Characterization of Android Malware Families by a Reduced Set of Static Features

Javier Sedano[1], Camelia Chira[2], Silvia González[1], Álvaro Herrero[3], Emilio Corchado[4], and José Ramón Villar[5]

[1]Instituto Tecnológico de Castilla y León
C/ López Bravo 70, Pol. Ind. Villalonquejar, 09001, Burgos, Spain
{javier.sedano,silvia.gonzalez}@itcl.es

[2]Department of Computer Science, University of Cluj-Napoca
Baritiu 26-28, Cluj-Napoca 400027, Romania
camelia.chira@cs.utcluj.ro

[3]Department of Civil Engineering, University of Burgos, University of Burgos
Avenida de Cantabria s/n, 09006, Burgos, Spain
ahcosio@ubu.es

[4]Department of Computer Science and Automation, University of Salamanca
Plaza de la Merced, s/n, 37008, Salamanca, Spain
escorchado@usal.es

[5]Computer Science Department, University of Oviedo, ETSIMO
33005, Oviedo, Spain
villarjose@uniovi.es

Abstract. Due to the ever increasing amount and severity of attacks aimed at compromising smartphones in general, and Android devices in particular, much effort have been devoted in recent years to deal with such incidents. However, accurate detection of bad-intentioned Android apps still is an open challenge. As a follow-up step in an ongoing research, preset paper explores the selection of features for the characterization of Android-malware families. The idea is to select those features that are most relevant for characterizing malware families. In order to do that, an evolutionary algorithm is proposed to perform feature selection on the Drebin dataset, attaining interesting results on the most informative features for the characterization of representative families of existing Android malware.

Keywords: Feature Selection, Genetic Algorithm, Android, Malware families

1 Introduction

Bad-intentioned people are taking advantage of the open nature of Android operating system to exploit its vulnerabilities. It is one of the main targets of mobile-malware creators because it is the most widely used mobile operating system [1]. The number of apps available at Android's official store has increased constantly from the very beginning, up to more than 2.1 million [2] that are available nowadays. With regard to the security issue, Android became the top mobile malware platform as well [3] and it is forecast that the volume of Android malware will spike to 20 million during 2016when it was 4.26million at the end of 2014 and 7.10 million in first half of 2015 [4]. This operating system is an appealing target for bad-intentioned people, reaching unexpected heights, as there are cases where PC malware is now being transfigured as Android malware [5].

To fight against such a problem, it is required to understand the malware and its nature, given that this nature is constantly evolving as it happens with most software. Without understanding the malware, it will not be possible to practically develop an effective solution [6]. Thus, present study is focused on the characterization of Android malware families, trying to reduce the amount of app features needed to distinguish among all of them. To do so, a real-life benchmark dataset [7], [8] has been analyzed by means of several feature selection strategies.

To more easily identify the malware family an app belongs to, authors address this feature selection problem using a genetic algorithm guided by information theory measures. Each individual encodes the subset of selected features using the binary representation. The evolutionary search process is guided by crossover and mutation operators specific to the binary encoding and a fitness function that evaluates the quality of the encoded feature subset. In the current study, this fitness function is defined as the mutual information.

Feature selection methods are normally used to reduce the number of features considered in a classification task by removing irrelevant or noisy features [9], [10]. Filter methods perform feature selection independently from the learning algorithm while wrapper models embed classifiers in the search model [11], [12].Filter methods select features based on some measures that determine their relevance to the target class without any correlation to a learning method.

There are many advantages of feature selection for malware detection; however, little effort has been devoted until now to apply these methods of machine learning to deal with malware features [13]. In [14] just information gain is used to rank the 32 static and dynamic features from a self-generated malware dataset containing 14,794 instances, comprising 30 legitimate apps. Samples of malware come from five different families (GoldDream, PJApps, DroidKungFu2, Snake and Angry Birds Rio Unlocker). To rank the features, four machine learning classifiers (Naïve Bayes, RandomForest, Logistic Regression, and Support Vector Machine) were applied. The top 10 selected features were (in decreasing order of importance): Native_size, Native_shared, Other_shared, Vmpeak, Vmlib, Dalvik_RSS, Rxbytes, VmData, Send_SMS, and CPU_Usage. In a different work [15], 88 dynamic features from 43 apps were collected and then analysed to discriminate between games and tools. The underlying idea of this study was that distinguishing between games and tools would provide a positive indication about the ability of detection algorithms to learn and model the behavior applications and potentially detect malware. To do so, feature selection was applied to identify the 10, 20 and 50 best features, according to Information Gain, Chi Square, and Fisher Score. A similar analysis [16] by same authors proposed a selection from 22,000 static features about 2,285 apps to distinguish between games and tools apps once again. The following classifiers were applied: Decision Tree, Naïve Bayes, Bayesian Networks, PART, Boosted Bayesian Networks, Boosted Decision Tree, Random Forest, and Voting Feature Intervals. The obtained results shown that the combination of Boosted Bayesian Networks and the top 800 features selected using Information Gain yield an accuracy level of 0.918 with a False Positive Rate of 0.172.

