Hybrid malware detection model

Abstract

With a grown of malware usage on mobile platform, practitioners and researchers attempt to provide a good way to produce suitable malware detection model. During the last decades, some models built are based on program analysis (static and/or dynamic). Nowadays, hackers make malware able to avoid some detection with evasion techniques. In this way, we introduce a new approach to deal with these evasion techniques. To perform that our process performs static and dynamic analysis but between these two steps, our model makes a filtering as from call graph mining executed during the static analysis. This model provides a detection with a respectable degree of accuracy near 0.98 and an F1_Score of 0.95 while avoiding a waste of time generated by dynamic analysis with limited scope and a boolean detection.

Keywords

reverse engineering, malware detection, dynamic analysis, static analysis, hybrid analysis, android, mobile malware, machine learning.

Introduction

Every day when we watch the news, we could hear some information about data leakage, criminal activities, or global breakdown when malwares are implicated. Last month, we saw a lot of examples of banking frauds using banking trojans or ransomware. In the near past, these criminal activities did not use informatic programs because critical services like service banking were not easily accessible to customers. Today, these services are always connected to customers by many devices (computer, smartphone, IoT's…). Smartphone represent a critical target because this type of device is always activated and connected to the Internet to ensure its functioning (sending message, receive data from SDN[1]…). As we saw, this is a trendy topic.

Now, we focus our attention on malware analysis techniques. During the two previous decades, many researchers have been working on model analysis conception. All models are based on two principal technics: static and dynamic analysis. We will introduce these concepts later. The research evolved very quickly on this topic and provide many tools to perform malware detection. These tools used a plenty of mathematical theory as "graph theory", "machine learning", "classification", "data modeling", "deep learning" and more other. To differentiate these models, we have two dimensions: metrics using by the model and analysis approach implemented on its.

Features: uses to measure some aspect of malware behaviors, there are a plenty. We could regroup it on two family: syntactic and resource centric [07]. Usually, model extract some features to build metrics and concentrate is work on one type of metrics. The first models used API calls to evaluate potential suspect behavior [01] [02] [04]. These models are based on graph analysis and semantic analysis, but they have some limitation especially when programs are obfuscated. Some improvements have been done to avoid this limitation; notably semi-level language designing called "Aposcopy" [15]. Year after year, researcher include many other types of metrics like app structure [01], network data flow [01], CPU consumption [05], function calls [01] [04] and system events. The choice made by researcher determines the process and the approach use on a model.

[1] Social network like Facebook, Linkedin….

Approach: that's represent the logical path implemented on model. The first models used one analysis technic (dynamic or static). In the last decade, researchers combined the two technics to avoid some limitation encountered by the old models. These limitations are as following: data extraction is impossible due to binaries obfuscation [01][17], high proportion of true positive, especially with dynamic analysis [01] [02] [04] and unrepresentative dataset [16]. Usually, this combination aggregate results of two technics to make a signature or a program behavior synthesis. In the last years, we could see the emergence of hybrid models. This evolution is represented by the grown of hybrid analysis usage. Furthermore, our approach uses this development in a specific way: firstly, features are extracted and filtered to provide a targeting of analysis points to investigate them on the next step This filtering based on graph mining provides a good way to focus the dynamic analysis on the main nodes. During our process, features are extracted and stored on vector to make a classification of program in "benign" and "malware".

Hypothesis: It is possible to reduce the dynamic analysis scope with a filtering based on central node detection to produce a suitable detection of malware and benign program.

The evolution of using metrics and approaches provide a significant change on malware detection model [2] Models could improve their lifetime [17] and become more tolerant in front of new malware type. Now, we focus our attention on the last evolution of MDM as the rest of this study.

Overview

Method Analysis

To perform data extraction to classify a program in boolean statement (benign or malware), we could use two techniques: static analysis and dynamic analysis. Third technique exists but it is a combination of the two previous. Now, we introduce static analysis.

