SVM-based Technique for Mobile Malware Detection

Sergii Lysenko[0000-0001-7243-8747], Kira Bobrovnikova[0000-0002-1046-893X], Andrii Nicheporuk[0000-0002-7230-9475] and Roman Shchuka

Khmelnitsky National University, Khmelnitsky, Ukraine
{sirogyk@ukr.net, kirabobrovnikova@gmail.com
andrey.nicheporuk@gmail.com, schuka.roman@gmail.com}

Abstract. A paper presents a new technique for the mobile malware detection based on the malware's network features analysis is proposed. It uses SVM for malicious programs detection. The novel approach provides the ability to detect malware in the mobile devices. As the inference engine for malware detection the support vector machine was used. The detection process is performed by taking into account the malware's features, captured in the mobile devices. Experimental research showed that the SVMs are able to produce the accurate classification results. Experiments demonstrated, that technique is able to detect different types of malware up to 98.21%.

Keywords: Malware, Mobile device, Cybersecurity, Behavior, Computer system, Network, Android, Cyberattack

1 Introduction

Last year the threats against mobile devices in particular backdoors, malicious cryptomining, banking trojans, especially during the second half of 2018, were growing [1]. The both new mobile device infection techniques (for example, DNS hijacking) and the use of tried-and-tested distribution schemes (for example, SMS spam) were applied. The one third of all mobile attacks - from building botnets, to stealing banking credentials, perpetrating click fraud, or threatening reputation - were performed by hidden apps. McAfee researchers identified 65 000 of fake mobile apps, up more than six-fold from the 10 000 detected only six months earlier. However Android remains by far the most popular mobile operating system, being used by four fifths of respondents. iOS is the only other mobile OS to reach double figures, at 15% [2].

2 Related works

Today variety of approaches to identify of android malware is widely present on the literature.

A malware detection technique called Personal Mobile Malware Guard – PMMG that classifies malwares based on the mobile user feedback is presented in [3]. PMMG controls permissions of different applications and their behavior according to the user needs. These preferences are built incrementally on a personal basis according to the feedback of the user. Performance analysis showed that it is theoretically feasible to build PMMG tool and use it on mobile devices. However in case of sensitive mobile resources that require a large portion of permissions, the proposed technique may lead to increase a number of a false alarm.

Similar to above mentioned except permission, approach that use a combination of permissions and intents supplemented with multiple stages of classifiers for malware detection is proposed in [4]. Their experiments were performed on 1745 applications samples starting with a performance comparison between MLP, Decision Table, Decision Tree, Random Forest, Naive Bayes and Sequential Minimal Optimization classifiers. The Decision Table, MLP, and Decision Tree classifiers were then combined using three schemes: average of probabilities, product of probabilities and majority voting.

In work [5] authors have proposed an approach to detect Android malware using system call logs. The proposed technique mainly consists of three stages. At first using emulator Genymotion the system call log of applications are observed. To do this, each application was executed in emulator for approximately five minutes. To improve the quality of dataset on the second stage involves the using of filtering algorithm. The third step is the implementation of the dataset on machine-learning algorithms. The results of experiments have shown the high accuracy of detection, however authors didn't took into account capable of some applications identify a sandbox type environment.

In [6] a malware detection system that uses a deep convolutional neural network (CNN) is proposed. Malware classification is performed based on static analysis of the raw opcode sequence from a disassembled program. Features indicative of malware are automatically learned by the network from the raw opcode sequence thus removing the need for hand-engineered malware feature. Their experiments demonstrate that CNNs can effectively learn to detect sequences of opcodes that are indicative of malware.

Another static approach for android malware detection that uses multiple features to analyze is presented in [7]. A set of features such as hardware, permission, application components, filtered intents, opcodes and strings are extracted from the samples to form a vector space model. Feature selection methods such as Entropy based Category Coverage Difference (ECCD) and Weighted Mutual Information (WI) are used to choose the prominent features. The performance of the system is analyzed using classifiers, Random Forest, Rotation Forest and Support Vector Machine (SVM).

Authors in [8] propose a static-analysis based system that named MaMaDroid. The operation of a MaMaDroid goes through four stages. First, it extract the call graph from each app by using static analysis, then, it obtain the sequences of API calls using all unique nodes after which it abstract each call to class, package, or family. The third stage involves modeling of the behavior of each app by constructing Markov chains from the sequences of abstracted API calls for the app, with the transition probabilities used as the feature vector toclassify the app as either benign or malware using a machine learning classifier.