Although MRMR has also been previously applied to the detection of malware [17], present study differentiates from previous work as feature selection is now applied from a new perspective, trying to characterize the different Android malware families, to gain deeper knowledge of malware nature. Additionally, to the best of the authors knowledge, this is the very first proposal applying feature selection to an up-to-date and large Android malware dataset in general terms and the Drebin one, more precisely.

More recently, static analysis of Android malware families was already proposed in [18], trying to identify the malware family of malicious apps. The main difference when compared to present work is that family identification relied on apps payload. That is, authors analyzed the Java Bytecode produced when the source code of apps is compiled. It was analyzed through formal methods, being the system behaviour represented as an automaton. With regard to authors previous work [19], an improved evolutionary algorithm for feature selection is now applied, and a more comprehensive and recent dataset is considered in present paper.

The rest of this paper is organized as follows: the proposed evolutionary feature selection algorithm is described in section 2, the experiments for the Drebin dataset are presented in section 3, the results obtained are discussed in section 4 and the conclusions of the study are drawn in section 5.

2 Feature Selection

The big amount of features in the analysed dataset (see Section 3), the various feature subsets that may be defined can be extensively evaluated using different methods. The result of these methods can then be aggregated in a ranking scheme. It is proposed to determine an ordered list of selected features using a genetic algorithm based on mutual information as fitness function. The methods described in this section assume a matrix X of N feature values in M samples and an output value y for each sample.

The proposed Genetic Algorithm (GA) encodes in each individual the feature selection by using a binary representation of features. The size of each individual equals the number of features and the value of each position can be 0 or 1, where 1 means that the corresponding feature is selected (the number of features is N). It is proposed to evaluate feature selection results using the mutual information (I).

Defined by means of their probability distribution, the mutual information between two variables has a high value for higher degrees of relevance between the two features. Let $I(X,Y)$be the mutual information between two features, given by:

$$I(X,Y) = \iint p(x,y) * \log\left(\frac{p(x,y)}{p(x)*p(y)}\right) dxdy \quad (1)$$

The genetic algorithm resulting by using $I(X,Y)$ as the fitness function, named GA-I, is outlined below. N G and t denote the population size, the maximum number of generations and the current generation, respectively.

Algorithm: GA-I Feature Selection

```
Require: X the input variables data set
Require: Y the output vector
P ← a vector of N Individual objects
t ← 0
Generate the initial population P(t): randomly initialize the value of each individual
while t <G do
        Evaluate each individual IND in P(t): calculate I(IND, Y) value
        P(t +1) ← roulette wheel selection from P(t)
        for all individuals IND in P(t + 1) do
                Select mate J from P(t + 1)
                K ←two-point crossover (IND, J)
                if fitness(K) > fitness(IND) then
                        IND ← K
                end if
                L ← mutation(IND)
                if fitness(L) > fitness(IND) then
                        IND ← L
                end if
        end for
        t ← t+1
end while
Return Best Individual in P(t)
```

The GA follows a standard scheme in which roulette wheel selection, two-point crossover and swap mutation are used to guide the search. Each individual is evaluated based on the correlation between the current subset of selected features and the output, given by I.

Furthermore, a second variant of the GA-I algorithm (called *GA-I-W*, where *W* stands for *weighted*) is proposed in order to control the number of features selected in an individual. In order to do that, the fitness function of *GA-I-W* is based on a weighted scheme between the information theory measure and the number of selected features.