Static

This technique extracts data from the app package without running it. Firstly, the app package is a compressed file with a dedicated extension: ".apk". All assets of the app are present in this archive. To extract data, this technique decompresses the file and decompile the program to reach the targeted features [01][09][11]. This technique provides good accuracy [01] but presents major shortcoming especially when app obfuscated [09]. In this situation, a lot of data are not accessible even if multiple decompiler are used during the analysis. To avoid this limitation some improvements are made especially with semantic approach and graph analysis approach. With the last approach, the model could interpret the program structure from a function call graph or control flow graph [14][18][20]. This evolution constitutes a well improvement of MDM but the accuracy provided by this technique decreases in time. Another shortcoming of it is the huge set generated by this graph. For an average app size, between 30 to 50 Mo, the average size of these graph is respectively located between 5 and 10 Mo for CG [3]and 120Mo and 2,5 Go for CFG[4]. The amount of calc to analyze these graphs is significant. To avoid it, researchers focus their model on an extract of this graph, especially for the CFG. But results provided by this technique generate more false positives than other models and are not representative of the global program behavior. There are many libraries to perform static analysis on android package but in our approach, we work with "androguard[5]" python library. Transposed on android architecture,

[2] Also note MDM in this study

[3] Call graph for function call graph

[4] Control flow graph

[5] https://github.com/androguard/androguard

static analysis can have a lot of features but many of they could be corrupted or unrepresentative. That is the first argument to implement our vision in MDM.

Dynamic

This technique requires an environment (virtual or physical) to run a targeted application. During this execution, the analysis performs some hook to gather some data [11][12]. Here again, some types of data could be extracted. To use this technique, an analyst could install or inject a hook function on an app's package that focuses on a predefined target list.

Major shortcoming of this technique is the waste of time necessary to perform it [11]. Where a static analysis provides a result in a few seconds a dynamic analysis takes more time to reach a result. Another default resides in the fact this technique generates a high proportion of false positive. Despite these limitations, this technique is robust in front of an obfuscated app's because it focuses its resources on input, output and architectural symbols (like memory address) not on the obfuscated code. Here again, there are many libraries to perform dynamic analysis on an android package but in our approach, we work with "Frida[6]" python library. As we saw, we understand the need to implement the third solution in our approach. Now we introduce the last analysis technique.

Hybrid method

This technique consists of combining dynamic and static analysis on the same process [01][02][04][05]. We could use this technique with different approaches, usually researchers use a complementarity approach [01][02][04][05]. They run separately the two previous techniques and aggregate results on the same artifact (vector, matrix or anything else). This combination is quite complex because we gather a heterogeneous dataset (graph, vector…) and from multiple levels (system, classes, methods, environment…). We see the need to create a data structure able to combine all data into it. We could perform an abstraction to provide a representative and exploitable dataset.

Another approach consists of using static analysis to select targeted malicious points that will be analyzed by dynamic analysis on the next step. Our work uses this approach to limit waste of time generated by the dynamic analysis include in our process.

Malware classification

Analysis of malware is performed to gather some crucial information like permission usage, argument distribution to build some features with it. Usually, these features are mining by computing algorithms to classify or cluster all application analyzed by the model. To perform this, researchers use machine learning and/or deep learning algorithms. Machine learning algorithms learn and discover the structure of data from a dataset. These algorithms can be exploited in a supervised an unsupervised context. Researchers perform features extraction to build efficient model to provide the best result for the aim of their study. It exists a plenty of algorithms and the most used are AdaBoost, K-Means, Random Forest, Lib-SVM and DecisionTree. These algorithms showed some limitations like the amount of data can be supported with its.

On the other side, deep learning algorithms are also used to perform the same goal, but they don't require features selection. These algorithms mine the dataset to detect the best neural structure to classify the data. Moreover, they can support a dataset with a large size, and they improve their performance when the dataset is growing. These algorithms can be recurrent, artificial, or convolutional. However, these algorithms showed some limitations like the computing power needed to exploit them and the minimum size of dataset to produce a good accuracy.

[6] https://frida.re/

In our context, we use machine learning algorithms to classify numerical data because the size of our dataset is more suitable with machine learning algorithms. We introduce our approach in the next part of this paper. We try this set of algorithms to evaluate the performance provided by our model: SVM, RandomForest, AdaBoost and Naïve Bayes. Now, we analyze some recent hybrid MDM to discuss about results provided by these models.

