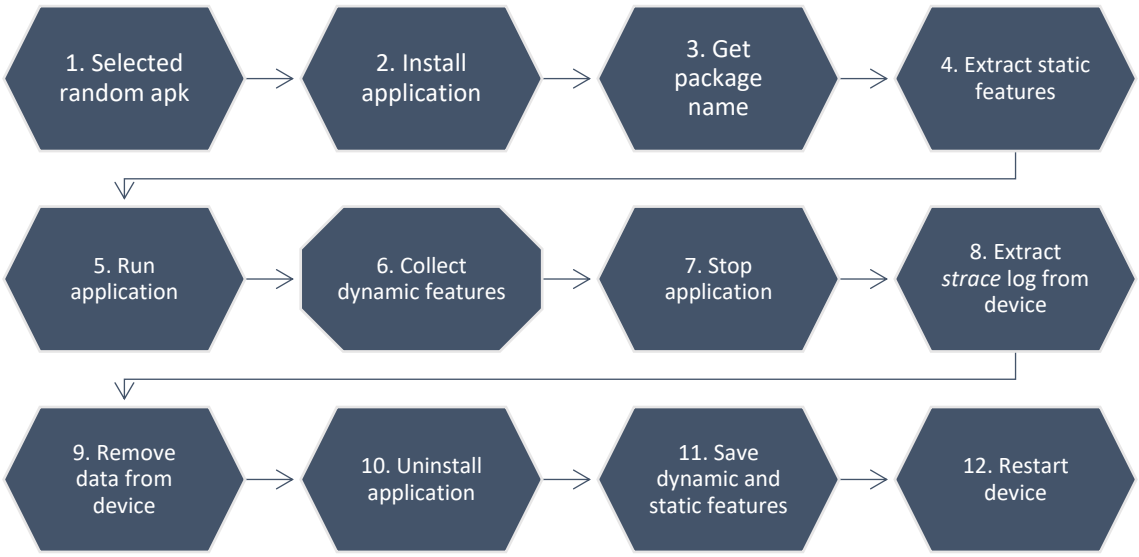
For this thesis purpose, requested permissions were collected from AndroidManifest.xml file (included in every apk file) using Android Asset Packaging Tool (aapt). A log file was created containing the requested permissions and analyzed for each application. Android OS has 147 standard permissions [19]. Custom permissions can be defined by app developers in order to share resources and capabilities with other apps [62]. Only standard permissions were collected and analyzed from dataset applications. After collection, a permission profile vector (binary codification of each application set/unset) was created and used as input to feed a machine learning classifier model.

3.1.5 Feature extraction

More concretelly, feature extraction was performed using two tools included in Android SDK Tools: Android Debug Bridge (ADB) and Android Asset Packaging Tool (aapt). Android Debug Bridge (ADB) is an Android SDK Tool used to communicate with Android devices (virtualized or via USB port) that allows the user to perform multiple features directly on the device via a command terminal from the computer: install application packages, uninstall them, execute commands, etc. For this project, it has been used to install and uninstall apk files and also to run *strace* tool command. Everything was done with an automated script using bash programming language. Aapt tool has been used to extract information from *AndroidManifest.xml* file, included inside all apk formatted files.

3.1.6 System calls and permissions' collection

An automated bash script was used to perform the following process over dataset apk's containing folder:



This process was performed for all 3000 random dataset samples that were compatible with x86 architecture. Not architecture-compatible apps were discarded at step 2, being unable to install them. This process was performed until 1000 legitimate, 1000 *old malware* and 1000 *new malware* applications with 2000 boot syscalls were collected.

1. Selected random apk from folder (using randomising function).
2. Install application using Android Debug Bridge (adb) from Android Sdk platform-tools folder (install command).
3. Get package name from AndroidManifest.xml using Android Asset Packaging Tool (aapt) from Sdk build-tools folder and regular expression.
4. Extract static features (requested permissions) from AndroidManifest.xml using aapt and regular expression. Saved in <package name>.perm file as plaintext.
5. Run application using *monkey* tool (executing Android.intent.category.LAUNCHER).
6. Collect dynamic features (first 2000 system calls) using *strace* tool attached to the main process run by *monkey* executed package name.
7. Stop application using SIGKILL to application's main process.
8. Extract *strace* log from device to computer using *adb* tool *pull* option.

9. Remove *strace* log from device.
10. Uninstall application using adb (uninstall command).
11. Save dynamic and static features into classified folders according to package name and randomized number.
12. Restart device.

After whole dataset information was collected, another script was created to read and collect features data. Regarding system calls, information about all applications was stored as CSV file, including package name, malware/legitimate classification and frequency of each system call. Regarding permissions, another CSV file was created to store information about this feature. It included package name, malware/legitimate classification and binary codification of each present/absent permission.

3.1.7 Statistical hypothesis testing

Malware detection relies on the assumption that legitimate and malicious applications have differential characteristics that allow us to discriminate between them. In order to know whether this assumption is correct or not, providing rationalization of later steps, statistical hypothesis testing is required. Statistical hypothesis testing involves three main steps [63]:

1. Making an initial assumption.
2. Collection of evidence (data).
3. Based on the collected and available evidence (data), decide if initial assumption was true or not.

3.1.7.1 System call hypothesis

Our initial assumption suggests that malware could be detected using system calls and permissions. In order to know if that discrimination is possible, our first step is to check wheter both kind of applications differ significantly in each of these parameters. That is, if system calls and permissions are significantly different. Regarding system calls, as a frequency count vector is used, means are compared, that could be translated that the mean of each system call should be sensitively different from malware dataset to

legitimate dataset, allowing such a discrimination of samples. For this thesis purpose, two competing hypothesis are stated for each system call in each of composed datasets:

H_0: $\mu_L = \mu_{OM}$ $\quad\quad$ H_0: $\mu_L = \mu_{NM}$

H_A: $\mu_L \neq \mu_{OM}$ $\quad\quad$ H_A: $\mu_L \neq \mu_{NM}$

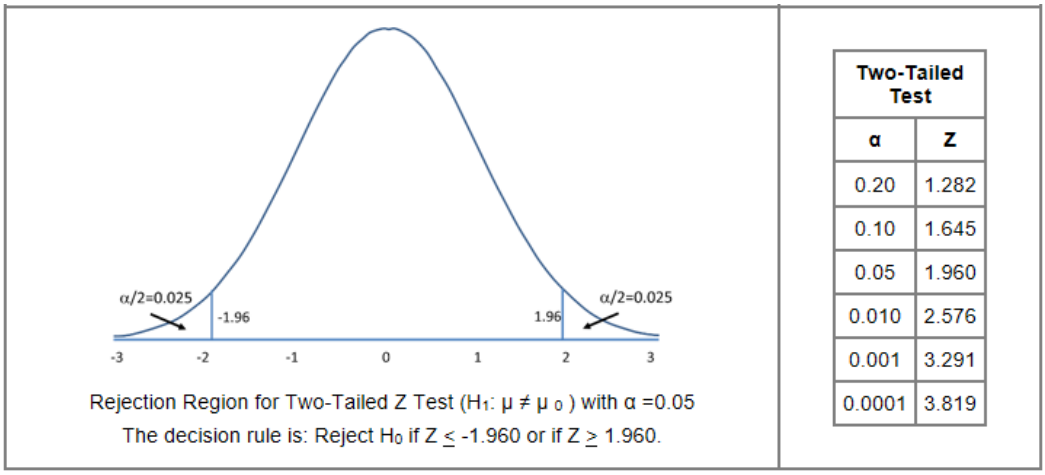
where μ_L stands for mean of particular system call in legitimate applications; μ_{OM} for mean of particular system call in old malware applications and μ_{NM} stands for mean of particular system call in new malware applications.

H_0 also called null hypothesis is tested for each call separetely. Null hypothesis stands that the mean of each system call is the same for each application, indistinctable from malware to legitimate applications. Contrary, H_A states that they are different, making them distinct from legitimate applications to malware applications. In statistics, it is always assumed that null hypothesis is true so that alternative hypothesis will be only chosen if it demonstrated that null hypothesis is false, being rejected. Sample data is collected and statistical test applied in order to reject or not null hypothesis. For this thesis, 3000 samples dataset of applications were used to collect the mean of each system call both in legitimate (n=1000) and malware sub-datasets (n=1000 on both malware datasets) and then applied Welch's t-test. Results were analyzed to reject or not our specific null hypothesis.

For each particular subset (legitimate, old malware and new malware), mean and standard deviation of each particular system call was evaluated, as shown in Appendix 1. In order to test hypothesis, Welch's t-test, also called z-test, was applied between legitimate vs. old malware datasets and legitimate vs. new malware dataset for each of the system calls. Z-test hypothesis testing of two population means was performed and Z-test score was analyzed in order to reject or not null hypothesis. In this case, z-test score for the comparison of two samples means is calculated with the following equation:

$$z = \frac{\bar{\chi}_L - \bar{\chi}_M}{\sqrt{\frac{\sigma_L^2}{n_L} - \frac{\sigma_M^2}{n_M}}}$$

In this stadistic test notation, $\bar{X}$ corresponds to sample mean, σ to standard deviation and n to size of data sample. Once every z score test was calculated, z-score value was analyzed using two-tailed z test (when H_1 refers to $\mu_1 \neq \mu_2$). Two-tailed z test establishes two regions within a standard normal distribution curve: rejection region of H_0 and acceptance region of H_0. If z score lies in rejection region, Null hypothesis is rejected, thus accepting alternative hypothesis (H_1) as true. Consequently, if z score lies in acceptance or non rejection region, Null hypothesis is accepted and considered true. Those regions are limited by already tabulated values, according to specific level of significance (α), which refers to the probability of rejecting the null hypothesis when it is true, that is, estimation error. For example, a significance level of 0.05 indicates a 5% risk of concluding that a difference in means exist when there is no difference. According to two-tailed z test, choosing a level of significance of 0.05 establishes an acceptance region delimited by ±1.96 z score values, that corresponds to (-1.96, 1.96), as shown in Figure 4. Then, decision rule establishes that if z score value is within this gap, H_0 is accepted as true, otherwise it is rejected, as can be seen in below figure example.



Two-Tailed Test	
α	Z
0.20	1.282
0.10	1.645
0.05	1.960
0.010	2.576
0.001	3.291
0.0001	3.819

Figure 4. Rejection area example using Z test score [64].

Depending on alpha or level of significance, rejection area is narrower or wider, requiring higher or lower values of z score. Previous figure, right side table shows z score value for each significance level, showing that for lower significance level (less error), higher values of z score are needed. Z-test score were calculated on both cases (Legitimate vs. Old Malware and Legitimate vs. New Malware) and degree of rejection of null hypothesis was assessed, as shown in Appendix 2.

Results show that, regarding legitimate vs. old malware testing sample data:

- H_0 is accepted in 59 system calls.
- H_0 is rejected in 36 system calls. Thus accepting H_A, meaning that means are significatively different between legitimate and malware applications regarding that specific system call.
- When there is not enough data, H_0 is considered to be true (117 system calls).

Regarding those 36 system calls that reject H_0, consequently being able to discriminate between the two samples according to that system calls, level of significance is:

- Less than 0.05 (5% error) in 4 system calls.
- Less than 0.01 (1% error) in 3 system calls.
- Less than 0.001 (0.1% error) in 3 system calls.
- Less than 0.0001 (0.01% error) in 26 system calls.

In the other case, analyzing legitimate vs. new malware testing sample data scores:

- H_0 is accepted in 57 system calls.
- H_0 is rejected in 42 system calls. Thus accepting H_A, meaning that means are significatively different between legitimate and malware applications regarding that specific system call.
- When there is not enough data, H_0 is considered to be true (113 system calls).

Regarding those 42 system calls that reject H_0, consequently being able to discriminate between the two samples according to that system calls, level of significance is:

- Less than 0.05 (5% error) in 9 system calls.
- Less than 0.01 (1% error) in 4 system calls.
- Less than 0.001 (0.1% error) in 5 system calls.
- Less than 0.0001 (0.01% error) in 24 system calls.

As can be stated from previous calculations, there is a solid statistical fundation to confirm that malware (on both datasets) and legitimate applications differ significantly in system calls behaviour, posing it as potentially discriminatory feature that can be used to detect malware from legitimate applications.

3.1.7.2 Android permissions hypothesis

Regarding permissions, as it is categorical or nominal data, in order to discriminate between malware dataset and legitimate dataset the notion of connection should be assessed as a discriminatory treat. For this thesis purpose, two competing hypothesis are stated for each system call on each dataset (Legitimate vs. Old malware and Legitimate vs. New Malware):

- H_0 or null hypothesis: There is no connection between legitimate/malware applications by observing whether a permission is set or unset.
- H_A or alternative hypothesis: There is connection between legitimate/malware applications by observing whether a permission is set or unset.

In order to establish this connection, the concept of proportion is introduced. H_0 states that the proportion of each permission is the same for each application, indistinctable from malware to legitimate applications. Contrary, H_A states that they are different, making them distinct. In this case, a categorical statistical hypothesis test is applied in order to accept or reject null hypothesis. For this thesis, 3000 samples dataset of applications were used to collect the proportion of each permission in legitimate (n=1000) and malware sub-datasets (n=1000 on both malware datasets) and then applied X^2 test (chi square test). Results were analyzed to reject or not our specific null hypothesis.

For each particular subset (legitimate, old malware and new malware), frequency (count) of each permission attribute (absent or present) was evaluated, as shown in Appendix 3. X^2 (Chi square) test is commonly used to test the relationship between categorical variables. In this case it was applied between legitimate/old malware datasets and legitimate/new malware dataset for each of the permissions. X^2 test hypothesis testing was used to test the connection or independence of two categorical

variables using crosstabulations (bivariate table) and its score was analysed in order to reject or not null hypothesis. Given for each permission a cross tabulated observed frequency count, like the following:

E.g.: READ_PHONE_STATE permission in L/O malware dataset:

Permission	Legitimate dataset		Old Malware Dataset	
	Absent	Present	Absent	Present
READ_PHONE_STATE	669	331	88	912

	Legitimate	Malware	
Absent	669	88	757
Present	331	912	1243
	1000	1000	2000

X^2 statistic is calculated from the previous table with the following equation:

$$X^2 = \sum \frac{(f_o - f_e)^2}{f_e}$$

Where f_o is the observed frequency (the observed counts in the cells) and f_e is the expected frequency if no relationship existed between the variables. This second table is constructed as follows:

	Legitimate	Malware	
Absent	1000*757/2000	1000*757/2000	757
Present	1000*1243/2000	1000*1243/2000	1243
	1000	1000	2000

With all these previous data in tables, X^2 is calculated and used to evaluate the connection between the two variables. Using that information, confidence level (alpha) and degrees of freedom, calculated as:

$$df = (n - 1)*(m - 1)$$

where *n* stands for number of rows and *m* stands for number of columns.

Probability value (p-value) can be found in Chi-square table, in order to state the rejection or not of the null hypothesis [65]. If p-value is less than confidence level (usually 0.05), it can be concluded that the variables are not independent and that there

is a statistical relationship between the categorical variables. X^2 test score was calculated on both cases (Legitimate vs. Old Malware and Legitimate vs. New Malware) and degree of rejection of null hypothesis was assessed, as shown in Appendix 4.

Results show that, regarding legitimate vs. old malware testing sample data:

- H_0 is accepted in 47 permissions.
- H_0 is rejected in 82 permissions. Thus accepting H_A, meaning that permission proportions are significatively different between legitimate and malware applications regarding that specific permission.
- When there is not enough data, H_0 is considered to be true (18 permissions).

Regarding those 82 system calls that reject H_0, consequently being able to discriminate between the two samples according to that system calls, level of significance is:

- Less than 0.05 (5% error) in 14 permissions.
- Less than 0.01 (1% error) in 7 permissions.
- Less than 0.001 (0.1% error) in 4 permissions.
- Less than 0.0001 (0.01% error) in 57 permissions.

In the other case, analyzing legitimate vs. new malware testing sample data values:

- H_0 is accepted in 48 permissions.
- H_0 is rejected in 79 permissions. Thus accepting H_A, meaning that permission proportions are significatively different between legitimate and malware applications regarding that specific permission.
- When there is not enough data, H_0 is considered to be true (20 permissions).

Regarding those 79 system calls that reject H_0, consequently being able to discriminate between the two samples according to that system calls, level of significance is:

- Less than 0.05 (5% error) in 15 permissions.
- Less than 0.01 (1% error) in 7 permissions.

- Less than 0.001 (0.1% error) in 6 permissions.
- Less than 0.0001 (0.01% error) in 51 permissions.

As a result, there is a solid statistical foundation (more than half of overall permissions are significantly different) to confirm that malware and legitimate applications differ significantly in permission settings, posing permissions as potentially discriminatory feature that can be used to detect malware from legitimate applications.

3.2 Phase 2. Data pre-processing

Machine learning models rely on data quality to achieve its purpose: make computers actually "learn" and predict accurately about the unknown. In order to create an accurate prediction model, thus enhancing data quality, data pre-processing for data mining and machine learning purposes is an important step that aims to transform the raw collected data into a new better representation before processing. Feature selection is the first step in classification process. Real data may contain features with different relevance and importance for predicting class labels. Less relevant features could harm the accuracy of the classification model and additionally be a source computational inefficiency [66]. Feature selection algorithms are designed to select the most informative features regarding to the class label [66], what translates as choosing the best predictive features that could provide as good or better accuracy whilst requiring less data acquisition and processing than using all collected features [67]. Regarding this, feature selection, called variable selection or attribute selection, methods help to identify and remove irrelevant and redundant features from data that do not actually contribute in a positive way to the accuracy of the predictive model and may, in fact, decrease or not affect the accuracy of the proposed model. The objective of feature selection is three-fold: improve prediction performance, supply faster and more cost-effective label predictors and provide a better knowledge of the underlying process that generated the data [68].

3.2.1 Feature selection

There are three primary feature selection methods in machine learning classification models, that are [66]:

- Filter models: involve the use of a mathematical criterion to evaluate the quality of the feature and use it to filter out irrelevant features.
- Wrapper models: consists in the progressive and iterative addition of features according to model accuracy output. An initial set of features F is used and accuracy is evaluated. Next, another feature is added and accuracy is evaluated in order to accept or reject the new added feature. This process is repeated iteratively with all features [66].
- Embedded models: main idea is that the solutions to many classification formulations provide important hints about the most relevant features to be used. Recursive feature elimination is used. After each elimination of features, the classifier is retrained on the new set of pruned features to re-estimate the weights. Next iteration will eliminate features with least absolute weight. This procedure is repeated until all remaining features are confirmed as sufficiently relevant [66].

In this present research, filter model was selected as feature selection method because wrapper model usually takes more time to execute and embedded models require some hints about data, and we decided to start from scratch, without any hints about data. Using this method, in practice, the features are evaluated independently of one another and the most discriminative ones are selected. The method used depends on the nature of the data attribute, whether it is categorical or numerical. System call feature selection requires a mathematical criterion suitable for quantitative attributes while permissions require one that works with categorical attributes.

3.2.1.1 System call feature selection

As demonstrated before, system calls are sensitive enought to discriminate between malware and legitimate applications. Fisher score is oriented concretely on a classifier construction, in order to test if the differences stated by statistical hypothesis testing are sensitive enough to construct a machine learning classifier.

Fisher Score is a criterion designed for numeric features to describe the discriminative power for classifier construction. Fisher Score (F) of a feature is calculated using the following equation:

$$F = \frac{\sum_{j=1}^{k} p_j(\mu_j - \mu)^2}{\sum_{j=1}^{k} p_j {\sigma_j}^2}$$

Where μ_j and σ_j refer, respectively, to the mean and standard deviation of data points belonging to class j for a particular feature and p_j to the fraction of data points belonging to class j. Whereas μ refers to the global mean of the data on the feature under evaluation. The numerator quantifies the average interclass separation, whereas the denominator quantifies the average intraclass separation. Thus, a larger Fisher score value implies a greater discriminatory power of the attribute. Attributes with largest Fisher score value should be selected to build the classifier model [66].

Fisher Score (F) criterion has been applied in this thesis to select best potentially discriminatory features from system calls whole group of features. Fisher score has no direct threshold that establish what features must be selected. Features are selected in comparison with other Fisher score values, the higher, the better. In this particular case, as can be seen in Appendix 5, as all F values were relatively low, selected features were the ones with F over 0.15, resulting in:

- 21 system calls from legitimate vs. old malware dataset.
- 12 system calls from legitimate vs. new malware dataset.

These variables are, from feature selection point of view, candidates to provide the best discriminatory power from the whole system call domain (212 system calls) regarding malware detection. A deeper look inside those selected attributes state that 11 of them are common in both datasets (showing different discriminatory power in each dataset). Table 1 shows the selected system calls and its Fisher score value. Highlighted in red are those common system calls while in green the specific ones.

System call	Legitimate vs. Old Malware	System call	Legitimate vs. New Malware
		Mprotect	0.19452250717
Readlinkat	0.688956982862	Readlinkat	0.586917830628
Sigaltstack	0.228203557978	Sigaltstack	0.205008675264
Munmap	0.749835053001	Munmap	0.569561624686
Sigaction	0.290309155851	Sigaction	0.302835602324

clock_gettime	0.839285565601	clock_gettime	1.10628298947
Madvise	0.541897726902	Madvise	0.475477241315
Connect	0.673586005831	Connect	0.518975491867
Prctl	0.614407016018	Prctl	0.532512665818
Openat	0.216978053454	Openat	0.164595592304
mmap2	0.630509802457	mmap2	0.473747280576
Ppoll	0.30531528393	Ppoll	0.249381065644
Futex	0.300893946098		
eventfd2	0.219313445317		
Clone	0.206699686896		
getdents64	0.15469663511		
Recvfrom	0.181364659841		
Sendto	0.194429708766		
epoll_create1	0.228419526142		
Close	0.173596597527		
Getppid	0.21559412016		
rt_sigprocmask	0.244162179462		

Table 1. System calls and Fisher score value

As can be seen in Table 1, legitimate/old dataset has more system calls selected and in general with higher values of Fisher Score than legitimate/new malware dataset (except for sigaction and clock_gettime system calls which is actually lower). This implies that separability is less obvious in legitimate/new malware dataset than in legitimate/old dataset. Thus, malware system call behaviour is becoming more similar to legitimate application, reducing its discriminatory power.

Although this general decrease of the discriminatory effect of some of the variables (except two, that actually increase), 11 of them are showing especially good discriminatory effect. A closer look on their purpose can highlight were in system call behaviour this discriminatory effect relies on. From better to lower Fisher score they are [69]:

- clock_gettime → retrieves the time of the clock.
- munmap → unmap files or devices into memory.

- readlinkat → read value of a symbolic link relative to a directory file descriptor.
- connect → initiate a connection on a socket.
- prctl → perform operations on a process.
- mmap2 → map files or devices into memory.
- madvise → give advice about use of memory.
- ppoll → wait for some event on a file descriptor.
- futex → provides a method for a program to wait for a value at a given address to change, and a method to wake up anyone waiting on a particular address.
- sigaction → used to change the action taken by a process on receipt of a specific signal.
- rt_sigprocmask → examine and change blocked signals.
- epoll_create1 → open an epoll (I/O event notification facility) file descriptor.
- sigaltstack → allows a process to define a new alternate signal stack and/or retrieve the state of an existing alternate signal stack.
- eventfd2 → create a file descriptor for event notification.
- openat → open a file relative to a directory file descriptor.
- clone → create a child process.
- getppid → get process identification.
- recvfrom → receive a message from a socket.
- mprotect → set protection on a region of memory.
- sendto → send a message on a socket.
- close → close a file descriptor.
- getdents64 → get directory entries.

As can be stated, system calls' candidates to possess best discriminatory power are related with socket connection, process management or file operations. Nevertheless, the best candidate is related with clock time, an interesting issue that will be further analyzed in next section.

3.2.1.2 System calls: top discriminative feature

Clock_gettime, arises as the most potentially discriminatory feature from the ones selected in this research using Fisher score value. Fisher score value for this specific

system call is very high (over 1 in L/N dataset), highlighting it as a potentially optimal predictor. Figure 5 shows the scatter plot of clock_gettime combined with readlinkat system call (second most potentially discriminatory feature).

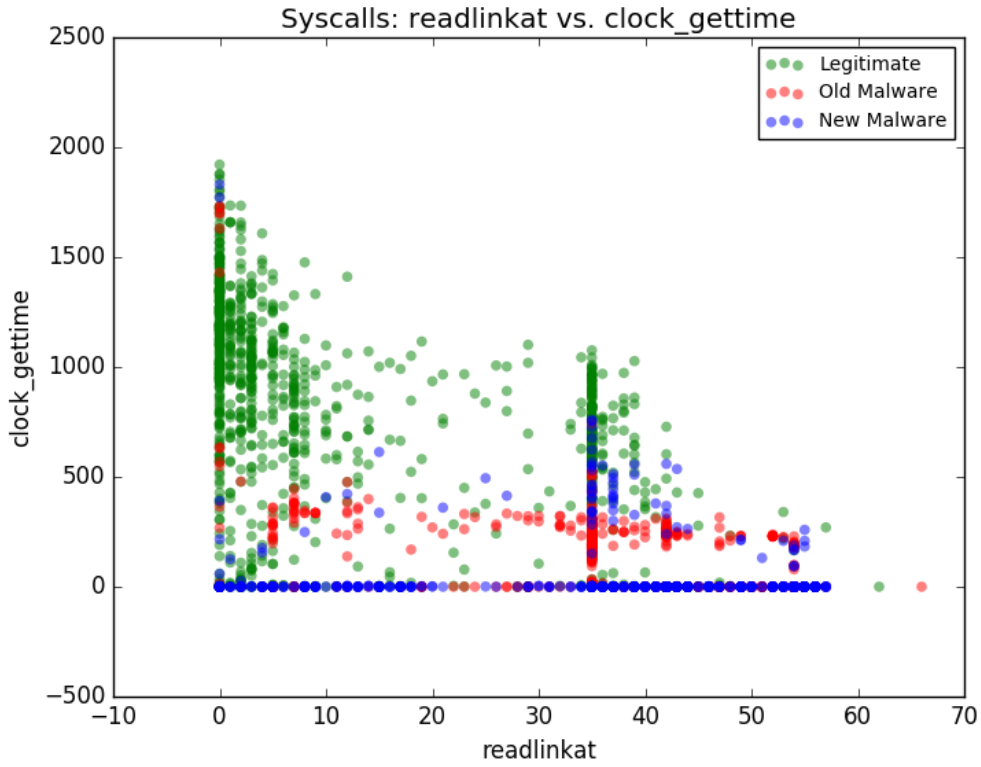


Figure 5. Scatter plot clock_gettime vs. readlinkat

As can be stated from Figure 5, malware (old and new) differ significantly in the use of clock_gettime and readlinkat from legitimate applications, creating a well defined area where malware points are concentrated. This could be used by a classifier to create a well defined decision boundary. This fact is also confirmed in Figure 6, where munmap, the third best common system call, is plotted against clock_gettime. The same issue appears, a well defined decision boundary is created by old malware, putting almost all new malware behind it. Legitimate applications use of clock_gettime is much greater than old malware and even more than new malware, which creates a significant difference that could be used for classification issues.

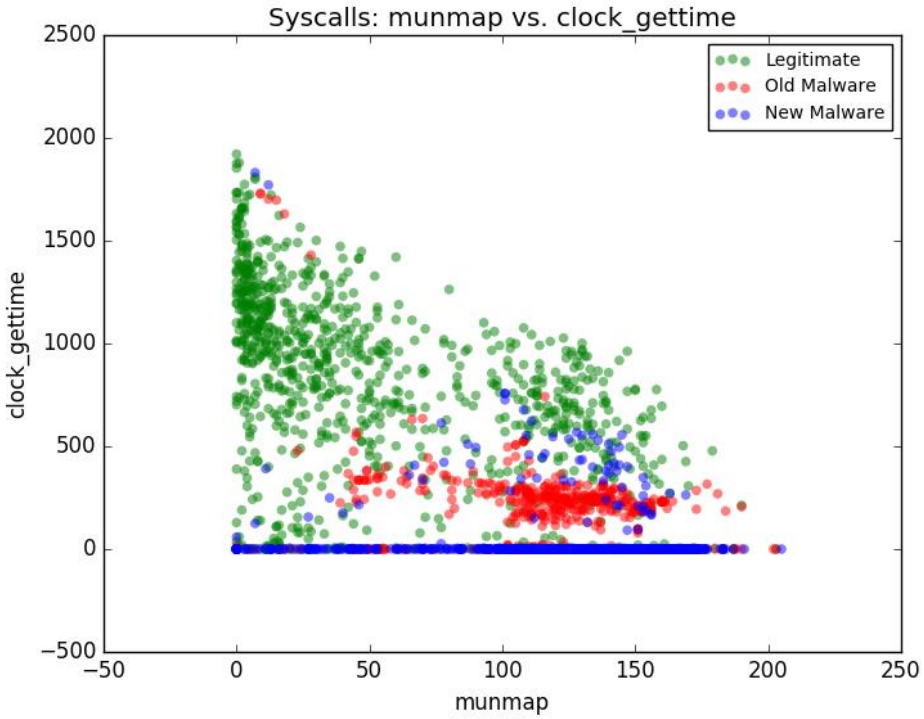


Figure 6. Scatter plot munmap vs. clock_gettime

Once again, plotting clock_gettime with any other rellevant feature, like mmap2 as can be stated in Figure 7.