The framework TriFlow that use a triage mechanism to rank applications considering their potential risk is developed [9]. TriFlow combines a probabilistic model to predict the existence of information flows with a metric of how significant a flow is in benign and malicious apps. Based on this, TriFlow provides a score for each application that can be used to prioritize analysis. The experimental results show that it can predict the presence of information flows very accurately and that the overall triage mechanism enables significant resource saving.

The literature review has shown that the problem of android malware detection is extremely actuals. However considered approaches have some common disadvantages which are manifested in ignoring packed malware and impossibility to protect the device from Zero Day attacks and malwares capable of modifying themselves.

In [10–13] an approaches for malware detection based on the network and hosting behavior analysis are presented. They use the self-adaptive idea for the malware detection and are based on usage of the semi-supervised fuzzy c-means clustering, where the objects of clustering were the feature vectors which elements may indicate the appearance of cyber threats networks and computer systems.

The mentioned above methods of the malware detection in the mobile devices have shown a high level of effectiveness, but also demonstrate high rate of false positives.

The common weakness of the aforementioned approaches is the requirement of large amounts of computing resources and the fact that they aren't able to respond adaptively to known and unknown attacks performed by malware against the mobile devices.

3 Support Vector Machine

The support vector machines (SVMs) are the high-potential approach for the object classification. SVMs are the supervised learning models with associated learning algorithms [14–15]. They are able to produce accurate and robust classification results, even when input data is non-monotone and non-linearly separable, and to evaluate more relevant information in a convenient way, providing high accuracy of classification with small training sets [16].

Basically, SVM performs classification by finding the hyperplane that maximizes the margin between two classes. Vectors (cases) that define the hyperplane are the support vectors. To define an optimal hyperplane we need to maximize the width of the margin:

$$w \cdot x + b = 0, \tag{1}$$

where x is a classification object lying on the hyperplane, w is normal to the hyperplane, b is the basis and $\frac{|b|}{\|w\|}$ is the perpendicular distance from the hyperplane to the origin, with $\|w\|$ the Euclidean norm of w [14, 17].

Let φ be a mapping function which projects the training data into a Hilbert high dimensional space *H*, $\varphi : R^q \to H$. The data point x is represented in space *H* as $\varphi(x)$.

In the context of SVM, the kernel function defines the hypothesis space, and is defined as:

$$K(x, x_i) = (\varphi(x) \cdot \varphi(x_i)), \tag{2}$$

where x_i is a training data.

It leads to decision functions of the following form:

$$f(x) = \operatorname{sgn}\left(\sum_{i=1}^{r} \alpha_i \alpha_i K(x, x_i) + b \right), \tag{3}$$

where $\alpha_i, i = 1, ..., r$ are Lagrange multipliers, the maximal magnitude of which is governed by *C*.

In order to compute the separating hyperplane without explicitly carrying out the mapping into the feature space, different kernel functions can be used [18, 19]. In this approach, the classification process involved the kernels: linear, polynomial, Gaussian, exponential, and B-Spline.

The linear kernel is the simplest kernel function. It is given by the inner product (x, x_i) plus an optional constant *c*:

$$K(x, x_i) = x^T x_i + c, \; c \in R. \tag{4}$$

The polynomial kernel is a non-stationary kernel, where the adjustable parameters are the slope *α*, the constant term c and the polynomial degree *p*:

$$K(x, x_i) = (\alpha x^T x_i + c)^p, \; \alpha \in R, \; c \in R, \; p \in N. \tag{5}$$

The Gaussian kernel is an example of radial basis function kernel:

$$K(x, x_i) = e^{-\frac{1}{2\sigma^2} \|(x - x_i)\|^2}, \; \sigma > 0, \tag{6}$$

where σ is the parameter that controls the width of the Gaussian kernel.

The exponential kernel is closely related to the Gaussian kernel, with only the square of the norm left out. It is also a radial basis function kernel:

$$K(x, x_i) = e^{-\frac{1}{2\sigma^2} \|(x - x_i)\|}. \tag{7}$$

The B-Spline is a radial basis function kernel, and is defined on the interval [−1, 1]. It is given by the recursive formula [18]:

$$K(x,x_i) = B_{2p+1}(x - x_i),\ where\ p \in N\ with\ B_{i+1} := B_i \otimes B_0 \tag{8}$$

In order to perform the multi-class classification, the ”one against all” and ”one against one” SVM-based methods are used [16].