Related Works

There are plenty of MDM, we focus our discussion on recent hybrid models. All of these models implement different approaches. For each model, we describe its process, its logical architecture and its goals. Hybrid MDM use one approach as follow:

Approach	Static analysis role	Possible output
Complementarity approach	Make results aggregation provided by the two methods	Score, signature, or decision

Marvin [01]

Designed by Lindofer and al, this model combines static and dynamic analysis. It performs the first type to establish a compromised score of an apps. This analysis is based on DroidMat dataset. Result is a signature. Then, it performs a dynamic analysis and aggregates all results to provide a global evaluation of the app. We should note this model use a complementarity approach. Aggregation provides a global evaluation score to send some advice to the users about app management (deploying or deleting). At the end, users make the final decision. This model has the following component blocks on its architecture:

- static analysis.
- dynamic analysis.
- database.
- parser and extractor for app package.
- area with specific scripts.
- parser and tracer for binaries in app package.

Nodaway, this architecture is usual for MDM. We find the use high level programming language like “Python” and ‘Javascript” for specific needs. However, the singularity of it is the employment of machine learning to perform the classification. This model provides global scoring based on probabilistic calc with a good accuracy level, but it isn’t resilient in front of obfuscated app.

Droid ranger [01]

This model performs an analysis on app previously downloaded on the Google Play Store. It uses downloading mechanism directly connected to the store and based on similar interface present on apkpure platform[7].The analysis scope of it could be large and adaptable. This can provide results so quickly with a good accuracy level. Another advantage of it is its interconnection with the main market store for android application mentioned previously. A specificity of it is that static analysis is not automated, just assisted. Machine learning techniques are not implemented on it.This configuration provides a high level of accuracy, but this gain is relative. Recently papers tended to reach a good accuracy level with a full automation of the process. We will highlight this in the next model's analysis. Droid Ranger don’t compare its results with a signature database or any dataset. It uses a heuristic method to analyze function calls. This method provides a good resilience to deal with new malware type

[7] https://apkpure.com

coming because it integrates an abstraction on its extracted data. We find this analysis technique on intrusion detection systems.

Finally, this model has a similar architecture than the Marvin model.

M0Droid [04]

Designed by Damshenas and al, this model uses static analysis in the first step to extract required permissions and notes obfuscation technique use. Furthermore, it analyses class structure and declared attributes for each class contained in the app package. In the next step, these results are classified and enriched by the dynamic analysis. This model uses a complementarity approach, and we note machine learning implementation to classify data during the dynamic analysis. Here again, injected inputs are generated by an automated process using "Monkey Runner". Also, the tracing of function calls is realized with "Trace" a usual tool present on some Linux distributions. A specificity of this model is the role of machine learning. Machine learning is also used to integrate a new malware signature, when it is emerging. This point will have several consequences for the lifetime model. Firstly, it could be resilient in front of new malware type. Finally, it could keep its accuracy level longer than other models. This improvement of hybrid models will integrate in our approach.

Droidmat [05]

Designed by Wu and al, this model uses static analysis in the first step to extract required permissions, natives' functions calls, and APIs' calls present on the app. Also, it extracts library imports and classes imports mentioned on the code. This allows the model to integrate a complete set of interactions on its scope. Furthermore, it analyses information presents in the app certificate (as signature, author) like the famous tool "Google Bouncer" to make a trust score of the app. In the next step, it uses dynamic analysis to trace system and function calls perform during the app execution. However, this model measures the energy consumption for each process directly linked with the app process. This metrics provides a relevant information because malware programs use several techniques (as logic bomb) requiring a high level of energy consumption. This data is registered with "taintdroid" tool to automate the process. This model has an architecture very similar to the M0Droid model.

Madam [23]

Designed by Saracino and al, this model makes a multilevel analysis (kernel, flow and apps level). Flow analysis isn't a usual method as we saw in the previous models. However, this metric combined with other features could provide a new perspective for the analysis. Furthermore, this model integrates energy consumption on its scope. This metric provides a good way to avoid obfuscation techniques present in the app. To dynamic analysis, this model extracts some usual features (memory area access, function and system calls...). These features will be correlated with the previous metrics. The static analysis extracts some metadata included in the app package (app certificate, manifest…) and other usual dataset (required permissions, class declaration…). This set of data will be aggregated with dynamic analysis results to classify the app behavior. The static analysis makes the first level of classification that will be correlated with dynamic analysis results. As we see, this model uses a complementarity approach.

Bride maid [02]

Designed by Delorenzo and al, this model uses a complementarity approach. In the first step it uses static analysis to extract some features. These results will be aggregate with results provided by the second step (dynamic analysis). Architecture model is like the Marvin model. However, this model uses a multilevel hooking (local and distant environment). On the local environment it listens to kernel and dalvik events. In the distant environment it listens to data transfer.