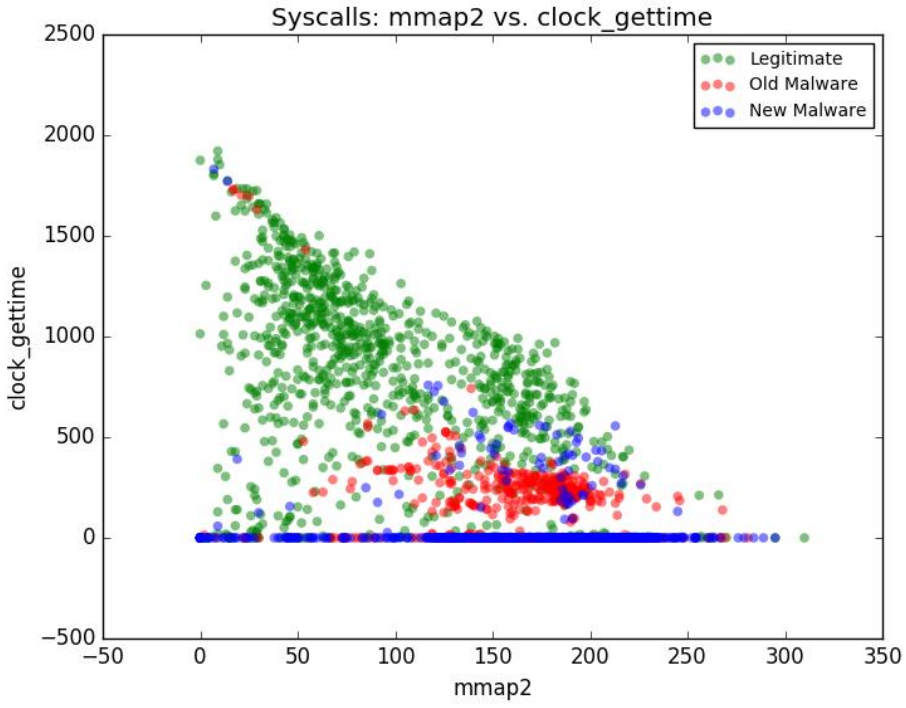


Figure 7. Scatter plot mmap2 vs. clock:_gettime

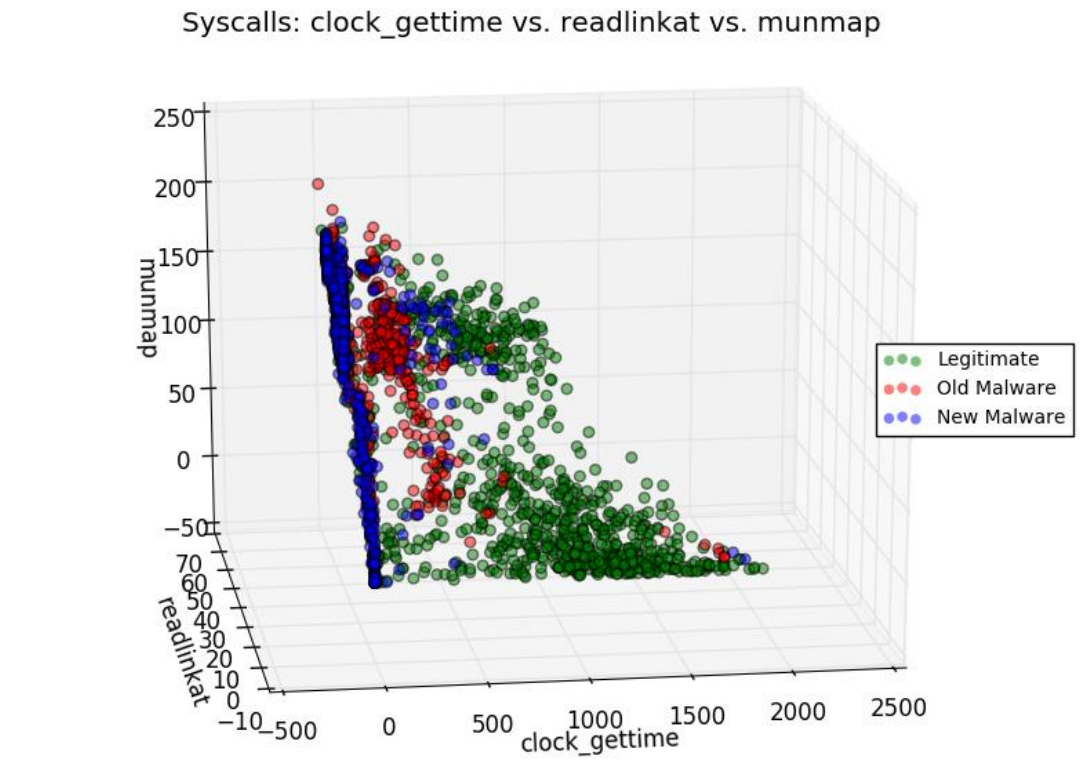


Figure 8. Scatter plot clock_gettime vs. readlinkat vs. munmap

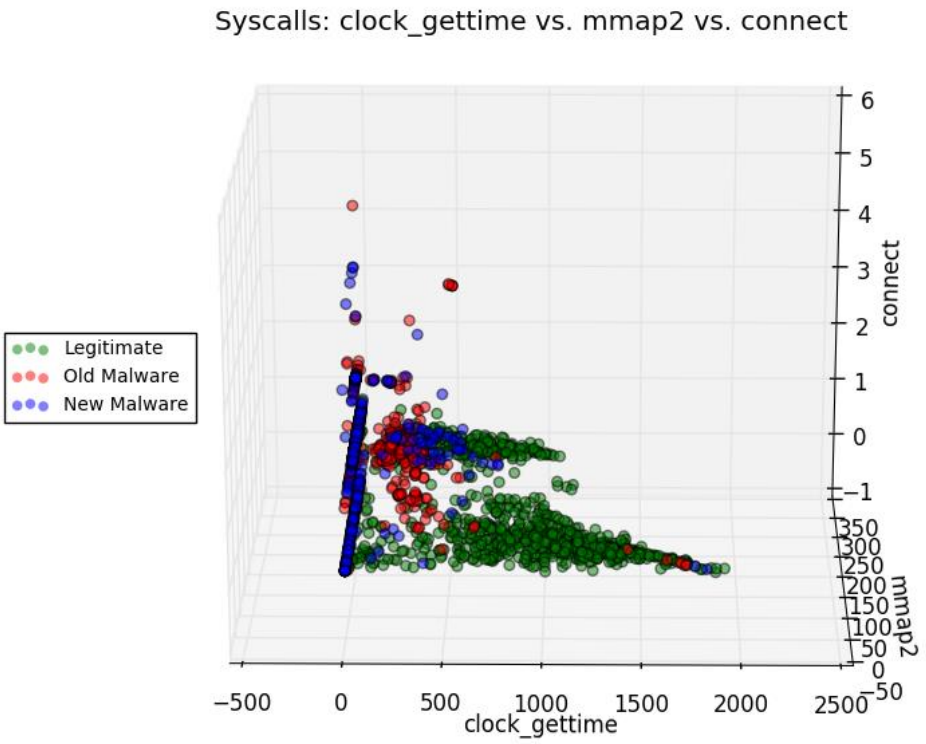


Figure 9. Scatter plot clock_gettime vs. mmap2 vs. connect

Figure 5, Figure 6 and Figure 7 point out that top selected system calls provide a good separability for potential discrimination of legitimate applications from malware and even old malware from new malware. Figure 8 and Figure 9 provide a 3D overview of previous facts.

Separability regarding the combination of this system calls is stated from Figure 8 and Figure 9, a well defined decision boundary appears to establish a good border for a classifier model. However, this separability decreases if analysing other features, as shown in Figure 10 and 11.

As can be seen on Figure 10 and Figure 11, separability decreases when using lower discriminatory features from common selected features (Figure 10) and even more if not common features are selected (Figure 11). Cloud of points become more mixed, providing, consequently, less separability.

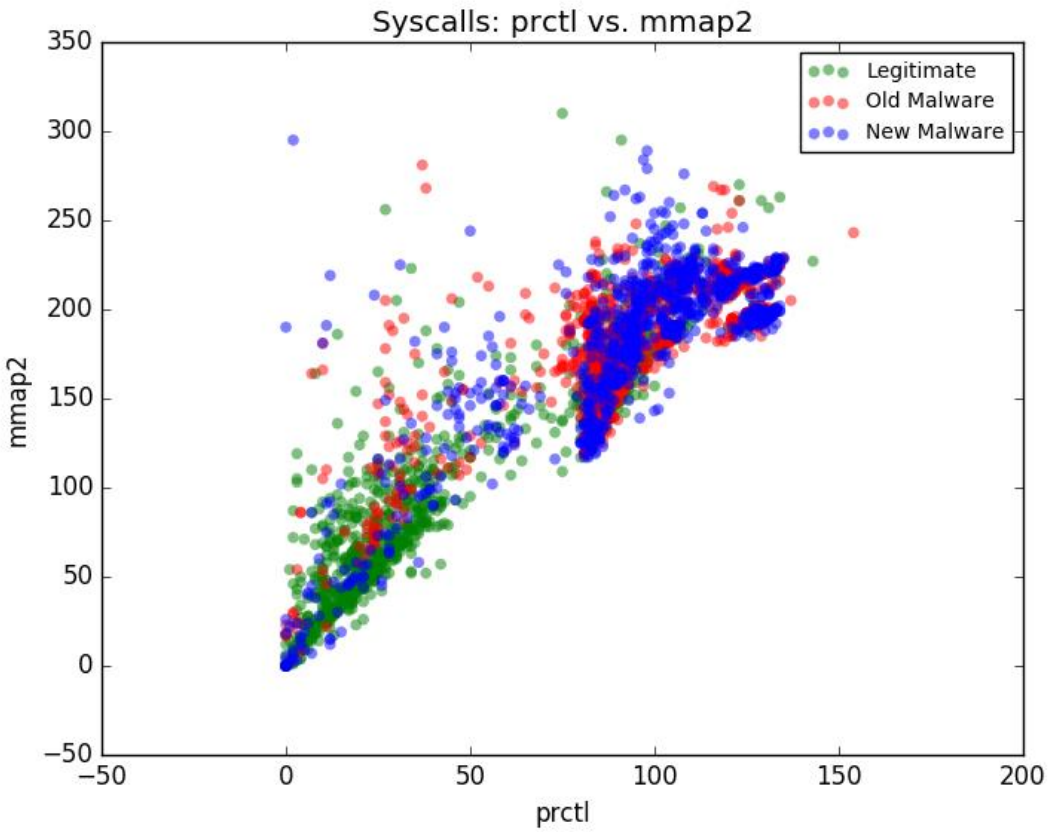


Figure 10. Scatter plot prctl vs. mmap2

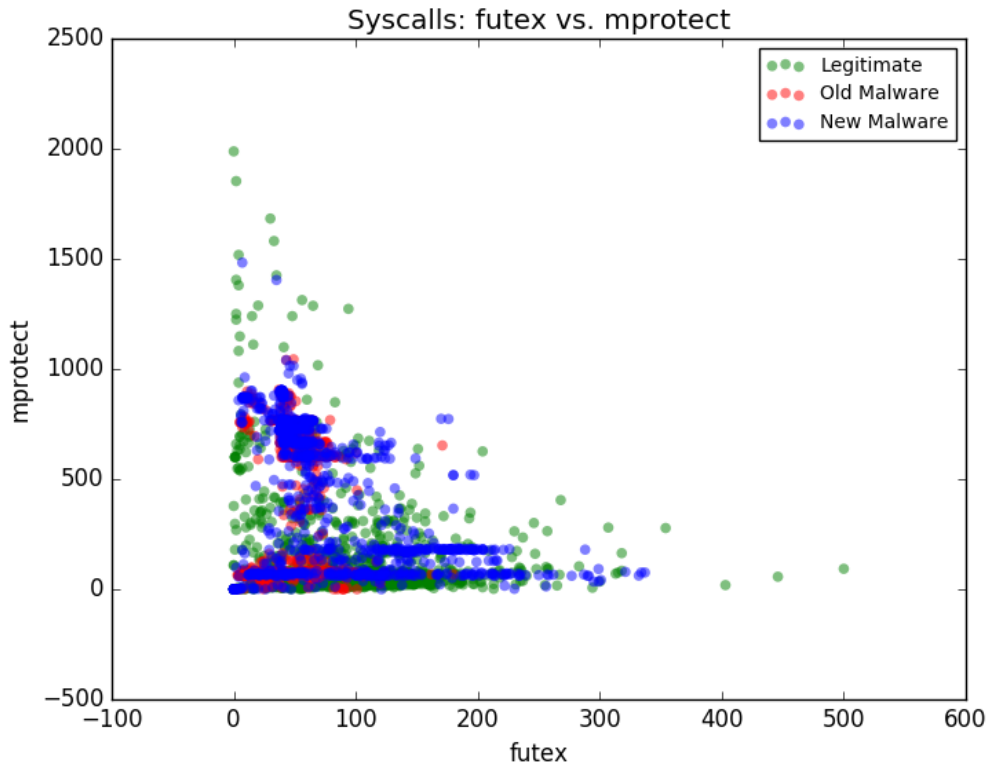


Figure 11. Scatter plot futex vs. mprotect

Classifier construction and model validation will provide output for this hypothesis, using mainly clock_gettime as most important feature could lead to a well classifier model.

3.2.1.3 Permissions feature selection

Gin Index is a criterion used for categorical feature selection, but it can be also used with numerical attributes via discretization process [66]. It is calculated using the following equations:

$$(1) \quad G(v_i) = 1 - \sum_{j=1}^{k} p_j^2$$

$$(2) \quad G = \sum_{j=1}^{k} \frac{n_i}{n} G(v_i)$$

$G(v_i)$ refers to the Gini index of the value v_i of a categorical attribute. In this equation (1), v_1 ... v_r refer to the r possible values of a particular categorical attribute and p_j to the fraction of data points containing attribute value v_i that belong to the class j ∈ {1 . . . k} for the attribute value v_i. If classes are distributed evenly for a particular attribute value, Gini index value is 1 − 1/k. Contrarily, if all data points for an attribute value, belong to the same class, Gini index value will be 0 [66]. Consequently, lower values of the Gini index imply greater discriminative power. This value-specific Gini index is not enough to measure the discriminative power of an attribute, needing it to be converted to an attributewise Gini index (2). In this equation (2), the weighted average over the different attribute values defines the overall Gini index for the specific attribute [66], being n_i the number of data points that take on the value v_i for the attribute and n is defined as the whole dataset data number of points ($\sum_{i=1}^{r} n_i$). Again, lower values of the Gini index imply greater discriminative power of the attribute. The attributes with the lowest Gini index values may be selected for use with the classification algorithm.

Gini Index (G) criterion has been applied in this thesis to select best features from permissions group of features. Gini index has no direct threshold that establish what features must be selected. Features are selected in comparison with other Gini index values, the lower, the better. In this particular case, as can be seen in Appendix 6, as all G values were relatively high, selected features were the ones with G under 0.47, resulting in:

- 13 permissions from legitimate vs. old malware dataset.
- 9 permissions from legitimate vs. new malware dataset.

This variables are, from feature selection point of view, the ones with best potentially discriminatory power from the whole standard Android permissions domain (147 permissions) regarding malware detection. A deeper look inside those selected attributes state that 4 are common in both datasets (showing different discriminatory power in each dataset). Next table shows the selected permissions and its Gini Index value. Highlighted in red are those common system calls while in green the specific ones.

Android Permission	Legitimate vs. Old Malware	Android Permission	Legitimate vs. New Malware
		VIBRATE	0.4412
		SYSTEM_ALERT_WINDOW	0.463627839885
		GET_TASKS	0.453245610181
		MOUNT_UNMOUNT_FILESYSTEMS	0.444434031227
		GET_ACCOUNTS	0.465315938431
WAKE LOCK	0.450313191498	WAKE LOCK	0.385793353195
READ_PHONE_STATE	0.320627747885	READ_PHONE_STATE	0.446529030013
ACCESS_NETWORK_STATE	0.46061566122	ACCESS_NETWORK_STATE	0.408704620648
INSTALL_PACKAGES	0.421355286521	INSTALL_PACKAGES	0.407988872281
CAMERA	0.465936440235		
USE_FINGERPRINT	0.466950959488		
BIND_REMOTEVIEWS	0.468932554434		
SEND_SMS	0.434005731136		
READ_EXTERNAL_STORAGE	0.334183622631		
ACCESS_FINE_LOCATION	0.469946356562		
READ_LOGS	0.441188030605		
BLUETOOTH	0.461664295186		
READ_CONTACTS	0.428342100938		

Table 2. System calls and Fisher score value

Unlike system calls, from Table 2 data it is not possible to infer that separability is less obvious within legitimate/old dataset than legitimate/new dataset. There are 4 common system calls, which vary its score from one dataset to the other. More specifically, one reduces its discrimination power (READ_PHONE_STATE), but other 3 increase its discrimination power (WAKE_LOCK, ACCESS_NETWORK_STATE, INSTALL PACKAGES). This change is not significantly enough to support such separability issue, unlike system calls.

A closer look on their purpose can highlight were in permissions this potentially discriminatory effect relies on. From better to lower Gini Index they are [19]:

- READ_PHONE_STATE → Allows read only access to phone state, including the phone number of the device, current cellular network information, the status of any ongoing calls, and a list of any PhoneAccounts registered on the device. Protection level: dangerous.
- READ_EXTERNAL_STORAGE → Allows an application to read from external storage. Protection level: dangerous.
- WAKE_LOCK → Allows using PowerManager WakeLocks to keep processor from sleeping or screen from dimming. Protection level: normal. INSTALL_PACKAGES → Allows an application to install packages. Not for use by third-party applications.
- ACCESS_NETWORK_STATE → Allows applications to access information about networks. Protection level: normal.
- READ_CONTACTS → Allows an application to read the user's contacts data. Protection level: dangerous.
- SEND_SMS → Allows an application to send SMS messages. Protection level: dangerous.
- VIBRATE → Allows access to the vibrator. Protection level: normal.
- READ_LOGS → Allows an application to read the low-level system log files. Not for use by third-party applications, because Log entries can contain the user's private information.
- MOUNT_UNMOUNT_FILESYSTEMS → Allows mounting and unmounting file systems for removable storage. Not for use by third-party applications.
- GET_TASKS → Deprecated in API level 21. *No longer enforced.*
- GET_ACCOUNTS → Allows access to the list of accounts in the Accounts Service. Protection level: dangerous.
- SYSTEM_ALERT_WINDOW → Allows an app to create windows using the type TYPE_APPLICATION_OVERLAY, shown on top of all other apps. Protection level: signature.
- CAMERA → Required to be able to access the camera device. Protection level: dangerous.

- USE_FINGERPRINT → Allows an app to use fingerprint hardware. Protection level: normal.
- BIND_REMOTEVIEWS → Must be required by a RemoteViewsService, to ensure that only the system can bind to it.
- BLUETOOTH → Allows applications to connect to paired bluetooth devices. Protection level: normal.
- ACCESS_FINE_LOCATION → Allows an app to access precise location. Protection level: dangerous.

As can be stated, candidates to best discriminatory requested Android permissions are mainly permissions with associated dangerous protection level which characterizes higher-risk permissions that could provide unauthorized access to user's private data or device information [70], thus needing to be specially approved by him/her.

3.3 Phase 3. Classification, Training and Validation

After data collection, pre-processing and statistical hypotheis testing, data, using appropiated feature selection criteria, is conformed into input vectors and ready to be processed and build a machine learning classification model.

3.3.1 Malware detection: binary classification problem

In this thesis' case, that deals with malware detection, supervised learning is performed in the form of a binary classification problem. Labeled samples or *instances* are provided along with measurements of attributes that characterize each instance. These attributes that characterise each sample are: system calls and permissions. System calls are measured numerically (frequency or count of each system call) while permissions are categorical (whether the specific permission is present or absent). The categorical label of each instance defines whether if the application is malicious or not (two possible options, binary classification). As a result, this classification problem aims to discriminate between two classes or labels of instances, malware or legitimate application, using feature input vectors in the form of frequency of system calls and set/unset permissions. Classification problems require that from dataset of instances, some of them should be used to train the model and the remaining as a validation or testing. Training data is provided labeled while test data is used to test the system make

a prediction of the label. Then, both predicted and real label are compared and classification performance and results are analyzed in the form of percentage of well classified and bad classified instances.

In supervised machine learning, datasets are splitted into training/testing sets where usually, 70% is used as training set for the machine learning classifier algorithm and 30% of them as validation or testing set. Nevertheless, *k-fold cross-validation* uses a similar but more complete approach. K-fold cross-validation split data into k consecutive folds (without shuffling by default) and performs k validation/training cycles into the model. "Each fold is then used once as a validation while the k-1 remaining folds form the training set" [71]. The output is then k test results which provide a better overall picture of the performance than the 70/30 split. Both approaches are used in this thesis as model performance results.

3.3.2 Malware detection: performance

Regarding binary classification results and performance, they are usually assessed using confusion matrix and performance metrics. Confusion matrix also known as error matrix is a tabulated data that condenses the performance of a classification algorithm. In this table, each row of the matrix represents the instances in a predicted class while each column represents the instances in an actual class (or viceversa). An example of confusion matrix is as follows:

		Actual class	
		Malware	Legitimate
Predicted class	Malware	C_{11}	C_{10}
	Legitimate	C_{01}	C_{00}

Cells labelled values as C_{00} and C_{11} indicate correct predictions of the machine learning model. C_{00} indicates an actual legitimate application predicted as legitimate application and C_{11} an actual malware application predicted as malware application. Cells labelled values as C_{01} and C_{10} indicate incorrect predictions or errors of the machine learning model. C_{10} denotes actual malware applications that were predicted as legitimate applications while C_{01} indicates actual legitimate applications that were classified as malware by the model. More concretely, C_{11} correct predicted value is often called *True*

Positive (a positive value well predicted) while C_{00} is often called *True Negative* (a negative value well predicted). Regarding errors, C_{01} is often called *False Positive*, *False alarm* or *Type I error* while C_{10} is called *False Negative*, *Miss* or *Type II error*. In our case, a *false negative* implies a security threat as malware application could not be detected, posing the user in an imminent threat while *false positive* implies a wrong classification, but not any kind of security threat to the user. Based on these concepts, new measurements are created in order to verify algorithms performance. *Precision* represents how correct the model is when it predicts a 1 or positive class [17], by using the following equation:

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

Where TP stands for *True Positive* and FP for *False Positive*. *Recall* or *True Positive Ratio* (TPR) or *Sensitivity* measures how many of the positive cases or 1 were identified by the model [17], using the following equation:

$$recall\ (TPR) = \frac{TP}{TP + FN}$$

Where FN stands for *False Negative*. Using this last metric, *False Positive Ratio* (FPR) stands for the amount of positive samples wrongly classified as negative, using:

$$FPR = \frac{FP}{FP + TN}$$

Ideally a good model should perform well on both precision and recall values [17], obtaining high scores as a result and low scores in FPR. *Accuracy* measures the whole amount of samples that were correctly classified over the total amount of samples, using the following equation:

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

Finally, *F-score* is a "comprehensive indicator which combines precision and recall together by a harmonic mean way. Higher F-score value proves a better performance in the system" [36]. It is calculated as follows:

$$F - score = \frac{2 * prc * rcl}{prc + rcl}$$

where *prc* stands for precisión and *rcl* for recall values.

In this thesis' case, malware detection machine learning classification model will be assessed in the light of the previous metrics.

4 Malware detection – practical implementation

In order to implement this malware detection machine learning classification model, Python was used as the base programming language and *scikit-learn* [72] library as machine learning algorithm resource implementation. *Scikit-learn* is an open-source Python library that provides "simple and efficient tools for data mining and data analysis" [72]. The library is built on other well-known and widely used Python mathematical libraries such as NumPy, SciPy and matplotlib. *Scikit-learn* can be used in machine learning in Python, providing important resources in classification, regression, clustering, dimensionality reduction, model selection and preprocessing of data [72]. For this thesis, *scikit-learn* was used to train and test machine learning used algorithms: Decision Tree, Support Vector Machines, K-nearest neighbors and Logistic Regression, and also to provide metrics output. Schematic of the whole model is described in Figure 12.

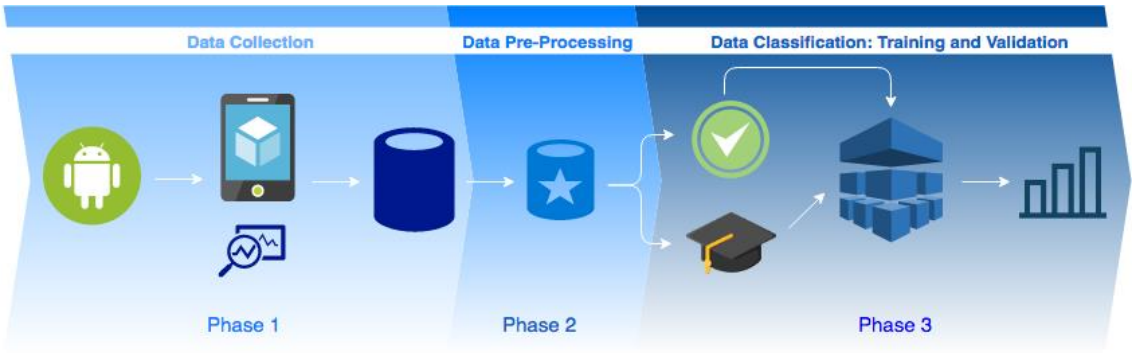


Figure 12. Malware detection practical implementation workflow

4.1 Classification algorithms

Machine learning classification problems use machine learning classification algorithms in order to discriminate and classify testing samples from the different classes that the problem deals with, according with training set used. As there are many options to build

a classification model, once built, it should be compared with other classifiers in order to choose the best one. Key elements in this comparison are [73]:

- *Predictive accuracy*, that refers to the model's ability to classify correctly every new, unkwnown sample;
- *Speed*, referring to how quickly the model can process data;
- *Robustness*, which alludes to the model's ability to make accurate predictions even in the presence of 'noise' in data;
- *Scalability*, that mainly refers to the model's ability to process increasingly larger volumen of data or to the ability of processing data from different fields;
- *Interpretability*, that deals with the fact that how easy the model can be understood or interpreted;
- *Simplicity*, which deals with the model's ability to be not too complicated, despite its efectiveness.

Classification algorithms use different mathematical approaches to deal with its common class discrimination purpose. Briefly explained, the most used ones in classification applications are [73][74]:

- Decision trees → it creates some conditions or questions based on the values of the input features it receives to make predictions about discrete or continous outputs.
- Bayesian classifiers/Naïve Bayes classifiers → based on Bayes' Thorem, it makes the assumption of independence (no relation) of all predictors to perform classification.
- Support Vector Machines (SVM) → focus on points that are far away from an hyperplane created by the training set. Works better with small datasets.
- Logistic Regression → uses the logistic or *sigmoid* function (from statistics) to perform classification.
- Neural networks → inspired by biological neural networks of animal brains. Uses "connection" between neurons and layers to perform classification.
- Random Forest → creates multiple decision trees and estimates the output based on weighted outputs of them.

- K-nearest neighbor classifier → classification is performed by a assigning to each point the most common class among itds k nearest neighbors.

For this thesis purpose, decision trees, logistic regression, k-nearest neighbor and Support Vector Machines (SVM) algorithms were selected as competing classifier models. Those are the most common ones used in malware detection as they provide good feedback regarding the key elements of comparison previously stated. They all have provided good predictive accuracy in prior studies with quick processing of data, easy interpretability and simplicity of the built models. In order to select the best one, four test data case scenarios were trained/tested using all models and results were analyzed. Case scenarios and results are summarized in Table 3 (best results are highlighted in green, second best results in lighter green).

Scenario	Decision Tree	Logistic Regression	k-Nearest Neighbor	SVM (linear kernel)
Test 1 Dataset used: L/O Features: 21 syscalls Validation: 70/30 split	Precision: 0.97 Recall: 0.97 F-score: 0.97	Precision: 0.97 Recall: 0.97 F-score: 0.97	Precision: 0.97 Recall: 0.97 F-score: 0.97	Precision: 0.97 Recall: 0.97 F-score: 0.97
Test 2 Dataset used: L/N Features: 12 syscalls Validation: 5-fold	Accuracy: 0.9015	Accuracy: 0.9005	Accuracy: 0.8995	Accuracy: 0.8960
Test 3 Dataset used: L/O Features: 13 perms Validation: 70/30 split	Accuracy: 0.9375	Accuracy: 0.9430	Accuracy: 0.9215	Accuracy: 0.9430
Test 4 Dataset used: L/N Features: 9 perms Validation: 5-fold	Precision: 0.91 Recall: 0.91 F-score: 0.91	Precision: 0.90 Recall: 0.90 F-score: 0.90	Precision: 0.90 Recall: 0.90 F-score: 0.90	Precision: 0.90 Recall: 0.90 F-score: 0.90

Table 3. Case scenarios, algorithms and results.

As can be seen from Table 3, comparing the different performance metrics of each row, Decision Tree algorithm was the best overall classifier in the all 4 scenarios. As a result, this thesis next steps will be built using Decision Tree algorithm as a classifier.

4.1.1 Decision Tree Algorithm

Decision trees are a non-parametric supervised machine learning algorithm used to address classification and regression problems [75]. When the target variable is a continous or numerical value, they are named Regression Trees. If the target is categorical, then Classification Trees name is used. Usually both types are referred together and called CART (Classification and Regression Tree). Non-parametric involves that the model structure is not specified a priori neither based on underlying assumptions but is instead only determined from acquired data (training dataset features). It is one of the most understandable machine learning algorithms, as they use simple decision rules, inferred from the data features of the training set, to predict the target value of the test set. The classification process, is then, built using a set of hierarchical decisions on the feature variables, in a tree-like structure or, more technically, in a directed acyclic graph [66]. Decision trees are composed by *nodes*, *branches* and *leaves*. A node is a decision point (also called split criterion) which is a condition on one or more feature variables in the training data. A node divides the training data into two or more parts [66], as can be seen in Figure 13. *Branches* are symbolized by lines that connect decision nodes with other decision nodes (called edges) or leaves. *Leaves* are end points of the tree where a final value is assigned (class or numerical) for cases that fulfilled all conditions from the top node to that specific leave [17].

Decision tree's main goal is to find a split criterion so that the level of mixing of the class variables in each brand of the three is minimal. Each node in the tree-like structure represents a subset of the data characterized by the combination of the split criteria in the nodes above it [17]. As a hierarchical decision, the most *important* or discriminatory attribute is placed at the top (root node) and create branches for each possible value of the attribute. Below it, each subsequent nodes decrease in importance, adding branches and ending, always, in a leaf, which will determine the corresponding class label. In general, a deeper tree involves more complex decision rules and a fitter model [75]. A special case of decision tree is *binary decision tree* where all decisions in the tree are two-valued, as can be seen in yellow highlight in left part of Figure 13.

Hunt's algorithm is used to achieve the goal of every decision tree, to make the optimal choice at the end of each node. Hunt's algorithm is greedy and recursive. Greedy stands for making at each step the most optimal decision and recursive means that larger questions are split in smaller questions and these are resolved the same way. The decision to split at each node is made using a metric called purity. In this regard, a node is 100% impure when it splits data evenly 50/50 and 100% pure when all node's data belongs to a single class [76]. The goal is then, achieve maximum purity and avoid impurity. To obtain it, Gini impurity and other metrics such as entropy or chi-square are applied. Information gain is another metric used, in this case to decide what feature to split at each step of the tree. The performance of a tree can be increased significantly using pruning. Pruning is a technique that removes the branches that use low-importance features, thus reducing tree's complexity and increasing tree's predictive power by reducing overfitting [77].

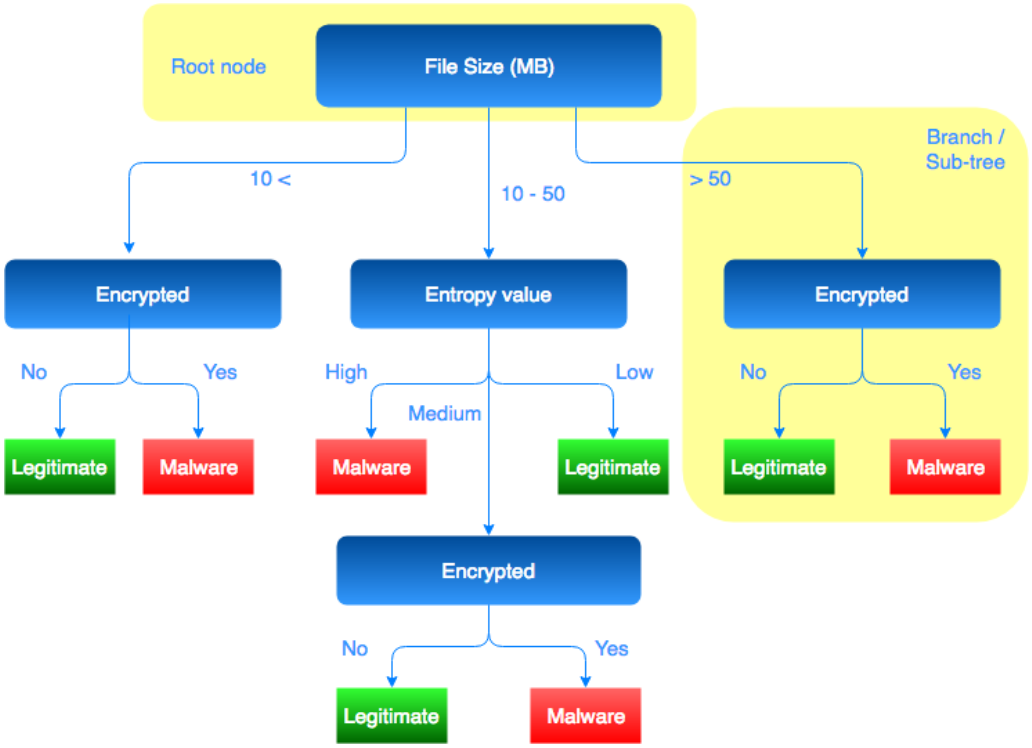


Figure 13. Example of Decision Tree in invented malware detection scenario.

CART main advantages are:

- Simple to understand, interpret, visualize and draw.

- There is no significant need of data preprocessing as CART performs implicitly some kind of feature selection.
- Works well with all types of data: categorical and numerical.
- Can handle multi-output problems.
- Its performance is not affected by nonlinear relationships between parameters.
- Can be validated using statistical tests, thus providing reliability to the model.
- The cost of using the tree is logarithmic in the number of data points in the training set.

CART main disadvantages are:

- Overfitting: which involves the creation of over-complex trees that correspond to an excessive fit of the model to the training data and may fail to fit additional data correctly or predict future observations reliably. Pruning is required to solve this problem.
- Unstability: small variations in the data might result in a completely different tree being generated. Ensemble techniques are needed to solve this problem.
- Not optimal: greedy algorithms cannot guarantee to return the globally optimal decision tree. Multiple trees should be created to reduce this problem
- Biased: can create biased trees if some classes dominate. Balanced data is needed to solve this problem.

All functionalities designed to built and traing a decision tree model to deal with classification and regression problems is included in *scikit-learn* Python library package [75]. Scikit-learn will be used as a basis of the following tests to create, train and test decision tree algorithm in malware detection using system calls and Android permissions.

5 Malware Detection Model Validation

Next section will show different test and results with the collected dataset, that it is composed by 1000 legitimate applications, 1000 old malware and 1000 new malware samples. From both, dynamic features were collected in the form of system calls and static features in the form of requested features contained in AndroidManifest.xml. The discriminatory power is evaluated and assessed in the following sections.