Let *k(x)* be the size of the feature subset encoded in an individual *x* and *w* a real parameter between 0 and 1, denoting the weight of each fitness component. The weighted fitness function for an individual *x* is depicted in Eq. 2.

$$f(x) = w \cdot I(x) + (1-w) \cdot 1/k(x) \quad (2)$$

The maximization of *f* would also lead to a minimum number of possible selected features in the individual. It should be noted that the features would only be selected as long as a high value of *I(x)* still emerges in the current individual. This balance is ensured by the value of the weight parameter *w*. A 0.5 value *w* gives the same relevance to both measurements, while higher values of *w* can be used to give a relative higher importance to the information theory measure value compared to the size of the feature subset.

3 Experimental Study

As previously explained, several approaches for features selection have been applied to the characterization of Android malware. The analysed dataset is described in sub-section 3.1 and the obtained results are introduced and described in sub-section 3.2.

3.1 Drebin Dataset

The Drebin dataset [7],[8] is a collection of Android apps gathered from the Android official market (Google Play) and from some other un-official sources (alternative markets, websites, forums…) between 2010 and 2012. The gathered apps were analysed through the VirusTotal [20] service, being declared as malicious when more than one of the applied scanners identified the app as an anomalous one. As a result, the dataset contains123,441 benign applications and 5,554 malicious applications (128,995 in total), being one of the largest publicly-available datasets containing legitimate and malicious Android apps. Aps from 179 different families were collected.

Data were extracted from the manifest and the disassembled dex code of the apps, obtained by a linear sweep over the application's content [8]. Every sample in the dataset is associated to an analysed app and the values of the sample represent the given values of app characteristics, such as permissions, intents and API calls.

The following feature sets were extracted from the manifest file of every app [8]:

- Hardware components: contains information about the hardware components requested by the app.
- Requested permissions: contains information about the permission system, the main security mechanism of Android. Permissions declared by the app, and hence requested before installation, are taking into account.
- App components: contains information about the different types of components in the app, each defining different interfaces to the system.
- Filtered intents: contains information about intents (passive data structures exchanged as asynchronous messages for inter-process and intra-process communication).

Additionally, some other feature sets were extracted from the dex information extracted from the apk files of the apps [8]:

- Restricted API calls: contains information about the calls defined in the app to those APIs defined as critical. Although that information must be declared in the manifest file, exactly for being malware, some APIs may be accessed without declaring that in the manifest file (root exploits) and hence the information is double checked with the API calls from the dex code.

- Used permissions: contains information about the permissions that must be granted for the calls identified in previous feature subset. It is once again a way of double-checking the manifest file; the permissions in this case.
- Suspicious API calls: contains information about calls defined in the app to those APIs identified by the authors of the dataset as potentially dangerous. It includes calls for accessing sensitive data, communicating over the network, sending and receiving SMS messages, execution of external commands, and obfuscation.
- Network addresses: contains information about IP addresses, hostnames and URLs found in the dex code.

The previously defined features sets resulted in an initial set of features for the analysed apps. Each one of these features takes binary values: 0 if the app does not contain such feature and 1 otherwise. To aggregate this information at a family level, feature data were summarized for each family, taking binary values as well: 1 if any app from the family does contain such feature and 0 otherwise.

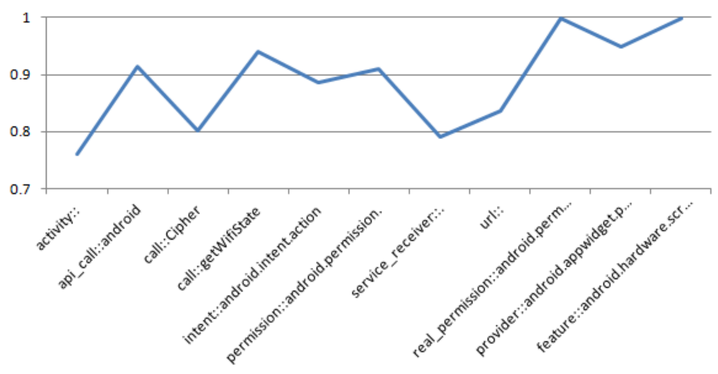
3.2 Results

The previously described GA-I and GA-I-W algorithms (see Section 2) have been applied for the selection of features from the Drebin dataset (see Section 3.1). For each one of the algorithms, three sets of experiments have been carried out, with the maximum number of features to be selected taking values of 10, 30 and 50. Two intermediate values of w (0.5 and 0.7) have been selected for comparison purposed. On the other hand, the following values were setting for the parameters of the GA:

- Population size: 100.
- Number of generations: 100.
- Number of runs for the algorithm for each experiment: 50.

