Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 192 (2021) 2340–2346

Procedia Computer Science

www.elsevier.com/locate/procedia

25th International Conference on Knowledge-Based and Intelligent Information & Engineering Systems

# Android Collusion Detection by means of Audio Signal Analysis with Machine Learning techniques

Rosangela Casolare[a,*], Umberto Di Giacomo[a], Fabio Martinelli[b], Francesco Mercaldo[c,b,*], Antonella Santone[c]

[a] *Department of Biosciences and Territory, University of Molise, Pesche (IS), Italy*
[b] *Institute for Informatics and Telematics, National Research Council of Italy (CNR), Pisa, Italy*
[c] *Department of Medicine and Health Sciences "Vincenzo Tiberio", University of Molise, Campobasso, Italy*

**Abstract**

Smartphones, tablets and other mobile devices have become objects that we can no longer do without, as a matter of fact for us they are like an extension of our body and many people are addicted to them; this behavior is a consequence of the use we make of it, since these devices allow us to manage sensitive data (i.e., financial ones) and access information of different types (i.e., photos, messages or health data). For this reason it is essential to detect the harmful behaviors present within our smartphones, taking into account the weaknesses of the current anti-malware mechanisms. In this article we propose an approach capable of discriminating trusted applications from those that instead have malicious behavior, since they are involved in a colluding attack. We resort to the processing of the audio signal extracted from the conversion of an application into an audio file. The processing allows to generate a vector of characteristics to be analyzed with different classifiers. The experimental analysis is performed on a set of Android applications consisting of 359 trusted and (colluding) untrusted applications, showing the effectiveness of our method in detecting colluding applications.



*Keywords:* malware; audio; Android; colluding; malware detection; machine learning; security; classification

## 1. Introduction

People's daily habits have been changed by technological development which has grown very quickly in recent years. Among the various technological fields, the one that has most influenced the change in our habits is related to the communication devices i.e., smartphone, tablet, phablet, wearable.

* Corresponding author: Rosangela Casolare, Francesco Mercaldo
*E-mail address:* rosangela.casolare@unimol.it, francesco.mercaldo@unimol.it

1877-0509 
10.1016/j.procs.2021.08.224

Like everything, technological development has brought with it pros and cons: if on the one hand these devices allow us to be connected at any time with people all over the world by breaking down every barrier, on the other hand their rapid spread does not has allowed adequate education on the correct use of them, so as to avoid creating and suffering damage. The adoption of smartphones has become so obvious that people have no real perception of the data stored within their devices and how dangerous this data can be if in the hands of malicious people such as cybercriminals. The latter could take possession of data such as personal photographs of the user, financial data, emails, messages and so on, creating various types of damage [1].

Surely the operating system installed on the devices plays a fundamental role in this scenario. When we think of an operating system, speaking of smartphones, the first that comes to mind is Android, this is because the Android platform is the most popular on the global market [2, 3]. The features that make Android interesting for attackers is its ”open nature” [4, 5], in fact it is possible to download application in official markets (i.e., Google Play) and in third-party ones. In this way it is easy to come across malicious applications: often the unofficial markets are used to download free applications that are paid on the official ones, therefore it is very likely that the free application found on the unofficial market is infected [6]. However, this does not exclude the possibility of finding infected applications even in the official market.

The described scenario is composed by two parts: the user and the attacker, but we can introduce a third part characterized by the defender. With the word defender in the malware analysis context we refer to anti-malware, tools with the aim of analyzing software and detecting the presence of harmful actions in them.

To evade the anti-malware detection, attackers continuously work to develop more and more complex and undetectable payload [7]: in this context a new type of developed malicious payload is the so-called *Collusion Attack*. The collusion is a communication between two (or more) applications stored in the same device that have minimum permission to execute their role, in this way they do not make the anti-malware that analyses them suspicious, because it is their collaboration aimed to launch the attack, while individually they are completely harmless [8] (because current anti-malware can only scan one application at a time).

