Android Malware Detection and Crypto-mining Recognition Methodology with Machine Learning

Sorin Soviany
R&D Department
BEIA Consult International
Bucharest, Romania
sorin.soviany@beia.ro

Andrei Scheianu
R&D Department
BEIA Consult International
Bucharest, Romania
andrei.scheianu@beia.ro

George Suciu
R&D Department
BEIA Consult Internationa
Bucharest, Romania
george@beia.ro

Alexandru Vulpe
Telecommunication Department
University Politehnica of Bucharest
Bucharest, Romania
alex.vulpe@radio.pub.ro

Octavian Fratu
Telecommunication Department
University Politehnica of Bucharest
Bucharest, Romania
octavian.fratu@upb.ro

Cristiana Istrate
R&D Department
BEIA Consult International
Bucharest, Romania
cristiana.istrate@beia.ro

***Abstract*— The paper proposes a Machine Learning methodology for Android malware detection and recognition, including crypto-mining applications using the blockchain. The design is based on a hierarchical classification method, with several decision stages. A combination of functional and statistical features is proposed to be applied for data classification in order to provide a high-performance malware recognition process. The specific contribution of this design methodology is the hierarchical classifier with detection and discrimination stages, respectively. Further works should be done for various features sets in order to achieve an optimized and high-accuracy modeling process supporting an innovative Machine Learning-based solution for Android malware detection.**

***Keywords*— Android; malware; hierarchical classifier; machine learning; crypto-mining; blockchain; detector**

I. INTRODUCTION

Since the last few years the mobile devices have become an extending support for various applications in areas like financial, medicine, science. This is because of the need for mobility and accessibility of mobile communications networks for many end-users acting in residential or corporate sites and performing various tasks according to their applications. Within this general framework, one can see that Android is currently the most popular operating system for mobile devices [1]. This trend is proven by the evolution of the market shares of Android OS (operating system) during the last years. Its fast adoption leads to an increasing rate of malware occurrences comparing to the previous years. The increasing functional capabilities that Android platform provides to the various applications of the end-users become a source for new vulnerabilities and attack points that can be exploited the malware developers [1]. In many cases, the malware spreading opportunity is supported by the third-party applications stores availability. This is because these third-party developer applications are typically hosted in Google Play [1].

The malwares for mobile devices (with Android OS) include viruses, Trojans, adware, backdoors, worms, botnets, spyware, ransomware and other applications that are designed for malicious purposes using various implementation methods such as code obfuscation, dynamic execution, stealth techniques, encryption and repackaging in order to avoid the actual anti-malware mechanisms for Android [1], [2]. The most applied techniques that are used to attack the Android platforms (devices, OS and installed applications) include sending messages without the target's awareness and deleting them by itself, fraudulent sending of user's private data [1].

The Android malware could also be classified based on their behavior, as following [2]: Information extraction malware, Premium Rate Calls and SMS, Root Exploits, Search Engine Optimization, Dynamically Downloaded Code. According to the Malwarebytes LABS report published in 2017, in 2016 ransomware increased to the top, targeting especially business [3]. It is a cyber criminality industry based on the new paradigm of Ransomware as a Service (RaaS). During the last quarter of 2016, nearly 400 variants of ransomware were identified [3]. As concerning the specific Android malware, the the Malwarebytes LABS report shows that the most important trend in 2016 was the increasing use of randomization as an approach of the malware developers to bypass the detection mechanisms [3].

The purpose of this research activity is to define a design methodology for an anti-malware solution addressing Android platforms, based on advanced machine learning techniques. The application goals are the detection and recognition of various malware having as their targets the mobile devices and apps. The modeling process should allow to accurately recognize and detect the malicious apps before producing serious damages by compromising the end-users sensitive data and their privacy.

The rest of the paper is structured as following. Section II presents related works about the most relevant developments in the area of anti-malware solutions while Section III proposes a methodological framework for design and development of Android malware detection solutions; Section IV draws conclusions and outlines further research lines in order to design, develop and implement optimized solutions taking in account the various real applications constraints.

II. RELATED WORK

The recent trends in the Android malware forced the anti-malware products developers to design, implement and release customized solution to detect, recognize and block the different malware types for Android devices. Most of the existing solutions are grouped into 2 categories: static methods and dynamic methods [1]. Any of these main approaches is used by most of the actual anti-virus applications in order to prevent or minimize the malware attacks against the mobile devices and the installed apps. Usually the security systems are designed to operate according to some thresholds that are defined based on the

978-1-5386-8296-8/18/$31.00 ©2018 IEEE
DOI 10.1109/EUC.2018.00010

IEEE computer society

threat level and the data sensitivity. So, the increasing of the threat or alert level leads to the corresponding increase of the detection rate of the target malware. Given these trends and the limitations of the actual anti-malware solutions, the overall risk situation of the Android devices and their supported apps users is difficult to be assessed [1]. A taxonomy of the anti-malware techniques is given in Fig. 1 [1].