5.1 Selected system calls

Previous steps have shown that from 212 system calls gathered, 21 shown good potential discriminatory power in legitimate vs. old malware dataset and 11 in legitimate vs. new malware dataset. Each whole dataset is composed by 2000 samples. For this test, different number of features, from the ones with better potential discriminatory power, were selected and used to train and test a decision tree model using same dataset. Table 4 shows accuracy metric performance for each different model (range 0-1), using 5-fold cross validation. Appendix 7 shows a more complete picture of this analysis, including classification confusion matrix using 70-30% split.

# features	Legitimate vs. Old Malware Dataset (accuracy value)	Legitimate vs. New Malware Dataset (accuracy value)
Best feature*	0.8695	0.8910
3 best common features**	0.9005	0.8770
6 best common features***	0.9120	0.8850
11 common features	0.9305	0.8905
22 features****	0.9660	0.9075
All features (212)*****	0.9700	0.9270

Table 4. System calls model accuracy performance

*Best feature: *clock_gettime*

**3 best common features: *clock_gettime*, *readlinkat* and *munmap*.

***6 best common features: *clock_gettime*, *readlinkat*, *munmap*, *connect*, *prctl* and *mmap2*.

****21 features of L/O and *mprotect*.

*****All system calls gathered, without performing feature selection.

Table 4 shows that:

- A single feature (the one with higher Fisher score value) is capable of discriminate between malware and legitimate applications (in both datasets) with an accuracy over 87%. Providing also relatively low values of *False Negatives*.
- In L/O dataset, discriminative power increases gradually when more features are added but 22 best Fisher score features provide as good prediction as whole system call domain, 212 features (~ 97%). The importance of feature selection provides here optimum output with less information.
- In L/N dataset, best single feature has more discriminative power alone than using from 2 to 11 top Fisher score features. This single feature has more discriminative power in L/N dataset than in L/O dataset, confirming the increase in discriminatory power suggested by its Fisher score value from L/O dataset to L/N dataset (from ~0.8 to ~1.1).
- Unlike L/O dataset, 22 best Fisher score does not provide in L/N dataset similar accuracy obtained with whole system call domain. This fact, combined with a general lower accuracy values in L/N dataset than in L/O dataset suggests that separability between legitimate and malware applications is higher in L/O dataset than in L/N dataset.

5.2 Selected permissions

Previous steps have shown that from 147 standard Android permissions, a total amount of 13 shown potential good discriminatory power in legitimate vs. old malware dataset and 9 in legitimate vs. new malware dataset. For this test, different number of features, from the ones with potential better discriminatory power, will be selected and used to train and test a decision tree model using same dataset. Table 5 shows accuracy metric performance for each different model (range 0-1), using 5-fold cross validation.

Appendix 8 shows a more complete picture of this analysis, including classification confusion matrix using 70-30% split.

# features	Legitimate vs. Old Malware Dataset (accuracy value)	Legitimate vs. New Malware Dataset (accuracy value)
Best feature of L/O dataset*	0.7905	0.6635
Best feature of L/N dataset**	0.6420	0.7310
2 best features of L/O dataset***	0.8880	0.6635
2 best features of L/N dataset****	0.6995	0.7310
4 common features	0.8580	0.8460
9 features of L/N dataset	0.8955	0.8940
13 features of L/O dataset	0.9350	0.9065
18 features*****	0.9410	0.9170
All features (147)******	0.9505	0.9210

Table 5. Permissions model accuracy performance

*READ_PHONE_STATE permission
**WAKE_LOCK permission
***READ_PHONE_STATE and READ_EXTERNAL_STORAGE permissions
****WAKE_LOCK and INSTALL_PACKAGES permissions
*****13 features of L/0 and *VIBRATE, SYSTEM_ALERT_WINDOW, GET_TASKS, MOUNT_UNMOUN_FILESYSTEMS, GET_ACCOUNTS.*
******All permissions, without performing feature selection.

Table 5 shows that:

- Unlike system calls, there is no single or two best common features capable to discriminate with good accuracy on both datasets, as those best features vary within datasets. As can be seen, using one or two features, greater discriminatory power is achieved when they are used in its own dataset and not in the other dataset (e.g.: best feature in L/O shows 79% accuracy in L/O dataset and 66%

accuracy when they are used in L/N dataset), stating that best permissions, in this case, cannot be used cross-dataset detection with acceptable results.

- When using 4 common features, good discriminatory power is achieved in both datasets (~ 85%). From that minimum number of features, increased discriminatory values are obtained in both datasets when more features are added.
- In this case, in both datasets, when using 18 best Gini index features show similar accuracy as using all features (~ 95% in L/O dataset and ~ 92% in L/N dataset). The importance of feature selection provides here optimum output with less information.
- Compared to system calls, in L/O dataset, permissions provide lower top discrimination value (~ 95%) than in system calls (~ 97%). Regarding L/N dataset, maximum discrimination power is almost the same (~ 92%). This suggests that separability between legitimate and malware applications is higher in L/O dataset than in L/N dataset, but lower than in system calls case.

5.3 Combination of features: permissions and system calls

Previous tests used just a set of features, or system calls (numerical) or permissions (categorical). Following tests use mixing of features in order to find whether the results improve by the addition of another type of feature. For this test, different number of features, from the ones with better discriminatory power of each variable, will be selected and used to train and test a decision tree model using same dataset. Table 6 shows accuracy metric performance for each different model (range 0-1), using 5-fold cross validation. Appendix 9 shows a more complete picture of this analysis, including classification confusion matrix using 70-30% split.

# features	Legitimate vs. Old Malware Dataset (accuracy value)	Legitimate vs. New Malware Dataset (accuracy value)
Best system call + best permission L/O*	0.8965	0.9070
Best system call + best permission L/N**	0.8800	0.8900
2 best system calls L/O + 2 best permissions L/O	0.9450	0.8990
2 best system calls L/N + 2 best permissions L/N	0.9035	0.8950

All common system calls and permissions	0.9505	0.9210
22 system calls + 18 permissions	0.9740	0.9390
All features (212+147)	0.9765	0.9400

Table 6. Hybrid model accuracy performance

*clock_gettime system call and READ_PHONE_STATE permission

**clock_gettime system call and WAKE_LOCK permission

****clock_gettime* and *munmap* system calls and READ_PHONE_STATE and READ_EXTERNAL_STORAGE permissions.

*****clock_gettime* and *readlinkat* system calls and WAKE_LOCK and INSTALL_PACKAGES permissions.

Table 6 shows that:

- When using combination of each best feature of both domains, accuracies are slightly better than using system calls alone. System calls alone provide more or less the same discrimination power than when it is used combined with permissions in L/O dataset. In L/N dataset, permissions have a slightly greater influence, improving detection ratio by 1-2% than system calls alone. Thus, in general, system calls provide better classification performance than permissions.
- Maximum accuracy in L/N dataset is improved using hybrid approach (~ 94%) than static or dynamic approaches alone (~ 92%). In this case, hybrid approach shows an improvement not applied to L/O dataset.
- Using all features selected (40) provide the same discriminatory power than using 357 features (all permissions and system calls) in both datasets (~ 97% in L/O and 94% in L/N dataset). Feature selection arises as an important fact that provides optimum results with less information.

5.4 Old vs. New Malware discrimination

As was stated in previous sections, old and new malware show similarities, as they have common features that allow discriminate them from legitimate applications. This experimental model eliminate legitimate applications from the stage and use a dataset

composed only by malware: 1000 old and 1000 new malware applications. New malware was labelled as 0 and old malware as 1. As for the other tests, different number of features, from the ones with better discriminatory power, will be selected and used to train and test a decision tree model using same dataset. Table 7 shows accuracy metric performance for each different model (range 0-1), using 5-fold cross validation. Appendix 10 shows a more complete picture of this analysis, including classification confusion matrix using 70-30% split.

Feature	Number of features	New vs. Old Malware Dataset (accuracy value)
System calls	Best system call*	0.6460
	Common system calls (11)	0.8175
	22 selected system calls	0.8955
	All system calls (212)	0.8990
Permissions	Best permission of L/O dataset**	0.6270
	Best permission of L/N dataset***	0.5890
	Common permissions (4)	0.6655
	18 selected permissions	0.9310
	All permissions (147)	0.9430
Hybrid (system calls + permissions)	2 best system calls L/O + 2 best permissions L/O dataset****	0.7825
	2 best system calls L/N + 2 best permissions L/N dataset*****	0.7720
	All common system calls and permissions (11+4)	0.8190
	22 system calls + 18 permissions (40)	0.9300
	All features (212+147)	0.9345

Table 7. Old. vs. New Malware discrimination accuracy performance

*clock_gettime

**READ_PHONE_STATE

***WAKE_LOCK

****_clock_gettime_ and _munmap_ system calls and READ_PHONE_STATE and READ_EXTERNAL_STORAGE permissions.
*****_clock_gettime_ and _readlinkat_ system calls and WAKE_LOCK and INSTALL_PACKAGES permissions

Table 7 shows that:

- Malware itself can be discriminated from old samples to new samples using a small amount of features, also used to discriminate between legitimate and malware applications.
- Unlike legitimate vs. malware detection, permissions can be used with best accuracy when discriminating between different malware (~ 94%) than system calls (~ 90%) and similar with hybrid approach (~ 94%). Permissions appear to be the most important discriminative variable in this case, having higher detection ratio than system calls. Thus, separability between malware permissions is greater than malware from legitimate applications.
- Using hybrid approach, system calls provide good accuracies to the model when a small amount of permissions are used. When more than common permissions are used, system calls does not provide any significant improvement than permissions alone.
- When using 18 selected permissions, it can be stated that they provide the same discrimination power than all features combined (system calls and permissions) and all selected features combined (~ 94%), stating the importance of permissions in this particular case.

5.5 Cross-dataset malware detection validation

This experimental setup analyzes the accuracy of different scenarios using system calls and permissions. For each feature, different number of features are selected and the model is trained. Training is performed with one dataset (e.g.: legitimate/old malware dataset) and testing is performed with the other dataset (e.g.: legitimate/new malware dataset). Accuracy metric performance results are shown in Table 8, Table 9 and Table 10 for each model (range 0-1). In this case 70-30 split was performed as training/testing split and all datasets shuffled prior of the random selection of training/testing samples. Results regarding training and testing with same dataset are included as a reference (5-

fold cross validated). Appendix 11 shows a more complete picture of this analysis, including classification confusion matrix using 70-30% split.

5.5.1 Dynamic approach: System calls

System Calls (using best feature)* *clock_gettime	Training \ Testing	Old Dataset (accuracy)	New Dataset (accuracy)
	Old Dataset	0.8695	0.9166
	New Dataset	0.795	0.8910

System Calls (using 3 best features)* *clock_gettime, munmap and readlinkat	Training \ Testing	Old Dataset (accuracy)	New Dataset (accuracy)
	Old Dataset	0.9025	0.8833
	New Dataset	0.815	0.8820

System Calls (using 11 common features)	Training \ Testing	Old Dataset (accuracy)	New Dataset (accuracy)
	Old Dataset	0.9305	0.8100
	New Dataset	0.865	0.8905

System Calls (using 22 selected features)	Training \ Testing	Old Dataset (accuracy)	New Dataset (accuracy)
	Old Dataset	0.9650	0.7330
	New Dataset	0.8766	0.9075

System Calls (using all features)	Training \ Testing	Old Dataset (accuracy)	New Dataset (accuracy)
	Old Dataset	0.9700	0.7150
	New Dataset	0.8450	0.9270

Table 8. Dynamic approach: system call results

Table 8 shows that:

- If a small amount of features is used (1-3 best features), classification accuracies are better using old dataset as a training set for detecting both old and new malware (~ 90%).

- If more than 3 best features are used (from 11 to all system call domain), classification results are better using new dataset as a training or learning set for detecting both old and new malware.
- Best cross-dataset results are provided when using 3-11 selected common system calls (over ~ 80% in all cases), stating the importance of feature selection.
- This tables show that it is possible to obtain acceptable accuracy results in cross-dataset detection using a single training dataset, altough best results are achieved when testing with same dataset.

5.5.2 Static approach: Permissions

Permissions (using best feature L/O dataset)* *READ_PHONE_STATE	Training \ Testing	Old Dataset (accuracy)	New Dataset (accuracy)
	Old Dataset	0.7905	0.6633
	New Dataset	0.7850	0.6635

Permissions (using best feature L/N dataset)* *WAKE_LOCK	Training \ Testing	Old Dataset (accuracy)	New Dataset (accuracy)
	Old Dataset	0.6420	0.7450
	New Dataset	0.6783	0.7310

Permissions (using 4 common features)	Training \ Testing	Old Dataset (accuracy)	New Dataset (accuracy)
	Old Dataset	0.8580	0.8400
	New Dataset	0.8800	0.8460

Permissions (using 18 features)	Training \ Testing	Old Dataset (accuracy)	New Dataset (accuracy)
	Old Dataset	0.9410	0.7100
	New Dataset	0.8716	0.9170

Permissions (using all features)	Training \ Testing	Old Dataset (accuracy)	New Dataset (accuracy)
	Old Dataset	0.9505	0.6500
	New Dataset	0.7916	0.9245

Table 9. Static approach: permission results

Table 9 shows that:

- Unlike system calls, when a small amount of features are used (1-4), detection performance is low (not higher than 80% in any case) with no significant difference in malware discrimination by using one set or another as a training set.
- When using 4 common features, best overall across dataset performance is achieved, over 84% of discrimination in all cases.
- When using all features, its discriminative power decreases across datasets, having only significant discriminative power when testing with the same dataset.

5.5.3 Hybrid approach: system calls and permissions

Hybrid (best syscall and best L/O dataset permission)* *clock_gettime and READ_PHONE_STATE	Training \ Testing	Old Dataset (accuracy)	New Dataset (accuracy)
	Old Dataset	0.8985	0.9200
	New Dataset	0.8700	0.9070

Hybrid (using best syscall and best L/N dataset permission)*	Training \ Testing	Old Dataset (accuracy)	New Dataset (accuracy)
	Old Dataset	0.8800	0.9033
	New Dataset	0.8066	0.8835

Hybrid (using 11 common syscalls + 4 common permissions)	Training \ Testing	Old Dataset (accuracy)	New Dataset (accuracy)
	Old Dataset	0.9500	0.8660
	New Dataset	0.9016	0.9210

Hybrid (using 22 selected syscalls + 18 selected permissions)	Training \ Testing	Old Dataset (accuracy)	New Dataset (accuracy)
	Old Dataset	0.9740	0.7066
	New Dataset	0.9116	0.9390

Hybrid (using all dynamic and static features)	Training \ Testing	Old Dataset (accuracy)	New Dataset (accuracy)
	Old Dataset	0.9765	0.6966
	New Dataset	0.8983	0.9400

Table 10. Hybrid approach: system call+permission results

Table 10 shows that:

- The combination of best system call and best permission from L/O dataset provides the best accuracy results in cross-dataset performance, using the less amount of features (2). System call discriminatory power is empowered with best L/O dataset permission to provide improved cross-dataset accuracies than using system call alone in some specific cases.
- When using a small amount of features combined (from 2 to 15 features), old dataset provides better discriminative power as a training set than new dataset in cross-dataset performance.
- When using more than 15 features, new dataset provides better discriminative power as a training set than new dataset in cross-dataset performance.
- Combination of all selected features (40) provide greater performance (over 91%) than all 357 features combined (over 89%). Feature selection allows to achieve optimum results with lower amount of data.

5.6 Mixed malware detection validation

This experimental setup analyzes the accuracy of different scenarios using system calls, permissions and hybrid approach. One mixed malware dataset is created from old malware dataset and new malware dataset (500 applications of each one selected randomly) and used as training/testing tested in different scenarios. This pretends to simulate real conditions where it is not possible to classify if a malware sample belongs to old or new dataset. For each feature, different number of features are selected and the model is trained with them. Accuracy results are shown in next the following tables. In this case 70-30 split was performed as training/testing split and all datasets shuffled prior of the random selection of training/testing samples. Appendix 12 shows a more complete picture of this analysis, including classification confusion matrix using 70-30% split.

5.6.1 System call features

Training set	clock_gettime (accuracy)	11 common (accuracy)	22 selected (accuracy)	All features (accuracy)
Old Dataset	0.8966	0.9000	0.8450	0.866
Mixed malware	0.8675	0.8805	0.9080	0.9195
New Dataset	0.8516	0.9133	0.9400	0.9266

*5-fold cross validation againsta same dataset.

Table 11. Mixed malware: system call results

5.6.2 Permission features

Training set	READ_PHONE _STATE (accuracy)	WAKE_LOCK (accuracy)	4 common (accuracy)	18 selected (accuracy)	All features (accuracy)
Old Dataset	0.7300	0.6533	0.8550	0.8466	0.8316
Mixed malware	0.7265	0.6780	0.8525	0.9160	0.9225
New Dataset	0.7150	0.6550	0.846	0.9016	0.8700

*5-fold cross validation againsta same dataset.

Table 12. Mixed malware: permission results

5.6.3 Hybrid approach

Training set	Clock_gettime & READ_PHONE_STATE (accuracy)	Clock_gettime & WAKE_LOCK (accuracy)	11 common syscalls & 4 common perms (accuracy)	22 selected syscalls & 18 selected permissions	All syscalls & all permissions (accuracy)
Old Dataset	0.9116	0.9150	0.9366	0.8650	0.8533
Mixed malware	0.8965	0.8730	0.9300	0.9415	0.9375
New Dataset	0.9133	0.8800	0.9450	0.9433	0.9266

*5-fold cross validation againsta same dataset.

Table 13. Mixed malware: hybrid results

Table 11 show that, regarding system calls:

- As stated before, if only the most important feature is used, training with old dataset provides better overall cross-dataset detection (~ 90%) than training with just new or mixed malware dataset (below 87% on both cases).
- When more features are used, training with new malware datasets provide better overall cross-dataset detection (over 91%) than training with old or mixed.
- Best results are achieved training with new dataset and using 22 selected features (94% accuracy). Feature selection arises as an important model builder.

Table 12 shows that, regarding permissions:

- Using a small amount (1-4) of permissions does not provide good accuracies (below 86% in all cases), no matter the training set used.
- Best cross-datasets detection results are achieved using mixed malware dataset. When using 18 features, 91% accuracy is achieved, slightly below than the 92% accuracy achieved with all permissions. This states the importance of feature selection to create good predictive models with less data.
- In general, permissions alone show less discriminative power (92% accuracy) than system calls alone (94% accuracy).

Table 13 shows that, regarding hybrid approach:

- In general, hybrid approach provides better overall results than using static or dynamic features alone. Regarding that, system calls are shown to be the most discriminative feature and when combined with permissions they provide slightly better results in accuracies when system calls alone does not provide good detection ratios.
- Best results in this case are achieved training with new malware dataset and using from 15 to 40 features combinated (over 94%).
- Using all features domain (357) does not provide better accuracy (~ 92%) than using selected features. Feature selection arises, again, as an important step in model building and improvement in accuracies.

5.7 Decision Tree graphs

As was previously stated, Decision Tree is a *hierarchical* set of decisions that place in root node the most important or discriminatory feature of the dataset. In order to check how decision trees were implemented in this malware detection project, nine random test trees were plotted and analysed, using all available features. *Pruning* was not performed, so trees structure shown are more complex or deep than if pruning (remove unimportant branches) were performed. Next diagrams show the trees' structure.

5.7.1 System calls

5.7.1.1 Legitimate / Old Malware Dataset Trained Decision Tree

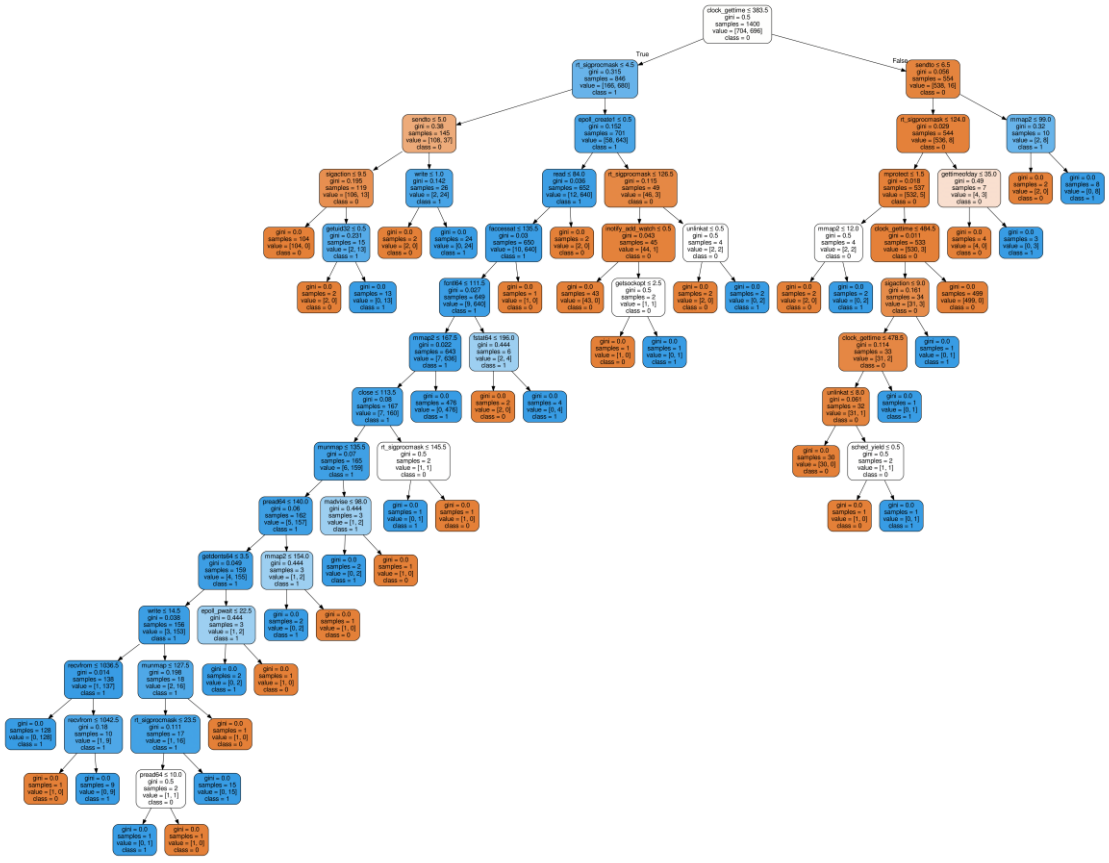


Figure 14. Decision Tree L/O Dataset

This decision tree uses as root node a *clock_gettime* decision rule, the most important feature selected also by Fisher score:

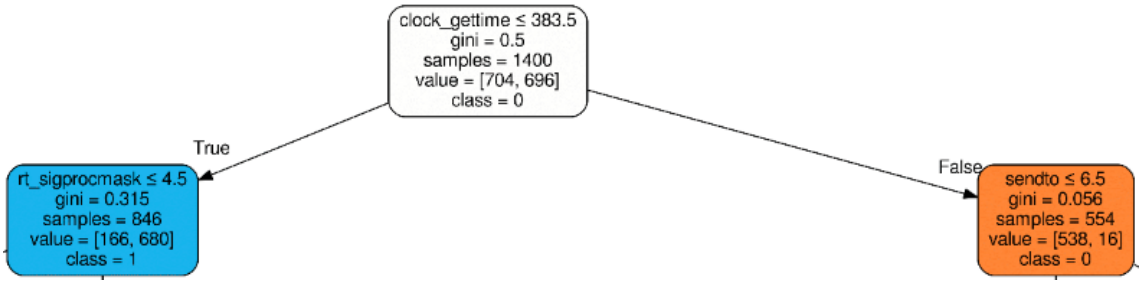


Figure 15. Decision Tree L/O Dataset root node

5.7.1.2 Legitimate / New Malware Dataset Trained Decision Tree

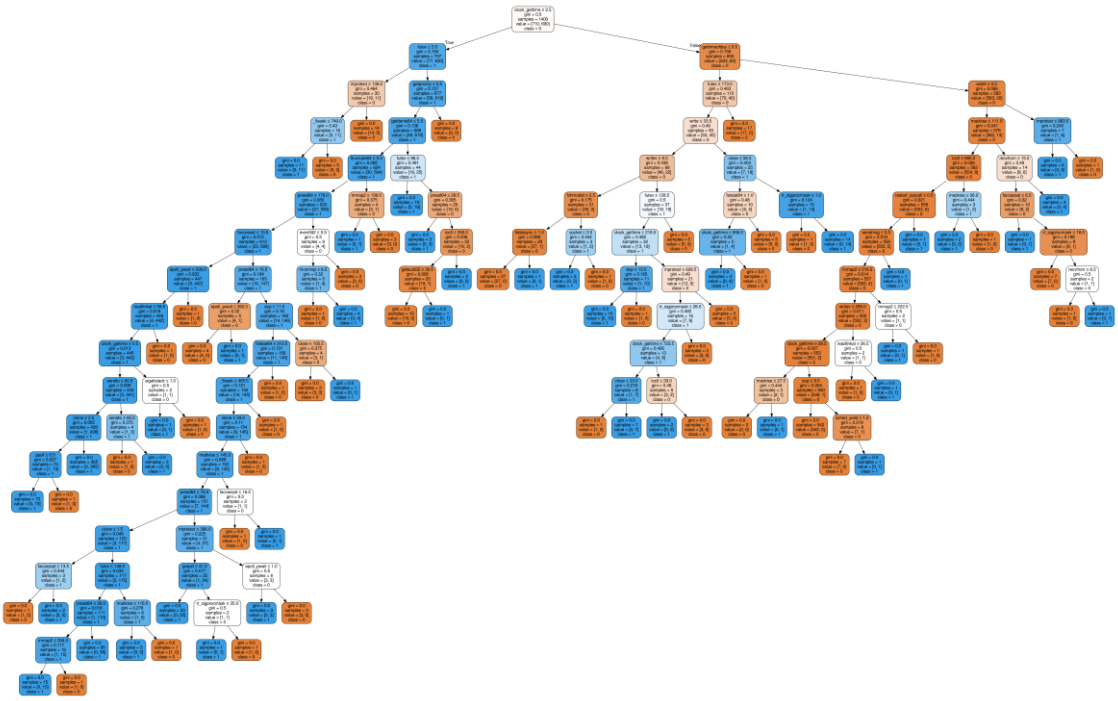


Figure 16. Decision Tree L/N Dataset

Again, if we look at first layers on left side of the tree, we can state that root node is placed by *clock_gettime*, the most discriminative feature in this present research for malware detection:

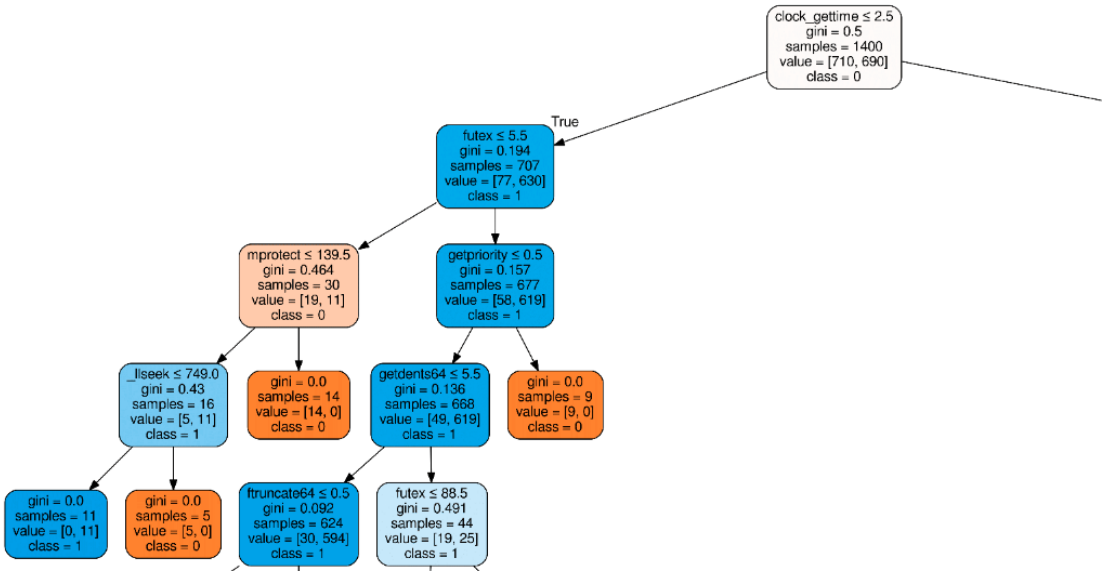


Figure 17. Decision Tree L/N Dataset root node

5.7.1.3 Old Malware / New Malware Dataset Trained Decision Tree

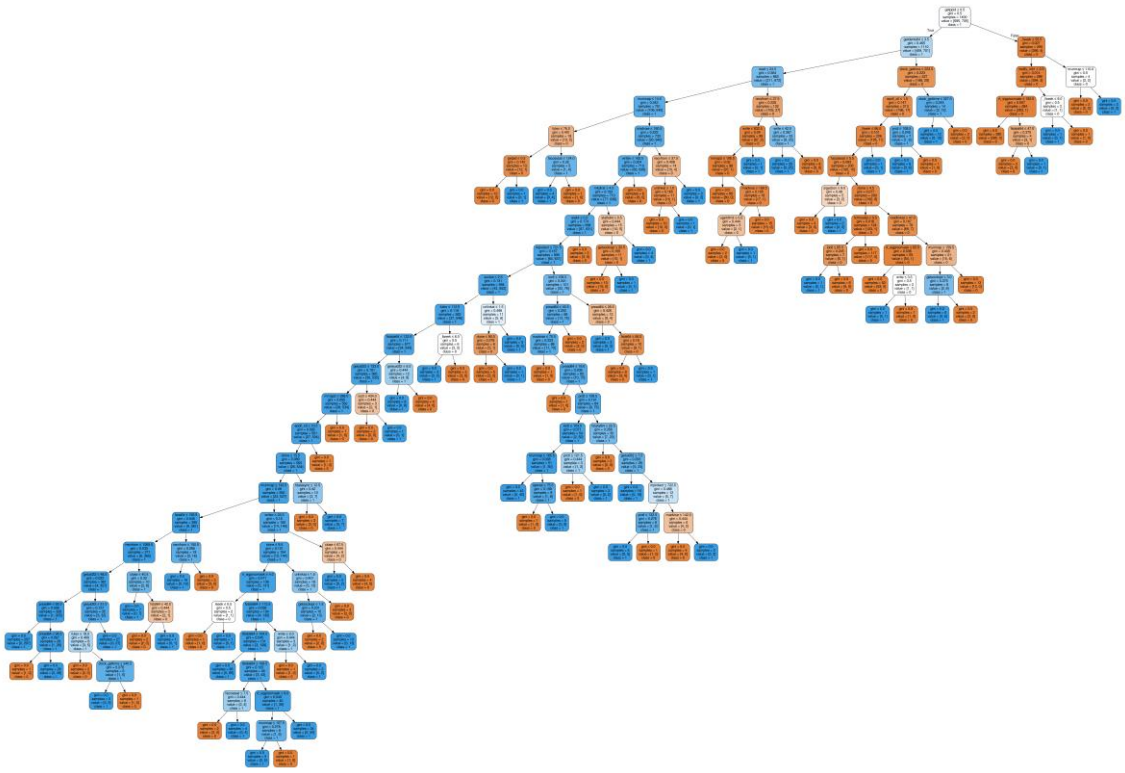


Figure 18. Decision Tree O/N Malware Dataset

By looking again top node or root, we can see now that it is not *clock_gettime* this time, as this comparison was not analyzed by Fisher score. Nevertheless, top node is placed by another import selected feature, *getppid* which is the most discriminant feature in this particular all-malware dataset (discrimination across malware).