In order to perpetrate a colluding attack at least two applications must be involved in the harmful collusion: these applications play two different roles, since to be unsuspected, one must only be able to access information (without being able to share it), and this implies that the application is able to read the data thanks to the *GET* property, while the second application is able to write and then modify the data, thanks to the *PUT* property. This is why the presence of an application capable of performing a PUT action could be considered as a prerequisite for the presence of a collusive attack.

In this work we propose a detector of colluding application in Android environment. Our approach is based on the conversion of an Android application into an audio file, then we generate a *.csv* file to extract the applications' features that are analyzed to understand whether we are in presence of colluding applications. For the classification have been considered as algorithms: J48, BayesNet, RandomForest, LogisticModelTrees, JRip and DecisionTable. From this analysis it emerges that the best performances are obtained with BayesNet algorithm, with Precision of 0,976 and a Recall of 0,975.

The paper is organized in the following way: in section 2 current state-of-the-art literature is analyzed and discussed; in section 3 is described the proposed approach to convert the apk in audio file, analyse their audio signals and detect the presence of colluding applications, using classifier on the features extracted from the audio signals; in section 4 is described the dataset composition and the algorithms involved in the analysis, showing the effectiveness of the experimental analysis executed; as last, in section 5 conclusion and future research plans are presented.

## 2. Related Work

In this paper we propose a method that analyze the signal features to detect the presence of colluding applications, so we have focused our attention on the exploitation of audio signal for malicious behaviour detection. In literature instead are present different approaches based on behavioural [9] or network features analysis [10] and so on.

SigMal [11] is a framework that performs a generalized signal analysis for the classification of malware. It directly analyzes binary samples through the nearest-neighbours technique and it also support the classification with a dataset of known benign executables.

The authors in [12] have worked on a technique similar to ours: they start extracting the program bytes and converting them into an audio signal. Specifically, these bytes are converted into musical notes (i.e., MIDI note) and audio files are generated after the conversion. Audio features such as Chromagram and MFCC are used to classify music and machine learning classifier like KNN is applied. Differently from the approach we propose, they do not consider colluding applications.

Researchers in [13] developed an approach based on the use of audio signal processing and image classification techniques. They transform the bytes of the program under analysis into audio signals and subsequently apply Fourier transformation for the conversion of the signal from time-domain to frequency-domain and thus generate spectrograms. Convolutional Neural Network as a standard image-classification task is used to analyze the spectrograms.

In [14] the authors propose a method to detect Android malware, detecting the family of the infected sample under analysis. Similar to our method, they convert the application executable into an audio file and using audio signal processing techniques, extract a set of numerical characteristics from each sample. On this data they build different machine learning models to evaluate their effectiveness with regard to malware detection and family identification. Also the authors of this paper do not take into account malicious applications aimed to perform a collusion.

Other techniques were used for colluding applications detection. For instance, in [15, 16] the authors propose a method based on model checking; they represent an Android application as an automata and exploit a set of logic properties to reduce the number of comparisons and another set of properties, automatically generated, to detect colluding applications.

## 3. The Method

In this section we present the proposed method whose purpose is the detection of colluding applications. In Figure 1 is showed the pipeline related to the proposed method: we start converting a set of Android applications (*.apk* files) into audio files (with *.wav* extension), from these audio we extract the features represented by numerical values. Once obtained the features, we use them as a dataset to be given as input to the properly trained classifier for the prediction of colluding applications present in the initial dataset.