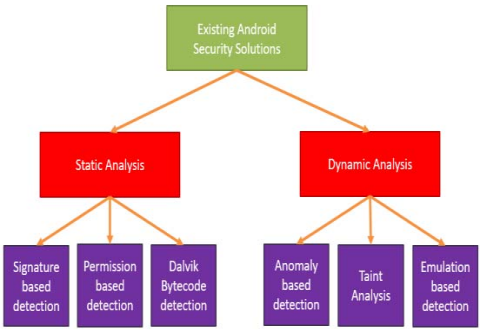

Fig.1. Categories of actual Android Anti-Malware Solutions[1]

The static anti-malware methods check the functionalities and the malicious potential of an application while its operational mode is disabled; the source code of the application is analyzed outside any run-time execution. This approach allows to recognize potential malicious behaviors that do not activate until some particular conditions are met. The static solutions include the following techniques [1], [2], [4], [5]: Signature-based detection, Permission-based detection, Dalvik Bytecode detection. Therefore, the static anti-malware techniques are used to detect malware without executing the suspicious apps. The modalities include extracting permissions requested from the Manifest file, analyzing information passed through Inter-Component Communication and API calls [2]. These methods are not always very efficient in providing the desired protection level. Many malware developers already implemented and applied obfuscation techniques that proved to be very effective against the static anti-malware solutions [2].

In the dynamic approach, the antimalware engines perform the application analysis during the execution time. These solutions could miss some code segments that are not yet executed, however the malicious behaviors that are not recognized by using static methods could be easily identified with the dynamic approach. Anyway, the dynamic methods require more resources in order to perform the malware detection and recognition, while focusing on polymorphic and metamorphic code obfuscation techniques that are used in some new discovered malwares. The dynamic anti-malware solutions include the following techniques [1], [2]: Anomaly-based detection, Taint analysis, Emulation based detection. Therefore, the dynamic anti-malware techniques are used to detect malware with analysis involving running the app in a controlled environment, monitoring the application behavior at runtime [2]. Many actual developments with dynamic solutions for Android malware detection are also based on novel approaches including [2]: Cloud-based anti-malware protection, Reputation-based application recognition.

As concerning the security performance evaluation of the Android malware detection, one can mention the method called PAMD (Permission analysis for Android Malware Detection) [6]. According to their authors, this method allows to evaluate the security level of Android apps based on their permission.

The technical literature and reporting in this area are very comprehensive. A comprehensive overview for the development of mobile malware with a survey of the existing methods for malware mitigation on mobile devices is presented in [7]. Here there are given the mobile malware specifics, followed by the main malware detection methods, with a set of comparison criteria. Among the particular classes of anti-malware solutions for Android, in the cited study there are mentioned [7]:

- Permission analysis, that considers the information in app description and its manifest. This information includes the conditions for the security configurations of apps under which they become potentially dangerous. The correspondence between the application-required permissions and their description should be verified;
- Machine learning, a very promising approach to design, develop and release innovative and efficient anti-malware solutions. The ML-based malware detection methods can be considered as static or dynamic, depending also on the way in which the learning process is applied. The learning stage of the overall malware recognition process performs the detection model design, actually training the classifier with the suitable data comprising the most informative features of Android malware. The model becomes able to separate between malware and not-malware apps (a binary classification in this case). Some of these methods consider the API calls that are used within the Android apps and perform the malware recognition with machine learning algorithms while taking as features some parameters of the API calls. Among the classification algorithms, the best performances seemed to be provided by KNN (K-Nearest Neighbors) and SVM (Support Vector Machine);
- Battery life monitoring: the main idea is that a malicious app running on a mobile device could increase its energy consumption. This factor is significant for the overall functionality of the mobile device, its operating system and cloud-based detection. This detection approach is based on methods for efficient analysis transferring data from the mobile device to an analysis site;
- Cloud-based detection, approach that is also justified by the limited resources of mobile devices (storage, processing). This detection approach is based on methods for efficient analysis transferring data from the mobile device to analysis site.

A survey of evaluation techniques for Android anti-malware using transformation attacks is given in [8]. According to that survey authors, it is mandatory to evaluate the reliability of the current anti-malware solutions for Android with respect to the various malware. A systematic framework to evaluate the robustness of the actual anti-malware products is presented while considering several transformation attack techniques. Such tools are very useful for the decision-making process within the overall secure system development [8], [9]. The systematic framework named DroidChameleon uses some common transformations techniques to automatically transform the Android applications allowing to generate new variants of malware that are used to assess the effectiveness of the most applied anti-malware products [9].

Many security tools for Android platform combine several design, development and implementation techniques.