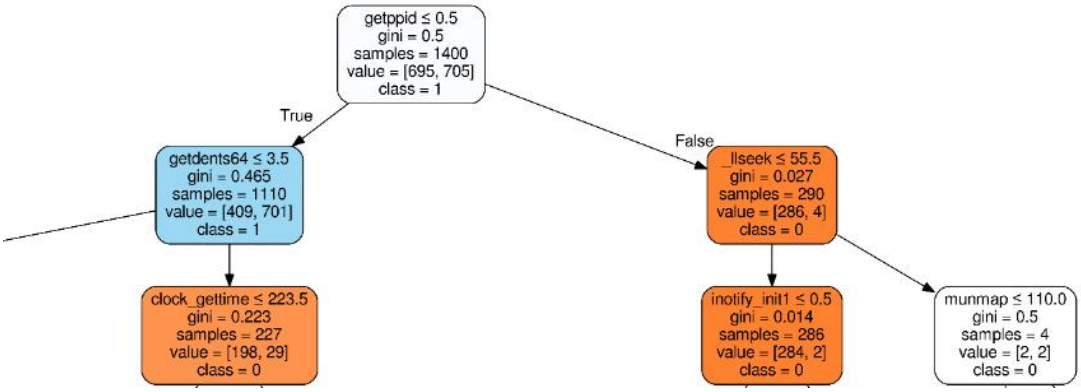


Figure 19. Decision Tree O/N Malware Dataset root node

5.7.2 Permissions

5.7.2.1 Legitimate / Old Malware Dataset Trained Decision Tree

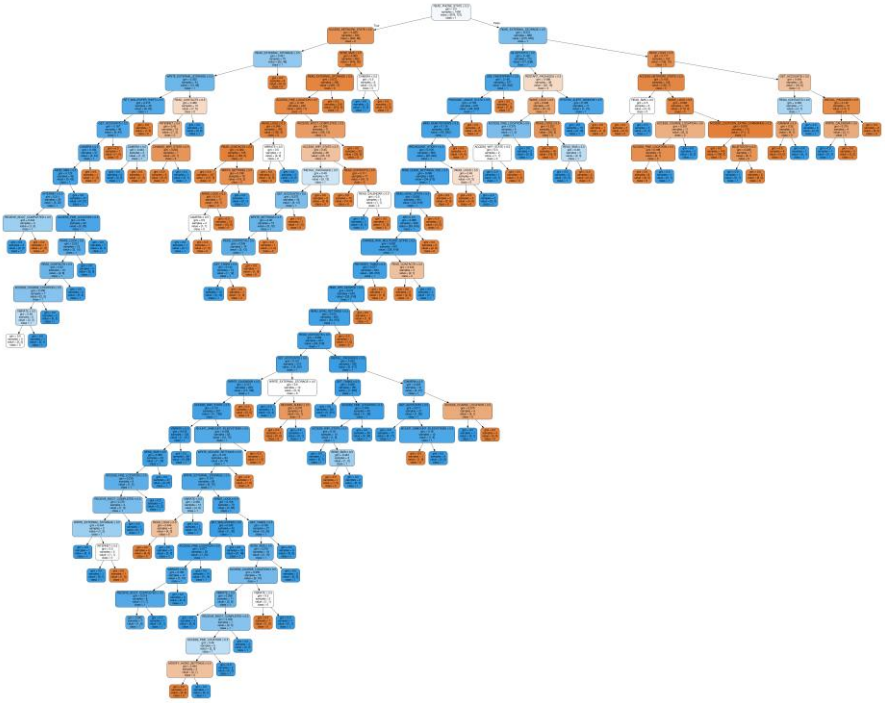


Figure 20. Decision Tree L/O Malware Dataset

This particular tree places as top node READ_PHONE_STATE, the one that was already highlighted as the best Gini index predictor in L/O dataset. Next two decision points or branches are placed by the second and third better Gini index values, as can be seen in the following picture:

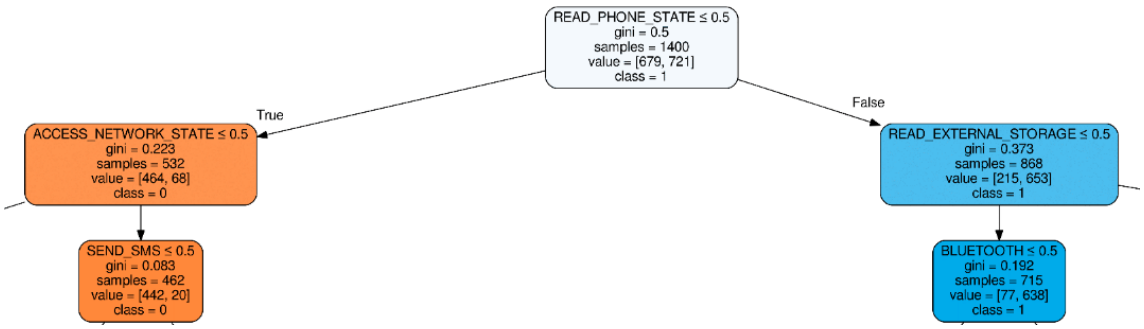


Figure 21. Decision Tree L/O Malware Dataset root node

5.7.2.2 Legitimate / New Malware Dataset Trained Decision Tree

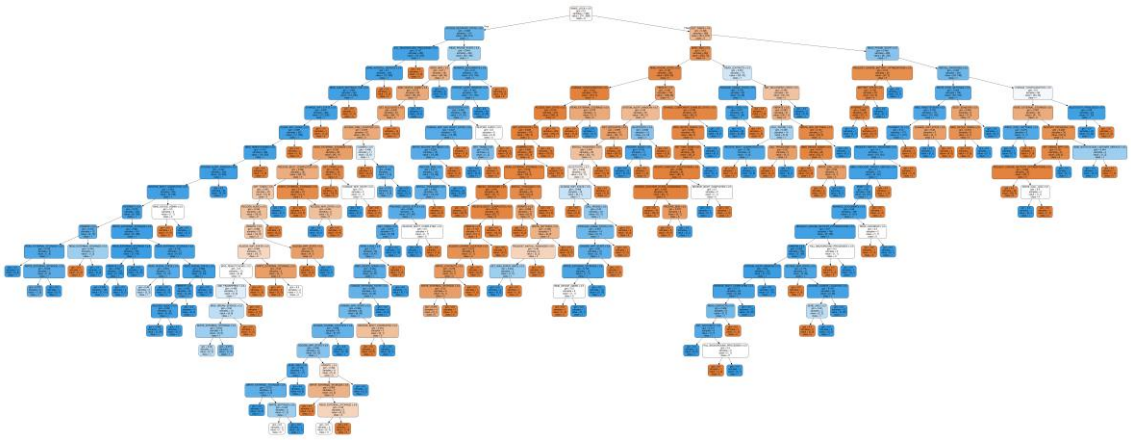


Figure 22. Decision Tree L/N Malware Dataset

Top node is placed by WAKE_LOCK permission, the most discriminative one stated by Gini index in feature selection performed by this research.

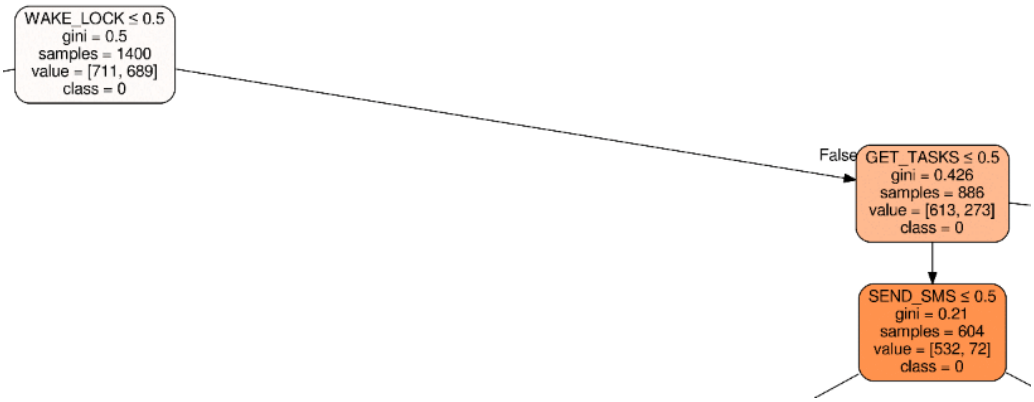


Figure 23. Decision Tree L/N Malware Dataset root node

5.7.2.3 Old Malware / New Malware Dataset Trained Decision Tree

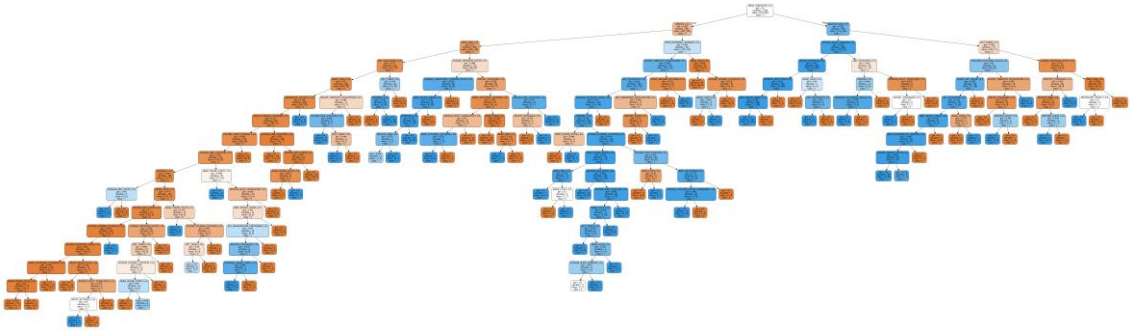


Figure 24. Decision Tree O/N Malware Dataset

In cross-malware tree, top node is placed by READ_CONTACTS, which is the best discriminative permission across malware.

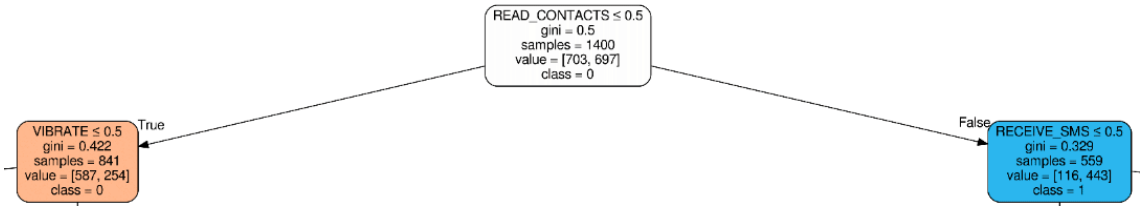


Figure 25. Decision Tree O/N Malware Dataset root node

5.7.3 Hybrid trees

5.7.3.1 Legitimate / Old Malware Dataset Trained Decision Tree

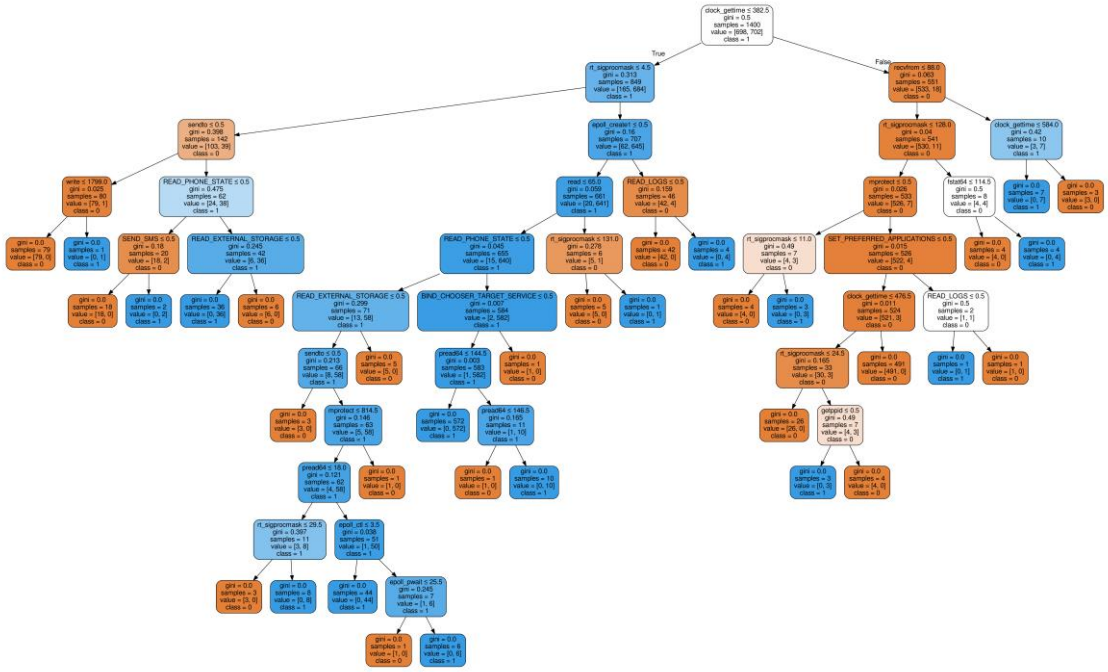


Figure 26. Decision Tree L/O Malware Dataset

Using permissions and system calls together, root node is placed by a system call, the most discriminative feature stated by this resarch, clock_gettime system call.

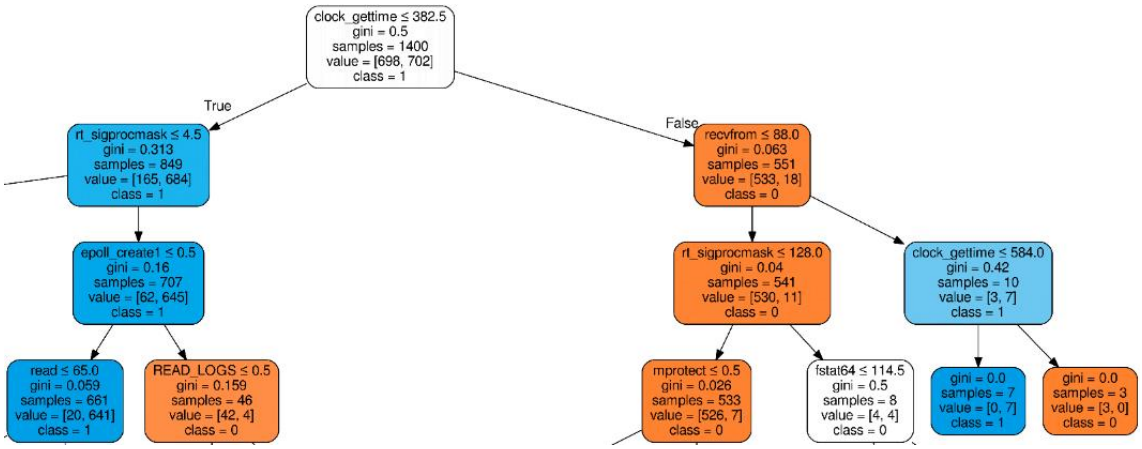


Figure 27. Decision Tree L/O Malware Dataset root node

5.7.3.2 Legitimate / New Malware Dataset Trained Decision Tree

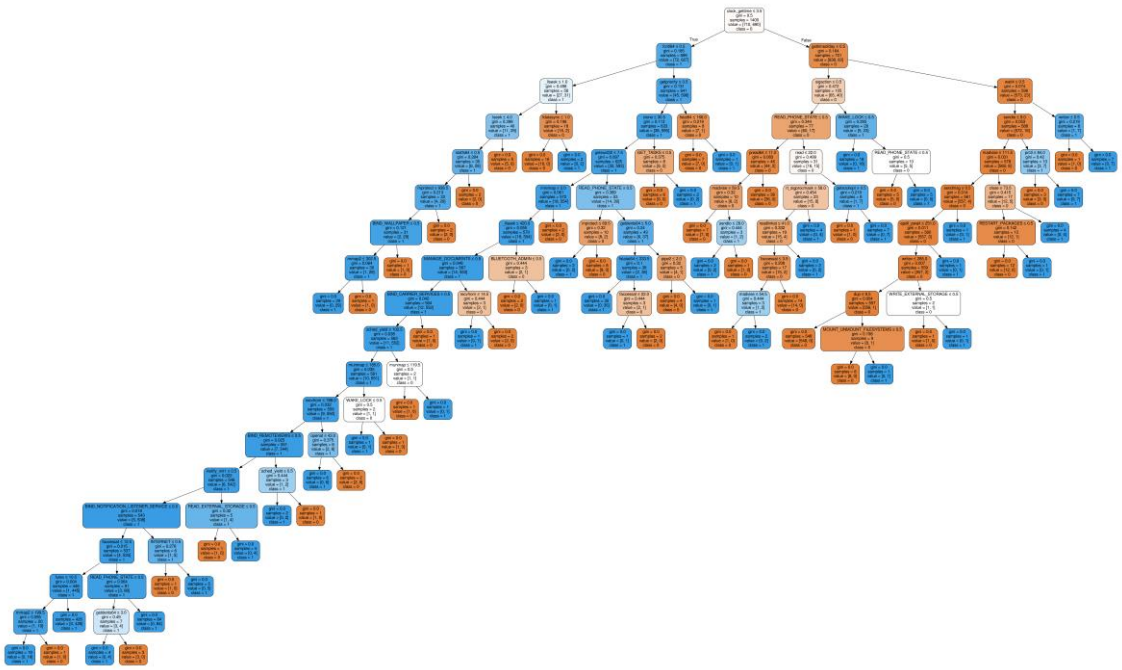


Figure 28. Decision Tree L/N Malware Dataset

Once again, top node is placed by a system call, clock_gettime:

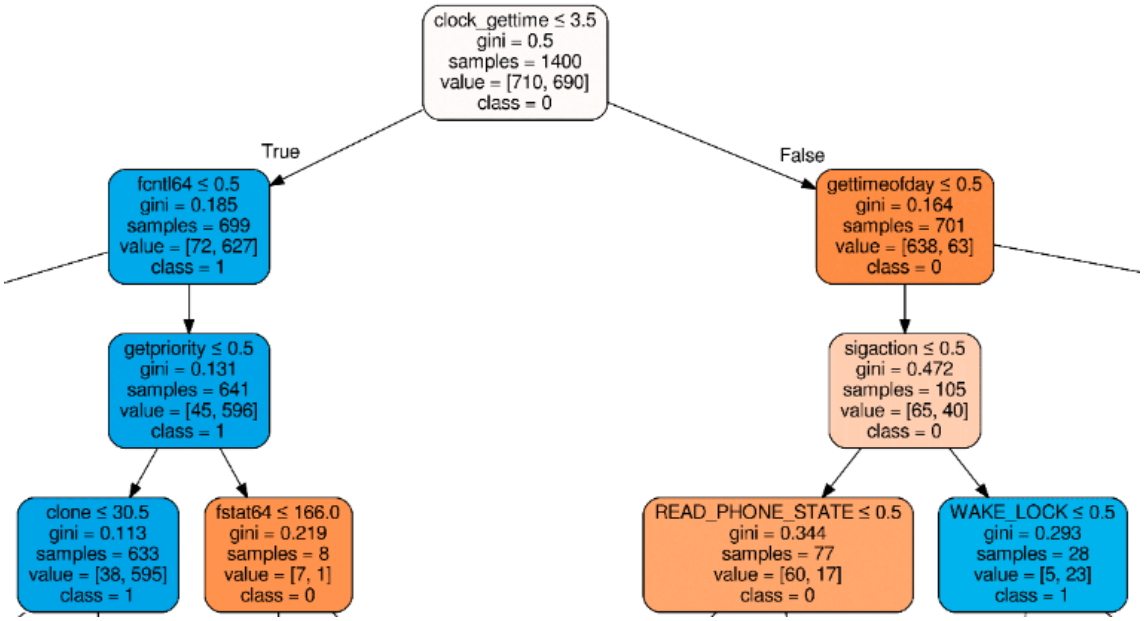


Figure 29. Decision Tree L/N Malware Dataset root node

5.7.3.3 Old Malware / New Malware Dataset Trained Decision Tree

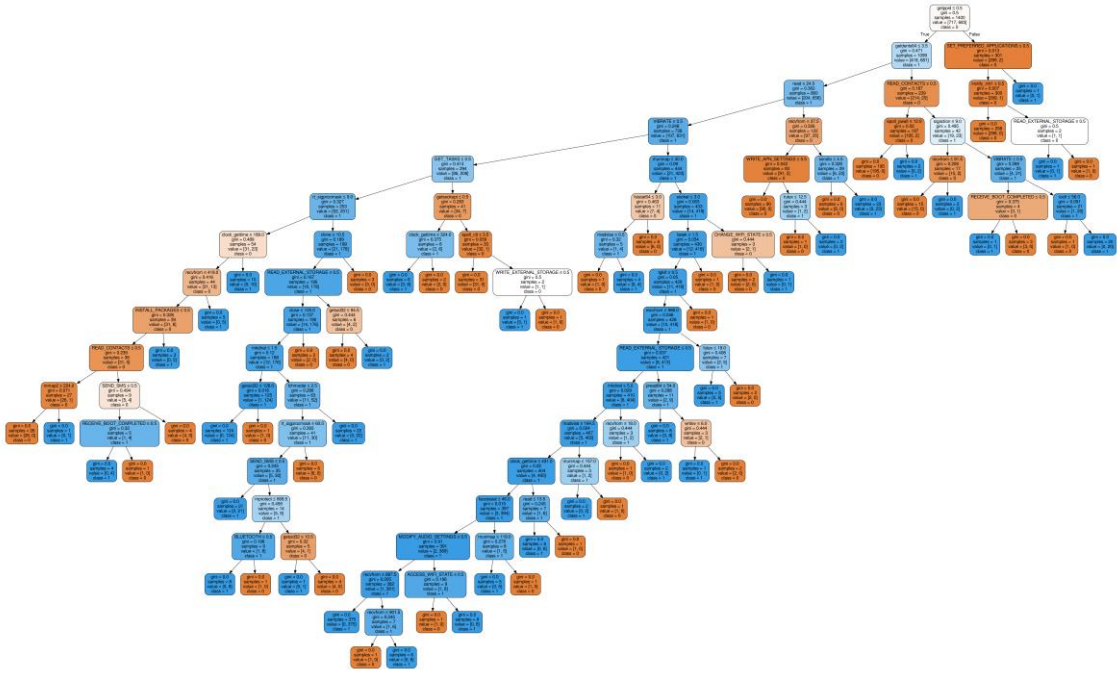


Figure 30. Decision Tree O/N Malware Dataset

This last picture shows again that system calls have more discriminant power than permissions when using together:

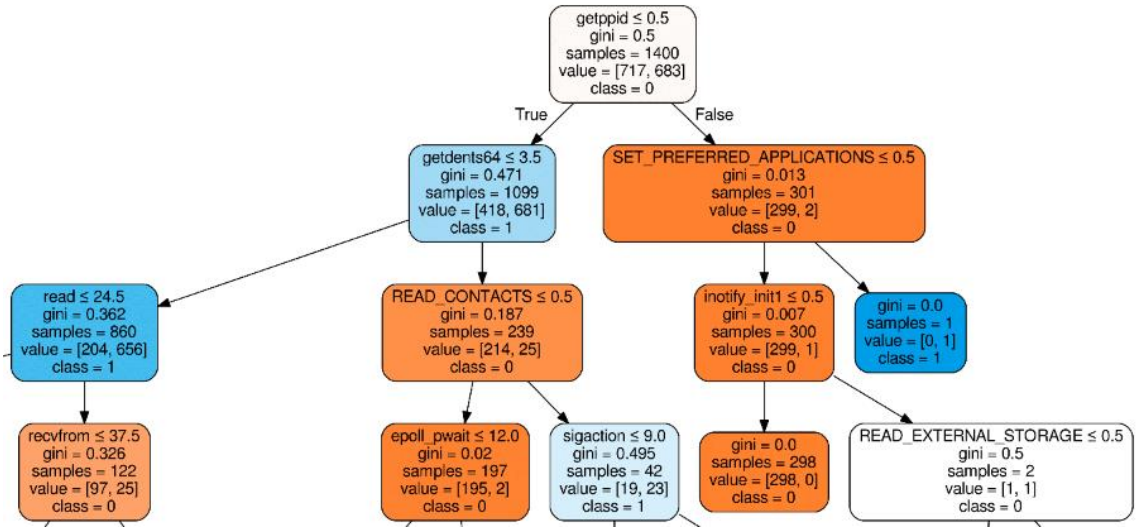


Figure 31. Decision Tree O/N Malware Dataset root node

6 Conclusions

In this thesis, applicability of dynamic features (system calls) and static features (permissions) were investigated with respect to build a classifier for detect malware in Android environment. Feature selection methods provided a reduced number of features in system calls and permission whole domains that were used to train and test classification models. System calls analysis stated that they are distributed with greater differentiation between legitimate and malware applications than permissions, providing greater discriminative power. Analysis of feature selection states that best predictive variables have changed in the two malware datasets used in this research, with a time difference of 8 years between them (2010 to 2018). In this regard, in some variables malware has become more similar to legitimate applications, reducing the discriminative power of such variables while others have increased its discriminative impact. In this last case, when using only one specific feature, old malware used as training set could create a well-defined decision boundary against legitimate applications that could be used to accurately predict unknown new malware samples. This fact provides the possibility to train classifiers only with old malware that will accurately discriminate new malware, overcoming the efforts of new malware to become undetectable by becoming similar to legitimate applications. From the point of view of malware detection, accuracy results show that it is possible to detect malware minimizing the features used, obtaining good accuracy results using a small number of features as predictors (from 1 to 40 features) with as good or better than using all features. More specifically, results show that if using a single feature as predictor, training with old dataset could provide better results in detecting both new and old malware, as old dataset creates a well-defined decision boundary where new malware lies behind it. When more features are used as predictors, new dataset as training set provides better results in cross-dataset performance accuracy. This finding points out that depending on features selected, it would be preferable to use a different dataset to train the model to get optimal predictions. Regarding dataset used, mixed malware dataset does not provide significant difference in performance than using old or new

datasets separately. Old vs. new malware comparison provided that permissions are a better feature to discriminate between malware samples than system calls. This fact suggests that new malware evolved to become more similar to legitimate applications regarding permissions than system calls, increasing separability from old malware in this feature, but not as much in its dynamic behaviour.

Feature selection has been highlighted as a key point in malware detection, allowing to minimize features used with optimal detection accuracies in cross-malware datasets. This fact may lead to build faster and less complex classifiers that focus on main predictors, that may vary depending on dataset used and detection needs. This research shows that depending on detection objectives and requirements, different features and dataset should be used in order to accomplish them with optimal malware detection performance.

References

[1] Go-Globe. “Mobile vs. Desktop Internet Usage – Statistics and Trends”, Nov, 2016. [Online]. Available: https://www.go-globe.hk/blog/mobile-vs-desktop-internet-usage/ [Accessed: March 1, 2018].

[2] J. Stevens. “Internet Stats & Facts for 2017”. Aug, 2017. [Online]. Available: https://hostingfacts.com/internet-facts-stats-2016/ [Accessed: March 1, 2018].

[3] McAfee. “McAfee Mobile Threat Report Q1, 2018”. Mar, 2018. [Online]. Available: https://www.mcafee.com/es/resources/reports/rp-mobile-threat-report-2018.pdf. [Accessed: March 31, 2018]

[4] Kaspersky. “Mobile malware evolution 2017”. Mar, 2018. [Online]. Available: https://securelist.com/mobile-malware-review-2017/84139/. [Accessed: March 31, 2018].

[5] Trendmicro. “2017 Mobile Threat Landscape”. Feb, 2018. [Online]. Available: https://www.trendmicro.com/vinfo/us/security/research-and-analysis/threat-reports/roundup/2017-mobile-threat-landscape. [Accessed: April 1, 2018].

[6] R. Fedler, J. Schütte and M. Kulicke. “On the Efectiveness of Malware Protection on Android. An Evaluation of Android Antivirus Apps”. Fraunhofer, AISEC. Apr, 2013.

[7] Statista. “Global mobile OS market share in sales to end users from 1st quarter 2009 to 2nd quarter 2017”. 2018. [Online]. Available: https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/. [Accessed: April 1, 2018].

[8] Statista. “Distribution of Android operating systems used by Android phone owners in February 2018, by platform versions”. 2018. [Online]. Available: https://www.statista.com/statistics/271774/share-of-android-platforms-on-mobile-devices-with-android-os/. [Accessed: April 1, 2018].

[9] D. Arp, M. Spreitzenbarth, M. Huebner, H. Gascon and K. Rieck. "Drebin: Efficient and Explainable Detection of Android Malware in Your Pocket", 21th Annual Network and Distributed System Security Symposium (NDSS), February 2014.

[10] Z. Yuan, Y. Lu, Z. Wang and Y. Xue. “Droid-Sec: Deep Learning in Android Malware Detection”, in *SIGCOMM Computer Communication Review*, vol. 44, number 4, pp. 371-372, October 2014.

[11] J. Sahs and L. Khan. “A Machine Learning Approach to Android Malware Detection”, in *Intelligence and security informatics conference (eisic), 2012 european.* IEEE, 2012.

[12] A. Feizollah, N.B. Anuar, R. Salleh and A. Wahid. “A review on feature selection in mobile malware detection”, in *Digital Investigation*, vol. 13, pp. 22-37, March 2015.

[13] Android. “App Permissions”, 2018. [Online]. Available: https://developer.android.com/guide/topics/permissions/index.html. [Accessed: April 3, 2018].

[14] Android. “Glossary”, 2018. [Online]. Available: https://developer.android.com/guide/appendix/glossary.html. [Accessed: April 3, 2018].

[15] Y. Zhou and X. Jiang. “Dissecting Android Malware: Characterization and Evolution”, in *2012 IEEE Symposium on Security and Privacy*, San Francisco, CA, 2012, pp. 95-109.

[16] T. M. Mitchell. “Machine Learning”, McGraw-Hill, 1997.

[17] M. Hoogendoorn and B. Funk. “Machine Learning for the Quantified Self”, Springer, 2018.

[18] B. Baskaran and A. Ralescu. “A Study of Android Malware Detection Techniques and Machine Learning”, in *MAICS: The Modern Artificial Intelligence and Cognitive Science Conference,* 2016, pp. 15-23.

[19] Android. “Manifest.permission”, 2018. [Online]. Available: https://developer.android.com/reference/android/Manifest.permission.html. [Accessed: April 3, 2018].

[20] Android. “Permissions Overview”, 2018. [Online]. Available: https://developer.android.com/guide/topics/permissions/overview.html. [Accessed: April 1, 2018].

[21] M. Nezhadkamali, S. Soltani and S.A.H. Seno. “Android malware detection based on overlapping of static features”, in *7th International Conference on Computer and Knowledge Engineering (ICCKE 2017),* 2017.

[22] N. Peiravian and X. Zhu. “Machine Learning for Android Malware Detection Using Permission and API Calls”, in *IEEE 25th International Conference on Tools with Artificial Intelligence*, 2013.

[23] X. Liu and J. Liu. “A Two-layered Permission-based Android Malware Detection Scheme”, in *IEEE International Conference on Mobile Cloud Computing, Services, and Engineering,* 2014.

[24] K. A. Talha, D. I. Alper and C. Aydin. “APK Auditor: Permission-based Android malware detection system”, in *Digital Investigation*, vol. 13, March, 2015, pp. 1-14.

[25] Contagio. “Contagio Malware Dump”, 2018. [Online]. Available: http://contagiodump.blogspot.com. [Accessed: April 5, 2018].

[26] D.-J. Wu, C.-H. Mao, T.-E. Wei, H.-M. Lee and K.-P. Wu. “DroidMat: Android Malware Detection through Manifest and API Calls Tracing”, in Information Security (Asia JCIS), 2012 Seventh Asia Joint Conference, 2012, pp. 62-69.

[27] Z. Aung and W. Zaw. “Permission-Based Android Malware Detection”, in International Journal of Scientific & Technology Research, vol.2, number 3, March 2013.

[28] B. Sanz, I. Santos, C. Laorden, X. Ugarte-Pedrero, P. Garcia Bringas and G. Alvarez. “PUMA: Permission Usage to detect Malware in Android”, in International Joint Conference CISIS’12-ICEUTE´12-SOCO´12 Special Sessions, 2012, pp 289-298.

[29] S. Liang and X. Du. “Permission-Combination-based Scheme for Android Mobile Malware Detection”, in IEEE Mobile and Wireless Networking Symposium, 2014.

[30] R. V. Varma P., K. P. Raj and K. V. Subba Raju. “Android mobile security by detecting and classification of malware based on permissions using machine learning algorithms”, in International conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud), 2017.

[31] Google. “Android platform bionic C library supported syscalls”, 2018. [Online]. Available: https://android.googlesource.com/platform/bionic/+/cd58770/libc/SYSCALLS.TXT. [Accessed: March 3, 2018].

[32] I. Burguera, U. Zurutuza and S. Nadjm-Tehrani, “Crowdroid: Behavior-Based Malware Detection System for Android”, in Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices, 2011, pp. 15.

[33] Strace. “Strace – Linux syscall tracer”, 2018. [Online]. Available: https://strace.io/. [Accessed: March 2, 2018].

[34] M. Kerrisk. “Linux Programmer’s Manual - Ptrace”, 2018. [Online]. Available: http://man7.org/linux/man-pages/man2/ptrace.2.html. [Accessed: March 2, 2018].