Synthesis

Wwe can conclude the major approach present in these models is the complementarity approach. Complementarity approach provides a good resilience in front of obfuscated app but the lifetime model is a shortcoming. Another problem is the waste of time required to perform the dynamic analysis during the process. This problem is well known and today we could have two strategies to avoid it. The first consists of paying attention to a few samples of dynamic interactions. The lack of this way resides in the risk of exploiting an unrepresentative dataset for the classification. The second consists of determining the main suspect target without extracting a huge proportion of dynamic interaction. We think the second solution is a good way to build an MDM. in the next part we introduce our approach to build a malware detection model exploiting a representative dataset of app behavior and generalizable in any execution context.

In our work, we define a new approach call “Exploration approach”. This approach is similar to “Complementarity approach” but results provided in static analysis is stored on vector and also used to make a filtering to reduce the dynamic analysis on the most important point. We can define our approach as follow:

Approach	Static analysis role	Possible output
Exploration approach	Make a targeting to perform local dynamic analysis	Signature (with an enrichment mechanism or not) deep analysis, decision

Methodology & result

Then, in this part we expose the functioning of our model. Firstly, we focus our attention on the model architecture. Our model is composed by two area: extraction area and classification area. In the first, our model receives an android application to perform data gathering datas. This area includes the following logical process:

1. Firstly, a static analysis, driving by python script using “androguard” library, gather some datas to build a staticvector and the callgraph of the app. Static vector is defined in tab 1 ;
2. Secondly, a filtering is performing on the callgraph to extract “critical nodes”. These nodes have a high proximity with other call include the graph. These nodes have the following properties:
    - C1: High degree of centrality and proximity (1000 first high score nodes).
    - C2: Theses nodes manage a minimum of two arguments.
    - C3: Theses nodes are not Google addmanaging or credential API.
    - C4: Theses nodes have indegree and out degree included in the 0.95 quantile for the twice degree.

    This filtering is performing by the algorithm present in the appendix (1).
3. Thirdly, a dynamic analysis is performing on a virtual machine based under “Android x86” versionto run the mobile app. During the app execution a data extraction is performed to extract some information on the mother class concerned by all nodes targeted in the previous step. This extraction is driving by python script and Frida library. Also, we found a set of hooking function dynamically generated to perform the extraction. At the end the dynamic vector is create. Dynamic vector is defined in tab 1.
4. Finally, the static and dynamic vector are aggregate to form the final vector of the application. This aggregation is driving by python script.

All data extracted from the app is include in the following tab (1). Then the extraction of data is achieved, the model uses the classification module. This area uses three instances: the final app vector built on the previous area, a dataset and a neural network classification model. This area is driving by python scripts and a set of libraries (sklearn, pandas, numpy…). Then we introduce the processing of this area and after we describe the fine processing of dataset and classification model. Our model instantiates the neural network model and train his model with the dataset. Then, it analyzes the final vector app to predict the type of the application (benignware or malware).

Tab 1: Extracted Data and vector composition

App Data	Vector index and type of data	Model step	Vector
Oppacification	Encrypting and Renaiming : boolean data	Static analysis	Static
Local components interaction	NbPermissions : total number of used permissions NbPermissionsDangerous : total number of dangerous used permissions NbPermissionsSignature : total number of used permissions needed an user signal NbandroidAPI : Android API number used in the app	Static analysis	Static
Data storage interaction	Nbprovider : number of providers include in the app Nbreceiver : number of receivers include in the app	Static analysis	Static
Outside environnement interaction	Nbservices : number of services include in the app	Static analysis	Static
Local artefacts	stringNBGlobal : total number of strings (sum of all following type) UniCodeNb : number of URI strings UrlNb : number of URL strings AffectNb : number of variable instantiation BaliseNb : number of tag strings SpecialNb : number of special character strings CallNb : number of calls ClasseNb : number of classes ChaineNb : number of strings not corresponding on previous types excluding stringNBGlobal	Static analysis	Static
Argument distribution	AD_SYSTEM : number of arguments provided by OS or the language Java. This metrics is extracted for all classes targeted in the filtering step. AD_CUSTOM : number of custom arguments defined by the programmer. This metrics is extracted for all classes targeted in the filtering step.	Dynamic analysis	Dynamic vector
Library Call nature	NLC_INTERN : Percentage of call custom. This metrics is extracted for all classes targeted in the filtering step. NLC_EXTERN : Percentage of call system. This metrics is extracted for all classes targeted in the filtering step.	Dynamic analysis	Dynamic vector
Average of size instances	ASI : Average of instances size for each classes. This metrics is extracted for all classes targeted in the filtering step.	Dynamic analysis	Dynamic vector
Dynamic loaded origins	DLCOI : number of dex uses in the app to store the app classes.	Dynamic analysis	Dynamic vector
Number of arguments type	NBTYPE : Total number of arguments types in the app.	Dynamic analysis	Dynamic vector
WMC	Ratio of number method include in class for each class	Dynamic analysis	Dynamic vector
RFC	Average of intern and extern methods call in each class. This metrics is extracted for all classes targeted in the filtering step.	Dynamic analysis	Dynamic vector
NOC	Average of subclass in each class. This metrics is extracted for all classes targeted in the filtering step.	Dynamic analysis	Dynamic vector