One of these tools is Andrubis, a completely automated, publicly available analysis system for Android apps. This tool combines static and dynamic analysis techniques, while applying some operations into several stages: Static analysis, Dynamic analysis and Post-processing. The feature set generated by Andrubis is used within a behavioral profiling process while applying an unsupervised learning (with clustering) [10].

ThinAV is a truly lightweight mobile cloud-based anti-malware solution for Android that uses pre-existing web-based file scanning services for malware detection; this security tool combines a lightweight footprint on an Android device with the capability of leveraging several free existing anti-malware services that are already available on Internet [11].

DroidClone is another security tool that allows to detect Android malware variants by exposing code clones. The main idea that support this implementation for malware detection is that matching parts of a malware family with parts of an application provides a strong evidence about the malware presence within that program or even that the program itself is a malware. DroidClone proceeds to expose code clones (actually code segments that are similar) in Android apps in order to detect malware variants [12].

Another approach for detection and identification of Android malware is based on information flow monitoring [13]. The support technique is called Dynamic Information Flow Tracking (DIFT) for monitoring an application during its execution. It allows to monitor where the sensitive data goes within a given environment at execution time. This technique is applied on Android to find out if an application is leaking sensitive data. Here the malware profile is built with a system information flow graph [13].

HinDroid is an intelligent Android malware detection system based on structured heterogeneous information network (HIN). This anti-malware method relies on a multi-kernel learning process and a system architecture that includes: an unzipper and decompiler, a feature extractor, the HIN constructor, the multi-kernel learner and, finally, the malware detector functionality [14]. This solution integrates several machine learning-based tasks with some optimizations that are performed at various processing stages, including the multi-kernel approach. This is a relevant example for the potential of Machine Learning to support the design of reliable security applications including those for malware detection/recognition for the Android platform.

The Machine Learning-based approaches present a significantly potential to increase the security performances of both approaches (static and dynamic) in malware detection for Android, with an optimal combination of unsupervised and supervised learning methods and also with an appropriate feature selection. Some recent developments with special focus of feature-related issues are described as following. These are relevant for our work, an ongoing research to design, develop a ML-based methodology for malware detection using some optimizations for feature generation/selection and classification process.

In [15] it is approached the detection of malware on Android based on application-level features. The reason to use application features for Android malware detection is that most of the existing anti-malware solutions are able to only detect the malware in their original version, but not after some transformations or even obfuscation. Another issue is that the permission feature-based detection can lead to many undetected malwares [15]. The referred study proposes a more comprehensive static analysis method in which the machine learning-based detection model is applied for additional features exceeding the conventional permission-based features that are already extensively used in many of the actual products. The classification process for malware detection/recognition uses different features including permissions and suspicious API (Application Programming Interface) calls. The classification output decisions are malware and non-malware (benign app), respectively.

The features examination for Android malware detection is also researched in [16], by considering the Java API call data as features sources for malware detection. The extracted features are weighted in order to provide some reliable indicators of the malicious potential of the source data or Android app. A hyperparameter optimization is applied in order to enhance the malware recognition accuracy. This process concerns the learning rate (hyperparameter), not the features weight (parameter). The parameters are the values of certain variables used in the model and that are affected by the overall training process. The hyperparameters are the higher-level properties of the model that are chosen by the user. Other typical examples of hyperparameters are: the decay rate of the learning rate, the number of training steps, the size of training dataset [16].

Another Machine Learning approach to Android malware detection is presented in [17]. Here a certain number of features are extracted and then a One-Class Support Vector Machine is trained in an off-line (off-device) way, in order to leverage the higher computing power of application servers. The main idea of that approach was to design a classifier able to recognize most of the training samples as belonging to the positive class, and also to classify the validation data as negative only there are sufficient differences from the training samples. For the classification, the design is based on the SVM classifier that is a linear model in a high-dimensional feature space. The resulting discriminant function is given based on a constrained quadratic optimization problem [17]. The considered features derive from the permissions to access some restricted functionalities of Android OS. The list of the requested permissions provides the features for the further classification process. These permission-based features are divided into 2 groups: standard and non-standard permissions features, respectively. The classification approach was a kernel-based one.

A comprehensive study concerning the Android malware detection techniques with Machine Learning is presented in [18]. It concerns the ML algorithms that are designed to analyze features extracted from malicious apps and their using to recognize and detect the malware applications. This is an overview of the actual trends in design anti-malware detection solutions using ML-based approach and with focus on the Android OS.

The ML methods are applied also to analyze the features of some particular malware families for Android in order to find out some hidden attributes and behavioral features that could be exploited to recognize and detect similar behaviors belonging to other malware families. An example is using DroidDream Android malware behavior for identification of other malware families [19]. In this example, the Naive-Bayes classifier is applied to analyze the DroidDreamLight family also using a dataset of the most popular Google Play apps.