[35] M. Dimjasevic, S. Atzeni, and I. Ugrina, “Evaluation of Android Malware Detection Based on System Calls,” In *Proc. ACM on International Workshop on Security And Privacy Analytics*, New Orleans, LA, USA , 2016, pp. 1-8.

[36] S. Zhang and X. Xiao. “CSCdroid: Accurately Detect Android Malware via Contribution-Level-based System Call Categorization”, in IEEE Trustcom/BigDataSE/ICESS, 2017.

[37] J. Maestre Vidal, A.L. Sandoval Orozco, L. J. García Villalba. “Malware Detection in Mobile Devices by Analyzing Sequences of System Calls”, in *World Academy of Science, Engineering and Technology International Journal of Computer, Electrical, Automation, Control and Information Engineering* Vol:11, No:5, 2017.

[38] X. Xiao, X. Xiao, Y. Jiang, X. Liu and R. Ye. “Identifying Android malware with system call co-occurrence matrices”, in *Transactions on Emerging Telecommunications Technologies*, 2016, pp. 675-684.

[39] Y.-D. Lin, Y.-C. Lai, C.-H. Chen and H.-C. Tsai. “Identifying android malicious repackaged applications by thread-grained system call sequences”, in Computers & Security, 2013, pp. 340-350.

[40] VirusTotal. “VirusTotal a free online virus, malware and URL scanner”. [Online]. Available at: https://www.virustotal.com/. [Accessed: April 4, 2018].

[41] V. M. Afonso, M. F. de Amorim, A. R. A. Grégio, G. B. Junquera and P. L. de Geus. “Identifying Android malware using dynamically obtained features”, in Journal of Computer Virology and Hacking Techniques 2014, pp. 1–9.

[42] APIMonitor. “Droidbox – APIMonitor.wiki”. [Online]. Available: https://code.google.com/archive/p/droidbox/wikis/APIMonitor.wiki. [Accessed: April 4, 2018].

[43] V. Wahanggara and Y. Prayudi. “Malware Detection Through Call System on Android Smartphone Using Vector Machine Method”, in Fourth International Conference on Cyber Security, Cyber Warfare, and Digital Forensic, 2015.

[44] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer and Y. Weiss."'Andromaly': a behavioral malware detection framework for android devices”, in Journal of Intelligent lnformation Systems, 2012, pp.161-190.

[45] C. Da, Z. Hongmei and Z. Xiangli. “Detection of Android Malware Security on System Calls”, in *IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*, 2016.

[46] A. Saracino, D. Sgandurra, G. Dini and F. Martinelli. “MADAM: Effective and Efficient Behavior-based Android Malware Detection and Prevention” in IEEE Transactions on Dependable and Secure Computing, vol. 15, 2018, pp. 83-97.

[47] S. Hou, A. Saas, L. Chen and Y. Ye. “Deep4MalDroid: A Deep Learning Framework for Android Malware Detection Based on Linux Kernel System Call Graphs”, in IEEE/WIC/ACM International Conference on Web Intelligence Workshops, 2016.

[48] L. Singh and M. Hofmann. “Dynamic Behavior Analysis of Android Applications for Malware Detection”, in International Conference on Intelligent Communication and Computational Techniques (ICCT), 2017.

[49] A. Ferrante, E. Medvet, F. Mercaldo, J. Milosevic and C. A. Visaggio. “Spotting the Malicious Moment: Characterizing Malware Behavior Using Dynamic Features”, in 11th International Conference on Availability, Reliability and Security, 2016.

[50] G. Canfora, E. Medvet, F. Mercaldo and C.A. Visaggio. “Detecting Android Malware using Sequences of System Calls”, in Proceedings of the 3rd International Workshop on Software Development Lifecycle for Mobile, 2015, pp. 13-20.

[51] X. Xiao, P. Fu, X. Xiao, Y. Jiang, Q. Li and R. Lu. “Two Effective Methods to Detect Mobile Malware”, in 4th International Conference on Computer Science and Network Technology, 2015.

[52] M. Lindorfer, M. Neugschwandtner and C. Platzer. “MARVIN: Efficient and Comprehensive Mobile App Classification Through Static and Dynamic Analysis”, in IEEE 39th Annual International Computers, Software & Applications Conference, 2015.

[53] APKMirror. “APKMirror.com”. [Online]. Available: https://www.apkmirror.com/. [Accessed: January 7, 2018].

[54] VirusTotal. “How to use VirusTotal Community”. [Online]. Available: https://www.virustotal.com/es/documentation/virustotal-community/ [Accessed: February 20, 2018].

[55] A. Mullis. “Android virtual devices: AVD Manager versus Genymotion“, November 20, 2015. [Online]. Available: https://www.androidauthority.com/android-virtual-devices-avd-manager-versus-genymotion-653093/. [Accessed: March 31, 2018].

[56] W. Mauerer. “Professional Linux Kernel Architecture”, Wiley Publishing, 2015.

[57] Google. “Git repositories on Android: Android x86 Kernel”. [Online]. Available: https://android.googlesource.com/kernel/x86/+/android-x86-anthracite-3.10-nougat-mr1-wear-release/include/linux/ [Accessed: February 20, 2018].

[58] GNU. “The GNU C Library (glibc)”. [Online]. Available: https://www.gnu.org/software/libc/libc.html [Accessed: February 21, 2018].

[59] Google. “Android Anatomy and Physiology”. [Online]. Available: https://web.archive.org/web/20160408053917/http://androidteam.googlecode.com/files/Anatomy-Physiology-of-an-Android.pdf [Accessed: February 22, 2018].

[60] M. Kerrisk. “Linux Programmer’s Manual: *ptrace*”. [Online]. Available: http://man7.org/linux/man-pages/man2/ptrace.2.html [Accessed: March 22, 2018].

[61] Google. “Git repositories on Android: Nougat Syscalls.txt”. [Online]. Available: https://android.googlesource.com/platform/bionic/+/nougat-mr1.1-release/libc/SYSCALLS.TXT [Accessed: January 30, 2018].

[62] Android. “Define a custom app Permission”. [Online]. Available: https://developer.android.com/guide/topics/permissions/defining.html [Accessed: February 30, 2018].

[63] Eberly College of Science. “Hypothesis Testing”. [Online]. Available: https://onlinecourses.science.psu.edu/statprogram/node/136 [Accessed: February 30, 2018].

[64] Boston University School of Public Health. “Hypothesis testing for Means and Proportions”. [Online]. Available: http://sphweb.bumc.bu.edu/otlt/MPH-Modules/BS/BS704_HypothesisTest-Means-Proportions/BS704_HypothesisTest-Means-Proportions3.html [Accessed: February 30, 2018].

[65] Y. Narathy. “Chi-square Distribution Table”. [Online]. Available: https://people.smp.uq.edu.au/YoniNazarathy/stat_models_B_course_spring_07/distributions/chisqtab.pdf [Accessed: April 28, 2018].

[66] C. C. Aggarwal. “Data Mining: The textbook”, Springer, 2015.

[67] J. Brownlee. “An introduction to feature selection”. Oct, 2014. [Online]. Available: https://machinelearningmastery.com/an-introduction-to-feature-selection/ [Accessed: February 30, 2018].

[68] I. Guyon and A. Elisseeff. “An Introduction to Variable and Feature Selection” in Journal of Machine Learning Research 3, pp. 1157-1182, 2003.

[69] Die.net. “Linux Man Pages”. [Online]. Available: https://linux.die.net/man/ [Accessed: March 20, 2018].

[70] Google. “<permission>”. [Online]. Available: https://developer.android.com/guide/topics/manifest/permission-element [Accessed: January 30, 2018].

[71] Scikit-learn. “sklearn.model_selection.KFold”. [Online]. Available: http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html [Accessed: January 30, 2018].

[72] Scikit-learn. “Scikit-learn: Machine learning in Python”. [Online]. Available: http://scikit-learn.org [Accessed: January 30, 2018].

[73] F. Gorunescu. “Data Mining”, Springer, 2011.

[74] Sifium. “Types of classification algorithms in machine learning”, Feb, 2017. [Online]. Available: https://medium.com/@sifium/machine-learning-types-of-classification-9497bd4f2e14 [Accessed: February 30, 2018].

[75] Scikit-learn. “Decision Trees”. [Online]. Available: http://scikit-learn.org/stable/modules/tree.html [Accessed: January 28, 2018].

[76] T. Plapinger. “What is a decision tree?”, Jul, 2017. [Online]. Available: https://towardsdatascience.com/what-is-a-decision-tree-22975f00f3e1 [Accessed: January 28, 2018].

[77] P. Gupta. “Decision Trees in Machine Learning”, May, 2017. [Online]. Available: https://towardsdatascience.com/decision-trees-in-machine-learning-641b9c4e8052 [Accessed: January 28, 2018].

Appendix 1 – System call's statistics

Next table shows the mean value and standard deviation for each dataset composed by 1000 applications and 212 system calls:

System Call	Legitimate Dataset		Old Malware Dataset		New Malware Dataset	
	Mean	Std Dev	Mean	Std Dev	Mean	Std Dev
sched_get_priority_min	0.004	0.0894	0.002	0.0632	0.002	0.0632
Lseek	0.852	4.8925	0.6235	3.4842	0.916	4.0779
Pipe	0.004	0.0894	0.002	0.0632	0.002	0.0632
epoll_ctl	1.072	1.5965	1.317	1.6645	1.402	1.8194
rt_sigtimedwait	0.0	0.0	0.0	0.0	0.0	0.0
Setfsuid	0.0	0.0	0.0	0.0	0.0	0.0
Tee	0.0	0.0	0.0	0.0	0.0	0.0
Uname	0.026	0.4973	0.014	0.3547	0.013	0.3519
Kill	0.0	0.0	0.0005	0.0224	0.0	0.0
Swapoff	0.0	0.0	0.0	0.0	0.0	0.0
Readahead	0.0	0.0	0.0	0.0	0.0	0.0
clock_getres	0.005	0.0705	0.0025	0.0499	0.0025	0.0499
Preadv	0.0	0.0	0.0	0.0	0.0	0.0
setresgid32	0.0	0.0	0.0	0.0	0.0	0.0
Gettid	0.225	6.4683	0.1125	4.5751	0.1125	4.5751
Sethostname	0.0	0.0	0.0	0.0	0.0	0.0
timer_delete	0.0	0.0	0.0	0.0	0.0	0.0
Umask	0.02	0.3682	0.01	0.2606	0.01	0.2606
sched_getaffinity	0.0	0.0	0.0	0.0	0.0	0.0
Writev	12.449	39.6861	12.952	29.2155	27.3565	86.0063
sched_setparam	0.0	0.0	0.0	0.0	0.0	0.0
Fchmod	0.026	0.2265	0.013	0.1607	0.096	0.5242
_llseek	30.889	130.2178	18.2055	93.0928	19.8055	96.0433
Getpid	0.371	4.3746	0.278	3.7406	0.7905	8.1252
setreuid32	0.0	0.0	0.0	0.0	0.0	0.0
signalfd4	0.0	0.0	0.0	0.0	0.0	0.0
epoll_wait	0.001	0.0316	0.0005	0.0224	0.0005	0.0224
Capset	0.0	0.0	0.0	0.0	0.0	0.0
Personality	0.0	0.0	0.0	0.0	0.0	0.0
delete_module	0.0	0.0	0.0	0.0	0.0	0.0

dup3	0.0	0.0	0.0	0.0	0.0	0.0
sched_setaffinity	0.0	0.0	0.0	0.0	0.0	0.0
Read	48.165	155.4994	29.188	111.8058	53.328	175.6358
Getppid	0.313	0.4637	0.159	0.3657	0.3645	0.4813
getgid32	0.0	0.0	0.0	0.0	0.0	0.0
Capget	0.0	0.0	0.0	0.0	0.0	0.0
getgroups32	0.0	0.0	0.0	0.0	0.0	0.0
Readlinkat	13.326	15.7562	25.346	18.8199	25.246	19.6004
sched_rr_get_interval	0.0	0.0	0.0	0.0	0.0	0.0
Setfsgid	0.0	0.0	0.0	0.0	0.0	0.0
Renameat	0.845	5.4607	0.8825	4.3868	0.6685	4.0679
Fsync	1.296	5.7327	1.242	4.6054	1.0835	4.388
geteuid32	3.167	6.6419	3.0705	6.2316	2.236	6.2344
Unshare	0.0	0.0	0.0	0.0	0.0	0.0
epoll_pwait	8.954	28.6496	13.287	33.3084	27.833	65.453
Recvfrom	13.748	78.436	109.2905	243.8441	43.038	121.0542
sched_get_priority_max	0.004	0.0894	0.002	0.0632	0.002	0.0632
Symlinkat	0.0	0.0	0.0	0.0	0.0	0.0
Settimeofday	0.0	0.0	0.0	0.0	0.0	0.0
timer_create	0.0	0.0	0.0	0.0	0.0	0.0
Sendto	1.093	4.9121	9.5155	20.8756	3.6615	9.7667
Mkdirat	2.597	3.5359	2.01	2.6885	2.123	2.9339
Lgetxattr	0.0	0.0	0.0	0.0	0.0	0.0
Linkat	0.0	0.0	0.0	0.0	0.0	0.0
Shutdown	0.0	0.0	0.0	0.0	0.001	0.0316
epoll_create1	0.338	0.4876	0.1715	0.3861	0.3705	0.4901
getresuid32	0.0	0.0	0.0	0.0	0.0	0.0
Adjtimex	0.0	0.0	0.0	0.0	0.0	0.0
Sync	0.0	0.0	0.0	0.0	0.0	0.0
Syslog	0.0	0.0	0.0	0.0	0.0	0.0
Fchownat	0.0	0.0	0.0	0.0	0.0	0.0
setgid32	0.0	0.0	0.0	0.0	0.0	0.0
setregid32	0.0	0.0	0.0	0.0	0.0	0.0
fadvise64_64	0.0	0.0	0.0	0.0	0.0	0.0
Getsockname	0.008	0.1412	0.0075	0.1202	0.0075	0.1159
Close	45.603	25.6286	55.4745	25.6668	54.606	26.2713
Flock	0.442	5.0194	0.221	3.5561	0.428	3.7184
Lsetxattr	0.0	0.0	0.0	0.0	0.0	0.0
Pwritev	0.0	0.0	0.0	0.0	0.0	0.0
Llistxattr	0.0	0.0	0.0	0.0	0.0	0.0
Tgkill	0.0	0.0	0.0	0.0	0.001	0.0316
getegid32	0.0	0.0	0.0	0.0	0.0	0.0
pselect6	0.0	0.0	0.0	0.0	0.0	0.0

rt_sigprocmask	12.602	27.3517	40.095	62.0614	18.9485	37.9334
accept4	0.0	0.0	0.0	0.0	0.0	0.0
Reboot	0.0	0.0	0.0	0.0	0.0	0.0
Fremovexattr	0.0	0.0	0.0	0.0	0.0	0.0
setgroups32	0.0	0.0	0.0	0.0	0.0	0.0
Setsid	0.0	0.0	0.0	0.0	0.0	0.0
Exit	0.0	0.0	0.0	0.0	0.0	0.0
timerfd_create	0.0	0.0	0.0	0.0	0.0	0.0
Sigaltstack	0.129	0.6873	0.5385	0.95	0.515	0.9358
inotify_rm_watch	0.0	0.0	0.0	0.0	0.0	0.0
Munmap	55.594	51.9374	92.9135	57.0101	90.9865	58.753
Socketpair	0.006	0.1094	0.004	0.0894	0.016	0.1782
Setrlimit	0.001	0.0316	0.0005	0.0224	0.001	0.0316
Getitimer	0.0	0.0	0.0	0.0	0.0	0.0
Fchmodat	2.603	5.9088	2.4865	4.8525	2.2015	4.6682
Setpgid	0.0	0.0	0.0	0.0	0.0	0.0
Getrandom	0.042	0.4672	0.021	0.331	0.021	0.331
Getcwd	0.032	0.3674	0.016	0.2603	0.016	0.2603
fstatfs64	0.003	0.0706	0.0015	0.05	0.0015	0.05
fstatat64	56.303	92.9289	56.6735	73.6869	60.771	89.0602
rt_sigsuspend	0.0	0.0	0.0	0.0	0.0	0.0
inotify_add_watch	0.042	0.2055	0.0215	0.1485	0.0285	0.1694
getresgid32	0.0	0.0	0.0	0.0	0.0	0.0
timer_settime	0.0	0.0	0.0	0.0	0.0	0.0
Fchdir	0.0	0.0	0.0	0.0	0.0	0.0
timer_gettime	0.0	0.0	0.0	0.0	0.0	0.0
Getpeername	0.0	0.0	0.0	0.0	0.0	0.0
Sigaction	0.614	2.2466	2.6615	4.3166	2.5725	4.0622
Getsid	0.0	0.0	0.0	0.0	0.0	0.0
Mknodat	0.086	1.0033	0.043	0.7107	0.043	0.7107
Setsockopt	0.057	0.939	0.055	0.7007	0.041	0.6851
Munlockall	0.0	0.0	0.0	0.0	0.0	0.0
Msync	0.113	1.7367	0.0565	1.2294	0.0565	1.2294
process_vm_readv	0.0	0.0	0.0	0.0	0.0	0.0
pipe2	0.096	0.7327	0.058	0.5564	0.1045	1.024
set_thread_area	0.0	0.0	0.0	0.0	0.0005	0.0224
Times	0.0	0.0	0.0	0.0	0.0	0.0
prlimit64	0.0	0.0	0.0	0.0	0.0	0.0
pwrite64	14.041	30.3444	12.956	27.4921	9.469	27.4994
Vfork	0.016	0.1725	0.0105	0.1319	0.023	0.2499
Nanosleep	0.057	1.2457	0.0285	0.8813	0.0285	0.8813
Sendmsg	0.0	0.0	0.005	0.0773	0.004	0.0631
Clone	11.936	9.0007	8.7695	7.6508	9.131	7.8919

Flistxattr	0.0	0.0	0.0	0.0	0.0	0.0
Getcpu	0.0	0.0	0.0	0.0	0.0	0.0
Splice	0.0	0.0	0.0	0.0	0.0	0.0
rt_sigaction	0.07	2.2125	0.035	1.5649	0.035	1.5649
clock_gettime	749.879	466.3368	426.187	479.1833	389.67	497.0268
Ugetrlimit	0.014	0.1944	0.022	0.1831	0.0485	0.2432
umount2	0.0	0.0	0.0	0.0	0.0	0.0
rt_sigreturn	0.0	0.0	0.0	0.0	0.004	0.1581
rt_sigpending	0.0	0.0	0.0	0.0	0.0	0.0
Setxattr	0.0	0.0	0.0	0.0	0.0	0.0
Getpgid	0.0	0.0	0.0	0.0	0.0	0.0
Brk	0.0	0.0	0.0	0.0	0.0	0.0
Fsetxattr	0.0	0.0	0.0	0.0	0.0	0.0
Acct	0.0	0.0	0.0	0.0	0.0	0.0
clock_settime	0.0	0.0	0.0	0.0	0.0	0.0
Getsockopt	3.665	12.9546	3.212	9.616	3.683	11.0707
exit_group	0.0	0.0	0.0	0.0	0.0	0.0
Getpriority	0.094	0.3304	0.048	0.2402	0.048	0.2402
Listxattr	0.0	0.0	0.0	0.0	0.0	0.0
Sysinfo	0.0	0.0	0.0	0.0	0.0	0.0
Removexattr	0.0	0.0	0.0	0.0	0.0	0.0
Faccessat	22.461	22.7219	18.258	18.8957	17.8985	21.8602
Dup	3.502	3.1366	3.871	3.6028	3.682	3.1831
Fdatasync	5.344	11.2058	5.3635	10.8609	3.89	10.1719
Setns	0.0	0.0	0.0	0.0	0.0	0.0
Mprotect	169.244	255.1326	268.415	300.745	295.302	312.3796
Listen	0.008	0.1093	0.0045	0.0805	0.0045	0.0805
fchown32	0.0	0.0	0.0	0.0	0.0	0.0
getuid32	19.574	27.4583	21.417	25.6297	32.1925	48.4912
init_module	0.0	0.0	0.0	0.0	0.0	0.0
sched_yield	1.865	8.0529	1.302	6.6048	1.5675	11.058
statfs64	0.268	0.7498	0.261	0.6464	0.1805	0.6066
ftruncate64	0.213	1.4077	0.1065	1.0011	0.114	1.0213
Fgetxattr	0.0	0.0	0.0	0.0	0.0	0.0
Ptrace	0.0	0.0	0.0	0.0	0.0	0.0
Vmsplice	0.0	0.0	0.0	0.0	0.0	0.0
fcntl64	20.851	38.314	20.108	35.4502	15.495	35.3675
Swapon	0.0	0.0	0.0	0.0	0.0	0.0
Fallocate	0.0	0.0	0.0	0.0	0.0	0.0
Execve	0.0	0.0	0.0	0.0	0.0	0.0
Socket	0.456	1.8166	0.7525	1.3602	0.766	1.4924
wait4	0.001	0.0316	0.0005	0.0224	0.122	0.6067
Chdir	0.219	6.9219	0.1095	4.8958	0.1095	4.8958

Madvise	52.268	40.0585	78.3115	43.9307	79.094	47.256
fstat64	60.813	43.9699	69.8725	42.8535	65.273	41.3114
Mount	0.0	0.0	0.0	0.0	0.0	0.0
timerfd_gettime	0.0	0.0	0.0	0.0	0.0	0.0
rt_sigqueueinfo	0.0	0.0	0.0	0.0	0.0	0.0
Getrusage	0.0	0.0	0.0	0.0	0.0	0.0
timerfd_settime	0.0	0.0	0.0	0.0	0.0	0.0
sched_setscheduler	0.016	0.1891	0.008	0.1339	0.008	0.1339
Utimensat	0.0	0.0	0.0	0.0	0.0075	0.2036
Bind	0.016	0.2318	0.0085	0.1656	0.0085	0.1656
eventfd2	0.324	0.4765	0.1645	0.3761	0.3635	0.4851
Connect	0.285	0.458	0.665	0.599	0.649	0.6227
getdents64	3.693	4.2697	2.436	3.4342	7.343	84.2587
Readv	0.0	0.0	0.0	0.0	0.0	0.0
Mremap	0.073	0.9485	0.0365	0.6717	0.0365	0.6717
pread64	47.026	77.9913	51.9545	68.7889	48.6345	81.9527
setuid32	0.0	0.0	0.0	0.0	0.0	0.0
Prctl	43.82	34.3746	68.4385	39.9062	68.3895	41.6805
stat64	0.009	0.2023	0.0045	0.1431	0.0045	0.1431
Recvmmsg	0.0	0.0	0.0	0.0	0.0	0.0
Chroot	0.366	11.5681	0.183	8.182	0.183	8.182
sched_getparam	0.004	0.0894	0.002	0.0632	0.002	0.0632
truncate64	0.0	0.0	0.0	0.0	0.0	0.0
setresuid32	0.0	0.0	0.0	0.0	0.0	0.0
Write	33.224	145.0314	22.8055	119.0203	48.8405	158.9098
Munlock	0.009	0.1641	0.0045	0.1161	0.0045	0.1161
Setpriority	0.024	0.2131	0.0265	0.1918	0.0495	0.2409
Recvmsg	0.0	0.0	0.0	0.0	0.0	0.0
inotify_init1	0.042	0.2006	0.0215	0.145	0.0285	0.1664
Mincore	0.0	0.0	0.0	0.0	0.0	0.0
Sendfile	0.0	0.0	0.0	0.0	0.0	0.0
Sendmmsg	0.0	0.0	0.0	0.0	0.0	0.0
timer_getoverrun	0.0	0.0	0.0	0.0	0.0	0.0
restart_syscall	0.0	0.0	0.001	0.0316	0.005	0.0705
Truncate	0.0	0.0	0.0	0.0	0.0	0.0
Lremovexattr	0.0	0.0	0.0	0.0	0.0	0.0
Openat	47.721	25.2029	59.808	28.6254	57.8275	26.8831
Waitid	0.0	0.0	0.0	0.0	0.0	0.0
Getxattr	0.0	0.0	0.0	0.0	0.0	0.0
sched_getscheduler	0.006	0.1094	0.003	0.0774	0.003	0.0774
Ioctl	144.14	89.137	145.575	101.0203	152.9925	103.7091
clock_nanosleep	0.0	0.0	0.0	0.0	0.0	0.0
Unlinkat	2.044	9.9126	1.9375	7.4045	1.875	7.4984

clock_adjtime	0.0	0.0	0.0	0.0	0.0	0.0
set_tid_address	0.0	0.0	0.0	0.0	0.0	0.0
Setitimer	0.0	0.0	0.0	0.0	0.0	0.0
Gettimeofday	33.914	77.6484	19.896	58.4256	17.184	57.4809
Futex	92.637	61.6389	67.264	52.7577	87.2165	61.9276
mmap2	99.496	60.6225	140.0055	65.1437	138.424	68.6594
sendfile64	0.0	0.0	0.0	0.0	0.0	0.0
Mlockall	0.0	0.0	0.0	0.0	0.0	0.0
Ppoll	0.042	0.2006	0.2565	0.4435	0.2315	0.4242
Mlock	0.044	0.9023	0.022	0.6384	0.022	0.6384

Appendix 2 – System calls' Welch's test

Next table shows the relation between Welch's test or z test score on both analyzed cases (Legitimate vs. Old Malware and Legitimate vs. New Malware) and degree of rejection of null hypothesis (green coloured), if applied, or not (red coloured). If statistic could not be calculated (not enough data to statistic test), *null* value is indicated (orange coloured).

System Call	Legitimate vs. Old Malware	Reject Null Hypothesis	Legitimate vs. New Malware	Reject Null Hypothesis
sched_get_priority_min	0.577672893		0.57767289	
Lseek	1.203028112		-0.31776043	
Pipe	0.577672893		0.57767289	
epoll_ctl	-3.35920016	Yes with α=0.001 and p = 0.000782	-4.31122938	Yes with α=0.0001 and p < 0.00001
rt_sigtimedwait	*null*		*null*	
setfsuid	*null*		*null*	
tee	*null*		*null*	
uname	0.621237041		0.67479824	
kill	-0.70586555		*null*	
swapoff	*null*		*null*	
readahead	*null*		*null*	
clock_getres	0.915299082		0.91529908	
preadv	*null*		*null*	
setresgid32	*null*		*null*	
gettid	0.449029547		0.44902955	
sethostname	*null*		*null*	
timer_delete	*null*		*null*	
umask	0.701027886		0.70102789	
sched_getaffinity	*null*		*null*	
writev	-0.32277216		-4.9768951	Yes with α=0.0001 and p < 0.00001
sched_setparam	*null*		*null*	
fchmod	1.480269752		-3.87641892	Yes with α=0.0001
_llseek	2.505674149	Yes with α=0.05 and p = 0.012224	2.16612685	Yes with α=0.05 and p = 0.030304
getpid	0.510948802		-1.43755456	

setreuid32	*null*		*null*	
signalfd4	*null*		*null*	
epoll_wait	0.408204751		0.40820475	
capset	*null*		*null*	
personality	*null*		*null*	
delete_module	*null*		*null*	
dup3	*null*		*null*	
sched_setaffinity	*null*		*null*	
read	3.133353112	Yes with α=0.01 and p = 0.001729	-0.69600165	
getppid	8.246335691	Yes with α=0.0001 and p < 0.00001	-2.4367732	Yes with α=0.05 and p = 0.014851
getgid32	*null*		*null*	
capget	*null*		*null*	
getgroups32	*null*		*null*	
readlinkat	-15.4862132	Yes with α=0.0001 and p < 0.00001	-14.9888673	Yes with α=0.0001 and p < 0.00001
sched_rr_get_interval	*null*		*null*	
setfsgid	*null*		*null*	
renameat	-0.16929849		0.81967172	
fsync	0.23222089		0.93081511	
geteuid32	0.335062023		3.23188737	Yes with α=0.01 and p = 0.00123
unshare	*null*		*null*	
epoll_pwait	-3.1187604	Yes with α=0.01 and p = 0.001821	-8.3557529	Yes with α=0.0001 and p < 0.00001
recvfrom	-11.795177	Yes with α=0.0001 and p < 0.00001	-6.42128104	Yes with α=0.0001 and p < 0.00001
sched_get_priority_max	0.577672893		0.57767289	
symlinkat	*null*		*null*	
settimeofday	*null*		*null*	
timer_create	*null*		*null*	
sendto	-12.4193868	Yes with α=0.0001 and p < 0.00001	-7.42958313	Yes with α=0.0001 and p < 0.00001
mkdirat	4.178955277	Yes with α=0.0001 and p = 0.000029	3.26234992	Yes with α=0.01 and p = 0.001105
lgetxattr	*null*		*null*	
linkat	*null*		*null*	
shutdown	*null*		-1.00072078	
epoll_create1	8.465563367	Yes with α=0.0001 and p < 0.00001	-1.4865904	
getresuid32	*null*		*null*	
adjtimex	*null*		*null*	

sync	*null*		*null*	
syslog	*null*		*null*	
fchownat	*null*		*null*	
setgid32	*null*		*null*	
setregid32	*null*		*null*	
fadvise64_64	*null*		*null*	
getsockname	0.08526729		0.08655462	
close	-8.60636194	Yes with α=0.0001 and p < 0.00001	-7.7571543	Yes with α=0.0001 and p < 0.00001
flock	1.136097036		0.07087279	
lsetxattr	*null*		*null*	
pwritev	*null*		*null*	
llistxattr	*null*		*null*	
tgkill	*null*		-1.00072078	
getegid32	*null*		*null*	
pselect6	*null*		*null*	
rt_sigprocmask	-12.8190496	Yes with α=0.0001 and p < 0.00001	-4.29145044	Yes with α=0.0001 and p = 0.000018
accept4	*null*		*null*	
reboot	*null*		*null*	
fremovexattr	*null*		*null*	
setgroups32	*null*		*null*	
setsid	*null*		*null*	
exit	*null*		*null*	
timerfd_create	*null*		*null*	
sigaltstack	-11.0438705	Yes with α=0.0001 and p < 0.00001	-10.5129786	Yes with α=0.0001 and p < 0.00001
inotify_rm_watch	*null*		*null*	
munmap	-15.302532	Yes with α=0.0001 and p < 0.00001	-14.2723198	Yes with α=0.0001 and p < 0.00001
socketpair	0.447653407		-1.51231425	
setrlimit	0.408204751		0	
getitimer	*null*		*null*	
fchmodat	0.4818302		1.68605217	
setpgid	*null*		*null*	
getrandom	1.159819529		1.15981953	
getcwd	1.123702878		1.12370288	
fstatfs64	0.548294544		0.54829454	
fstatat64	-0.09878932		-1.09770256	
rt_sigsuspend	*null*		*null*	
inotify_add_watch	2.556863137	Yes with α=0.05 and p = 0.010564	1.60298392	
getresgid32	*null*		*null*	

timer_settime	*null*		*null*	
fchdir	*null*		*null*	
timer_gettime	*null*		*null*	
getpeername	*null*		*null*	
sigaction	-13.3054876	Yes with α=0.0001 and p < 0.00001	-13.3417634	Yes with α=0.0001 and p < 0.00001
getsid	*null*		*null*	
mknodat	1.105948429		1.10594843	
setsockopt	0.053981126		0.43529028	
munlockall	*null*		*null*	
msync	0.839685706		0.83968571	
process_vm_readv	*null*		*null*	
pipe2	1.306134824		-0.21347451	
set_thread_area	*null*		-0.70586555	
times	*null*		*null*	
prlimit64	*null*		*null*	
pwrite64	0.837944479		3.53052865	Yes with α=0.001 and p = 0.000415
vfork	0.800947527		-0.72898373	
nanosleep	0.590623386		0.59062339	
sendmsg	-2.04545774	Yes with α=0.05 and p = 0.040855	-2.00461341	Yes with α=0.05 and p = 0.04507
clone	8.476541826	Yes with α=0.0001 and p < 0.00001	7.40999343	Yes with α=0.0001 and p < 0.00001
flistxattr	*null*		*null*	
getcpu	*null*		*null*	
splice	*null*		*null*	
rt_sigaction	0.408413126		0.40841313	
clock_gettime	15.30862261	Yes with α=0.0001 and p < 0.00001	16.7131788	Yes with α=0.0001 and p < 0.00001
ugetrlimit	-0.94731311		-3.50407153	Yes with α=0.001 and p = 0.000458
umount2	*null*		*null*	
rt_sigreturn	*null*		-0.80007025	
rt_sigpending	*null*		*null*	
setxattr	*null*		*null*	
getpgid	*null*		*null*	
brk	*null*		*null*	
fsetxattr	*null*		*null*	
acct	*null*		*null*	
clock_settime	*null*		*null*	
getsockopt	0.887912555		-0.03340316	
exit_group	*null*		*null*	