We describe the global functioning of our model, then we define two crucial components of the classification area. Now, we study the dataset used in this module. Our dataset was created with a set of malware sample provided by the “VirusShare” organization and a random list of benign applications provided by “Apkpure”. Malwares samples was extracted from the period 2012 to 2018 of archived present on the “VirusShare” website. To qualify these samples, we send it by API services of “VirusTotal” company to determine the most probable type of the application. Then the result is receiving, at xml format, all detection results are consigned in a global database and an frequency calculation is performing for all record. For each app, 72 detection results are provided by the analysis engines

integrate in the “VirusTotal” platform. The real type for a record is the detection result with the highest frequency.

After, all types are determined the process of our model, previously exposed, is performing an all apps include in our dataset. We can find the dataset population composition on the tab (2). As we can see in the next part, our dataset construction is quite particular because our model has interdependent steps. For each step, the result of the previous step affects the functioning of the following step. For the two first step included in the extraction area, the probability to perform the analysis is good, near 92% of android app can be analyzed. But it is not the same for the dynamic analysis, especially with the malware samples. These present a lot of way to perform evading processing and our model need some adjustment to deal with them. After a lot of integration, our model has a probability to perform the dynamical analysis around 62%. That involved, we use a large population of samples malwares to construct our actual dataset population.

Tab 2: Dataset population

Benign	Rootkit	Trojan	Spyware	Botnet	Ransomware	Total
461	62	289	89	56	115	1072
43.00 %	5,78 %	26,96 %	8,30 %	5,22 %	10,73 %	

Our dataset is used with machine learning algorithms in our classification model. A set of usual algorithms are used to test our model detection performance. We discuss more about this on the result part (RandomForest, LibSVM, AdaBoost and NaivesBayes). Now, we describe the results obtained after applying our model’s process. Firstly, our model is evaluated with the following measures: Accuracy, F1_Score, Recall and Precision. All of these features are calculated as follow:

$Accuracy = \frac{TP + TN}{Total}$	$Recall = \frac{TP}{TP + FN}$
$F1\ Score = \frac{2}{\frac{1}{Precision} + \frac{1}{Recall}}$	$Precision = \frac{TP}{TP + FP}$

TP, TN, FP and FN mean respectively the true positive, true negative, false positive and false negative results provided by the classification process. These measures are used because the aim of our model is to produce a classification and they are usual to evaluate the model’s performance on our context.

Tab 3: Model performance

Algorithms	*Accuracy*	*F1_Score*	*Recall*	*Precision*
Random Forest	0.984	0.955	0.955	0.955
SVM	0.979	0.945	0.955	0.955
AdaBoost	0.930	0.935	0.935	0.935
Naives Bayes	0.966	0.908	0.908	0.908

As we see in the tab (3), our model provides a precision near 0.984% percent with Random Forest. This result is obtained in a k-folds execution of our model with a k equal to 10. The precision is very similar with other hybrid models and that show we can obtain a good level of malware detection with features representative of the global app behavior in a hybrid model. Like in our model, we use filtering to limit the scope of the dynamic analysis the result of this filtering shows the behavior of the central calls included in the application.

Results validate our process and our hybrid model with a classification based on boolean statement. The filtering produced in the second step of our process provides a good targeting illustrated by the accuracy provided in the detection. Also, our main hypothesis is validated.

