Artificial & Computational Intelligence/Published Online: July 2021
https://acors.org/ijacoi/VOL2_ISSUE2_9.pdf

Artificial & Computational Intelligence

Exploring Deep Reinforcement Learning for Android Malware Detection

*Simran Gill[1], Surabhi Gogte[1], Chahak Sharma[1], Prathmesh Pathwar[1], Viraj Desai[2], Ashutossh Gupta[3], Aamod Vyas[4] and O.P. Vyas[1]
[1]Indian Institute Of Information Technology, Allahabad
[1]{gillsimu98,surabhigogte27,chahaksharma1010,prathmeshpathwar1998}@gmail.com,[1]opvyas@iiita.ac.in
[2] Veermata Jijabai Technological Institute
[2]desaiviraj28@gmail.com
[3] Indian Institute of Technology, Bhubhneshwar
[3]ag35@iitbbs.ac.in
[4] University of Mannheim, Germany
[4]aamod.vyas@gmail.com

Abstract

In today's world of technology, we are surrounded by electronic gadgets and connected through the internet. The risk of privacy invasion is at an all-time high. Cyber security plays an important role in preventing and precluding these threats and provides a safeguard against these attacks. In this paper, we have focused on Android Malware detection, since its causes have been seen skyrocketing due to easy access to a host device, which makes it susceptible to attacks and data breach. Moreover, Android is open-source, due to which it exposes the application to foreign attacks. Drebin dataset - the most extensive dataset for android malware detection, which is extracted from 15036 application files, was used for analyzes. The feature extraction was performed using Random Forest Classifier and Extra Trees Classifier, top 15 features were selected, and the accuracies were compared. The key focus of our work is to map the Android malware detection problem into the Markov Decision Process which is the mathematical foundation of the Reinforcement learning algorithm. We have implemented Q-Learning, a Deep Reinforcement Learning technique, for the classification of android malware on the Drebin dataset. The accuracy of RL (94.30%) was compared with the performance of other malware detection techniques.

Keywords: Android Malware Detection, Markov Decision Process, Q-Learning, Reinforcement Learning, Extra Trees Classifier, Random Forest Classifier.

1. Introduction

Cybersecurity is the protection of all hardware devices, software-related technologies, and data associated with the applications and devices connected with the internet. Cybersecurity is very important in today's world, as there are hackers and scammers all over the internet, who try to steal sensitive user data, scam people, and cause disruptions in the normal flow of internet traffic. Today, Mobile devices are present in every nook and corner of the world, and the privacy of users and their data is of utmost importance. Android is the most used and popular platform for mobile devices [1]. A recent mobile threat report summarized the comparison between malware samples collected from different devices with varied mobile OS platforms (218 samples). They were analyzed in four quarters of 2012. Interestingly, other platforms, such as iOS, Blackberry, J2ME, Windows mobile, and Symbian, showed a decline in malware samples, while Android showed a twofold increase in the reported cases of malware [2].

Android has been detected with an increasing amount of

malware in the past few years. This number has increased gradually in the past years, unlike other OS platforms of malware that targeted other mobile platforms [3]. The main cause behind this was the open-source policy of Android. Android easily allows the malware to be inserted into an Android app. There has been a wide range of research for the detection of malware in Android [4][5]. After various researches, it was concluded that there are some patterns on which we can divide the malware detection techniques to be effective or not [6][7]. However, these researchers could not find an effective technique that would give satisfactory results for the detection of unknown malware files. Deep learning methods have also been studied for detecting malware detection.

The dataset employed for malware detection is the most comprehensive dataset available for research based on android malware detection. The dataset used is the Drebin Dataset and is provided by the Technische Universitat Braunschweig. To get accurate and precise results, the dataset was initially cleaned for extracting any erroneous or lacking information, by employing Extra Trees Classifier and Random Forest Classifier, in tandem with the Q-Learning, for the identification of malware.

A novel approach to complement Q-Learning has been efficient in yielding desirable accuracy for the recognition of malicious content without hampering the critical attributes. The influence of this extensive research paper will help curb malware attacks and strengthen the security of data. Over the next sections we formalize our objective and challenges through the literature review of the present detection techniques and show our methodology and results.

2. Literature Review

Most of the Android-targeted malware is divided into four categories: Trojan, Spyware, Root Permission Acquisition (exploit), and Installer (dropper).

Trojan: Trojan appears to be valid as a normal program or application but when opened or executed, it installs malware on the device. It may spread to other parts of the device [8].

Spyware: Spyware is encompassed in the original software and deployed to users. The main aim of spyware is to collect data about people, groups, and organizations without their known consent or by fooling them. It can also gain control of the device secretly without users' knowledge [9].

Root permission acquisition (exploit): Some applications trick the user into giving root permission. This gives hackers or exploiters almost full access to the data and control of the device.

Adware: It keeps opening different pop-ups and tries to redirect users to misleading sites and malware [10].

A shocking increase in the cases of malware intrusion reported has led to a massive increase in the research being conducted on detecting and analyzing Android malware.

Vinit B. Mohata and Dhananjay M. Dakhane [11] advised 5 types of malware detection techniques: Signature - Based, Behavioural - based, Specification - based, Data - Mining based, Cloud Based. In signature Based, malware detection is done during program compilation. In specification-based, depending on the behavior of a normal program, a rule set is defined. In behavioral-based, different malware families are studied on a target system. In data-Mining based, defined patterns are detected from large amounts of data, and classifiers are used for malware detection. In cloud Based, Google uses a service called Bouncer to check for malware in the apps uploaded on the play store.