The number of features selected by the two different algorithms is 188. As it is too high taking into account the aim of present paper, in order to reduce it, selected features are grouped according to their similar properties. As a result, the following 11 different groups of selected features were generated: activity::, api_call::android/, call::Cipher, call::getWifiState, intent::android.intent.action, permission::android.permission, service_receiver::, url::, real_permission::android.permission, provider::android.appwidget.provider, and feature::android.hardware.screen.landscape. Average values of I() for each one of these groups of features, when running the GA, are shown in Fig. 1. The Y axis shows the values of I and the X axis represents the associated group of features.

Fig. 1. Average values of *I()* for each group of features.



To summarize results of the 50 runs for each experiment, percentages were calculated for each group of features and algorithm. In the case of GA-I, percentages are shown in Table 1 while percentages of GA-I-W are shown in Table 2. Additionally, general results are shown in Fig. 2.

Table 1. Percentage of runs in which groups of features are selected by the GA-I algorithm.

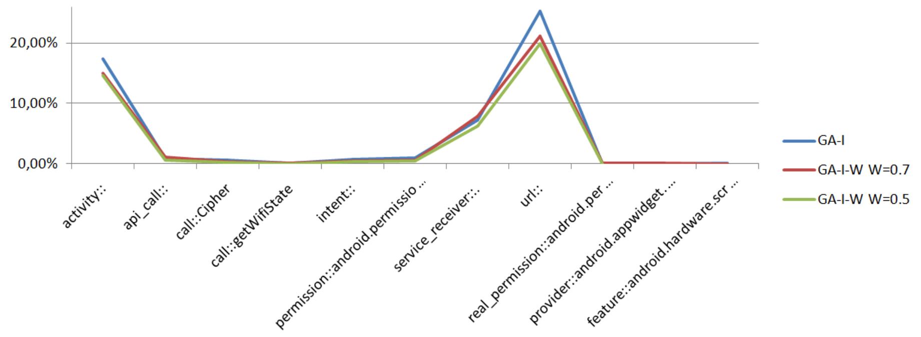
Feature	10 Features max	30 Features max	50 Features max
activity::	3,33%	5,26%	8,81%
api_call::android/	0,16%	0,00%	0,64%
call::Cipher	0,21%	0,21%	0,21%
call::getWifiState	0,00%	0,00%	0,05%
intent::android.intent.action	0,00%	0,21%	0,48%
permission::android.permission.	0,11%	0,21%	0,64%
service_receiver::.	1,29%	2,42%	3,60%
url::	3,97%	7,63%	13,64%
real_permission::android.permission.	0,00%	0,00%	0,00%
provider::android.appwidget.provider	0,00%	0,00%	0,00%
feature::android.hardware.screen.landscape	0,05%	0,00%	0,00%

Table 2. Percentage of runs in which groups of features are selected by the GA-I-W algorithm.

	10 Features max		30 Features max		50 Features max	
Feature	**W=0.7**	**W=0.5**	**W=0.7**	**W=0.5**	**W=0.7**	**W=0.5**
activity::	2,58%	3.23%	4,30%	3.63%	8,16%	7.78%
api_call::android/	0,32%	0.17%	0,11%	0.06%	0,64%	0.40%
call::Cipher	0,00%	0.11%	0,11%	0.00%	0,27%	0.11%
call::getWifiState	0,00%	0.00%	0,05%	0.00%	0,00%	0.00%
intent::android.intent.action	0,05%	0.11%	0,05%	0.06%	0,32%	0.23%
permission::android.permission.	0,11%	0.06%	0,05%	0.11%	0,43%	0.34%

service_receiver::.	1,13%	0.91%	2,26%	1.87%	4,46%	3.46%
url::	4,24%	4.20%	4,67%	4.26%	12,30%	11.46%
real_permission::android.permission.	0,00%	0.00%	0,00%	0.00%	0,11%	0.00%
provider::android.appwidget.provider	0,05%	0.00%	0,00%	0.00%	0,05%	0.00%
feature::android.hardware.screen.landscape	0,00%	0.00%	0,00%	0.00%	0,00%	0.00%