Conclusion

With the implementation of our filtering process, we can obtain a representative view of central nodes included in the android application. As we saw, this process provides a respectable result for malware detection, based on simple boolean statement (benign or malware). On the other side our model needs a heavy and restrictive preprocessing to build our dataset. That induces a waste of time and malware samples to perform our analysis before able to classify the result.

To avoid this limitation, we have multiple opportunities. Firstly, and the easiest, we improve the size of our dataset to increase the precision of our model. According to this idea a floor of 10000 records will be reached for the size of our dataset. Now the data extraction of our model is robust we can apply with an external dataset. To do that, we will use CiCMalDroid dataset in our next work. Secondly, to improve the dynamic analysis efficiency we can implement more circumvention technique to deal with evasion technique implemented in malwares. Finally, we should investigate more on nodes filtering to integrate in our dataset some measures from these extracted nodes. It is possible these data could provide a highlight on their behavior.

Appendix

Appendix 1 : Filtering Algorithm – Pseudo-Code

```
Vars :
        Integer n1, n2, NbArguments, SizeNode
        Float AVGDegreeProximity, AVGDegreeCentrality, DegreeCentrality, DegreeProximity, InDegree, OutDegree,
        ProximityQuantile, CentralityQuantile, TabDegreeProximity[n2], TabDegreeCentrality[n2], x
        Character Methods, Nodes, TypeNode, MethodsReg
MatrixFiltered[n1][9] FilteringNode(MatrixBrut[n2][9])
Begin
        Integer i
        For i = 1 to n2 Do
        TabDegreeCentrality[i] <- MatrixBrut[3][i]
        TabDegreeProximity [i] <- MatrixBrut[6][i]
End

x = 0,95
ProximityQuantile <- Quantile(x, TabDegreeProximity)
CentralityQuantile <- Quantile(x, TabDegreeCentrality)

Begin
        Integer i
        For i = 1 To n2 Do

                Methods <- MatrixBrut [2][i]
                Nodes <- MatrixBrut [1][i]
                NbArguments <- MatrixBrut [9][i]
                SizeNode <- MatrixBrut [8][i]
                DegreeProximity <- MatrixBrut [6][i]
                DegreeCentrality <- MatrixBrut [3][i]
                OutDegree <- MatrixBrut [4][i]
                InDegree <- MatrixBrut [5][i]
                TypeNode<- MatrixBrut [7][i]
                If DegreeProximity > ProximityQuantile and DegreeCentrality > CentralityQuantile and Methods contains
                MethodsReg Then
```

MatrixFiltered (i) <- [Nodes, Methods, DegreeCentrality, OutDegree, InDegree, DegreeProximity, TypeNode, SizeNode, NbArguments]

End If

End For

End

Reference

[01] LINDORFER, Martina, NEUGSCHWANDTNER, Matthias, et PLATZER,Christian. Marvin: Efficient and comprehensive mobile app classification throughstatic and dynamic analysis. In: 2015 IEEE 39th annual computer software andapplications conference. IEEE, 2015. p. 422-433.

[02] DE LORENZO, Andrea, MARTINELLI, Fabio, MEDVET, Eric, et al.Visualizing the outcome of dynamic analysis of Android malware with VizMal.Journal of Information Security and Applications, 2020, vol. 50, p. 102423.

[03] https://github.com/Dam4323/EGDataset

[04] DAMSHENAS, Mohsen, DEHGHANTANHA, Ali, CHOO, Kim-KwangRaymond, et al-. M0droid: An android behavioral-based malware detectionmodel. Journal of Information Privacy and Security, 2015, vol. 11, no 3, p.141-157.

[05] WU, Dong-Jie, MAO, Ching-Hao, WEI, Te-En, et al. Droidmat: An-droid malware detection through manifest and api calls tracing. In: 2012 Sev-enth Asia Joint Conference on Information Security. IEEE, 2012. p. 62-69.

[06] KARBAB, ElMouatez Billah, DEBBABI, Mourad, DERHAB, Abde-louahid, et al. MalDozer: Automatic framework for android malware detectionusing deep learning. Digital Investigation, 2018, vol. 24, p. S48-S59.

[07] SUAREZ-TANGIL, Guillermo, DASH, Santanu Kumar, AHMADI, Man-sour, et al. Droidsieve: Fast and accurate classification of obfuscated androidmalware. In: Proceedings of the Seventh ACM on Conference on Data andApplication Security and Privacy. 2017. p. 309-320.