”One Against all” SVM Classifier. This method constructs *k* SVM models, where *k* is the number of classes. The *m*-th SVM is trained with all of the samples in the *m*-th class with positive labels, and all other samples with negative labels. The *m*-th SVM solves the task of training data mapping to a higher dimensional space for the given *l* training data $(x_1, a_1),...,(x_l, a_l)$, where $x_i \in R^n$, $i = 1,...,l$, $a_i \in \{i = 1,...,k\}$ is the class of training data x_i.

In order to perform the classification for x_i, we use *k* decision functions, where *k* is the number of classes:

$$(w_i)^T \varphi(x_i) + b_i,\ where\ i = 1,...,k, \tag{9}$$

where φ – is the mapping function, $\varphi : R^q \to H$.

The data x_i then belongs to class *a*, for which the above decision function has the largest value:

$$a \equiv \arg\max x_{i=1,...,k}((w_i)^T \varphi(x_i) + b_i). \tag{10}$$

”One Against One” SVM Classifier. The proposed technique uses the ”one against one” SVM classification method as well. Here, $k(k-1)/2$ classifiers are to be constructed for each pair of classes and the max-win strategy is to be followed. Specifically, if $sqn(w_{jl})^T \varphi(x_i) + b_{jl}$ evaluates x_i to be in *j*-th class, then the vote for the *j*-th class is incremented by one, else that for the *l*-th class is increased by one. Finally, the training data vector x_i is predicted to belong to the class with maximum number of votes [16].

Taking into account the advantages of the SVM, it can be used as the inference engine for the making the decision about the presence of the malware in the mobile devices.

4 SVM-based Approach of the Mobile Malware Detection

We propose a new technique for the mobile malware detection. An approach presents the SVM-based system for malware detection with the ability to classify the malicious program and blocking them.

The proposed approach includes learning and detection stages. The learning stage consists of the following steps:

1. Knowledge formation based on the features that may indicate the mobile malware presence.

2. Presentation of the knowledge about the mobile malware behavior as a set of feature vectors.

3. Formation the set of mobile malware classes using SVM.

The monitoring stage consists of the following steps:

1. Gathering the features in the mobile device, which may indicate the mobile malware presence.

2. Construction of the feature vectors based on the obtained information.

3. The detecting stage involves the implementation of SVM classification of the obtained feature vectors in order to assign them to one of the mobile malware's class.

4. The blocking of the malicious program execution.

Let us take a closer look at each step of the method.

4.1 Knowledge Formation Based on the Features that May Indicate Mobile Malware Presence

Let us denote the set of malware's classes as $A = \{a_m\}_{m=1}^{N_A}$, where a_1 – the Trojans; a_2 – the backdoors; a_3 – the mobile botnets; a_4 – the spammers; a_5 – the spyware; a_6 – smartphone trackers; a_7 – the mobile proxy-servers; a_8 – SMS malware; a_9 – the exploits; a_{10} – the rootkits; a_{11} – the Adware; a_{12} – the DDoS attackers, where N_A – the number of attacks, performed by DDoS botnets [20].

Let us denote the set of features, that may indicate mobile malware's attacks against the device and are to be analyzed as $B = \{b_j\}_{j=1}^{N_B}$, where N_B – the number of features. The list of features is presented in Table 1. Let us denote the set of devices attacked by mobile malware as $H = \{h_i\}_{i=1}^{N_H}$, where N_H – the number of mobile devices. Thus, the function of the malware's attack identifying f can be presented as: $f : h_i \times b_j \rightarrow a_m$.

Table 1. The features, that take place in the malware's detection process [21].