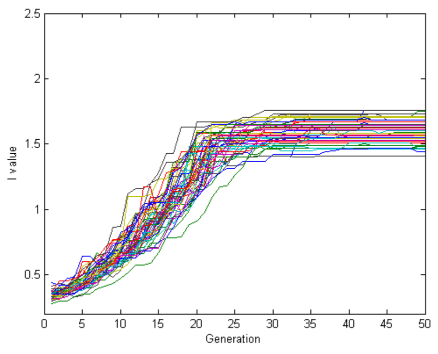
Fig. 2. Percentage of runs in which the group of features are selected by GA-I and GA-I-W algorithms. General results calculated from all the experiments.



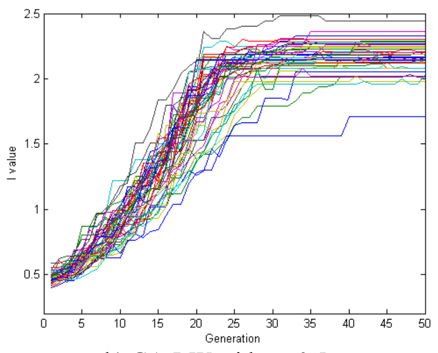
As it can be seen from the results shown in tables above, group of features url::, activity:: and service_receiver:: take the highest values of selection by the two algorithms. In general terms, it can be concluded that those groups of features are the most relevant ones for the characterization of Android malware families, according to the results from GA-I and GA-I-W.As url:: gets the highest scores, it means that looking into the URLs that are present in the disassembled core is critical in order to identify the family a malware belongs to. Malicious apps usually establish network connections to retrieve commands or exfiltrate data collected from the device. In a decreasing order of importance, the second group of features is activity::. It means that this kind of app components is the most informative ones. As service components are also important, it can be said that app components are very informative in order to discriminate between malware families.

11 types of features are present in the dataset: activity, api_call, call, feature, intent, network, permission, provider, real_permission, service_receiver, and url. Only one of them (network) is not present (even at a reduced percentage) in the features selected by any of the algorithms. As it can be seen, the group of features that have been selected in most executions are the same for the GA-I and GA-I-W with values of w equal to 0.5 and 0.7.

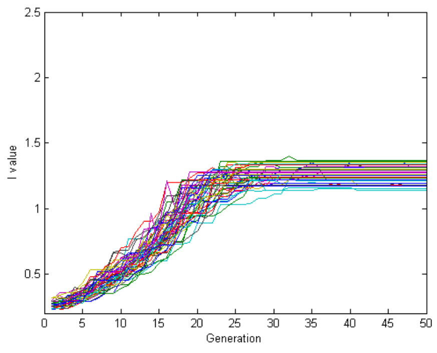
Fig. 3. Fitness evolution for *I()* values with 30 features limit.



a) GA-I (w=1)



b) GA-I-W with w=0.5



c) GA-I-W with w=0.7

It should be noted that the GA methods were able to reach the optimum values in the population during the second half of the search process – around generation 30 (see Fig. 3). Each line represents a run of the algorithm, some lines overlap in some executions that were similar - and that is why not every single line of the 50 total lines can be identified. This is due to the low feature quantity limits, which were set to 10, 30, and 50, and led to a size easier to handle (notwithstanding the actual size of the dataset) and enabled to quickly explore many feature subsets.

4 Conclusions and Future Work

By applying the proposed algorithms of feature selection, Android malware families are characterized. Thanks to such characterization, key features to distinguish from one malware family to the other ones are identified. This is a great contribution for malware detection tools as it is not only important to detect every single intrusion but also to know the family to run appropriate countermeasures.

Experimental results show that the two applied algorithms for feature selection agree on the selection of the 3 main groups of features. By large, url:: and activity::

are identified as the most important features for characterizing malware families as they get the highest percentages in both GA-I and GA-I-W. Feature dealing with URLs in the app code is the most important one to identify the family of a malware sample; it means that it is more important to take into account the external information (where the app is connecting to) rather than the characteristics of the app itself. Consequently, focusing on this piece of information could optimize the analysis of malware. Additionally, from the internal information, activities is the most relevant feature by large.

Future work will focus on proposing further adaptations of feature selection algorithms to ease the characterization of android malware.