As stated in [20], the most recent and sophisticated Android malware employ detection avoidance mechanisms

able to hide their malicious properties from most of the actual anti-malware tools. These mechanisms are based on various anti-emulator techniques in which the malware programs try to mask or even to hide their malicious nature with an emulator detection strategy. This is why the anti-emulation countermeasures become very important for the Android malware detection, requiring for an approach with analysis and detection based on real devices. This could be a reliable design solution to alleviate the issues of anti-emulation and also to improve the dynamic analysis effectiveness. In [20] it is presented a research concerning the machine learning-based malware detection with dynamic analysis on real devices. The dynamic features are automatically extracted from Android devices and also a comparative analysis of emulator vs. device-based detection with several ML algorithms is performed [20]. The same study concludes that several features for Android malware detection and recognition could be more effectively extracted within an on-device dynamic analysis process compared to the emulator-based approach. It states that all the considered ML-based detection algorithms performed better when applied to features that are extracted within the on-device dynamic analysis.

III. THE MACHINE LEARNING-BASED METHODOLOGY FOR ANDROID MALWARE DETECTION

Any machine learning solution design and development process requires the following elements: the design dataset, the features, the modelling process together with its evaluation and optimization in order to meet the target performances of the real application. Here the focus is on the Android malware detection and recognition, but the main steps of the design process follow the same operations sequence. In order to design a reliable malware detection and recognition machine learning-based solution the following elements should be considered for the overall modeling process:

- Input data for the model design: the input variables that allow to detect malware. These data types are used for the further feature generation and selection steps. The extracted data concern functional aspects about Android platform and the changes generated by various apps, providing behavioral information;
- Feature generation and selection: a process in which the most informative elements are extracted from the input data to provide the required features for the further classification stage. The feature generation uses statistics and some qualitative information related to the Android functionality and applications behaviors that could be relevant for various malware families. The feature selection looks to optimally adjust the dimensionality of the data for the classification;
- Data classification (classifier design): to select, design and develop the classification model in order to accurately recognize various classes of malware (in this case). The overall modeling process includes the following operations:
    - Model selection, in which several classifiers are considered for the malware detection function design;
    - Model training and testing, in which the model is built based on the training dataset. The testing should be performed on an independent dataset;
    - Optimization based on ROC curves: the best operating point is selected according to the application requirements. The classifier design also includes the model parameters tuning or adjustment. This is done according to the desired performance of the detection function.

This is an ongoing research, just at its beginning stage. We briefly consider the basics of the design process focusing on the main contributions of this research:

- the proposed framework to generate features for Android malware detection/recognition. It will be a combination of functional features and some statistical features that should be derived based on an history of the events, malware occurrences and their consequences on the end-users devices and their mobile apps, for the Android platform in our case;
- the proposed framework to perform data classification for Android malware detection/recognition. Here we propose a hierarchical classification model in which the several malware classes are considered;
- the proposed framework to conduct the performance estimation and optimization of the detection/recognition process in order to meet the real mobile applications security requirements. An optimization method based on ROC curves analysis will be adopted for this purpose.

A. Input variables and target performance

The overall detection and recognition for various malware classes is based on some functional/operational data about the devices and their operating systems. These information concerns permissions, mobile apps settings, device attributes, protocol-related information, OS-related attributes, as in the previously cited works [15] - [20]. All of them are grouped into the input variables that provide the required data sources for the feature generation and selection process. On the other hand, the overall design process should start from the target KPI (Key Performance Indicators) that are defined according to the real applications end-users requirements and also considering the complexity vs. performance ratio. A typical KPI set is True Positive Rate (TPR) vs. False Positive Rate (FPR). In the case of the malware detection/recognition process, the significance of these KPI are the following, based on a certain target malware class that is considered for the recognition process:

- TPR is the detection rate for the target malware;
- FPR is the alert rate (or false alarms rate) for the target malware

The typical targets for the security applications KPI, as the case of malware detection/recognition for Android, are to achieve a detection rate (TPR) of at least 90% for a false alarm rate (FPR) not exceeding 10%. This should be done by

applying one or several of the following design/development strategies:

- Optimizing the feature space design, that concerns the feature generation and selection process looking to adjust the dimensionality such as to provide an appropriate performance vs. complexity ratio based on an optimal informative degree of the retained features;
- Optimizing the classification design, that concerns the modelling process for data classification, including the model parameterization tuning (parameters and hyper-parameters, respectively) and with integration of the models into a hierarchical classification system;
- Adjusting the security thresholding for the real mobile application, such as to provide the optimal degree of sensitive data protection against an enlarging range of threats for mobile apps and devices.