getpriority	3.561077717	Yes with α=0.001 and p = 0.000369	3.56107772	Yes with α=0.001 and p = 0.000369
listxattr	*null*		*null*	
sysinfo	*null*		*null*	
removexattr	*null*		*null*	
faccessat	4.497484228	Yes with α=0.0001 and p < 0.00001	4.57589706	Yes with α=0.0001 and p < 0.00001
dup	-2.44277525	Yes with α=0.05 and p = 0.014606	-1.27373559	
fdatasync	-0.03951474		3.03816325	
setns	*null*		*null*	
mprotect	-7.95176191	Yes with α=0.0001 and p < 0.00001	-9.88352343	Yes with α=0.0001 and p < 0.00001
listen	0.815349465		0.81534946	
fchown32	*null*		*null*	
getuid32	-1.551624		-7.16064468	Yes with α=0.0001 and p < 0.00001
init_module	*null*		*null*	
sched_yield	1.709418529		0.68772835	
statfs64	0.223603033		2.8689857	Yes with α=0.01 and p = 0.004119
ftruncate64	1.949679661		1.8000973	
fgetxattr	*null*		*null*	
ptrace	*null*		*null*	
vmsplice	*null*		*null*	
fcntl64	0.450122912		3.2482551	Yes with α=0.01 and p = 0.001161
swapon	*null*		*null*	
fallocate	*null*		*null*	
execve	*null*		*null*	
socket	-4.13155473	Yes with α=0.0001 and p = 0.000036	-4.16970603	Yes with α=0.0001 and p = 0.000031
wait4	0.408204751		-6.29829625	Yes with α=0.0001 and p < 0.00001
chdir	0.408418476		0.40841848	
madvise	-13.8525535	Yes with α=0.0001 and p < 0.00001	-13.6934859	Yes with α=0.0001 and p < 0.00001
fstat64	-4.66601939	Yes with α=0.0001 and p < 0.00001	-2.33768036	Yes with α=0.05 and p = 0.019439
mount	*null*		*null*	
timerfd_gettime	*null*		*null*	
rt_sigqueueinfo	*null*		*null*	
getrusage	*null*		*null*	
timerfd_settime	*null*		*null*	

sched_setscheduler	1.091820623		1.09182062	
utimensat	*null*		-1.16488617	
bind	0.832539202		0.8325392	
eventfd2	8.308830557	Yes with α=0.0001 and p < 0.00001	-1.83696242	
connect	-15.9365216	Yes with α=0.0001 and p < 0.00001	-14.8910533	Yes with α=0.0001 and p < 0.00001
getdents64	7.254383525	Yes with α=0.0001 and p < 0.00001	-1.36811067	
readv	*null*		*null*	
mremap	0.99309778		0.99309778	
pread64	-1.49868521		-0.44960895	
setuid32	*null*		*null*	
prctl	-14.7808404	Yes with α=0.0001 and p < 0.00001	-14.3809775	Yes with α=0.0001 and p < 0.00001
stat64	0.574272569		0.57427257	
recvmmsg	*null*		*null*	
chroot	0.408419002		0.408419	
sched_getparam	0.577672893		0.57767289	
truncate64	*null*		*null*	
setresuid32	*null*		*null*	
write	1.756039343		-2.29539196	Yes with α=0.05 and p = 0.021733
munlock	0.707911111		0.70791111	
setpriority	-0.27574468		-2.50718653	Yes with α=0.05 and p = 0.012176
recvmsg	*null*		*null*	
inotify_init1	2.619065538	Yes with α=0.01 and p = 0.008819	1.63796695	
mincore	*null*		*null*	
sendfile	*null*		*null*	
sendmmsg	*null*		*null*	
timer_getoverrun	*null*		*null*	
restart_syscall	-1.00072078		-2.24275011	Yes with α=0.05 and p = 0.024961
truncate	*null*		*null*	
lremovexattr	*null*		*null*	
openat	-10.0218241	Yes with α=0.0001 and p < 0.00001	-8.67299164	Yes with α=0.0001 and p < 0.00001
waitid	*null*		*null*	
getxattr	*null*		*null*	
sched_getscheduler	0.707911111		0.70791111	
ioctl	-0.33682749		-2.04707487	Yes with α=0.05 and

				p = 0.040658
clock_nanosleep	*null*		*null*	
unlinkat	0.272195852		0.42997444	
clock_adjtime	*null*		*null*	
set_tid_address	*null*		*null*	
setitimer	*null*		*null*	
gettimeofday	4.561787918	Yes with α=0.0001 and p < 0.00001	5.47617955	Yes with α=0.0001 and p < 0.00001
futex	9.889374752	Yes with α=0.0001 and p < 0.00001	1.96178954	Yes with α=0.05 and p = 0.049797
mmap2	-14.3955245	Yes with α=0.0001 and p < 0.00001	-13.4400783	Yes with α=0.0001 and p < 0.00001
sendfile64	*null*		*null*	
mlockall	*null*		*null*	
ppoll	-13.9352536	Yes with α=0.0001 and p < 0.00001	-12.77069	Yes with α=0.0001 and p < 0.00001
mlock	0.629419829		0.62941983	

Appendix 3 – Permissions' statistics

Next table shows the count (frequency) of each permission attribute for each dataset composed by 1000 application samples:

	Legitimate Dataset		Old Malware Dataset		New Malware Dataset	
Permission	Absent	Present	Absent	Present	Absent	Present
ADD_VOICEMAIL	1000	0	1000	0	1000	0
USE_SIP	996	4	1000	0	971	29
ACCESS_NOTIFICATION_POLICY	1000	0	1000	0	1000	0
CAMERA	676	324	891	109	848	152
REQUEST_DELETE_PACKAGES	1000	0	1000	0	1000	0
BIND_CONDITION_PROVIDER_SERVICE	1000	0	1000	0	1000	0
BIND_QUICK_SETTINGS_TILE	965	35	1000	0	1000	0
MASTER_CLEAR	998	2	999	1	1000	0
BIND_DEVICE_ADMIN	978	22	1000	0	861	139
GET_ACCOUNTS_PRIVILEGED	995	5	1000	0	1000	0
READ_SYNC_SETTINGS	882	118	995	5	967	33
FACTORY_TEST	1000	0	997	3	1000	0
SET_ALWAYS_FINISH	1000	0	999	1	999	1
READ_CALENDAR	928	72	998	2	959	41
BIND_CARRIER_SERVICES	997	3	1000	0	1000	0
CHANGE_CONFIGURATION	981	19	983	17	867	133
SET_TIME	999	1	1000	0	997	3
PERSISTENT_ACTIVITY	998	2	996	4	971	29
USE_FINGERPRINT	876	124	1000	0	971	29
GET_PACKAGE_SIZE	955	45	995	5	945	55
ACCESS_LOCATION_EXTRA_COMMANDS	976	24	836	164	930	70
CONTROL_LOCATION_UPDATES	1000	0	997	3	999	1
SEND_RESPOND_VIA_MESSAGE	979	21	1000	0	986	14
CLEAR_APP_CACHE	986	14	996	4	949	51
BIND_INPUT_METHOD	983	17	999	1	998	2
WRITE_GSERVICES	1000	0	999	1	999	1
SIGNAL_PERSISTENT_PROCESS	1000	0	999	1	995	5

ES						
BIND_VOICE_INTERACTION	1000	0	1000	0	1000	0
BIND_REMOTEVIEWS	883	117	1000	0	998	2
BATTERY_STATS	979	21	985	15	947	53
READ_VOICEMAIL	1000	0	1000	0	1000	0
SET_WALLPAPER_HINTS	987	13	993	7	957	43
BIND_NFC_SERVICE	994	6	1000	0	1000	0
REQUEST_IGNORE_BATTERY_OPTIMIZATIONS	974	26	1000	0	996	4
RESTART_PACKAGES	988	12	929	71	911	89
CALL_PRIVILEGED	997	3	999	1	989	11
CAPTURE_SECURE_VIDEO_OUTPUT	999	1	1000	0	998	2
DISABLE_KEYGUARD	954	46	956	44	927	73
DELETE_PACKAGES	994	6	976	24	953	47
CHANGE_COMPONENT_ENABLED_STATE	999	1	990	10	993	7
BIND_APPWIDGET	996	4	998	2	973	27
RECORD_AUDIO	827	173	967	33	872	128
READ_PHONE_NUMBERS	1000	0	1000	0	1000	0
VIBRATE	432	568	319	681	768	232
WRITE_SECURE_SETTINGS	974	26	978	22	982	18
UNINSTALL_SHORTCUT	1000	0	1000	0	1000	0
WRITE_CALL_LOG	970	30	1000	0	958	42
ACCESS_CHECKIN_PROPERTIES	1000	0	998	2	998	2
PACKAGE_USAGE_STATS	951	49	1000	0	948	52
GLOBAL_SEARCH	993	7	999	1	1000	0
CHANGE_WIFI_STATE	795	205	920	80	759	241
BROADCAST_STICKY	943	57	992	8	933	67
KILL_BACKGROUND_PROCESSES	958	42	985	15	898	102
BIND_INCALL_SERVICE	997	3	1000	0	1000	0
SET_TIME_ZONE	1000	0	999	1	969	31
BLUETOOTH_ADMIN	872	128	977	23	942	58
BLUETOOTH_PRIVILEGED	996	4	1000	0	999	1
BIND_TEXT_SERVICE	1000	0	1000	0	1000	0
MANAGE_DOCUMENTS	967	33	1000	0	997	3
BIND_VR_LISTENER_SERVICE	1000	0	1000	0	1000	0
SET_WALLPAPER	951	49	811	189	936	64
WAKE_LOCK	141	859	425	575	603	397
WRITE_CALENDAR	949	51	987	13	964	36
BIND_SCREENING_SERVICE	999	1	1000	0	1000	0
BIND_AUTOFILL_SERVICE	996	4	1000	0	1000	0

REQUEST_INSTALL_PACKAGES	963	37	1000	0	1000	0
SET_PREFERRED_APPLICATIONS	999	1	981	19	999	1
NFC	927	73	1000	0	969	31
CALL_PHONE	895	105	923	77	865	135
BIND_PRINT_SERVICE	1000	0	1000	0	1000	0
INTERNET	16	984	19	981	39	961
BIND_VPN_SERVICE	982	18	1000	0	997	3
READ_SMS	917	83	778	222	800	200
ANSWER_PHONE_CALLS	997	3	1000	0	1000	0
MEDIA_CONTENT_CONTROL	978	22	1000	0	999	1
BROADCAST_PACKAGE_REMOVED	999	1	998	2	998	2
BIND_VISUAL_VOICEMAIL_SERVICE	998	2	1000	0	1000	0
BIND_NOTIFICATION_LISTENER_SERVICE	953	47	1000	0	989	11
REORDER_TASKS	980	20	999	1	963	37
MODIFY_AUDIO_SETTINGS	893	107	981	19	941	59
READ_PHONE_STATE	669	331	88	912	342	658
WRITE_SETTINGS	860	140	889	111	828	172
BIND_CARRIER_MESSAGING_SERVICE	998	2	1000	0	1000	0
BIND_WALLPAPER	991	9	1000	0	997	3
DUMP	990	10	998	2	998	2
UPDATE_DEVICE_STATS	1000	0	987	13	993	7
SEND_SMS	935	65	637	363	772	228
ACCESS_COARSE_LOCATION	643	357	520	480	709	291
READ_EXTERNAL_STORAGE	464	536	980	20	581	419
SYSTEM_ALERT_WINDOW	828	172	941	59	582	418
CHANGE_WIFI_MULTICAST_STATE	938	62	998	2	959	41
BIND_MIDI_DEVICE_SERVICE	999	1	1000	0	1000	0
EXPAND_STATUS_BAR	975	25	996	4	960	40
WRITE_APN_SETTINGS	995	5	953	47	955	45
BIND_TV_INPUT	1000	0	1000	0	1000	0
SET_ALARM	998	2	1000	0	1000	0
WRITE_CONTACTS	931	69	970	30	928	72
PROCESS_OUTGOING_CALLS	966	34	963	37	904	96
RECEIVE_BOOT_COMPLETED	502	498	454	546	658	342
MODIFY_PHONE_STATE	989	11	984	16	964	36
BIND_TELECOM_CONNECTION_SERVICE	998	2	1000	0	1000	0

RECEIVE_MMS	968	32	981	19	955	45
GET_TASKS	882	118	853	147	617	383
READ_INPUT_STATE	1000	0	999	1	1000	0
READ_CALL_LOG	945	55	1000	0	938	62
READ_SYNC_STATS	948	52	999	1	969	31
CAPTURE_AUDIO_OUTPUT	998	2	1000	0	997	3
REQUEST_COMPANION_RUN_IN_BACKGROUND	1000	0	1000	0	1000	0
RECEIVE_WAP_PUSH	996	4	982	18	956	44
MOUNT_UNMOUNT_FILESYSTEMS	976	24	960	40	745	255
REQUEST_COMPANION_USE_DATA_IN_BACKGROUND	1000	0	1000	0	1000	0
ACCESS_WIFI_STATE	414	586	470	530	505	495
INSTANT_APP_FOREGROUND_SERVICE	1000	0	1000	0	1000	0
ACCESS_FINE_LOCATION	604	396	359	641	717	283
BIND_DREAM_SERVICE	992	8	1000	0	1000	0
ACCESS_NETWORK_STATE	37	963	227	773	386	614
BROADCAST_WAP_PUSH	977	23	999	1	975	25
BODY_SENSORS	996	4	1000	0	972	28
DIAGNOSTIC	999	1	999	1	984	16
STATUS_BAR	992	8	999	1	998	2
READ_LOGS	946	54	678	322	893	107
BLUETOOTH	794	206	972	28	915	85
READ_FRAME_BUFFER	998	2	999	1	997	3
INSTALL_SHORTCUT	993	7	1000	0	999	1
SET_PROCESS_LIMIT	998	2	999	1	1000	0
WRITE_VOICEMAIL	1000	0	1000	0	1000	0
CAPTURE_VIDEO_OUTPUT	995	5	1000	0	998	2
TRANSMIT_IR	998	2	1000	0	972	28
CHANGE_NETWORK_STATE	897	103	933	67	794	206
WRITE_SYNC_SETTINGS	877	123	986	14	965	35
ACCOUNT_MANAGER	998	2	999	1	972	28
LOCATION_HARDWARE	997	3	1000	0	1000	0
BIND_ACCESSIBILITY_SERVICE	957	43	1000	0	972	28
GET_ACCOUNTS	561	439	753	247	806	194
RECEIVE_SMS	907	93	837	163	758	242
MOUNT_FORMAT_FILESYSTEMS	999	1	998	2	1000	0
DELETE_CACHE_FILES	999	1	941	59	998	2
WRITE_EXTERNAL_STORAGE	219	781	275	725	268	732
BIND_CHOOSER_TARGET_SERV	973	27	1000	0	1000	0

ICE						
MANAGE_OWN_CALLS	1000	0	1000	0	1000	0
REBOOT	996	4	996	4	997	3
INSTALL_PACKAGES	563	437	912	88	935	65
SET_DEBUG_APP	999	1	999	1	995	5
INSTALL_LOCATION_PROVIDER	1000	0	999	1	1000	0
SET_ANIMATION_SCALE	1000	0	999	1	998	2
READ_CONTACTS	734	266	357	643	836	164
BROADCAST_SMS	966	34	999	1	960	40

Appendix 4 – Permissions' Chi Square Test

Next table shows the relation between, X^2 test score on both analyzed cases (Legitimate vs. Old Malware and Legitimate vs. New Malware) and degree of rejection of null hypothesis (green coloured), if applied, or not (red coloured). If statistic could not be calculated (not enough data to statistic test), *null* value is indicated (orange coloured).

Permission	Legitimate vs. Old Malware	Reject Null Hypothesis	Legitimate vs. New Malware	Reject Null Hypothesis
ADD_VOICEMAIL	*null*	*null*	*null*	*null*
USE_SIP	4.008016032	0.045284408	19.2571367	1.1424E-05
ACCESS_NOTIFICATION_POLICY	*null*	*null*	*null*	*null*
CAMERA	136.2542391	1.7555E-31	81.563334	1.6973E-19
REQUEST_DELETE_PACKAGES	*null*	*null*	*null*	*null*
BIND_CONDITION_PROVIDER_ SERVICE	*null*	*null*	*null*	*null*
BIND_QUICK_SETTINGS_TILE	35.62340967	2.39393E-09	35.6234097	2.3939E-09
MASTER_CLEAR	0.333834084	0.563410114	2.002002	0.1570916
BIND_DEVICE_ADMIN	22.24469161	2.40021E-06	92.4685641	6.8403E-22
GET_ACCOUNTS_PRIVILEGED	5.012531328	0.025164486	5.01253133	0.02516449
READ_SYNC_SETTINGS	110.6158851	7.18217E-26	51.7551997	6.287E-13
FACTORY_TEST	3.00450676	0.083033246	*null*	*null*
SET_ALWAYS_FINISH	1.00050025	0.317189492	1.00050025	0.31718949
READ_CALENDAR	68.76034913	1.11186E-16	9.01369876	0.00267964
BIND_CARRIER_SERVICES	3.00450676	0.083033246	3.00450676	0.08303325
CHANGE_CONFIGURATION	0.113147771	0.736588514	92.5324675	6.6229E-22
SET_TIME	1.00050025	0.317189492	1.00200401	0.31682608
PERSISTENT_ACTIVITY	0.668672685	0.413514753	23.8863677	1.0219E-06
USE_FINGERPRINT	132.196162	1.35542E-30	63.8732302	1.3269E-15
GET_PACKAGE_SIZE	32.82051282	1.01073E-08	1.05263158	0.30490179
ACCESS_LOCATION_EXTRA_ COMMANDS	115.0720962	7.58877E-27	23.6208167	1.1731E-06
CONTROL_LOCATION_UPDATES	3.00450676	0.083033246	1.00050025	0.31718949

SEND_RESPOND_VIA_MESSAGE	21.22283982	4.08863E-06	1.42493639	0.2325926
CLEAR_APP_CACHE	5.606009642	0.017898978	21.769032	3.0752E-06
BIND_INPUT_METHOD	14.35138468	0.000151668	11.9556843	0.00054481
WRITE_GSERVICES	1.00050025	0.317189492	1.00050025	0.31718949
SIGNAL_PERSISTENT_PROCESSES	1.00050025	0.317189492	5.01253133	0.02516449
BIND_VOICE_INTERACTION	*null*	*null*	*null*	*null*
BIND_REMOTEVIEWS	124.2697823	7.35343E-29	118.165288	1.5952E-27
BATTERY_STATS	1.018329939	0.31291548	14.3695097	0.00015022
READ_VOICEMAIL	*null*	*null*	*null*	*null*
SET_WALLPAPER_HINTS	1.818181818	0.177529852	16.5343915	4.7776E-05
BIND_NFC_SERVICE	6.018054162	0.014160251	6.01805416	0.01416025
REQUEST_IGNORE_BATTERY_ OPTIMIZATION	26.34245187	2.85934E-07	16.3790186	5.1856E-05
RESTART_PACKAGES	43.75561715	3.72047E-11	61.8251399	3.7535E-15
CALL_PRIVILEGED	1.002004008	0.316826082	4.60365415	0.03190389
CAPTURE_SECURE_VIDEO_OUTPUT	1.00050025	0.317189492	0.33383408	0.56341011
DISABLE_KEYGUARD	0.046538685	0.829199566	6.51361023	0.0107052
DELETE_PACKAGES	10.96446701	0.000928757	32.5803607	1.1436E-08
CHANGE_COMPONENT_ENABLED_ STATE	7.404360346	0.006506597	4.51807229	0.0335386
BIND_APPWIDGET	0.668672685	0.413514753	17.3331804	3.1366E-05
RECORD_AUDIO	106.0709376	7.11424E-25	7.91945233	0.00489061
READ_PHONE_NUMBERS	*null*	*null*	*null*	*null*
VIBRATE	27.22604182	1.81004E-07	235.2	4.3789E-53
WRITE_SECURE_SETTINGS	0.341530055	0.558947374	1.48726529	0.22264125
UNINSTALL_SHORTCUT	*null*	*null*	*null*	*null*
WRITE_CALL_LOG	30.45685279	3.41375E-08	2.0746888	0.14976048
ACCESS_CHECKIN_PROPERTIES	2.002002002	0.157091601	2.002002	0.1570916
PACKAGE_USAGE_STATS	50.23065095	1.36696E-12	0.09384825	0.75934075
GLOBAL_SEARCH	4.518072289	0.0335386	7.02458605	0.00803981
CHANGE_WIFI_STATE	63.93534858	1.2857E-15	3.73980643	0.0531306
BROADCAST_STICKY	38.17928841	6.45338E-10	0.85975652	0.35380684
KILL_BACKGROUND_PROCESSES	13.16466668	0.000285278	26.9396552	2.0991E-07
BIND_INCALL_SERVICE	3.00450676	0.083033246	3.00450676	0.08303325
SET_TIME_ZONE	1.00050025	0.317189492	31.488065	2.0067E-08
BLUETOOTH_ADMIN	78.97592756	6.28714E-19	29.0452988	7.0705E-08
BLUETOOTH_PRIVILEGED	4.008016032	0.045284408	1.80451128	0.17916806
BIND_TEXT_SERVICE	*null*	*null*	*null*	*null*
MANAGE_DOCUMENTS	33.55363498	6.93257E-09	25.4582485	4.5206E-07
BIND_VR_LISTENER_SERVICE	*null*	*null*	*null*	*null*
SET_WALLPAPER	93.47666422	4.11018E-22	2.11038733	0.14630228
WAKE_LOCK	198.747234	3.91938E-45	456.826587	2.358E-101
WRITE_CALENDAR	23.30836777	1.37998E-06	2.70382321	0.10010793

BIND_SCREENING_SERVICE	1.00050025	0.317189492	1.00050025	0.31718949
BIND_AUTOFILL_SERVICE	4.008016032	0.045284408	4.00801603	0.04528441
REQUEST_INSTALL_PACKAGES	37.69740194	8.26145E-10	37.6974019	8.2615E-10
SET_PREFERRED_APPLICATIONS	16.36363636	5.22787E-05	0	1
NFC	75.76543851	3.19444E-18	17.8919182	2.3381E-05
CALL_PHONE	4.738935432	0.029487313	4.26136364	0.03898861
BIND_PRINT_SERVICE	*null*	*null*	*null*	*null*
INTERNET	0.26172301	0.608938888	9.89016125	0.00166165
BIND_VPN_SERVICE	18.16347124	2.0273E-05	10.8279795	0.00099978
READ_SMS	74.74636104	5.35243E-18	56.3436514	6.0849E-14
ANSWER_PHONE_CALLS	3.00450676	0.083033246	3.00450676	0.08303325
MEDIA_CONTENT_CONTROL	22.24469161	2.40021E-06	19.3969783	1.0617E-05
BROADCAST_PACKAGE_ REMOVED	0.333834084	0.563410114	0.33383408	0.56341011
BIND_VISUAL_VOICEMAIL_ SERVICE	2.002002002	0.157091601	2.002002	0.1570916
BIND_NOTIFICATION_LISTENER_ SERVICE	48.13108039	3.98657E-12	23.0121808	1.6098E-06
REORDER_TASKS	17.37289155	3.07176E-05	5.2189145	0.02234249
MODIFY_AUDIO_SETTINGS	65.59265471	5.5445E-16	15.1357885	0.00010005
READ_PHONE_STATE	717.4890085	4.7062E-158	213.88388	1.9523E-48
WRITE_SETTINGS	3.831443807	0.050299587	3.88868635	0.04861243
BIND_CARRIER_MESSAGING_ SERVICE	2.002002002	0.157091601	2.002002	0.1570916
BIND_WALLPAPER	9.040683074	0.00264037	3.01810865	0.08233944
DUMP	5.365526492	0.020538587	5.36552649	0.02053859
UPDATE_DEVICE_STATS	13.08505284	0.000297661	7.02458605	0.00803981
SEND_SMS	263.9770755	2.33149E-59	106.243914	6.5196E-25
ACCESS_COARSE_LOCATION	31.08386727	2.47117E-08	9.94411571	0.00161364
READ_EXTERNAL_STORAGE	663.2655095	2.9118E-146	27.4335529	1.6259E-07
SYSTEM_ALERT_WINDOW	62.49525865	2.67087E-15	145.48864	1.6793E-33
CHANGE_WIFI_MULTICAST_STATE	58.10950413	2.47926E-14	4.51402572	0.03361803
BIND_MIDI_DEVICE_SERVICE	1.00050025	0.317189492	1.00050025	0.31718949
EXPAND_STATUS_BAR	15.43064084	8.55892E-05	3.57781753	0.05855603
WRITE_APN_SETTINGS	34.82862107	3.60041E-09	32.8205128	1.0107E-08
BIND_TV_INPUT	*null*	*null*	*null*	*null*
SET_ALARM	2.002002002	0.157091601	2.002002	0.1570916
WRITE_CONTACTS	16.16374157	5.80956E-05	0.0686711	0.79328157
PROCESS_OUTGOING_CALLS	0.131426193	0.716957872	31.6248457	1.8702E-08
RECEIVE_BOOT_COMPLETED	4.616938393	0.031657696	49.9507389	1.5765E-12
MODIFY_PHONE_STATE	0.938596985	0.332638923	13.6178928	0.00022404
BIND_TELECOM_CONNECTION_ SERVICE	2.002002002	0.157091601	2.002002	0.1570916

RECEIVE_MMS	3.400436624	0.065179164	2.28268871	0.1308249
GET_TASKS	3.65831113	0.055790158	187.017559	1.4235E-42
READ_INPUT_STATE	1.00050025	0.317189492	*null*	*null*
READ_CALL_LOG	56.55526992	5.46408E-14	0.44482572	0.50480243
READ_SYNC_STATS	50.41137308	1.2467E-12	5.54329996	0.01855157
CAPTURE_AUDIO_OUTPUT	2.002002002	0.157091601	0.20050125	0.65431655
REQUEST_COMPANION_RUN_IN _BACKGROUND	*null*	*null*	*null*	*null*
RECEIVE_WAP_PUSH	9.008180899	0.002687738	34.1530055	5.0944E-09
MOUNT_UNMOUNT_ FILESYSTEMS	4.132231405	0.042073838	222.263875	2.9013E-50
REQUEST_COMPANION_USE_ DATA_IN_BACKGROUND	*null*	*null*	*null*	*null*
ACCESS_WIFI_STATE	6.357547155	0.011688302	16.6713809	4.4446E-05
INSTANT_APP_FOREGROUND_ SERVICE	*null*	*null*	*null*	*null*
ACCESS_FINE_LOCATION	120.2145738	5.67751E-28	28.4717585	9.5075E-08
BIND_DREAM_SERVICE	8.032128514	0.00459548	8.03212851	0.00459548
ACCESS_NETWORK_STATE	157.5373551	3.906E-36	365.181517	2.0958E-81
BROADCAST_WAP_PUSH	20.41160594	6.24499E-06	0.08538251	0.77013152
BODY_SENSORS	4.008016032	0.045284408	18.2926829	1.8943E-05
DIAGNOSTIC	0	1	13.3487586	0.00025859
STATUS_BAR	5.469055193	0.019356087	3.61809045	0.05715444
READ_LOGS	235.2478776	4.27487E-53	18.9746655	1.3247E-05
BLUETOOTH	153.3428193	3.22367E-35	58.879713	1.6761E-14
READ_FRAME_BUFFER	0.333834084	0.563410114	0.20050125	0.65431655
INSTALL_SHORTCUT	7.024586051	0.008039805	4.51807229	0.0335386
SET_PROCESS_LIMIT	0.333834084	0.563410114	2.002002	0.1570916
WRITE_VOICEMAIL	*null*	*null*	*null*	*null*
CAPTURE_VIDEO_OUTPUT	5.012531328	0.025164486	1.29023009	0.25600555
TRANSMIT_IR	2.002002002	0.157091601	22.8764805	1.7275E-06
CHANGE_NETWORK_STATE	8.331726133	0.003895862	40.6071358	1.8613E-10
WRITE_SYNC_SETTINGS	93.0999761	4.97183E-22	53.2167842	2.987E-13
ACCOUNT_MANAGER	0.333834084	0.563410114	22.8764805	1.7275E-06
LOCATION_HARDWARE	3.00450676	0.083033246	3.00450676	0.08303325
BIND_ACCESSIBILITY_SERVICE	43.94481349	3.37766E-11	3.28565483	0.06988774
GET_ACCOUNTS	81.79240385	1.51158E-19	138.736246	5.0301E-32
RECEIVE_SMS	21.95025803	2.79809E-06	79.6055757	4.5713E-19
MOUNT_FORMAT_FILESYSTEMS	0.333834084	0.563410114	1.00050025	0.31718949
DELETE_CACHE_FILES	57.80068729	2.90068E-14	0.33383408	0.56341011
WRITE_EXTERNAL_STORAGE	8.430515455	0.003689757	6.51709822	0.01068422
BIND_CHOOSER_TARGET_ SERVICE	27.36948809	1.68062E-07	27.3694881	1.6806E-07

MANAGE_OWN_CALLS	*null*	*null*	*null*	*null*
REBOOT	0	1	0.1433589	0.70496438
INSTALL_PACKAGES	314.5788539	2.19672E-70	368.044511	4.9885E-82
SET_DEBUG_APP	0	1	2.67469074	0.10195513
INSTALL_LOCATION_ PROVIDER	1.00050025	0.317189492	*null*	*null*
SET_ANIMATION_SCALE	1.00050025	0.317189492	2.002002	0.1570916
READ_CONTACTS	286.6315962	2.69473E-64	30.8221004	2.828E-08
BROADCAST_SMS	31.66848419	1.82866E-08	0.50517808	0.47723371

Appendix 5 – System calls' Fisher Score values

Next table shows Fisher Score (F) values for each of the system calls gathered (212 different system calls). As all F values were relatively lower, F values over 0.15 were selected for the model as most discriminative features (highlighted in green).