[08] AONZO, Simone, GEORGIU, Gabriel Claudiu, VERDERAME, Luca,et al. Obfuscapk: An open-source black-box obfuscation tool for Android apps.SoftwareX, 2020, vol. 11, p. 100403.

[08] YERIMA, Suleiman Y., SEZER, Sakir, MCWILLIAMS, Gavin, et al.A new android malware detection approach using bayesian classification. In:2013 IEEE 27th international conference on advanced information networkingand applications (AINA). IEEE, 2013. p. 121-128.

[09] SUAREZ-TANGIL, Guillermo, DASH, Santanu Kumar, AHMADI, Man-sour, et al. Droidsieve: Fast and accurate classification of obfuscated androidmalware. In: Proceedings of the Seventh ACM on Conference on Data andApplication Security and Privacy. 2017. p. 309-32022

[10] ALZAYLAEE, Mohammed K., YERIMA, Suleiman Y., et SEZER,Sakir. DynaLog: An automated dynamic analysis framework for characterizingandroid applications. In: 2016 International Conference On Cyber Security AndProtection Of Digital Services (Cyber Security). IEEE, 2016. p. 1-8.

[11] SCHMIDT, A.-D., BYE, Rainer, SCHMIDT, H.-G., et al. Static anal-ysis of executables for collaborative malware detection on Android. In: 2009IEEE International Conference on Communications. IEEE, 2009. p. 1-5.

[12] KANG, BooJoong, YERIMA, Suleiman Y., MCLAUGHLIN, Kieran, etal. N-opcode analysis for android malware classification and categorization. In:2016 International conference on cyber security and protection of digital services(cyber security). IEEE, 2016. p. 1-7.

[13] SHABTAI, Asaf, TENENBOIM-CHEKINA, Lena, MIMRAN, Dudu, etal. Mobile malware detection through analysis of deviations in application net-work behavior. Computers Security, 2014, vol. 43, p. 1-18.

[14] JANG, Jae-wook, WOO, Jiyoung, YUN, Jaesung, et al. Mal-netminer:malware classification based on social network analysis of call graph. In: Pro-ceedings of the 23rd International Conference on World Wide Web. 2014. p.731-734.

[15] FENG, Yu, ANAND, Saswat, DILLIG, Isil, et al. Apposcopy: Semantics-based detection of android malware through static analysis. In:

Proceedings ofthe 22nd ACM SIGSOFT international symposium on foundations of softwareengineering. 2014. p. 576-587.

[16] ALSMADI, Tibra et ALQUDAH, Nour. A Survey on malware detec-tion techniques. In: 2021 International Conference on Information Technology(ICIT). IEEE, 2021. p. 371-376.

[17] WU, Yueming, LI, Xiaodi, ZOU, Deqing, et al. MalScan: Fast market-wide mobile malware scanning by social-network centrality analysis. In: 201934th IEEE/ACM International Conference on Automated Software Engineering(ASE). IEEE, 2019. p. 139-150.

[18] YANG, Yang, DU, Xuehui, YANG, Zhi, et al. Android Malware De-tection Based on Structural Features of the Function Call Graph. Electronics,2021, vol. 10, no 2, p. 186.

[19] KINABLE, Joris et KOSTAKIS, Orestis. Malware classification basedon call graph clustering. Journal in computer virology, 2011, vol. 7, no 4, p.233-245.23

[20] XU, Ming, WU, Lingfei, QI, Shuhui, et al. A similarity metric methodof obfuscated malware using function-call graph. Journal of Computer Virologyand Hacking Techniques, 2013, vol. 9, no 1, p. 35-47.

[21] YERIMA, Suleiman Y., SEZER, Sakir, MCWILLIAMS, Gavin, et al.A new android malware detection approach using bayesian classification. In:2013 IEEE 27th international conference on advanced information networkingand applications (AINA). IEEE, 2013. p. 121-128.

[22] RASCAGNERES Paul. Securite informatique et malwares: analyse desmenaces et mise en œuvre des contre-mesures. St Herblain : Editions ENI, 2019.

[23] SARACINO, Andrea, SGANDURRA, Daniele, DINI, Gianluca, et al.Madam: Effective and efficient behavior-based android malware detection andprevention. IEEE Transactions on Dependable and Secure Computing, 2016,vol. 15, no 1, p. 83-97