To better understand how our approach works, let's describe each step more specifically:

- *step 1: Input dataset*, we start with the training phase to have a prediction model for the collusion detection. In this phase we consider a dataset composed by malicious Android applications (*.apk* files) and the label relative to type of application (i.e., *GET*, *PUT* and *Trusted*) analyzed. In this paper we consider colluding application sharing resources by exploiting the *SharedPreferences* mechanism natively provided by the Android platform[1]. Also trusted applications are considered, obtained with a Python crawler developed by authors aimed to automatically gather free applications from the Android official store[2]. Then we extract from each *.apk* the executable file (with *.dex* extension) containing only the binary of the application, without considering sounds, images and so on in the analysis of the application resources;
- *step 2: Audio signal generation*, once the *dex* file has been extracted, we have to convert it using binary bytes to obtain a digitized raw signal which is then transformed into a *.wav* file. For the generation of *wav* from *dex* we have developed a function that first generates a *wav* header and after that, the *dex* file is opened and each byte belonging to it is converted in *wav*.
  In step 2 we use the *wave* module[3] available in Python. More specifically, we invoke two methods: the *open* method to open the file and the *writeframes* method to write the *wav* file. With the *setparams* methods we used different parameters for the generation of the *wav* files as follows: the number of audio channels is set with a value equals to 1 (1 stands for mono and in this way there is one input equally distributed by the left and the right speakers), the sample width set to n bytes with $n = 1$, the frame rate instead is set to 32768 and the frames number is set to 0 without compression;
  The features computed on the audio samples obtained are shown below:

[1] https://developer.android.com/reference/android/content/SharedPreferences
[2] https://play.google.com/store?hl=it&gl=US
[3] https://docs.python.org/3/library/wave.html

  - *Chromagram*, provides a chromagram representation automatically collected from a waveform;
  - *Spectral Centroid*, is related to the "centre of mass" of a sound sample and this feature is obtainable as the mean related to the frequencies of the audio;
  - *Spectral Bandwidth*, how suggests the name, it is related to the bandwidth of the spectrum;
  - *Spectral Rolloff*, it is represented as the frequency relative to a specific percentage of the total spectral of energy;
  - *Zero Crossing Rate*, this feature is expressed as the radio belonging to the variation of the sign of the sound samples;
  - *Mel-Frequency Cepstral Coefficients*, indicated with the acronimous *MFCC*, it is used to represent the shape of a spectral envelope. We consider 20 different features belonging to this category.
- *step 3: Features extraction*, from the *.wav* files we obtain the feature vectors and in this step we export them into a *.csv* file. The *csv* contains on each row the feature values related to each application analyzed, and as last value of the row we have a label that describes the type of application (i.e., GET, PUT or Trusted);
- *step 4*, the resulting *csv* file is opened in the classifier (i.e., Weka[4]), here we set the parameters for the classification algorithms. For our classification we have chosen six different algorithms to demonstrate that the features extracted from wav files are useful for a discrimination on the type of applications. Models are shown below:
  - *J48 Decision Tree*, it is an algorithm to generate a decision tree starting from C4.5 algorithm. It splits the information into smaller subsets based on a decision; division strategies stop when there is a subset with a place similar class in all the instances. J48 uses the expected class estimations to develop a decision node, it is also able to dealt precise characteristics, missing (or lost) attribute estimations of the data and the attribute costs variation. Its accuracy can be expanded with a "pruning" [17];
  - *Bayes Net*, it is a classifier that assumes strong independence assumptions based on Bayes' Theorem. This classifier, also called naive Bayes classifier, assumes that the presence of a specific feature belonging to a class is not related to the presence of any other feature, the same is true for their absence [18];
  - *Random Forest*, it is an algorithm obtained from the bagging of decision trees; it is composed by hundreds of thousands of decision trees and belongs to those ensemble learning algorithms that use multiple machine learning algorithms to have more accurate predictions. The number of tree in the Random Forest depends on the nature of the training set and also on other parameter (i.e., number of classes, number of beans, maximum depth and so on) [19];
  - *Logistic Model Trees*, it is a classification model that combines logistic regression and decision tree learning. LMT is therefore based on a model tree, the decision tree has linear regression models on its leaves thus providing a piecewise linear regression model [20];
  - *JRip*, with it the classes are examined in increasing size; using incremental reduced error, a class rules dataset is generated and JRip works by treating all the examples of a particular judgment in the training data as a class and using rules able to cover all class members. It proceeds in this way for each subsequent class until all the classes are covered [21];
  - *Decision Table*, it is made up of machine learning algorithms. It is based on a decision table, consisting of a hierarchical table where each entry in a higher level table is decomposed according to the values of a pair of additional attributes to form another table [22].
- *step 5: Collusion detection*, in the last phase, the effectiveness of the built model is assessed. Once extracted the features from audio samples, we use them as input for the models to generate a prediction on whether there is a collusion between a pair of applications.