Acknowledgments

This research has been partially supported through the project of the Spanish Ministry of Economy and Competitiveness RTC-2014-3059-4. The authors would also like to thank the BIO/BU09/14 and the Spanish Ministry of Science and Innovation PID 560300-2009-11.

References

1. Statista - The Statistics Portal, http://www.statista.com/statistics/266219/global-smartphone-sales-since-1st-quarter-2009-by-operating-system/ (Last accessed: 2016-07-08)
2. AppBrain Stats, http://www.appbrain.com/stats/stats-index (Last accessed: 2016-07-08)
3. Micro, T.: The Fine Line: 2016 Trend Micro Security Predictions. (2015)
4. Mind the (Security) Gaps: The 1H 2015 Mobile Threat Landscape, http://www.trendmicro.com/vinfo/us/security/news/mobile-safety/mind-the-security-gaps-1h-2015-mobile-threat-landscape (Last accessed: 2016-07-08)
5. F-Secure: Q1 2014 Mobile Threat Report. (2015)
6. Yajin, Z., Xuxian, J.: Dissecting Android Malware: Characterization and Evolution. 2012 IEEE Symposium on Security and Privacy (2012) 95-109
7. Spreitzenbarth, M., Echtler, F., Schreck, T., Freling, F.C., Hoffmann, J.: Mobile-Sandbox: Having a Deeper Look into Android Applications. 28th International ACM Symposium on Applied Computing (SAC) (2013)
8. Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., Rieck, K.: DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket. 21st Annual Network and Distributed System Security Symposium (2014)
9. Guyon, I., Elisseeff, A.: An Introduction to Variable and Feature Selection. The Journal of Machine Learning Research 3 (2003) 1157-1182
10. Larrañaga, P., Calvo, B., Santana, R., Bielza, C., Galdiano, J., Inza, I., Lozano, J.A., Armañanzas, R., Santafé, G., Pérez, A.: Machine Learning in Bioinformatics. Briefings in Bioinformatics 7 (2006) 86-112
11. Ding, C., Peng, H.: Minimum Redundancy Feature Selection from Microarray Gene Expression Data. Journal of Bioinformatics and Computational Biology 3 (2005) 185-205
12. Liu, H., Liu, L., Zhang, H.: Ensemble Gene Selection by Grouping for Microarray Data Classification. Journal of Biomedical Informatics 43 (2010) 81-87

13. Feizollah, A., Anuar, N.B., Salleh, R., Wahab, A.W.A.: A Review on Feature Selection in Mobile Malware Detection. Digital Investigation 13 (2015) 22-37
14. Hyo-Sik, H., Mi-Jung, C.: Analysis of Android Malware Detection Performance using Machine Learning Classifiers. 2013 International Conference on ICT Convergence (2013) 490-495
15. Shabtai, A., Elovici, Y.: Applying Behavioral Detection on Android-Based Devices. In: Cai, Y., Magedanz, T., Li, M., Xia, J., Giannelli, C. (eds.): Mobile Wireless Middleware, Operating Systems, and Applications: Third International Conference, Mobilware 2010, Chicago, IL, USA, June 30 - July 2, 2010. Revised Selected Papers. Springer Berlin Heidelberg, Berlin, Heidelberg (2010) 235-249
16. Shabtai, A., Fledel, Y., Elovici, Y.: Automated Static Code Analysis for Classifying Android Applications Using Machine Learning. 2010 International Conference on Computational Intelligence and Security (2010) 329-333
17. Vinod, P., Laxmi, V., Gaur, M.S., Naval, S., Faruki, P.: MCF: MultiComponent Features for Malware Analysis. 27th International Conference on Advanced Information Networking and Applications Workshops (WAINA) (2013) 1076-1081
18. Battista, P., Mercaldo, F., Nardone, V., Santone, A., Visaggio, C.: Identification of Android Malware Families with Model Checking. 2nd International Conference on Information Systems Security and Privacy (2016)
19. Sedano, J., Chira, C., González, S., Herrero, Á., Corchado, E., Villar, J.: On the Selection of Key Features for Android Malware Characterization. In: Herrero, Á., Baruque, B., Sedano, J., Quintián, H., Corchado, E. (eds.): International Joint Conference, Vol. 369. Springer International Publishing (2015) 167-176
20. Virus Total, https://www.virustotal.com (Last accessed: 2016-07-08)