B. Feature Generation and Selection

The feature generation is the process in which the relevant features for the classification are derived from the raw data. This process could involve some computations, various statistics and sometimes a careful selection strategy in order to provide the best features able to ensure the target KPI as required (at least 90% detection rate against an alert rate not exceeding 10%). The proposed method for the feature generation and selection in order to perform the Android malware detection/recognition is depicted in Fig. 2. The overall feature generation process has the following stages:

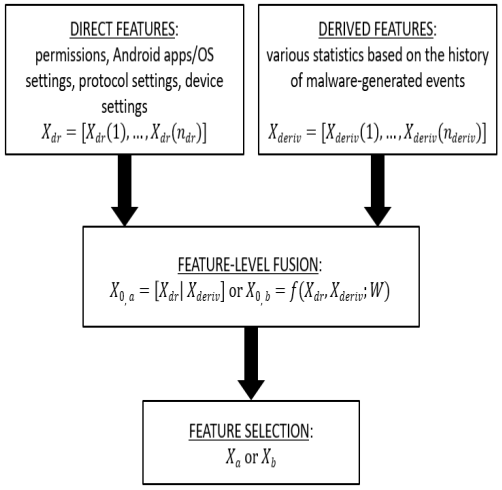

Fig. 2. The overall feature generation process for Android malware detection

- Feature extraction from the raw data concerning various functional issues of the Android devices, apps and OS, permissions, protocol issues. The extracted data are embedded into a set of attributes named direct features. These features are represented into a mathematical form as direct feature vector (X_{dir}). The size of this feature vector is n_{dir}. If the raw data contain some categorical variables, the One-Hot Encoding should be applied in order to provide numerical variables as features [21];
- Feature derivation, a process in which some statistics are applied on primary data concerning various malware events that are generated within the OS or Android apps running on the mobile device. It is an history-based approach in which new features are computed based on the direct features that were previously defined. The derived feature vector is X_{deriv} and the number of derived features is n_{deriv};
- Feature-level fusion, a process in which several features sets extracted from various sources are combined to generate a single feature vector for the further classification during the overall modeling. The approach for the feature fusion is an extension from the multimodal biometric security systems, in which various features extracted from different human traits are fused to improve the recognition accuracy [22]. Depending on the resulted features spaces sizes (for the direct and derived features, respectively), the fusion process could be applied with one of the following 2 variants [23]:
  - The *concatenation fusion*: a fusion process in which several feature vectors are concatenated to generate a single feature vector. This fusion is applied when the features are not homogeneous, therefore if the feature vectors have different sizes. For our design, the resulted feature vector is

$$X_{0,a} = [X_{dir} \mid X_{deriv}] \quad (1)$$

    If $n_{dir} \neq n_{deriv}$ (the direct and derived features sets have not the same size), then the concatenation fusion is the main design option for the overall feature space. An alternative could be to complete the lower-dimensional feature vector with null values in order to provide the common size for both features sets. This could be a design option for the missing values of some features [24]; however, it could lead to inconsistent classifier outputs if the functional fusion is not appropriately designed. The main drawback of this fusion technique is the significant dimensionality increasing of the feature space dimensionality, with impact on the processing computational complexity and the curse of dimensionality problem [24];
  - The *functional fusion*: a fusion process in which several feature vectors with a common size (the same dimensionality of the feature spaces) are combined using a certain function (mathematical rule), sometimes with application-specific chosen parameters (typical several weights). For this design, the resulted feature vector is

$$X_{0,b} = f(X_{dir}, X_{deriv}; W) \quad (2)$$

    where W is the set of weights that shows the contribution of the fused features to the overall feature set, depending on their significance for the real security application (in this case, malware detection/recognition for

Android). The fusion is based on a selected function *f*, that could be a weighted sum/average or other mathematical model that could be applied to compute the global feature vector from the direct and derived feature vectors, respectively. The critical condition for the functional feature-level fusion is to have a common feature space dimensionality of the fused data sources, in order to apply a certain function on vectors with the same size;

- Feature selection, a process in which the feature space dimensionality is optimally adjusted in order to provide the most relevant information for an accurate malware recognition/detection process. The main goal is to preserve the most relevant information with an optimal cost concerning the processing complexity within the further classification stage. In the proposed design methodology, the feature selection is performed just after the other feature generation operations, and even after the feature fusion. The reason for this design option is to reduce the computational and time expenses involved by running the feature selection algorithms on each of the generated feature vectors. In many of the actual ML-approaches for malware detection, the feature selection is applied just after the feature extraction from each of the several data sources required in the real application. From the existing feature selection algorithms, we only consider the following suboptimal techniques for this design: forward-searching feature selection, backward-searching feature selection, floating-search feature selection, individual ranking feature selection, random feature selection [24], [25]. The performances of these feature selection strategies should be compared on the datasets within the malware recognition application, in order to find out the suitable method to keep the best features, ensuring an optimal execution time vs. recognition accuracy ratio. However, some researches concerning these feature selection methods applied on biometric data suggests that the individual ranking has an optimal execution time for an overall feature space size that does not exceed 50 [26]. This is why the individual ranking could be considered if the feature extraction and fusion processes provide no more than 50 final features. The feature selection is important for the malware detection because this process requires to properly exploit the most relevant input data properties to provide the target KPI with the complexity/costs minimization. Within this development framework, the chosen feature selection algorithm will provide outliers and redundancy removal to only retain the most informative features [26].