## 4. Experimental Analysis

The experimental analysis is, in a nutshell, aimed to build several classifiers for the evaluation of feature accuracy to distinguish between *GET*, *PUT* colluding application and *Trusted* ones.

[4] https://www.cs.waikato.ac.nz/ml/weka/

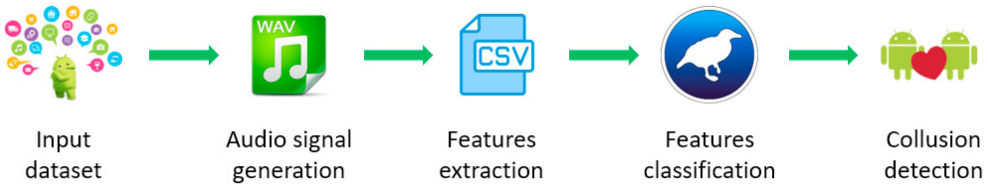


Fig. 1. Approach pipeline.

To this aim, we defined *APP* as a set of labeled application *(FA, l)*, where each FA is associated to a label $l \in \{GET, PUT, Trusted\}$.

For each *FA* we built a feature vector $F \in R_y$ , where $y$ is the number of the features used in training phase ($y = 25$).

We evaluate the effectiveness of the classification method by exploiting the following procedure:

1. build a training set $APP \subset D$;
2. build a testing set $APP' = D \div APP$;
3. run the training phase on $APP$;
4. apply the learned classifier to each element of $T'$.

Four different metrics are exploited to measure the effectiveness of the proposed method in Android colluding detection: Precision, Recall, F-Measure and Roc Area.

The precision represents the proportion of the Android applications truly belonging to a certain category (i.e., *GET*, *PUT* or *Trusted*) among all those which were labelled to belonging to the category. It represents the ratio of the number of relevant applications retrieved to the total number of irrelevant and relevant applications retrieved:

$$Precision = \frac{tp}{tp + fp} \tag{1}$$

where *tp* represents the true positives number and *fp* represents the false positives number.

The recall is defined as the proportion of Android applications assigned to a certain category (i.e., *GET*, *PUT* or *Trusted*), among all the Android applications truly belonging to the category under analysis; in other words, how many samples of the category under analysis were retrieved. It represents the ratio of the number of relevant applications retrieved to the total number of relevant applications:

$$Recall = \frac{tp}{tp + fn} \tag{2}$$

where *fn* is the false negatives number.

The F-measure represents the weighted average between the recall and the precision metrics:

$$F - Measure = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{3}$$

The Roc Area represents a precision/recall pair corresponding to a particular decision threshold. Basically in the Roc Area true positive rate is plotted in function of the false positive rate for different cut-off points of a parameter.

Relating to the experiments, we resort to the Weka[5] data mining tool, in particular among its characteristics we used the "Explorer" section: in this section we loaded the *.csv* file containing all the features of the *apk* converted into *.wav* files and selecting all the attributes, we went to classification section. For the classification, the algorithms in Table 1 were used and for each of them a cross-validation with fold equal to 10 was set. Cross-validation is a technique that consists in the subdivision of the dataset under analysis, in k parts of the same number (in our case we have set

[5] https://www.cs.waikato.ac.nz/ml/weka/

k equal to 10, taking into consideration the size of our dataset which is 359 samples) and at each step, the $k^{th}$ part of this dataset is validated, instead the remaining part of the data goes to make up the training set. Cross-validation is considered with the aim to avoid overfitting in model training[23].