Feature	Description
The CPU resource consumption features	
CPU_{PU}	the percentage of CPU time required for the user space by the operating system
CPU_{PK}	the percentage of CPU time required for the kernel space by the operating system
CPU_{PS}	the percentage of shared CPU assigned by the OS
CPU_{NP}	the total number of processes running for the application AUA
CPU_{NT}	the total number of threads running for the application AUA
The memory resource consumption features	
M_F	the global amount of idle memory
M_U	the global amount of used memory

M_A	the global amount of anonymous mapping maps, i.e., the area of the virtual memory (all processes) not backed by any file
M_S	the global amount of slab – a slab is the amount by which a cache can grow or shrink.
M_{TV}	the total amount of memory in swap and RAM
M_{PS}	the amount of application AUA memory shared with other processes, accounted in a way that the amount is divided evenly between the processes that share it
M_{RAM}	the size portion of memory occupied by the application AUA that is held in RAM
M_{UP}	the set of pages that are unique for the application AUA
The storage resource consumption features	
W_{kB}	the amount of information (expressed in kB) written on the storage by all the processes
R_{kB}	the amount of information (expressed in kB) read from storage by all the processes
R_B	the number of bytes which are read from storage by the application AUA
W_B	the number of bytes which are written into the storage by the application AUA
R_N	the value represents the number of read I/O operations
W_N	the value represents the number of write I/O operations
The network resource consumption features	
R_{AP}	the number of packets received by all the processes
T_{AP}	the number of packets transmitted by all the processes
R_{AB}	the number of bytes received by all the processes
T_{AB}	the number of bytes transmitted by all the processes
R_P	the number of packets received by the application AUA
T_P	the number of packets transmitted by the application AUA
R_B	the number of bytes received by the application AUA
T_B	the number of bytes transmitted by the application AUA
DNS-based features	
L_N	the length of the domain name
N_U	the number of unique characters in the domain name
E_N	entropy of the domain name
$T_{\text{mod}}, T_{med}, T_{aver}$	TTL-periods (mode, median, average value)
N_A	the number of A-records corresponding to domain name in the incomig DNS-message
N_{IP}	the number of IP-addresses concerned with the domain name
S_{IP}	the average distance between the IP-addresses concerned with domain name
S_A	the average distance between the IP-addresses in the set of A-records for domain name in the incoming DNS-message
N_{UA}	number of unique IP-addresses in sets of A-records corresponding to the domain name in the DNS-messages

S_{UA}	the average distance between unique IP-addresses in sets A-record corresponding to the domain name in the DNS-messages
N_D	number of domain names that share IP-address corresponding to the domain name
F_{UR}	the sign of the usage of uncommon types of the DNS-records, or DNS-records that are not commonly used by a typical client
E_R	entropy of the DNS-records, which are contained in the DNS-messages
L_P	maximum size of the DNS-messages about domain name
F_S	the sign of success of DNS-query

4.2 Presentation of the Knowledge About the Cyberattacks As the Set of the Feature Vectors

All the above-mentioned features are the base of the set of feature vectors $X = \{x_k\}_{k=1}^{N_x}$, where each of feature vector x_k describes the botnet' attack and the legitimate traffic, N_x – the number of the feature vectors.

4.3 Usage of the SVM-based inference engine for the malware detection

The main task of the SVM-based inference engine is to assign feature vector x_i to class a_t at, where $x_i \in X, a_t \in A, A = \{a_t\}_{t=1}^{N_A}$, N_A – is the number of classes, where each class corresponds to one specified type of attacks, performed by malware. The SVM-based inference engine makes a conclusion about the presence or absence of a malware in the mobile device and detects the possible type of the malware.

5 Experiments

In order to evaluate the effectiveness of the malware detection based on proposed method, experimental studies were conducted.

The experimental part of the study involved the solving of the following tasks:

- to investigate the applicability of SVM for making decisions about the presence of malware in the mobile device;
- to evaluate the detection efficiency of the malware detection in the mobile devices.

5.1 Evaluation Setting

In order to evaluate the technique's, a detection accuracy tests using malware's samples were conducted. For this purpose, a mobile malware datasets [22, 23] were employed. Used datasets were divided into training *T* and evaluation (test) *E* datasets. The training dataset includes 383 samples, 49.56% of which are malicious and the reminder contains normal behaviors. The test dataset includes 403 samples, 55.77% of which represent malicious behaviors.

To retrieve information about the CPU consumption the top tool, which is able to provide an overview of the CPU activity in real time, was used.

To extract the features concerning the memory resource usage, system monitoring tool vmstat and procrank tool were used.