C. *Data classification (the classifier general designs*

In our design, the data classification for Android malware recognition is performed with a hierarchical classifier in which the decisions are provided on several stages, according to the algorithm depicted in Fig. 3. Actually, it is an extension of the hierarchical multimodal algorithms that were applied to develop biometric security models [22], [23], [26]. The difference results from the features the algorithms apply to; in this design the features are extracted and derived from some behavioral characteristics of the Android devices and apps under various real-time operational environments.

Furthermore, special attention is given to recent number one malware threat of ransomware, which is being replaced by cryptocurrency mining various coins or tokens traded on the blockchain [27], anonymous blockchains being preferred by hackers.

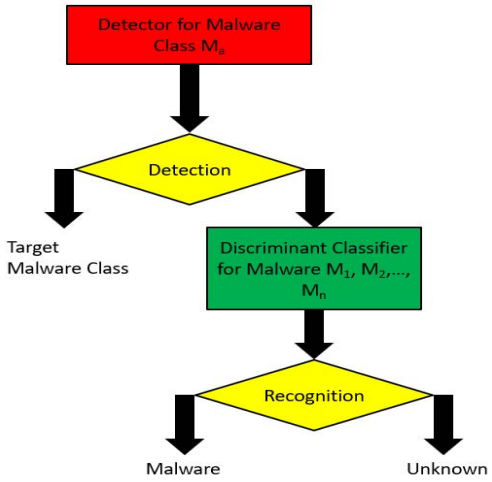

Fig. 3. The hierarchical classifier design for Android malware detection/recognition

The classification system design is based on a hierarchy of classifiers with 2 stages. In the first stage the classifier is a detector, therefore a target-vs.-non-target model that provides decisions to recognize a target malware class. In this way, the multi-class problem of the malware recognition is divided into 2 sub-tasks: a first detection stage that performs the target malware class identification and a second stage that performs the discrimination among all the other considered malware.

The main difference between a detector and a discriminant is that a detector is only focused on a target (most important) class, and for its training the data samples belonging to all the other classes are grouped into a single non-target class; the discriminant provides multi-class decisions (with more than 2 classes, depending on the application).

In the proposed design of the classifier for Android malware detection/recognition, if M_X is the class label for the malware type (or a class label for several malware apps with very similar features and severity), then a malware detector (target-vs.-non-target classifier) provides the following decisions (for a current datapoint X within the designed feature space), no matter the underlying classification model:

$M_X: M_T$ *for the target malware class label;*

$M_{non\text{-}T}$ *for the non-target malware class label.*

The target malware class could include several malware types that are grouped in a whole malware family for Android. The target malware class should be chosen based on its importance or severity for the mobile device, apps and OS functionalities and sensitive end-used data.

A malware discriminant model provides the following malware class labeling decisions:

$M_X: M_1$ *for the first malware class (individual malware or malware family, if case);*

M_2 for the second malware class (individual malware or malware family, if case);

……..

M_n for the n^{th} malware class (individual malware or malware family, if case);

$M_?$ for an unknown malware class

D. Model Selection

The model selection looks to find out the potentially best classifications models for both stages of the designed classifier for Android malware detection/recognition. In this case, given the estimated number of the extracted and generated features (direct and derived features, respectively), the Support Vector Machine classifier seems to be one of the most reliable design option, at least for the first classification stage (detection), that requires for a 2-class model.

Anyway, the training process should allow to fix or even to adjust both the suitable parameters and hyper-parameters of the chosen classifier. Also, the classifier design should consider finding the suitable kernel, given the non-linear characteristic of the direct and derived features for Android malware detection/recognition.

E. Training and testing

For the training and testing (validation) processes, the proposed methodology will divide the data depending on the time periods in which the malware details are acquired. Actually, this approach is justified by the actual dynamic in malware occurrences for Android devices and apps. This requires to periodically update the training data in order to ensure the suitable information for the malware detection and recognition. Therefore, the training process should be not a static one, but it must use the refreshed data about the new malware and various vulnerabilities in Android apps, OS and devices.

One of the most critical issues that should be carefully considered while designing a high-performance anti-malware solution for Android using the proposed detection/recognition methodology is the relevance or the malware classes (families) and their representativity within the training data. This issue relates to the occurrence frequency and the severity degree of those apps. The class representation remains actually a significant challenge in training classifiers able to perform in various real applications.

F. Performance estimation and optimization with ROC curves

The performance measures that are considered here are TPR and FPR, respectively. The target KPI for performance estimation should be the following:

- at least 90% for malware detection rate (TPR for a certain malware class or a malware family including several malicious apps);
- an alert rate (FPR for the target malware class/family) that does not exceed 10%.