Feature	Legitimate vs. Old Malware Dataset	Legitimate vs. New Malware Dataset
System Call	Fisher Score	Fisher Score
sched_get_priority_min	0.001002	0.001002
Lseek	0.00432	0.000246
Pipe	0.001002	0.001002
epoll_ctl	0.022145	0.034015
rt_sigtimedwait	0.0	0.0
Setfsuid	0.0	0.0
Tee	0.0	0.0
Uname	0.001146	0.001367
Kill	0.000501	0.0
Swapoff	0.0	0.0
Readahead	0.0	0.0
clock_getres	0.002513	0.002513
Preadv	0.0	0.0
setresgid32	0.0	0.0
Gettid	0.000605	0.000605
Sethostname	0.0	0.0
timer_delete	0.0	0.0
Umask	0.001475	0.001475
sched_getaffinity	0.0	0.0
Writev	0.000297	0.030974
sched_setparam	0.0	0.0
Fchmod	0.006586	0.018156
_llseek	0.018914	0.013497
Getpid	0.000619	0.002673
setreuid32	0.0	0.0
signalfd4	0.0	0.0
epoll_wait	0.000501	0.000501
capset	0.0	0.0

personality	0.0	0.0
delete_module	0.0	0.0
dup3	0.0	0.0
sched_setaffinity	0.0	0.0
Read	0.029663	0.000865
Getppid	0.215594	0.011583
getgid32	0.0	0.0
Capget	0.0	0.0
getgroups32	0.0	0.0
Readlinkat	0.688957	0.586918
sched_rr_get_interval	0.0	0.0
Setfsgid	0.0	0.0
Renameat	7.3e-05	0.001886
Fsync	0.000138	0.002351
geteuid32	0.00024	0.022809
Unshare	0.0	0.0
epoll_pwait	0.017214	0.090745
Recvfrom	0.181365	0.062184
sched_get_priority_max	0.001002	0.001002
Symlinkat	0.0	0.0
Settimeofday	0.0	0.0
timer_create	0.0	0.0
Sendto	0.19443	0.0743
Mkdirat	0.050058	0.026801
Lgetxattr	0.0	0.0
Linkat	0.0	0.0
Shutdown	0.0	0.001002
epoll_create1	0.22842	0.004416
getresuid32	0.0	0.0
Adjtimex	0.0	0.0
Sync	0.0	0.0
Syslog	0.0	0.0
Fchownat	0.0	0.0
setgid32	0.0	0.0
setregid32	0.0	0.0
fadvise64_64	0.0	0.0
Getsockname	1.7e-05	1.9e-05
Close	0.173597	0.133066
Flock	0.003877	1.4e-05
Lsetxattr	0.0	0.0
Pwritev	0.0	0.0
llistxattr	0.0	0.0
tgkill	0.0	0.001002

getegid32	0.0	0.0
pselect6	0.0	0.0
rt_sigprocmask	0.244162	0.028797
accept4	0.0	0.0
reboot	0.0	0.0
fremovexattr	0.0	0.0
setgroups32	0.0	0.0
setsid	0.0	0.0
exit	0.0	0.0
timerfd_create	0.0	0.0
sigaltstack	0.228204	0.205009
inotify_rm_watch	0.0	0.0
munmap	0.749835	0.569562
socketpair	0.000501	0.00316
setrlimit	0.000501	0.0
getitimer	0.0	0.0
fchmodat	0.000577	0.007452
setpgid	0.0	0.0
getrandom	0.004041	0.004041
getcwd	0.003793	0.003793
fstatfs64	0.000902	0.000902
fstatat64	2.5e-05	0.002523
rt_sigsuspend	0.0	0.0
inotify_add_watch	0.01944	0.006394
getresgid32	0.0	0.0
timer_settime	0.0	0.0
fchdir	0.0	0.0
timer_gettime	0.0	0.0
getpeername	0.0	0.0
sigaction	0.290309	0.302836
getsid	0.0	0.0
mknodat	0.003674	0.003674
setsockopt	8e-06	0.000546
munlockall	0.0	0.0
msync	0.002117	0.002117
process_vm_readv	0.0	0.0
pipe2	0.004685	6.9e-05
set_thread_area	0.0	0.000501
times	0.0	0.0
prlimit64	0.0	0.0
pwrite64	0.00156	0.028428
vfork	0.001743	0.000785
nanosleep	0.001047	0.001047

sendmsg	0.004202	0.004032
clone	0.2067	0.144596
flistxattr	0.0	0.0
getcpu	0.0	0.0
splice	0.0	0.0
rt_sigaction	0.000501	0.000501
clock_gettime	0.839286	1.106283
ugetrlimit	0.001913	0.020537
umount2	0.0	0.0
rt_sigreturn	0.0	0.000641
rt_sigpending	0.0	0.0
setxattr	0.0	0.0
getpgid	0.0	0.0
brk	0.0	0.0
fsetxattr	0.0	0.0
acct	0.0	0.0
clock_settime	0.0	0.0
getsockopt	0.002224	3e-06
exit_group	0.0	0.0
getpriority	0.038071	0.038071
listxattr	0.0	0.0
sysinfo	0.0	0.0
removexattr	0.0	0.0
faccessat	0.052051	0.045545
dup	0.010601	0.003208
fdatasync	3e-06	0.020859
setns	0.0	0.0
mprotect	0.122002	0.194523
listen	0.001894	0.001894
fchown32	0.0	0.0
getuid32	0.005198	0.072634
init_module	0.0	0.0
sched_yield	0.007319	0.000724
statfs64	0.000117	0.021252
ftruncate64	0.011447	0.009486
fgetxattr	0.0	0.0
ptrace	0.0	0.0
vmsplice	0.0	0.0
fcntl64	0.000439	0.023472
swapon	0.0	0.0
fallocate	0.0	0.0
execve	0.0	0.0
socket	0.049884	0.045093

wait4	0.000501	0.04142
chdir	0.000501	0.000501
madvise	0.541898	0.475477
fstat64	0.046783	0.011793
mount	0.0	0.0
timerfd_gettime	0.0	0.0
rt_sigqueueinfo	0.0	0.0
getrusage	0.0	0.0
timerfd_settime	0.0	0.0
sched_setscheduler	0.003581	0.003581
utimensat	0.0	0.001359
bind	0.002055	0.002055
eventfd2	0.219313	0.006673
connect	0.673586	0.518975
getdents64	0.154697	0.00188
readv	0.0	0.0
mremap	0.002962	0.002962
pread64	0.00516	0.000385
setuid32	0.0	0.0
prctl	0.614407	0.532513
stat64	0.00099	0.00099
recvmmsg	0.0	0.0
chroot	0.000501	0.000501
sched_getparam	0.001002	0.001002
truncate64	0.0	0.0
setresuid32	0.0	0.0
write	0.007722	0.009752
munlock	0.001505	0.001505
setpriority	0.00017	0.011328
recvmsg	0.0	0.0
inotify_init1	0.020383	0.006626
mincore	0.0	0.0
sendfile	0.0	0.0
sendmmsg	0.0	0.0
timer_getoverrun	0.0	0.0
restart_syscall	0.001002	0.005051
truncate	0.0	0.0
lremovexattr	0.0	0.0
openat	0.216978	0.164596
waitid	0.0	0.0
getxattr	0.0	0.0
sched_getscheduler	0.001505	0.001505
ioctl	0.000202	0.00734

clock_nanosleep	0.0	0.0
unlinkat	0.000207	0.000508
clock_adjtime	0.0	0.0
set_tid_address	0.0	0.0
setitimer	0.0	0.0
gettimeofday	0.061082	0.092552
futex	0.300894	0.007721
mmap2	0.63051	0.473747
sendfile64	0.0	0.0
mlockall	0.0	0.0
ppoll	0.305315	0.249381
mlock	0.001189	0.001189

Appendix 6 – Permissions' Gini Index values

Next table shows Gini Index (G) values for each of the permissions gathered (147 different permissions). As all G were relatively high, G values under 0.47 were selected for the model as most discriminative features (highlighted in green).

Feature	Legitimate vs. Old Malware Dataset	Legitimate vs. New Malware Dataset
Permission	Gini Index	Gini Index
ADD_VOICEMAIL	0.5	0.5
USE_SIP	0.498998	0.495186
ACCESS_NOTIFICATION_POLICY	0.5	0.5
CAMERA	0.465936	0.479609
REQUEST_DELETE_PACKAGES	0.5	0.5
BIND_CONDITION_PROVIDER_SERVICE	0.5	0.5
BIND_QUICK_SETTINGS_TILE	0.491094	0.491094
MASTER_CLEAR	0.499917	0.499499
BIND_DEVICE_ADMIN	0.494439	0.476883
GET_ACCOUNTS_PRIVILEGED	0.498747	0.498747
READ_SYNC_SETTINGS	0.472346	0.487061
FACTORY_TEST	0.499249	0.5
SET_ALWAYS_FINISH	0.49975	0.49975
READ_CALENDAR	0.48281	0.497747
BIND_CARRIER_SERVICES	0.499249	0.499249
CHANGE_CONFIGURATION	0.499972	0.476867
SET_TIME	0.49975	0.499749
PERSISTENT_ACTIVITY	0.499833	0.494028
USE_FINGERPRINT	0.466951	0.484032
GET_PACKAGE_SIZE	0.491795	0.499737
ACCESS_LOCATION_EXTRA_COMMANDS	0.471232	0.494095
CONTROL_LOCATION_UPDATES	0.499249	0.49975
SEND_RESPOND_VIA_MESSAGE	0.494694	0.499644
CLEAR_APP_CACHE	0.498598	0.494558
BIND_INPUT_METHOD	0.496412	0.497011
WRITE_GSERVICES	0.49975	0.49975
SIGNAL_PERSISTENT_PROCESSES	0.49975	0.498747
BIND_VOICE_INTERACTION	0.5	0.5

BIND_REMOTEVIEWS	0.468933	0.470459
BATTERY_STATS	0.499745	0.496408
READ_VOICEMAIL	0.5	0.5
SET_WALLPAPER_HINTS	0.499545	0.495866
BIND_NFC_SERVICE	0.498495	0.498495
REQUEST_IGNORE_BATTERY_OPTIMIZATIONS	0.493414	0.495905
RESTART_PACKAGES	0.489061	0.484544
CALL_PRIVILEGED	0.499749	0.498849
CAPTURE_SECURE_VIDEO_OUTPUT	0.49975	0.499917
DISABLE_KEYGUARD	0.499988	0.498372
DELETE_PACKAGES	0.497259	0.491855
CHANGE_COMPONENT_ENABLED_STATE	0.498149	0.49887
BIND_APPWIDGET	0.499833	0.495667
RECORD_AUDIO	0.473482	0.49802
READ_PHONE_NUMBERS	0.5	0.5
VIBRATE	0.493193	0.4412
WRITE_SECURE_SETTINGS	0.499915	0.499628
UNINSTALL_SHORTCUT	0.5	0.5
WRITE_CALL_LOG	0.492386	0.499481
ACCESS_CHECKIN_PROPERTIES	0.499499	0.499499
PACKAGE_USAGE_STATS	0.487442	0.499977
GLOBAL_SEARCH	0.49887	0.498244
CHANGE_WIFI_STATE	0.484016	0.499065
BROADCAST_STICKY	0.490455	0.499785
KILL_BACKGROUND_PROCESSES	0.496709	0.493265
BIND_INCALL_SERVICE	0.499249	0.499249
SET_TIME_ZONE	0.49975	0.492128
BLUETOOTH_ADMIN	0.480256	0.492739
BLUETOOTH_PRIVILEGED	0.498998	0.499549
BIND_TEXT_SERVICE	0.5	0.5
MANAGE_DOCUMENTS	0.491612	0.493635
BIND_VR_LISTENER_SERVICE	0.5	0.5
SET_WALLPAPER	0.476631	0.499472
WAKE_LOCK	0.450313	0.385793
WRITE_CALENDAR	0.494173	0.499324
BIND_SCREENING_SERVICE	0.49975	0.49975
BIND_AUTOFILL_SERVICE	0.498998	0.498998
REQUEST_INSTALL_PACKAGES	0.490576	0.490576
SET_PREFERRED_APPLICATIONS	0.495909	0.5
NFC	0.481059	0.495527
CALL_PHONE	0.498815	0.498935
BIND_PRINT_SERVICE	0.5	0.5
INTERNET	0.499935	0.497527

BIND_VPN_SERVICE	0.495459	0.497293
READ_SMS	0.481313	0.485914
ANSWER_PHONE_CALLS	0.499249	0.499249
MEDIA_CONTENT_CONTROL	0.494439	0.495151
BROADCAST_PACKAGE_REMOVED	0.499917	0.499917
BIND_VISUAL_VOICEMAIL_SERVICE	0.499499	0.499499
BIND_NOTIFICATION_LISTENER_SERVICE	0.487967	0.494247
REORDER_TASKS	0.495657	0.498695
MODIFY_AUDIO_SETTINGS	0.483602	0.496216
READ_PHONE_STATE	0.320628	0.446529
WRITE_SETTINGS	0.499042	0.499028
BIND_CARRIER_MESSAGING_SERVICE	0.499499	0.499499
BIND_WALLPAPER	0.49774	0.499245
DUMP	0.498659	0.498659
UPDATE_DEVICE_STATS	0.496729	0.498244
SEND_SMS	0.434006	0.473439
ACCESS_COARSE_LOCATION	0.492229	0.497514
READ_EXTERNAL_STORAGE	0.334184	0.493142
SYSTEM_ALERT_WINDOW	0.484376	0.463628
CHANGE_WIFI_MULTICAST_STATE	0.485473	0.498871
BIND_MIDI_DEVICE_SERVICE	0.49975	0.49975
EXPAND_STATUS_BAR	0.496142	0.499106
WRITE_APN_SETTINGS	0.491293	0.491795
BIND_TV_INPUT	0.5	0.5
SET_ALARM	0.499499	0.499499
WRITE_CONTACTS	0.495959	0.499983
PROCESS_OUTGOING_CALLS	0.499967	0.492094
RECEIVE_BOOT_COMPLETED	0.498846	0.487512
MODIFY_PHONE_STATE	0.499765	0.496596
BIND_TELECOM_CONNECTION_SERVICE	0.499499	0.499499
RECEIVE_MMS	0.49915	0.499429
GET_TASKS	0.499085	0.453246
READ_INPUT_STATE	0.49975	0.5
READ_CALL_LOG	0.485861	0.499889
READ_SYNC_STATS	0.487397	0.498614
CAPTURE_AUDIO_OUTPUT	0.499499	0.49995
REQUEST_COMPANION_RUN_IN_BACKGROUND	0.5	0.5
RECEIVE_WAP_PUSH	0.497748	0.491462
MOUNT_UNMOUNT_FILESYSTEMS	0.498967	0.444434
REQUEST_COMPANION_USE_DATA_IN_BACKGROUND	0.5	0.5
ACCESS_WIFI_STATE	0.498411	0.495832
INSTANT_APP_FOREGROUND_SERVICE	0.5	0.5

ACCESS_FINE_LOCATION	0.469946	0.492882
BIND_DREAM_SERVICE	0.497992	0.497992
ACCESS_NETWORK_STATE	0.460616	0.408705
BROADCAST_WAP_PUSH	0.494897	0.499979
BODY_SENSORS	0.498998	0.495427
DIAGNOSTIC	0.5	0.496663
STATUS_BAR	0.498633	0.499095
READ_LOGS	0.441188	0.495256
BLUETOOTH	0.461664	0.48528
READ_FRAME_BUFFER	0.499917	0.49995
INSTALL_SHORTCUT	0.498244	0.49887
SET_PROCESS_LIMIT	0.499917	0.499499
WRITE_VOICEMAIL	0.5	0.5
CAPTURE_VIDEO_OUTPUT	0.498747	0.499677
TRANSMIT_IR	0.499499	0.494281
CHANGE_NETWORK_STATE	0.497917	0.489848
WRITE_SYNC_SETTINGS	0.476725	0.486696
ACCOUNT_MANAGER	0.499917	0.494281
LOCATION_HARDWARE	0.499249	0.499249
BIND_ACCESSIBILITY_SERVICE	0.489014	0.499179
GET_ACCOUNTS	0.479552	0.465316
RECEIVE_SMS	0.494512	0.480099
MOUNT_FORMAT_FILESYSTEMS	0.499917	0.49975
DELETE_CACHE_FILES	0.48555	0.499917
WRITE_EXTERNAL_STORAGE	0.497892	0.498371
BIND_CHOOSER_TARGET_SERVICE	0.493158	0.493158
MANAGE_OWN_CALLS	0.5	0.5
REBOOT	0.5	0.499964
INSTALL_PACKAGES	0.421355	0.407989
SET_DEBUG_APP	0.5	0.499331
INSTALL_LOCATION_PROVIDER	0.49975	0.5
SET_ANIMATION_SCALE	0.49975	0.499499
READ_CONTACTS	0.428342	0.492294
BROADCAST_SMS	0.492083	0.499874

Appendix 7 – System call's model validation

<table>
<tr><th># features</th><th>Legitimate vs. Old Malware Dataset</th><th>Legitimate vs. New Malware Dataset</th></tr>
<tr><td>Best feature*</td><td>5-fold accuracy: 0.8695±0.01
70-30 confusion matrix
<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>265</td><td>41</td><td>306</td></tr><tr><td>0</td><td>29</td><td>265</td><td>294</td></tr><tr><td></td><td>294</td><td>306</td><td>600</td></tr></table></td><td>5-fold accuracy: 0.8910±0.02
70-30 confusion matrix
<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>300</td><td>39</td><td>339</td></tr><tr><td>0</td><td>27</td><td>234</td><td>261</td></tr><tr><td></td><td>327</td><td>273</td><td>600</td></tr></table></td></tr>
<tr><td>2 best features of L/O dataset**</td><td>5-fold accuracy: 0.8985±0.03
70-30 confusion matrix
<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>255</td><td>31</td><td>286</td></tr><tr><td>0</td><td>36</td><td>278</td><td>314</td></tr><tr><td></td><td>291</td><td>309</td><td>600</td></tr></table></td><td>5-fold accuracy: 0.8785±0.03
70-30 confusion matrix
<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>254</td><td>34</td><td>292</td></tr><tr><td>0</td><td>38</td><td>274</td><td>308</td></tr><tr><td></td><td>288</td><td>312</td><td>600</td></tr></table></td></tr>
<tr><td>2 best features of L/N dataset**</td><td>5-fold accuracy: 0.8985±0.02
70-30 confusion matrix
<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>283</td><td>35</td><td>318</td></tr><tr><td>0</td><td>27</td><td>255</td><td>282</td></tr><tr><td></td><td>310</td><td>290</td><td>600</td></tr></table></td><td>5-fold accuracy: 0.8920±0.02
70-30 confusion matrix
<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>283</td><td>39</td><td>322</td></tr><tr><td>0</td><td>23</td><td>255</td><td>278</td></tr><tr><td></td><td>306</td><td>294</td><td>600</td></tr></table></td></tr>
<tr><td>3 best common features***</td><td>5-fold accuracy: 0.9005±0.01
70-30 confusion matrix
<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>261</td><td>29</td><td>290</td></tr><tr><td>0</td><td>31</td><td>279</td><td>310</td></tr><tr><td></td><td>292</td><td>308</td><td>600</td></tr></table></td><td>5-fold accuracy: 0.8770±0.02
70-30 confusion matrix
<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>263</td><td>32</td><td>295</td></tr><tr><td>0</td><td>39</td><td>266</td><td>305</td></tr><tr><td></td><td>302</td><td>298</td><td>600</td></tr></table></td></tr>
<tr><td>6 best common features****</td><td>5-fold accuracy: 0.9120±0.02
70-30 confusion matrix</td><td>5-fold accuracy: 0.8850±0.03
70-30 confusion matrix</td></tr>
</table>

	Pred\True	1	0	
	1	293	23	316
	0	25	259	284
		318	282	600
	Pred\True	1	0	
	1	259	35	294
	0	37	269	306
		296	304	600
11 common features	5-fold accuracy: 0.9305±0.03 70-30 confusion matrix			
	Pred\True	1	0	
	1	299	15	314
	0	24	262	286
		323	277	600
	5-fold accuracy: 0.8905±0.03 70-30 confusion matrix			
	Pred\True	1	0	
	1	282	32	314
	0	32	254	286
		314	286	600
12 features of L/N dataset	5-fold accuracy: 0.9315±0.01 70-30 confusion matrix			
	Pred\True	1	0	
	1	287	22	309
	0	19	272	291
		306	294	600
	5-fold accuracy: 0.8950±0.03 70-30 confusion matrix			
	Pred\True	1	0	
	1	283	29	312
	0	33	255	288
		316	284	600
21 features of L/O dataset	5-fold accuracy: 0.9670±0.01 70-30 confusion matrix			
	Pred\True	1	0	
	1	298	11	309
	0	10	281	291
		308	292	600
	5-fold accuracy: 0.9065±0.02 70-30 confusion matrix			
	Pred\True	1	0	
	1	261	31	292
	0	24	284	308
		285	315	600
22 features*****	5-fold accuracy: 0.9660±0.01 70-30 confusion matrix			
	Pred\True	1	0	
	1	303	11	314
	0	10	276	286
		313	287	600
	5-fold accuracy: 0.9075±0.04 70-30 confusion matrix			
	Pred\True	1	0	
	1	272	24	296
	0	36	268	304
		308	292	600
All features******	5-fold accuracy: 0.9700±0.01 70-30 confusion matrix			
	Pred\True	1	0	
	1	295	7	302
	0	8	290	298
	5-fold accuracy: 0.9270±0.01 70-30 confusion matrix			
	Pred\True	1	0	
	1	279	21	300
	0	19	281	300

		303	297	600			298	302	600

*Best feature: *clock_gettime*

** 2 best features:

- Legitimate vs. Old Malware: *clock_gettime* and *munmap*.
- Legitimate vs. New Malware: *clock_gettime* and *readlinkat*.

***3 best common features: *clock_gettime, readlinkat* and *munmap*.

****6 best common features: *clock_gettime, readlinkat, munmap, connect, prctl* and *mmap2*.

*****21 features of L/O and *mprotect*.

******All system calls gathered, without performing feature selection.

Appendix 8 – Permissions' model validation

<table>
<tr><th># features</th><th>Legitimate vs. Old Malware Dataset</th><th>Legitimate vs. New Malware Dataset</th></tr>
<tr><td>Best feature of L/O dataset*</td><td>5-fold accuracy: 0.7905±0.03
70-30 confusion matrix
<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>271</td><td>98</td><td>269</td></tr><tr><td>0</td><td>30</td><td>201</td><td>231</td></tr><tr><td></td><td>301</td><td>299</td><td>600</td></tr></table></td><td>5-fold accuracy: 0.6635±0.04
70-30 confusion matrix
<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>196</td><td>100</td><td>296</td></tr><tr><td>0</td><td>98</td><td>206</td><td>304</td></tr><tr><td></td><td>294</td><td>306</td><td>600</td></tr></table></td></tr>
<tr><td>Best feature of L/N dataset**</td><td>5-fold accuracy: 0.6420±0.02
70-30 confusion matrix
<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>125</td><td>46</td><td>171</td></tr><tr><td>0</td><td>169</td><td>260</td><td>429</td></tr><tr><td></td><td>294</td><td>306</td><td>600</td></tr></table></td><td>5-fold accuracy: 0.7310±0.05
70-30 confusion matrix
<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>186</td><td>44</td><td>230</td></tr><tr><td>0</td><td>117</td><td>253</td><td>370</td></tr><tr><td></td><td>303</td><td>297</td><td>600</td></tr></table></td></tr>
<tr><td>2 best features of L/O dataset***</td><td>5-fold accuracy: 0.8880±0.01
70-30 confusion matrix
<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>272</td><td>35</td><td>307</td></tr><tr><td>0</td><td>36</td><td>257</td><td>293</td></tr><tr><td></td><td>308</td><td>292</td><td>600</td></tr></table></td><td>5-fold accuracy: 0.6635±0.03
70-30 confusion matrix
<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>192</td><td>103</td><td>295</td></tr><tr><td>0</td><td>99</td><td>206</td><td>305</td></tr><tr><td></td><td>291</td><td>309</td><td>600</td></tr></table></td></tr>
<tr><td>2 best features of L/N dataset****</td><td>5-fold accuracy: 0.6995±0.04
70-30 confusion matrix
<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>278</td><td>172</td><td>450</td></tr><tr><td>0</td><td>8</td><td>142</td><td>150</td></tr><tr><td></td><td>286</td><td>314</td><td>600</td></tr></table></td><td>5-fold accuracy: 0.7310±0.05
70-30 confusion matrix
<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>183</td><td>42</td><td>225</td></tr><tr><td>0</td><td>119</td><td>256</td><td>375</td></tr><tr><td></td><td>302</td><td>298</td><td>600</td></tr></table></td></tr>
<tr><td>4 common features</td><td>5-fold accuracy: 0.8580±0.03
70-30 confusion matrix</td><td>5-fold accuracy: 0.8460±0.02
70-30 confusion matrix</td></tr>
</table>

	Pred\True | 1 | 0 | 1 | 262 | 67 | 329 0 | 23 | 248 | 271 | 285 | 315 | 600	Pred\True | 1 | 0 | 1 | 281 | 63 | 344 0 | 29 | 227 | 256 | 310 | 290 | 600
9 features of L/N dataset	5-fold accuracy: 0.8955±0.02 70-30 confusion matrix Pred\True | 1 | 0 | 1 | 285 | 42 | 327 0 | 21 | 252 | 273 | 306 | 294 | 600	5-fold accuracy: 0.8940±0.02 70-30 confusion matrix Pred\True | 1 | 0 | 1 | 267 | 25 | 292 0 | 38 | 270 | 308 | 305 | 295 | 600
13 features of L/O dataset	5-fold accuracy: 0.9350±0.02 70-30 confusion matrix Pred\True | 1 | 0 | 1 | 294 | 16 | 310 0 | 20 | 270 | 290 | 314 | 286 | 600	5-fold accuracy: 0.9065±0.03 70-30 confusion matrix Pred\True | 1 | 0 | 1 | 296 | 35 | 331 0 | 22 | 247 | 269 | 318 | 282 | 600
18 features*****	5-fold accuracy: 0.9410±0.02 70-30 confusion matrix Pred\True | 1 | 0 | 1 | 284 | 17 | 301 0 | 19 | 280 | 299 | 303 | 297 | 600	5-fold accuracy: 0.9170±0.01 70-30 confusion matrix Pred\True | 1 | 0 | 1 | 274 | 33 | 307 0 | 19 | 274 | 293 | 293 | 307 | 600
All features******	5-fold accuracy: 0.9505±0.02 70-30 confusion matrix Pred\True | 1 | 0 | 1 | 281 | 15 | 296 0 | 14 | 290 | 304 | 295 | 305 | 600	5-fold accuracy: 0.9210±0.02 70-30 confusion matrix Pred\True | 1 | 0 | 1 | 284 | 28 | 312 0 | 20 | 268 | 288 | 304 | 296 | 600

*READ_PHONE_STATE permission

**WAKE_LOCK permission

***READ_PHONE_STATE and READ_EXTERNAL_STORAGE permissions

****WAKE_LOCK and INSTALL_PACKAGES permissions

*****13 features of L/0 and *VIBRATE, SYSTEM_ALERT_WINDOW, GET_TASKS, MOUNT_UNMOUN_FILESYSTEMS, GET_ACCOUNTS.*

******All permissions, without performing feature selection.

Appendix 9 – Hybrid model validation

<table>
<tr><th># features</th><th>Legitimate vs. Old Malware Dataset</th><th>Legitimate vs. New Malware Dataset</th></tr>
<tr><td>Best system call + best permission L/O*</td><td>5-fold accuracy: 0.8965±0.02
70-30 confusion matrix
<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>278</td><td>50</td><td>338</td></tr><tr><td>0</td><td>10</td><td>262</td><td>272</td></tr><tr><td></td><td>288</td><td>312</td><td>600</td></tr></table></td><td>5-fold accuracy: 0.9070±0.02
70-30 confusion matrix
<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>274</td><td>37</td><td>311</td></tr><tr><td>0</td><td>17</td><td>272</td><td>289</td></tr><tr><td></td><td>291</td><td>309</td><td>600</td></tr></table></td></tr>
<tr><td>Best system call + best permission L/N**</td><td>5-fold accuracy: 0.8800±0.02
70-30 confusion matrix
<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>294</td><td>48</td><td>342</td></tr><tr><td>0</td><td>23</td><td>235</td><td>258</td></tr><tr><td></td><td>317</td><td>283</td><td>600</td></tr></table></td><td>5-fold accuracy: 0.8900±0.03
70-30 confusion matrix
<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>293</td><td>45</td><td>338</td></tr><tr><td>0</td><td>23</td><td>239</td><td>262</td></tr><tr><td></td><td>316</td><td>284</td><td>600</td></tr></table></td></tr>
<tr><td>2 best system calls L/O + 2 best permissions L/O dataset***</td><td>5-fold accuracy: 0.9450±0.01
70-30 confusion matrix
<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>302</td><td>19</td><td>321</td></tr><tr><td>0</td><td>15</td><td>264</td><td>279</td></tr><tr><td></td><td>317</td><td>283</td><td>600</td></tr></table></td><td>5-fold accuracy: 0.8990±0.02
70-30 confusion matrix
<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>272</td><td>35</td><td>307</td></tr><tr><td>0</td><td>25</td><td>268</td><td>293</td></tr><tr><td></td><td>297</td><td>303</td><td>600</td></tr></table></td></tr>
<tr><td>2 best system calls L/N + 2 best permissions L/N dataset****</td><td>5-fold accuracy: 0.9035±0.03
70-30 confusion matrix
<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>286</td><td>36</td><td>322</td></tr><tr><td>0</td><td>21</td><td>257</td><td>278</td></tr><tr><td></td><td>307</td><td>293</td><td>600</td></tr></table></td><td>5-fold accuracy: 0.8950±0.03
70-30 confusion matrix
<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>269</td><td>40</td><td>309</td></tr><tr><td>0</td><td>23</td><td>268</td><td>291</td></tr><tr><td></td><td>292</td><td>308</td><td>600</td></tr></table></td></tr>
<tr><td>All common system calls and permissions</td><td>5-fold accuracy: 0.9505±0.02
70-30 confusion matrix</td><td>5-fold accuracy: 0.9210±0.02
70-30 confusion matrix</td></tr>
</table>

<table>
<tr><td></td><td><table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>284</td><td>16</td><td>300</td></tr><tr><td>0</td><td>14</td><td>286</td><td>300</td></tr><tr><td></td><td>298</td><td>302</td><td>600</td></tr></table></td><td><table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>278</td><td>31</td><td>309</td></tr><tr><td>0</td><td>17</td><td>274</td><td>291</td></tr><tr><td></td><td>295</td><td>305</td><td>600</td></tr></table></td></tr>
<tr><td>22 system calls + 18 permissions</td><td>5-fold accuracy: 0.9740±0.01
70-30 confusion matrix<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>308</td><td>7</td><td>315</td></tr><tr><td>0</td><td>5</td><td>280</td><td>285</td></tr><tr><td></td><td>313</td><td>287</td><td>600</td></tr></table></td><td>5-fold accuracy: 0.9390±0.02
70-30 confusion matrix<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>277</td><td>22</td><td>299</td></tr><tr><td>0</td><td>14</td><td>287</td><td>301</td></tr><tr><td></td><td>291</td><td>309</td><td>600</td></tr></table></td></tr>
<tr><td>All features (212+147)</td><td>5-fold accuracy: 0.9765±0.02
70-30 confusion matrix<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>311</td><td>9</td><td>320</td></tr><tr><td>0</td><td>7</td><td>273</td><td>280</td></tr><tr><td></td><td>318</td><td>282</td><td>600</td></tr></table></td><td>5-fold accuracy: 0.9400±0.01
70-30 confusion matrix<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>273</td><td>21</td><td>294</td></tr><tr><td>0</td><td>18</td><td>288</td><td>306</td></tr><tr><td></td><td>291</td><td>309</td><td>600</td></tr></table></td></tr>
</table>

*clock_gettime system call and READ_PHONE_STATE permission

**clock_gettime system call and WAKE_LOCK permission

****clock_gettime* and *munmap* system calls and READ_PHONE_STATE and READ_EXTERNAL_STORAGE permissions.

*****clock_gettime* and *readlinkat* system calls and WAKE_LOCK and INSTALL_PACKAGES permissions.

Appendix 10 – Malware discrimination model validation

<table>
<tr><th># features</th><th>New vs. Old Malware Dataset</th></tr>
<tr><td>Best system call*</td><td>5-fold accuracy: 0.6460±0.05

70-30 confusion matrix
<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>91</td><td>15</td><td>106</td></tr><tr><td>0</td><td>197</td><td>297</td><td>494</td></tr><tr><td></td><td>288</td><td>312</td><td>600</td></tr></table></td></tr>
<tr><td>Best permission of L/O dataset**</td><td>5-fold accuracy: 0.6270±0.03

70-30 confusion matrix
<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>274</td><td>197</td><td>471</td></tr><tr><td>0</td><td>28</td><td>101</td><td>129</td></tr><tr><td></td><td>302</td><td>298</td><td>600</td></tr></table></td></tr>
<tr><td>Best permission of L/N dataset***</td><td>5-fold accuracy: 0.5890±0.04

70-30 confusion matrix
<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>184</td><td>113</td><td>297</td></tr><tr><td>0</td><td>127</td><td>176</td><td>303</td></tr><tr><td></td><td>311</td><td>289</td><td>600</td></tr></table></td></tr>
<tr><td>2 best system calls L/O + 2 best permissions L/O dataset****</td><td>5-fold accuracy: 0.7825±0.05

70-30 confusion matrix
<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>259</td><td>79</td><td>338</td></tr><tr><td>0</td><td>51</td><td>211</td><td>262</td></tr><tr><td></td><td>310</td><td>290</td><td>600</td></tr></table></td></tr>
<tr><td>2 best system calls L/N + 2 best permissions L/N dataset*****</td><td>5- fold accuracy: 0.7720±0.03

70-30 confusion matrix</td></tr>
</table>

<table>
<tr><td></td><td>
<table>
<tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr>
<tr><td>1</td><td>235</td><td>79</td><td>314</td></tr>
<tr><td>0</td><td>55</td><td>231</td><td>286</td></tr>
<tr><td></td><td>290</td><td>310</td><td>600</td></tr>
</table>
</td></tr>
<tr><td>Common system calls (11)</td><td>5-fold accuracy: 0.8175±0.05

70-30 confusion matrix
<table>
<tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr>
<tr><td>1</td><td>245</td><td>57</td><td>302</td></tr>
<tr><td>0</td><td>53</td><td>245</td><td>298</td></tr>
<tr><td></td><td>298</td><td>302</td><td>600</td></tr>
</table>
</td></tr>
<tr><td>Common permissions (4)</td><td>5-fold accuracy: 0.6655±0.04