Since the tracking of the input-output activity information is disabled by default to retrieve it the modification of the specific configuration file (depending on the kernel and on the architecture of the device) was perfected and the kernel was recompiled. With purpose of the collection of input-output activity information the iostat command was used. To extract the features related to the use of the network, the means of open source tool Android Device Monitor was used.

Support vector machine was implemented using Matlab [24].

5.2 Performance Measures of the BotGRABBER System

The experimental results were estimated via standard sensitivity (*SN*), specificity (*SP*), and overall accuracy (*Q*) performance measures, taking into account the quantity measures of True Positives (*TP*), True Negatives (*TN*), False Positives (*FP*), False Negatives (*FN*):

$$SN = \frac{TP}{(TP + FN)},\ SP = \frac{TN}{(TN + FP)},\ Q = \frac{TP + TN}{TP + TN + FP + FN}. \tag{11}$$

In order to investigate the SVM-based inference engine efficiency we used different SVM kernel functions. Examples of classification results using linear, polynomial, Gaussian, exponential, and B-Spline kernels are presented in Fig. 1.

The process of classification is divided into several iterations.

In the first iteration, the classification objects are divided into two classes: malicious behavior and the benign one. Then classifiers divide objects into other two classes, for instance: malicious behavior and spam bot. The next iterations separate malicious behavior and other classes of malware and so on until all of them are totally divided.

Fig. 1 presents the placement of the classification objects on the 2-D plane and the objects' separation into two classes.

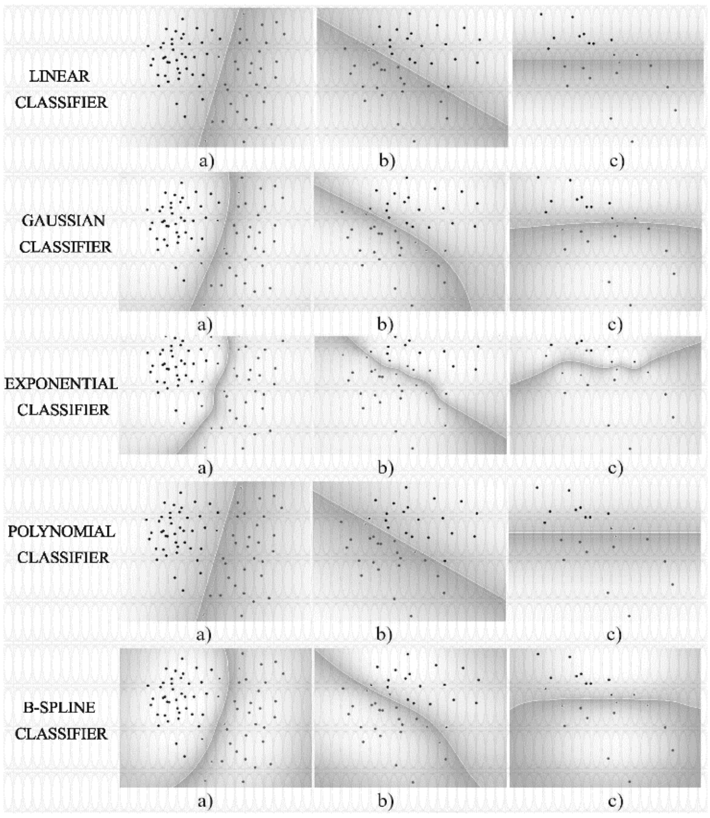

Fig. 3. Results of SVM classification using different kernel functions: a) malicious program/benign program; b) malicious program /spam bot; c) malicious program/trojn program

Experimental results of different SVM classifiers elucidated that the linear and polynomial classifiers had provided the worst results (Table 2). They were characterized by longer execution times, and higher rates of the overall classification accuracy.

Table 2. Experimental results for different SVM classifiers.

Classifier kernel	Classification accuracy		Execution time (sec)	Distance between hyperplanes
	one against	one against all SVM		
linear	92.53	96.35	0.5	0.18
Gaussian	96.18	97.39	0.4	0.25
B-spline	98.01	97.43	0.4	0.38
polynomial	93.56	96.79	0.5	0.37
exponential	97.02	97.64	0.5	0.33

Non-linear classifiers demonstrated better results, where B-Spline provided better results than others did.