The performance evaluation for a certain malware class should be performed using ROC (Receiver Operating Characteristic) curves that are generated from a set of classifier operating points. An operating point is a pair of the performance estimators that are achieved for a certain security threshold. Therefore, it is a thresholding-based approach in which the security thresholds are based on the real application specific requirements. The typical ROC curve for such a security system (in our case, a malware detection module) looks like the one depicted in Fig. 4.

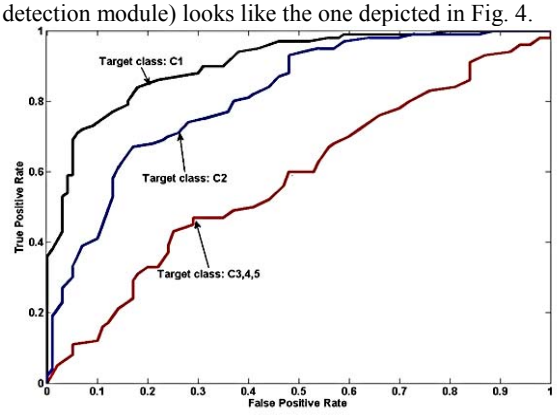

Fig. 4. A typical ROC curve for a multi-detector security system with hierarchical classification structure

Starting from the ROC curves that are generated using the set of operating points (based on a specific thresholding), one can define an iterative method for performance optimization that can be used for various applications requiring Machine Learning-based approaches. A feasible case is that of ML-based malware detection for Android, with an appropriate training of the classifier. The method steps are depicted in Fig. 5 [23].

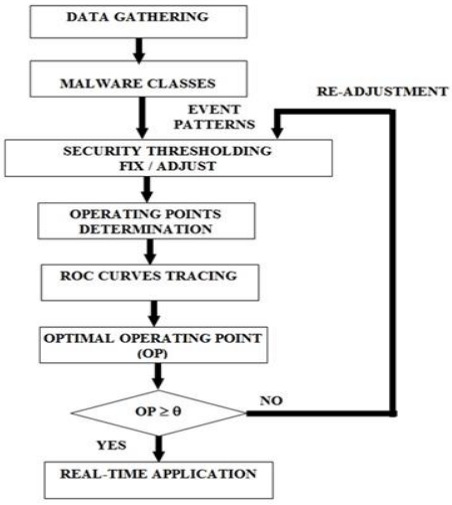

Fig. 5. Iterative optimization method for security systems, with application in malware detection

In Fig. 5, the optimization is based on a thresholding approach, in which the best operating point that is found out till the current moment should be compared with a security application specific threshold, that in our case is the detection threshold. The iterations are performed according to the real environment events (for instance to take in

account the new malware and vulnerabilities occurring for the Android apps and devices

IV. CONCLUSIONS

In this paper we defined a full methodological framework to design, develop and evaluate malware detection and recognition solutions for Android platform, while considering advanced Machine Learning approaches with a suitable optimization procedure in order to ensure the target KPI as concerning the malware detection rate, also with a corresponding reduction of the false alarms. This is an ongoing research in which the proposed methodology includes some data analytics for the best feature generation in order to support a high-performance design and modeling process and finally to implement an optimized solution.

Further work must be done especially as concerning the considered features for malware recognition, as much as there is a fast evolution in this domain, with new malware and potential malicious apps for Android devices and OS. Anyway, the Machine Learning approach becomes currently the most applied approach to design various security solutions for a continuously enlarging applications area.

ACKNOWLEDGMENT

This work has been supported in part by UEFISCDI Romania and MCI through projects ODSI, ALADIN, ToR-SIM, PARFAIT, and funded in part by European Union's Horizon 2020 research and innovation program under grant agreements No. 777996 (SealedGRID project) and No. 787002 (SAFECARE project).

REFERENCES

[1] S. Arshad, A. Khan, M. A. Shah and M. Ahmed. 2016. Android Malware Detection & Protection: A Survey, (IJACSA) *International Journal of Advanced Computer Science and Applications*, Vol. 7, No. 2.

[2] R. Raveendranath, R. Venkiteswaran and A. J. Babu. 2014. Android Malware Attacks and Countermeasures: Current and Future Directions

[3] Malwarebytes LABS: State of Malware Report 2017 https://www.malwarebytes.com/pdf/white-papers/stateofmalware.pdf

[4] R. Sato, D. Chiba, and S. Goto. 2013. Detecting Android Malware by Analyzing Manifest Files, *Proceedings of the Asia-Pacific Advanced Network 2013*, pp. 23–31,

[5] C.-Y. Huang, Y.-T. Tsai, and C.-H. Hsu. 2013. Performance Evaluation on Permission-based Detection for Android Malware, Adv. *Intell. Syst. Appl*. - Vol. 2, vol. 21, pp. 111–120,

[6] N. V. Duc, P. T. Giang and P. M. Vi. 2015. PERMISSION ANALYSIS FOR ANDROID MALWARE DETECTION, *The Proceedings of the 7th VAST - AIST Workshop "Research Collaboration: Review and perspective"*.