| Algorithm | Precision | Recall | F-Measure | ROC Area |
|---|---|---|---|---|
| *J48* | 0.950 | 0.950 | 0.950 | 0.959 |
| *BayesNet* | 0.976 | 0.975 | 0.975 | 0.999 |
| *RandomForest* | 0.970 | 0.969 | 0.969 | 0.998 |
| *LMT* | 0.955 | 0.953 | 0.953 | 0.977 |
| *JRip* | 0.931 | 0.930 | 0.930 | 0.947 |
| *DecisionTable* | 0.967 | 0.967 | 0.967 | 0.986 |

Table 1. Results obtained from each model.

In Table 1 we have reported the results obtained analyzing our dataset with the listed algorithms. The considered dataset is composed by 80 *GET* applications (i.e., able to read data), 80 *PUT* applications (i.e., able to write data) and 199 applications *Trusted* (i.e., not involved in a collusion attack).

Looking more closely at the classification results reported, we can immediately notice that all values for the considered metrics are above the 0.9 threshold: therefore our approach allows us to achieve generally satisfactory results. From the Table 1 analysis, however, among the various algorithms proposed, the BayesNet is the one that provides us the highest values: it reports a Precision of 0.976, a Recall of 0.975, an F-Measure of 0.975 and a Roc Area of 0.999 (very close to 1).

## 5. Conclusion and Future Work

Currently malware detection is a real problem, just think of the inability of signature-based anti-malware to detect new malicious payloads and the ability of malware authors to develop new malicious code. Another problem is characterized by users unaware of the malicious behavior that silently reads their data stored in the devices and sends them outside (i.e., colluding attack). By reflecting on the weaknesses of the signature-based methods provided by anti-malware, it is possible to obtain undetectable malware using only tools aimed at transforming the code structure, this is possible by applying code transformation techniques. In this regard, we propose a technique for classifying the type of application based on its behavior, so as to understand whether or not it is an application involved in a collusive type of attack. In detail, we analyze an audio stream, which we obtain from the Android application under analysis, to extract a set of numerical characteristics. The functionalities obtained constitute the input for various machine learning classifiers that we evaluate on a dataset consisting of 359 .apk files for the Android environment. We obtained an accuracy of 0.976 using BayesNet as a machine learning model; the proposed method proves to be effective for detecting the type of Android application.

As a future work, the dataset could be expanded with colluding applications having different types of Android shared resources (in this work only the *SharedPreferences* type resources are considered) and we could try to locate the specific pattern relating to the malicious behavior. Moreover, we will consider also malware applications belonging to widespread families in order to experiment whether the proposed method is aimed to distinguish between colluding malicious application and generic malware on Android environment. It is possible to experiment whether proposed method is cross-platform, by analysing a dataset composed of iOS applications, thus varying the type of operating system for mobile devices. We plan to use Multiple Instance Learning, a type of supervised machine learning with the aim to investigate whether audio files split in segments can be helpful to increase the obtained performances in collusion detection.

## ACKNOWLEDGEMENTS

This work has been partially supported by MIUR - SecureOpenNets, EU SPARTA, CyberSANE and E-CORRIDOR projects.

## References


[1] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones," *ACM Transactions on Computer Systems (TOCS)*, vol. 32, no. 2, pp. 1–29, 2014.

[2] W. Enck, "Defending users against smartphone apps: Techniques and future directions," in *International Conference on Information Systems Security*. Springer, 2011, pp. 49–70.

[3] F. Mercaldo and A. Santone, "Deep learning for image-based mobile malware detection," *Journal of Computer Virology and Hacking Techniques*, pp. 1–15, 2020.

[4] G. Iadarola, F. Martinelli, F. Mercaldo, and A. Santone, "Formal methods for android banking malware analysis and detection," in *2019 Sixth International Conference on Internet of Things: Systems, Management and Security (IOTSMS)*. IEEE, 2019, pp. 331–336.