70-30 confusion matrix
<table>
<tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr>
<tr><td>1</td><td>262</td><td>175</td><td>437</td></tr>
<tr><td>0</td><td>25</td><td>138</td><td>163</td></tr>
<tr><td></td><td>287</td><td>313</td><td>600</td></tr>
</table>
</td></tr>
<tr><td>All common system calls and permissions (11+4)</td><td>5-fold accuracy: 0.8190±0.03

70-30 confusion matrix
<table>
<tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr>
<tr><td>1</td><td>231</td><td>53</td><td>284</td></tr>
<tr><td>0</td><td>55</td><td>261</td><td>316</td></tr>
<tr><td></td><td>286</td><td>314</td><td>600</td></tr>
</table>
</td></tr>
<tr><td>22 selected system calls</td><td>5-fold accuracy: 0.8955±0.02

70-30 confusion matrix
<table>
<tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr>
<tr><td>1</td><td>264</td><td>43</td><td>307</td></tr>
<tr><td>0</td><td>20</td><td>273</td><td>293</td></tr>
<tr><td></td><td>284</td><td>316</td><td>600</td></tr>
</table>
</td></tr>
<tr><td>18 selected permissions</td><td>5-fold accuracy: 0.9310±0.01

70-30 confusion matrix
<table>
<tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr>
<tr><td>1</td><td>272</td><td>21</td><td>293</td></tr>
<tr><td>0</td><td>18</td><td>289</td><td>307</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><td></td><td><table><tr><td></td><td>290</td><td>310</td><td>600</td></tr></table></td></tr>
<tr><td>22 system calls + 18 permissions (40)</td><td>5-fold accuracy: 0.9300±0.03

70-30 confusion matrix<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>274</td><td>12</td><td>286</td></tr><tr><td>0</td><td>29</td><td>285</td><td>314</td></tr><tr><td></td><td>303</td><td>297</td><td>600</td></tr></table></td></tr>
<tr><td>All system calls (212)</td><td>5-fold accuracy: 0.8990±0.02

70-30 confusion matrix<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>273</td><td>36</td><td>309</td></tr><tr><td>0</td><td>30</td><td>261</td><td>291</td></tr><tr><td></td><td>303</td><td>297</td><td>600</td></tr></table></td></tr>
<tr><td>All permissions (147)</td><td>5-fold accuracy: 0.9430±0.03

70-30 confusion matrix<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>288</td><td>18</td><td>306</td></tr><tr><td>0</td><td>16</td><td>278</td><td>294</td></tr><tr><td></td><td>304</td><td>296</td><td>600</td></tr></table></td></tr>
<tr><td>All features (212+147)</td><td>5-fold accuracy: 0.9345±0.02

70-30 confusion matrix<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>277</td><td>16</td><td>293</td></tr><tr><td>0</td><td>23</td><td>284</td><td>307</td></tr><tr><td></td><td>300</td><td>300</td><td>600</td></tr></table></td></tr>
</table>

*clock_gettime

**READ_PHONE_STATE

***WAKE_LOCK

****_clock_gettime_ and _munmap_ system calls and READ_PHONE_STATE and READ_EXTERNAL_STORAGE permissions.

*****_clock_gettime_ and _readlinkat_ system calls and WAKE_LOCK and INSTALL_PACKAGES permissions

Appendix 11 – Cross-dataset malware model validation

Dynamic approach: System calls

<table>
<tr><td rowspan="3">System Calls (using best feature)*

*clock_gettime</td><td>Training \ Testing</td><td>Old Dataset</td><td>New Dataset</td></tr>
<tr><td>Old Dataset</td><td>0.8695±0.01</td><td>Accuracy: 0.9166

70-30 confusion matrix
<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>280</td><td>31</td><td>311</td></tr><tr><td>0</td><td>19</td><td>270</td><td>289</td></tr><tr><td></td><td>299</td><td>301</td><td>600</td></tr></table></td></tr>
<tr><td>New Dataset</td><td>Accuracy: 0.795

70-30 confusion matrix
<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>201</td><td>24</td><td>225</td></tr><tr><td>0</td><td>99</td><td>276</td><td>375</td></tr><tr><td></td><td>300</td><td>300</td><td>600</td></tr></table></td><td>0.8910±0.02</td></tr>
</table>

<table>
<tr><td rowspan="3">System Calls (using 3 best features)*

*clock_gettime, munmap and readlinkat</td><td>Training \ Testing</td><td>Old Dataset</td><td>New Dataset</td></tr>
<tr><td>Old Dataset</td><td>0.9025±0.03</td><td>Accuracy: 0.8833

70-30 confusion matrix
<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>234</td><td>9</td><td>243</td></tr><tr><td>0</td><td>61</td><td>296</td><td>357</td></tr><tr><td></td><td>295</td><td>305</td><td>600</td></tr></table></td></tr>
<tr><td>New Dataset</td><td>Accuracy: 0.815

70-30 confusion matrix
<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>211</td><td>20</td><td>231</td></tr><tr><td>0</td><td>91</td><td>278</td><td>369</td></tr></table></td><td>0.8820±0.02</td></tr>
</table>

				302	298	600

	Training \ Testing	Old Dataset	New Dataset
System Calls (using 11 common features)	Old Dataset	0.9305±0.03	Accuracy: 0.81 70-30 confusion matrix Pred\True 1 0 1 198 12 210 0 102 288 390 300 300 600
	New Dataset	Accuracy: 0.865 70-30 confusion matrix Pred\True 1 0 1 232 10 242 0 71 287 258 303 297 600	0.8905±0.03

	Training \ Testing	Old Dataset	New Dataset
System Calls (using 22 selected features)	Old Dataset	0.9650±0.01	Accuracy: 0.733 70-30 confusion matrix Pred\True 1 0 1 145 4 149 0 156 295 451 301 299 600
	New Dataset	Accuracy: 0.8766 70-30 confusion matrix Pred\True 1 0 1 242 16 258 0 58 284 342 300 300 600	0.9075±0.04

	Training \ Testing	Old Dataset	New Dataset
System Calls (using all features)	Old Dataset	0.9700±0.01	Accuracy: 0.715 70-30 confusion matrix

			Pred\True | 1 | 0 | 1 | 130 | 3 | 133 0 | 168 | 299 | 467 298 | 302 | 600
	New Dataset	Accuracy: 0.845 70-30 confusion matrix Pred\True | 1 | 0 | 1 | 217 | 8 | 225 0 | 85 | 290 | 375 302 | 298 | 600	0.9270±0.01

Static approach: Permissions

	Training \ Testing	Old Dataset	New Dataset
Permissions (using best feature L/O dataset)* *READ_PHONE_STATE	Old Dataset	0.7905±0.02	Accuracy: 0.6633 70-30 confusion matrix Pred\True | 1 | 0 | 1 | 194 | 96 | 290 0 | 106 | 204 | 310 300 | 300 | 600
	New Dataset	Accuracy: 0.785 70-30 confusion matrix Pred\True | 1 | 0 | 1 | 276 | 105 | 381 0 | 24 | 195 | 219 300 | 300 | 600	0.6635±0.04

	Training \ Testing	Old Dataset	New Dataset
Permissions (using best feature L/N dataset)* *WAKE_LOCK	Old Dataset	0.6420±0.02	Accuracy: 0.745 70-30 confusion matrix Pred\True | 1 | 0 | 1 | 184 | 42 | 226 0 | 111 | 263 | 374 295 | 305 | 600

	New Dataset	Accuracy: 0.6783 70-30 confusion matrix Pred\True 1 0 1 137 38 175 0 155 270 425 292 308 600	0.7310±0.05

Permissions (using 4 common features)	Training \ Testing	Old Dataset	New Dataset
	Old Dataset	0.8580±0.03	Accuracy: 0.84 70-30 confusion matrix Pred\True 1 0 1 275 69 344 0 27 229 256 302 298 600
	New Dataset	Accuracy: 0.88 70-30 confusion matrix Pred\True 1 0 1 277 50 327 0 22 251 273 299 301 600	0.8460±0.02

Permissions (using 18 features)	Training \ Testing	Old Dataset	New Dataset
	Old Dataset	0.9410±0.02	Accuracy: 0.71 70-30 confusion matrix Pred\True 1 0 1 142 11 153 0 161 286 447 303 297 600
	New Dataset	Accuracy: 0.8716 70-30 confusion matrix Pred\True 1 0 1 242 14 256 0 63 281 344	0.9170±0.01

				305	295	600

	Training \ Testing	Old Dataset	New Dataset
Permissions (using all features)	Old Dataset	0.9505±0.02	Accuracy: 0.65 70-30 confusion matrix
	New Dataset	Accuracy: 0.7916 70-30 confusion matrix	0.9245±0.02

Pred\True	1	0	
1	90	5	95
0	205	300	505
	295	305	600

Pred\True	1	0	
1	189	13	202
0	112	286	398
	301	299	600

Hybrid approach: mixing system calls and permissions

	Training \ Testing	Old Dataset	New Dataset
Hybrid (using best syscall and best L/O dataset permission)* *clock_gettime and READ_PHONE_STATE	Old Dataset	0.8985±0.02	Accuracy: 0.92 70-30 confusion matrix
	New Dataset	Accuracy: 0.87 70-30 confusion matrix	0.9070±0.02

Pred\True	1	0	
1	286	28	314
0	20	266	286
	306	294	600

Pred\True	1	0	
1	253	32	285
0	46	269	315
	299	301	600

Hybrid	Training \ Testing	Old Dataset	New Dataset

<table>
<tr><td rowspan="2">(using best syscall and best L/N dataset permission)*

*clock_gettime and WAKE_LOCK</td><td>Old Dataset</td><td>0.8800±0.02</td><td>Accuracy: 0.9033

70-30 confusion matrix<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>272</td><td>29</td><td>301</td></tr><tr><td>0</td><td>29</td><td>270</td><td>299</td></tr><tr><td></td><td>301</td><td>299</td><td>600</td></tr></table></td></tr>
<tr><td>New Dataset</td><td>Accuracy: 0.8066

70-30 confusion matrix<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>220</td><td>39</td><td>259</td></tr><tr><td>0</td><td>77</td><td>264</td><td>341</td></tr><tr><td></td><td>297</td><td>303</td><td>600</td></tr></table></td><td>0.8835±0.02</td></tr>
<tr><td rowspan="3">Hybrid
(using 11 common syscalls + 4 common permissions)</td><td>Training \ Testing</td><td>Old Dataset</td><td>New Dataset</td></tr>
<tr><td>Old Dataset</td><td>0.9500±0.02</td><td>Accuracy: 0.866

70-30 confusion matrix<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>228</td><td>8</td><td>236</td></tr><tr><td>0</td><td>72</td><td>292</td><td>364</td></tr><tr><td></td><td>300</td><td>300</td><td>600</td></tr></table></td></tr>
<tr><td>New Dataset</td><td>Accuracy: 0.9016

70-30 confusion matrix<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>258</td><td>12</td><td>270</td></tr><tr><td>0</td><td>47</td><td>283</td><td>330</td></tr><tr><td></td><td>305</td><td>295</td><td>600</td></tr></table></td><td>0.9210±0.02</td></tr>
<tr><td rowspan="2">Hybrid
(using 22 selected syscalls + 18 selected permissions)</td><td>Training \ Testing</td><td>Old Dataset</td><td>New Dataset</td></tr>
<tr><td>Old Dataset</td><td>0.9740±0.01</td><td>Accuracy: 0.7066

70-30 confusion matrix<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>120</td><td>3</td><td>123</td></tr><tr><td>0</td><td>176</td><td>301</td><td>477</td></tr></table></td></tr>
</table>

			296 304 600
	New Dataset	Accuracy: 0.9116 70-30 confusion matrix Pred\True 1 0 1 249 4 253 0 49 298 347 298 302 600	0.9390±0.02

Hybrid (using all Dynamic and static features)	Training \ Testing	Old Dataset	New Dataset
	Old Dataset	0.9765±0.02	Accuracy: 0.6966 70-30 confusion matrix Pred\True 1 0 1 119 2 121 0 180 299 479 299 301 600
	New Dataset	Accuracy: 0.8983 70-30 confusion matrix Pred\True 1 0 1 242 4 246 0 57 297 354 299 301 600	0.9400±0.01

Appendix 12 – Mixed malware detection validation

System call best feature: clock_gettime

<table>
<tr><th>Training \ Testing</th><th>Old Dataset</th><th>Mixed Malware Dataset</th><th>New Dataset</th></tr>
<tr><td>Old Dataset</td><td></td><td>Accuracy: 0.8966

70-30 confusion matrix
<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>274</td><td>36</td><td>310</td></tr><tr><td>0</td><td>26</td><td>264</td><td>290</td></tr><tr><td></td><td>300</td><td>300</td><td>600</td></tr></table></td><td></td></tr>
<tr><td>Mixed malware Dataset</td><td>Accuracy: 0.895

70-30 confusion matrix
<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>273</td><td>33</td><td>306</td></tr><tr><td>0</td><td>30</td><td>264</td><td>294</td></tr><tr><td></td><td>303</td><td>297</td><td>600</td></tr></table></td><td>Accuracy: 0.8675±0.01*

70-30 confusion matrix
<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>263</td><td>44</td><td>307</td></tr><tr><td>0</td><td>35</td><td>258</td><td>293</td></tr><tr><td></td><td>298</td><td>302</td><td>600</td></tr></table></td><td>Accuracy: 0.9133

70-30 confusion matrix
<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>278</td><td>27</td><td>305</td></tr><tr><td>0</td><td>25</td><td>270</td><td>295</td></tr><tr><td></td><td>303</td><td>297</td><td>600</td></tr></table></td></tr>
<tr><td>New Dataset</td><td></td><td>Accuracy: 0.8516

70-30 confusion matrix
<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>250</td><td>39</td><td>289</td></tr><tr><td>0</td><td>50</td><td>261</td><td>311</td></tr><tr><td></td><td>300</td><td>300</td><td>600</td></tr></table></td><td></td></tr>
</table>

*5-fold cross validation againsta same dataset.

System call 11 common features

<table>
<tr><th>Training \ Testing</th><th>Old Dataset</th><th>Mixed Malware Dataset</th><th>New Dataset</th></tr>
<tr><td>Old Dataset</td><td></td><td>Accuracy: 0.9

70-30 confusion matrix
<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>248</td><td>7</td><td>255</td></tr><tr><td>0</td><td>53</td><td>292</td><td>345</td></tr><tr><td></td><td>301</td><td>299</td><td>600</td></tr></table></td><td></td></tr>
<tr><td>Mixed malware Dataset</td><td>Accuracy: 0.9616

70-30 confusion matrix
<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>289</td><td>12</td><td>301</td></tr><tr><td>0</td><td>11</td><td>288</td><td>299</td></tr><tr><td></td><td>300</td><td>300</td><td>600</td></tr></table></td><td>Accuracy: 0.8805±0.03*

70-30 confusion matrix
<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>266</td><td>46</td><td>312</td></tr><tr><td>0</td><td>34</td><td>254</td><td>288</td></tr><tr><td></td><td>300</td><td>300</td><td>600</td></tr></table></td><td>Accuracy: 0.9383

70-30 confusion matrix
<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>265</td><td>7</td><td>272</td></tr><tr><td>0</td><td>30</td><td>298</td><td>328</td></tr><tr><td></td><td>295</td><td>305</td><td>600</td></tr></table></td></tr>
<tr><td>New Dataset</td><td></td><td>Accuracy: 0.9133

70-30 confusion matrix
<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>259</td><td>8</td><td>267</td></tr><tr><td>0</td><td>44</td><td>289</td><td>333</td></tr><tr><td></td><td>303</td><td>297</td><td>600</td></tr></table></td><td></td></tr>
</table>

System call: 22 selected features

<table>
<tr><th>Training \ Testing</th><th>Old Dataset</th><th>Mixed Malware Dataset</th><th>New Dataset</th></tr>
<tr><td>Old Dataset</td><td></td><td>Accuracy: 0.845

70-30 confusion matrix
<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>209</td><td>3</td><td>212</td></tr><tr><td>0</td><td>90</td><td>298</td><td>388</td></tr><tr><td></td><td>299</td><td>301</td><td>600</td></tr></table></td><td></td></tr>
</table>

<table>
<tr>
<td>Mixed malware Dataset</td>
<td>Accuracy: 0.975

70-30 confusion matrix
<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>290</td><td>4</td><td>294</td></tr><tr><td>0</td><td>11</td><td>295</td><td>306</td></tr><tr><td></td><td>301</td><td>299</td><td>600</td></tr></table></td>
<td>Accuracy: 0.9080±0.03*

70-30 confusion matrix
<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>279</td><td>28</td><td>307</td></tr><tr><td>0</td><td>26</td><td>267</td><td>293</td></tr><tr><td></td><td>305</td><td>295</td><td>600</td></tr></table></td>
<td>Accuracy: 0.9366

70-30 confusion matrix
<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>268</td><td>6</td><td>274</td></tr><tr><td>0</td><td>32</td><td>294</td><td>326</td></tr><tr><td></td><td>300</td><td>300</td><td>600</td></tr></table></td>
</tr>
<tr>
<td>New Dataset</td>
<td></td>
<td>Accuracy: 0.94

70-30 confusion matrix
<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>272</td><td>9</td><td>281</td></tr><tr><td>0</td><td>27</td><td>292</td><td>319</td></tr><tr><td></td><td>299</td><td>301</td><td>600</td></tr></table></td>
<td></td>
</tr>
</table>

System call: All features (212)

<table>
<tr>
<th>Training \ Testing</th>
<th>Old Dataset</th>
<th>Mixed Malware Dataset</th>
<th>New Dataset</th>
</tr>
<tr>
<td>Old Dataset</td>
<td></td>
<td>Accuracy: 0.866

70-30 confusion matrix
<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>223</td><td>2</td><td>225</td></tr><tr><td>0</td><td>78</td><td>297</td><td>375</td></tr><tr><td></td><td>301</td><td>299</td><td>600</td></tr></table></td>
<td></td>
</tr>
<tr>
<td>Mixed malware Dataset</td>
<td>Accuracy: 0.9766

70-30 confusion matrix
<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>296</td><td>5</td><td>301</td></tr><tr><td>0</td><td>9</td><td>290</td><td>299</td></tr><tr><td></td><td>305</td><td>295</td><td>600</td></tr></table></td>
<td>Accuracy: 0.9195±0.02*

70-30 confusion matrix
<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>278</td><td>20</td><td>298</td></tr><tr><td>0</td><td>27</td><td>275</td><td>302</td></tr><tr><td></td><td>305</td><td>295</td><td>600</td></tr></table></td>
<td>Accuracy: 0.9583

70-30 confusion matrix
<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>288</td><td>10</td><td>298</td></tr><tr><td>0</td><td>15</td><td>287</td><td>302</td></tr><tr><td></td><td>303</td><td>297</td><td>600</td></tr></table></td>
</tr>
</table>

<table>
<tr><td>New Dataset</td><td></td><td>Accuracy: 0.9266

70-30 confusion matrix<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>265</td><td>6</td><td>271</td></tr><tr><td>0</td><td>38</td><td>291</td><td>329</td></tr><tr><td></td><td>303</td><td>297</td><td>600</td></tr></table></td><td></td></tr>
</table>

Permission feature: READ_PHONE_STATE

<table>
<tr><th>Training \ Testing</th><th>Old Dataset</th><th>Mixed Malware Dataset</th><th>New Dataset</th></tr>
<tr><td>Old Dataset</td><td></td><td>Accuracy: 0.73

70-30 confusion matrix<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>236</td><td>99</td><td>335</td></tr><tr><td>0</td><td>63</td><td>202</td><td>265</td></tr><tr><td></td><td>299</td><td>301</td><td>600</td></tr></table></td><td></td></tr>
<tr><td>Mixed malware Dataset</td><td>Accuracy: 0.7966

70-30 confusion matrix<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>281</td><td>98</td><td>379</td></tr><tr><td>0</td><td>24</td><td>197</td><td>221</td></tr><tr><td></td><td>305</td><td>295</td><td>600</td></tr></table></td><td>Accuracy: 0.7265±0.02*

70-30 confusion matrix<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>241</td><td>105</td><td>346</td></tr><tr><td>0</td><td>60</td><td>194</td><td>254</td></tr><tr><td></td><td>301</td><td>299</td><td>600</td></tr></table></td><td>Accuracy: 0.6783

70-30 confusion matrix<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>205</td><td>99</td><td>304</td></tr><tr><td>0</td><td>94</td><td>202</td><td>296</td></tr><tr><td></td><td>299</td><td>301</td><td>600</td></tr></table></td></tr>
<tr><td>New Dataset</td><td></td><td>Accuracy: 0.715

70-30 confusion matrix<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>227</td><td>99</td><td>326</td></tr><tr><td>0</td><td>72</td><td>202</td><td>274</td></tr><tr><td></td><td>299</td><td>301</td><td>600</td></tr></table></td><td></td></tr>
</table>

*5-fold cross validation againsta same dataset.

Permission feature: WAKE_LOCK

<table>
<tr><th>Training \ Testing</th><th>Old Dataset</th><th>Mixed Malware Dataset</th><th>New Dataset</th></tr>
<tr><td>Old Dataset</td><td></td><td>Accuracy: 0.6533

70-30 confusion matrix
<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>146</td><td>52</td><td>198</td></tr><tr><td>0</td><td>156</td><td>246</td><td>402</td></tr><tr><td></td><td>302</td><td>298</td><td>600</td></tr></table></td><td></td></tr>
<tr><td>Mixed malware Dataset</td><td>Accuracy: 0.635

70-30 confusion matrix
<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>131</td><td>50</td><td>181</td></tr><tr><td>0</td><td>169</td><td>250</td><td>419</td></tr><tr><td></td><td>300</td><td>300</td><td>600</td></tr></table></td><td>Accuracy: 0.6780±0.03*

70-30 confusion matrix
<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>150</td><td>45</td><td>195</td></tr><tr><td>0</td><td>151</td><td>254</td><td>405</td></tr><tr><td></td><td>301</td><td>299</td><td>600</td></tr></table></td><td>Accuracy: 0.745

70-30 confusion matrix
<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>179</td><td>31</td><td>210</td></tr><tr><td>0</td><td>122</td><td>268</td><td>390</td></tr><tr><td></td><td>301</td><td>299</td><td>600</td></tr></table></td></tr>
<tr><td>New Dataset</td><td></td><td>Accuracy: 0.655

70-30 confusion matrix
<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>145</td><td>52</td><td>197</td></tr><tr><td>0</td><td>155</td><td>248</td><td>403</td></tr><tr><td></td><td>300</td><td>300</td><td>600</td></tr></table></td><td></td></tr>
</table>

Permission feature: 4 common permissions

<table>
<tr><th>Training \ Testing</th><th>Old Dataset</th><th>Mixed Malware Dataset</th><th>New Dataset</th></tr>
<tr><td>Old Dataset</td><td></td><td>Accuracy: 0.855

70-30 confusion matrix
<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>275</td><td>60</td><td>335</td></tr><tr><td>0</td><td>27</td><td>238</td><td>265</td></tr></table></td><td></td></tr>
</table>

		302 298 600	
Mixed malware Dataset	Accuracy: 0.8616 70-30 confusion matrix Pred\True 1 0 1 284 62 346 0 21 233 254 305 295 600	Accuracy: 0.8525±0.02* 70-30 confusion matrix Pred\True 1 0 1 276 65 341 0 28 231 259 304 296 600	Accuracy: 0.85 70-30 confusion matrix Pred\True 1 0 1 273 61 334 0 29 237 366 302 298 600
New Dataset		Accuracy: 0.846 70-30 confusion matrix Pred\True 1 0 1 278 65 343 0 27 230 257 305 295 600	

Permission feature: 18 selected permissions

Training \ Testing	Old Dataset	Mixed Malware Dataset	New Dataset
Old Dataset		Accuracy: 0.8466 70-30 confusion matrix Pred\True 1 0 1 219 7 226 0 85 289 374 304 296 600	
Mixed malware Dataset	Accuracy: 0.9566 70-30 confusion matrix Pred\True 1 0 1 290 15 305 0 11 284 295 301 299 600	Accuracy: 0.9160±0.04* 70-30 confusion matrix Pred\True 1 0 1 281 30 311 0 23 266 289 304 296 600	Accuracy: 0.9366 70-30 confusion matrix Pred\True 1 0 1 281 17 298 0 21 281 302 302 298 600

<table>
<tr><td>New Dataset</td><td></td><td>Accuracy: 0.9016

70-30 confusion matrix<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>248</td><td>6</td><td>254</td></tr><tr><td>0</td><td>53</td><td>293</td><td>346</td></tr><tr><td></td><td>301</td><td>299</td><td>600</td></tr></table></td><td></td></tr>
</table>

Permission feature: All permissions

<table>
<tr><th>Training \ Testing</th><th>Old Dataset</th><th>Mixed Malware Dataset</th><th>New Dataset</th></tr>
<tr><td>Old Dataset</td><td></td><td>Accuracy: 0.8316

70-30 confusion matrix<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>203</td><td>3</td><td>206</td></tr><tr><td>0</td><td>98</td><td>296</td><td>394</td></tr><tr><td></td><td>301</td><td>299</td><td>600</td></tr></table></td><td></td></tr>
<tr><td>Mixed malware Dataset</td><td>Accuracy: 0.9566

70-30 confusion matrix<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>289</td><td>12</td><td>301</td></tr><tr><td>0</td><td>14</td><td>285</td><td>299</td></tr><tr><td></td><td>303</td><td>297</td><td>600</td></tr></table></td><td>Accuracy: 0.9225±0.03*

70-30 confusion matrix<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>276</td><td>20</td><td>296</td></tr><tr><td>0</td><td>26</td><td>278</td><td>304</td></tr><tr><td></td><td>302</td><td>298</td><td>600</td></tr></table></td><td>Accuracy: 0.945

70-30 confusion matrix<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>289</td><td>17</td><td>306</td></tr><tr><td>0</td><td>16</td><td>278</td><td>294</td></tr><tr><td></td><td>305</td><td>295</td><td>600</td></tr></table></td></tr>
<tr><td>New Dataset</td><td></td><td>Accuracy: 0.87

70-30 confusion matrix<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>231</td><td>6</td><td>237</td></tr><tr><td>0</td><td>72</td><td>291</td><td>363</td></tr><tr><td></td><td>303</td><td>297</td><td>600</td></tr></table></td><td></td></tr>
</table>

Hybrid: clock_gettime and READ_PHONE_STATE

<table>
<tr><th>Training \ Testing</th><th>Old Dataset</th><th>Mixed Malware Dataset</th><th>New Dataset</th></tr>
<tr><td>Old Dataset</td><td></td><td>Accuracy: 0.9116

70-30 confusion matrix
<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>284</td><td>33</td><td>317</td></tr><tr><td>0</td><td>20</td><td>263</td><td>283</td></tr><tr><td></td><td>304</td><td>296</td><td>600</td></tr></table></td><td></td></tr>
<tr><td>Mixed malware Dataset</td><td>Accuracy: 0.9283

70-30 confusion matrix
<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>289</td><td>28</td><td>317</td></tr><tr><td>0</td><td>15</td><td>268</td><td>283</td></tr><tr><td></td><td>304</td><td>296</td><td>600</td></tr></table></td><td>Accuracy: 0.8965±0.01*

70-30 confusion matrix
<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>278</td><td>33</td><td>311</td></tr><tr><td>0</td><td>25</td><td>264</td><td>289</td></tr><tr><td></td><td>303</td><td>297</td><td>600</td></tr></table></td><td>Accuracy: 0.925

70-30 confusion matrix
<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>287</td><td>31</td><td>318</td></tr><tr><td>0</td><td>14</td><td>268</td><td>282</td></tr><tr><td></td><td>301</td><td>299</td><td>600</td></tr></table></td></tr>
<tr><td>New Dataset</td><td></td><td>Accuracy: 0.9133

70-30 confusion matrix
<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>282</td><td>33</td><td>315</td></tr><tr><td>0</td><td>19</td><td>266</td><td>285</td></tr><tr><td></td><td>301</td><td>299</td><td>600</td></tr></table></td><td></td></tr>
</table>

Hybrid: clock_gettime and WAKE_LOCK

<table>
<tr><th>Training \ Testing</th><th>Old Dataset</th><th>Mixed Malware Dataset</th><th>New Dataset</th></tr>
<tr><td>Old Dataset</td><td></td><td>Accuracy: 0.915

70-30 confusion matrix
<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>281</td><td>28</td><td>309</td></tr><tr><td>0</td><td>23</td><td>268</td><td>291</td></tr></table></td><td></td></tr>
</table>

		304 296 600	
Mixed malware Dataset	Accuracy: 0.8966 70-30 confusion matrix Pred\True 1 0 1 261 22 283 0 40 277 317 301 299 600	Accuracy: 0.8730±0.02* 70-30 confusion matrix Pred\True 1 0 1 262 37 299 0 39 262 301 301 299 600	Accuracy: 0.9166 70-30 confusion matrix Pred\True 1 0 1 281 28 309 0 22 269 291 303 297 600
New Dataset		Accuracy: 0.88 70-30 confusion matrix Pred\True 1 0 1 266 34 300 0 38 262 300 304 296 600	

Hybrid: 11 common syscalls + 4 common permissions

Training \ Testing	Old Dataset	Mixed Malware Dataset	New Dataset
Old Dataset		Accuracy: 0.9366 70-30 confusion matrix Pred\True 1 0 1 269 6 275 0 32 293 325 301 299 600	
Mixed malware Dataset	Accuracy: 0.965 70-30 confusion matrix Pred\True 1 0 1 287 8 295 0 13 292 305 300 300 600	Accuracy: 0.9300±0.02* 70-30 confusion matrix Pred\True 1 0 1 287 20 307 0 16 277 293 303 297 600	Accuracy: 0.9683 70-30 confusion matrix Pred\True 1 0 1 288 7 295 0 12 293 305 300 300 600

New Dataset		Accuracy: 0.945 70-30 confusion matrix Pred\True 1 0 1 273 6 279 0 27 294 321 300 300 600	

Hybrid: 22 selected syscalls + 18 selected permissions

Training \ Testing	Old Dataset	Mixed Malware Dataset	New Dataset
Old Dataset		Accuracy: 0.865 70-30 confusion matrix Pred\True 1 0 1 223 4 227 0 77 296 373 300 300 600	
Mixed malware Dataset	Accuracy: 0.99 70-30 confusion matrix Pred\True 1 0 1 299 3 302 0 3 295 298 302 298 600	Accuracy: 0.9415±0.03* 70-30 confusion matrix Pred\True 1 0 1 282 16 298 0 20 282 302 302 298 600	Accuracy: 0.9716 70-30 confusion matrix Pred\True 1 0 1 293 6 299 0 11 290 301 304 296 600
New Dataset		Accuracy: 0.9433 70-30 confusion matrix Pred\True 1 0 1 273 5 278 0 29 293 322 302 298 600	

Hybrid: all features

<table>
<tr><th>Training \ Testing</th><th>Old Dataset</th><th>Mixed Malware Dataset</th><th>New Dataset</th></tr>
<tr><td>Old Dataset</td><td></td><td>Accuracy: 0.8533

70-30 confusion matrix
<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>217</td><td>5</td><td>222</td></tr><tr><td>0</td><td>83</td><td>295</td><td>378</td></tr><tr><td></td><td>300</td><td>300</td><td>600</td></tr></table></td><td></td></tr>
<tr><td>Mixed malware Dataset</td><td>Accuracy: 0.9883

70-30 confusion matrix
<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>298</td><td>3</td><td>301</td></tr><tr><td>0</td><td>4</td><td>295</td><td>299</td></tr><tr><td></td><td>302</td><td>298</td><td>600</td></tr></table></td><td>Accuracy: 0.9375±0.02*

70-30 confusion matrix
<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>289</td><td>25</td><td>314</td></tr><tr><td>0</td><td>17</td><td>269</td><td>286</td></tr><tr><td></td><td>306</td><td>294</td><td>600</td></tr></table></td><td>Accuracy: 0.9616

70-30 confusion matrix
<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>289</td><td>7</td><td>296</td></tr><tr><td>0</td><td>16</td><td>288</td><td>304</td></tr><tr><td></td><td>305</td><td>295</td><td>600</td></tr></table></td></tr>
<tr><td>New Dataset</td><td></td><td>Accuracy: 0.9266

70-30 confusion matrix
<table><tr><td>Pred\True</td><td>1</td><td>0</td><td></td></tr><tr><td>1</td><td>269</td><td>8</td><td>277</td></tr><tr><td>0</td><td>36</td><td>287</td><td>323</td></tr><tr><td></td><td>305</td><td>295</td><td>600</td></tr></table></td><td></td></tr>
</table>

*5-fold cross validation againsta same dataset.