For an experimental evaluation samples, the most effective classifier using the SVM was the B-spline since it provided the greatest distance between hyperplanes, the shortest time of evaluation, and the best accuracy of the classification. Given that, it was employed as a basic kernel function in the SVM-based inference engine of the proposed technique.

5.3 Results

This subsection presents overall results of technique's efficiency, taking into account sensitivity, specificity, and the overall accuracy. Employing the datasets [21, 22] this stage involved the evaluation of the SVM-based inference engine accuracy concerning each type of malware separately. The combined results are given in Table 3.

Table 3. Test results for different mobile malware's classes: number of training set samples *T*, number of evaluation samples *E* (including malicious and benign traffic samples), sensitivity (*SN*), specificity (*SP*), overall accuracy (*Q*) of the proposed approach, true positives (*TP*), true negatives (*TN*), false positives (*FP*), false negatives (*FN*), the rate of successful reconfigurations (*SR*).

Mobile malware's classes	*T*	*E*				Results		
		Malicious		Benign		*SN*, %	*SP*, %	*Q*, %
		TP	*FN*	*TN*	*FP*			
trojans	36	38	3	34	1	92,68	97,14	94,74
backdoors	31	33	2	29	2	94,29	93,55	93,94
mobile botnets	28	27	1	25	0	96,43	100,00	98,11
spammers	32	34	3	35	2	91,89	94,59	93,24
spyware	26	29	2	30	0	93,55	100,00	96,72
smartphone trackers	37	39	3	35	3	92,86	92,11	92,50
mobile proxy-servers	24	30	0	25	1	100,00	96,15	98,21
SMS malware	41	40	3	39	2	93,02	95,12	94,05
exploits	35	36	3	34	2	92,31	94,44	93,33
rootkits	30	35	6	30	1	85,37	96,77	90,28
adware	34	27	2	28	2	93,10	93,33	93,22
DDoS attackers	29	35	2	30	1	94,59	96,77	95,59

Table 3 shows that the overall accuracy of technique is in the range from 90.28% to 98.21%. Moreover, sensitivity and specificity are in the range of 85.37–100% and 92.11–100%, respectively. Therefore, this approach indicates the capability of SVM for the botnets classification.

6 Discussion

As the technique uses the SVM-based engine, there are several factors which may affect the prediction accuracy. One of them is the diversity of used training samples. Most conspicuously, that not all possible feature vectors, that describe different malware classes, are adequately represented in the training set. Thus, system may be further improved by choosing more refined set of malicious samples for each malware classes.

In order to increase the classification accuracy the SVM prediction may be further improved by using different classification kernels, as well as the SVM optimization procedure and new mobile malware classes (and its feature vector selection) may also be improved.

Results of the experiments demonstrated, that the technique achieves the best results for detection of such mobile malware as DDoS, spyware, SMS malware, botnets, etc.

At the same time, the efficiency of the system concerning rootkits is rather lower. This is because the behavior of some malware is very similar to users' ones and some of malware's features weren't taken into account for the detection process.

7 Conclusions

A new technique for the mobile malware detection based on the malware's network features analysis is proposed. It uses SVM for malicious programs detection. The novel approach provides the ability to detect malware in the mobile devices.

As the inference engine for malware detection the support vector machine was used. The detection process is performed by taking into account the malware's features, captured in the mobile devices.

Experimental research showed that the SVMs are able to produce the accurate classification results. Implementation of the SVM-based inference engine into the mobile malware's detection process allowed to obtain its mean detection accuracy up to 98.01%. Experiments demonstrated, that technique is able to detect different types of malware in the range from 90.28 to 98.21%, while false positives is about 5%.

8 References

1. McAfee Mobile Threat Report Q1, 2019, https://www.mcafee.com /enterprise/en-us/assets/reports/rp-mobile-threat-report-2019.pdf
2. AV-Comparatives Security Survey, 2019, https://www.av-comparatives.org/wp-content/uploads/2019/02/Security_Survey_2019_en.pdf
3. Amro, B.: Personal Mobile Malware Guard PMMG: a mobile malware detection technique based on user's preferences. IJCSNS International Journal of Computer Science and Network Security, Vol. 18, No. 1, pp. 18–24 (2018)
4. Idrees, F., Rajarajan, M., Conti, M., Chen, T., Rahulamathavan, Y.: Pindroid: a novel android malware detection system using ensemble learning methods. Computers & Security, Vol. 68, pp. 36–46 (2017)
5. Chaba, S., Kumar, R., Pant, R., Dave, M.: Malware Detection Approach for Android systems Using System Call Logs, arXiv preprint arXiv:1709.0880 (2017)
6. McLaughlin, N., Martinez del Rincon, J., Kang, B, et al.: Deep android malware detection. In Proc. of the Seventh ACM on Conference on Data and Application Security and Privacy, pp. 301–308 (2017)