[7] A. Skovoroda and D. Gamayunov. 2015. Securing mobile devices: malware mitigation methods, *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications*, volume: 6, number: 2.

[8] O. Yeshvekar, S. Zende, D. Walvekar, N. Wabale, A. Korde, M. Saravanapriya, N. S. Patil. 2015 . A Survey of Evaluation Techniques for Android Anti-Malware using Transformation Attacks, *International Journal on Recent and Innovation Trends in Computing and Communication ISSN*: 2321-8169 Volume: 3 Issue: 11,

[9] V. Rastogi, Y. Chen and X. Jiang. 2014. Catch Me if You Can: Evaluating Android Anti-malware against Transformation Attacks, *IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY,*

[10] L. Weichselbaum, M. Neugschwandtner, M. Lindorfer, Y. Fratantonio, V. Veeny and C. Platzer. 2014. ANDRUBIS: Android Malware Under the Magnifying Glass, TECHNICAL REPORT TR-ISECLAB-0414-001,

[11] C. Jarabek, D. Barrera and J. Aycock. 2012. ThinAV: Truly Lightweight Mobile Cloud-based Anti-malware, *ACSAC '12 Dec*. 3-7, Orlando, Florida USA

[12] S. Alam, R. Riley, I. Sogukpinar and N. Carkaci. 2016. DroidClone: Detecting Android Malware Variants by Exposing Code Clones

[13] R. Andriatsimandetra and V. V. Triem Tong. Nov 2015, Detection and Identication of Android Malware Based on Information Flow Monitoring, *The 2nd IEEE International Conference on Cyber Security and Cloud Computing (CSCloud 2015)*, New York, United States

[14] S. Hou, Y. Ye, Y. Song and M. Abdulhayoglu. 2017. HinDroid: An Intelligent Android Malware Detection System Based on Structured Heterogeneous Information Network, Proceedings of KDD'17, August 13-17, Halifax, NS, Canada

[15] G. Kapse and A. Gupta. 2015. Detection of Malware on Android based on Application Features, *(IJCSIT) International Journal of Computer Science and Information Technologies*, Vol. 6 (4), 3561-3564

[16] M. Leeds, M. Keffeler and T. Atkison. 2017. Examining Features for Android Malware Detection, *Int'l Conf. Security and Management | SAM'17*

[17] J. Sahs and L. Khan. 2012. A Machine Learning Approach to Android Malware Detection, *2012 European Intelligence and Security Informatics Conference*

[18] B. Baskaran and A. Ralescu. 2016. A Study of Android Malware Detection Techniques and Machine Learning, *MAICS 2016*

[19] Y. Kim, K. J. Liszka and C. C. Chan. 2016. Using DroidDream Android Malware Behavior for Identification of Other Android Malware Families, *Int'l Conf. Security and Management, SAM'16*

[20] K. Alzaylaee, Y. Yerima and S. Sezer. 2017. EMULATOR vs REAL PHONE: Android Malware Detection Using Machine Learning, *IWSPA 2017 Proceedings of the 3rd ACM International Workshop on Security and Privacy Analytics, co-located with CODASPY'17*, pages 65-72, Scottsdale, Arizona, USA - March 24 – 24

[21] J. Brownlee. 2017. Why One-Hot Encode Data in Machine Learning?, https://machinelearningmastery.com/category/machine-learning-process/

[22] S. Soviany, C. Soviany and S. Puşcoci. 2016. A Multimodal Biometric System with Several Degrees of Feature Fusion for Target Identities Recognition, *The 2016 International Conference on Security and Management (SAM'16)*, Las Vegas, USA

[23] S. Soviany and S. Puşcoci. 2016. Soluţii multimodale pentru securizarea serviciilor mobile (Multimodal Solutions for Mobile Services Security), *editura AGIR (AGIR Publishing House*)

[24]] S. Theodoridis and K. Koutroumbas. 2009. Pattern Recognition 4th edition, *Academic Press Elsevier*

[25] S. Soviany. 2013. *Decision Optimization for Biometric Identification Systems*, Ph.D.Thesis

[26] S. Soviany, V. Săndulescu, S. Puşcoci, C. Soviany and M. Jurian. 2017. An Optimized Biometric System with Intra- and Inter-Modal Feature-level Fusion, ECAI 2017 - International Conference – 9th Edition Electronics, Computers and Artificial Intelligence, Târgovişte, România

[27] Rios, E., Prünster, B., Suzic, B., Prieto, E., Notario, N., Suciu, G., Ruiz, J.F., Orue-Echevarria, L., Rak, M., Franchetto, N. and Balboni, P., 2017. Cloud technology options towards Free Flow of Data.