[5] A. Ferrante, E. Medvet, F. Mercaldo, J. Milosevic, and C. A. Visaggio, "Spotting the malicious moment: Characterizing malware behavior using dynamic features," in *2016 11th International Conference on Availability, Reliability and Security (ARES)*. IEEE, 2016, pp. 372–381.

[6] T. Nguyen, J. Mcdonald, W. Glisson, and T. Andel, "Detecting repackaged android applications using perceptual hashing," in *Proceedings of the 53rd Hawaii International Conference on System Sciences*, 2020.

[7] A. M. Memon and A. Anwar, "Colluding apps: Tomorrow's mobile malware threat," *IEEE Security & Privacy*, vol. 13, no. 6, pp. 77–81, 2015.

[8] R. Casolare, F. Martinelli, F. Mercaldo, and A. Santone, "A model checking based proposal for mobile colluding attack detection," in *2019 IEEE International Conference on Big Data (Big Data)*. IEEE, 2019, pp. 5998–6000.

[9] N. K. Popli and A. Girdhar, "Behavioural analysis of recent ransomwares and prediction of future attacks by polymorphic and metamorphic ransomware," in *Computational Intelligence: Theories, Applications and Future Directions-Volume II*. Springer, 2019, pp. 65–80.

[10] H. M. Kim, H. M. Song, J. W. Seo, and H. K. Kim, "Andro-simnet: Android malware family classification using social network analysis," in *2018 16th Annual Conference on Privacy, Security and Trust (PST)*. IEEE, 2018, pp. 1–8.

[11] D. Kirat, L. Nataraj, G. Vigna, and B. Manjunath, "Sigmal: A static signal processing based malware triage," in *Proceedings of the 29th Annual Computer Security Applications Conference*, 2013, pp. 89–98.

[12] M. Farrokhmanesh and A. Hamzeh, "Music classification as a new approach for malware detection," *Journal of Computer Virology and Hacking Techniques*, vol. 15, no. 2, pp. 77–96, 2019.

[13] A. Azab and M. Khasawneh, "Msic: malware spectrogram image classification," *IEEE Access*, vol. 8, pp. 102 007–102 021, 2020.

[14] F. Mercaldo and A. Santone, "Audio signal processing for android malware detection and family identification," *Journal of Computer Virology and Hacking Techniques*, pp. 1–14, 2021.

[15] R. Casolare, F. Martinelli, F. Mercaldo, and A. Santone, "Detecting colluding inter-app communication in mobile environment," *Applied Sciences*, vol. 10, no. 23, p. 8351, 2020.

[16] ——, "Malicious collusion detection in mobile environment by means of model checking," in *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2020, pp. 1–6.

[17] N. anaN and V. thri, "Performance and classification evaluation of j48 algorithm and kendall's based j48 algorithm (knj48)," *International Journal of Computer Trends and Technology*, vol. 59, pp. 73–80, 05 2018.

[18] R. R. Bouckaert, "Bayesian network classifiers in weka," 2004.

[19] M. Pal, "Random forest classifier for remote sensing classification," *International journal of remote sensing*, vol. 26, no. 1, pp. 217–222, 2005.

[20] N. Landwehr, M. Hall, and E. Frank, "Logistic model trees," *Machine learning*, vol. 59, no. 1-2, pp. 161–205, 2005.

[21] W. Shahzad, S. Asad, and M. Khan, "Feature subset selection using association rule mining and jrip classifier," *International Journal of Physical Sciences*, vol. 8, pp. 885–896, 04 2013.

[22] V. J. Hodge, S. O'Keefe, and J. Austin, "A binary neural decision table classifier," *NeuroComputing*, vol. 69, no. 16-18, pp. 1850–1859, 2006.

[23] D. L. Parnas, "The real risks of artificial intelligence," *Communications of the ACM*, vol. 60, no. 10, pp. 27–31, 2017.