7. Varsha, M., Vinod, P., Dhanya, K.: Identification of malicious android app using manifest and opcode features. Journal of Computer Virology and Hacking Techniques, Vol. 13, Issue 2, pp. 125–138 (2016)
8. Mariconti, E., Onwuzurike, L., Andriotis, P., De Cristofaro, et al.: MaMaDroid: Detecting Android Malware by Building Markov Chains of Behavioral Model (Extended Version). ACM Trans. Priv. Sec., Vol. 1, No. 1, pp. 1–33 (2019)
9. Mirzaei, O., Suarez-Tangil, G., Tapiador, J., M.de Fuentes, J.: Triflow: Triaging android applications using speculative information flows. In Proc. of the 2017 ACM on Asia Conference on Computer and Communications Security, pp. 640-651 (2017)
10. Lysenko, S., Pomorova, O., Savenko, O., Kryshchuk, A., Bobrovnikova, K.: DNS-based anti-evasion technique for botnets detection. In Proc. of the 8th International Conference Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS), Vol. 1, pp. 453–458 (2015)
11. Pomorova, O., Savenko, O., Lysenko, S., Kryshchuk, A., Bobrovnikova, K.: Antievasion technique for the botnets detection based on the passive DNS monitoring and active DNS probing. In Proc. of the 23rd International Conference, CN 2016, Computer Networks, pp. 83–95 (2016)
12. Lysenko, S., Savenko, O., Bobrovnikova, K., Kryshchuk, A., Savenko, B.: Information Technology for Botnets Detection Based on Their Behaviour in the Corporate Area Network. In Proc. of the 24th International Conference, CN 2017, Computer Networks, pp. 166–181 (2017)
13. Lysenko, S., Savenko, O., Bobrovnikova, K., Kryshchuk, A.: Self-adaptive System for the Corporate Area Network Resilience in the Presence of Botnet Cyberattacks. In Proc. of the 25th International Conference, CN 2018, Computer Networks, pp. 385–401 (2018)
14. Weston, J., Mukherjee, S., Chapelle, O., Pontil, M., Poggio, T. Vapnik, V.: Feature selection for SVMs. In Proc. of the 2000 Neural Information Processing Systems (NIPS) Conference, pp. 668–674 (2001)
15. Chapelle, O., Vapnik, V., Bousquet, O. Mukherjee, S.: Choosing multiple parameters for support vector machines. Machine learning, Vol. 46, Issue 1-3, pp.131-159 (2002)
16. Foody, G. M., Mathur, A.: A relative evaluation of multiclass image classification by support vector machines. IEEE Transactions on geoscience and remote sensing, Vol. 42, Issue 6, pp. 1335–1343 (2004)
17. Deng, N., Tian, Y., Zhang, C.: Support vector machines: optimization based theory, algorithms, and extensions. Chapman and Hall/CRC, Data Mining and Knowledge Discovery Series, 363 p. (2012)
18. Hofmann, T., Schölkopf, B., Smola, A. J.: Kernel methods in machine learning. The annals of statistics, Vol. 36, No. 3(2008), pp. 1171-1220 (2008)
19. Larranaga, P., Atienza, D., Diaz-Rozo, J., et al.: Industrial Applications of Machine Learning / CRC Press, 336 p (2018)
20. Forensic Blog. Available: https://forensics.spreitzenbarth.de/android-malware/
21. Canfora, G., Medvet, E., Mercaldo, F., Visaggio, C. A.: Acquiring and analyzing app metrics for effective mobile malware detection. In Proc. of the 2016 ACM on International Workshop on Security And Privacy Analytics. ACM., pp. 50–57 (2016)
22. The Drebin Dataset, https://www.sec.cs.tu-bs.de/~danarp/drebin
23. Android malware genome project, http://www.malgenomeproject.org
24. MathWorks, https://www.mathworks.com/