F Martinelli, F. Mercaldo, A. Saracino, and C. A. Visaggio [12] in their research implemented a hybrid model of n-gram (for extracting the opcode), and feature selection techniques, while the app is in run mode. They achieved an accuracy of 99.7% for the detection of malware.

Vinod, P., Akka Zemmari, and Mauro Conti. [13] implemented 2 feature extraction techniques: Absolute Difference of Weighted System Calls (ADWSC) and Ranked System Calls using Large Population Test (RSLPT). After feature extraction, the accuracies were compared on different datasets for malware detection and most of the classifiers had an accuracy of 99%.

Ham, Hyo-Sik [14] employed an SVM classifier for the detection of malware targeting the Android platform, most of the malware was detected through this approach, however, still, some undetectable malware existed in the

application.

As part of literature review, we have analysed classification based on machine learning models and highlighted the results accordingly in Figure 1.

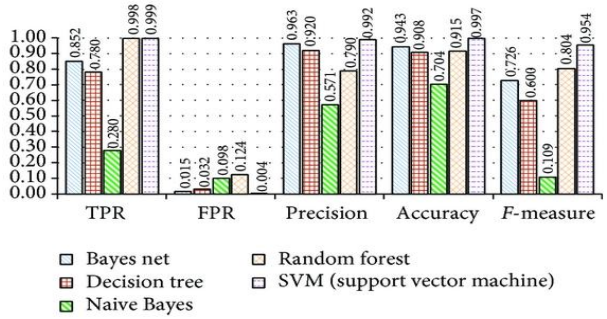

Figure 1 The figure shows comparison between outputs of different machine learning classifiers namely, Bayesian networks, Decision Tree, Random Forest, Naive Bayes and SVM.It can be seen that SVM gives better results than others [14].

3. Objectives and Challenges

Android has been detected with a number of malwares in the past few years, as depicted in Figure 2. This number has increased gradually in the past years, unlike other OS platforms of malware that targeted other mobile platforms. The main cause behind this was the open source policy of Android . Android easily allows malware to be inserted in an Android app. There has been a wide range of research for detection of malware in Android. After a period of time, it was concluded that there are some patterns on which we can divide the malware detection techniques to be effective or not. However,these researchers could not find an effective technique which would give satisfactory results for detection of unknown malware files. Deep learning methods have also been studied for detecting malware detection. These methods extract features from famous Android apps,some of which contain malware and some of them don't. After extracting features,machine learning algorithms were implemented on them to train the model on these features and to test these models for detection of unknown malware files in Android apps [15].

The main motivation behind this research is to apply reinforcement learning for cyber security issues and compare it with the previous works done using Deep Learning.

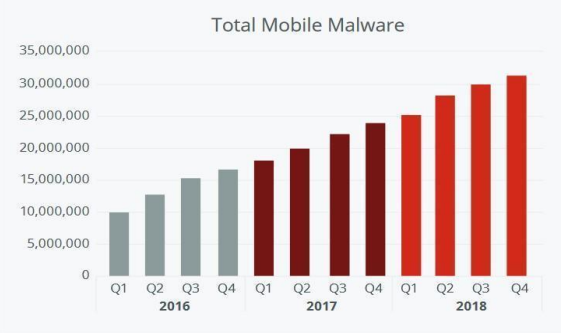

Figure 2 The figure shows a rapid increase in malware samples collected from different devices over the years on the android platform. They were analysed in 2016, 2017 and 2018 [15].

4. Methodology

Q-Learning with Random Forest analyses: The dataset used is the Drebin Dataset and is provided by the Technische Universitat Braunschweig. The dataset consists of 215 attributes and their feature vectors. It is extracted from 15,036 applications consisting of 5,560 malware files collected from August 2010 to October 2012 and 9,476 benign files [16]. All malware samples are labeled as 1 of 179 malware families. Drebin is one of the most popular benchmark datasets for Android malware detection.

For training and subsequently testing our Q-Learning model, the dataset was divided into a 3:2 ratio. The training dataset comprised 75% of the available samples, i.e. 11,277 samples; consequently, the testing dataset comprised 25% of the available samples, i.e. 3,759 samples, for the analyses and comparison of accuracies of the algorithm. Table 1 depicts the distribution of the Drebin dataset.

Dataset	#Samples	#Malign	#Benign
Original	15,036	5,560	9,476
Training	11,277	4,170	7,107
Testing	3,759	1,390	2,369

Table 1 Distribution of Drebin Dataset

Preliminary processing of data: Since the datasets consist of a massive probability of having erroneous attributes and entries increases with the amount of data. The Drebin dataset is gigantic, so to eradicate the chance

of having false positives or skewness in results due to the impact of fallacious data entries, data was initially processed, cleaned, and normalized.

Normalizing the data points: For an identical weight of each attribute and contribution to the model, the data needs to be normalized into a [0,1] range. This was achieved by implementing the Min-max normalization method using the following algorithm [17].

Cleaning of data: To avoid the dataset being contaminated with values that are missing and redundant columns, such tuples are removed for increasing the accuracy and quality of our results [18].

Feature Selection: Computational costs skyrocket with an increase in the attributes or dimensions of a dataset and also an increase in the records. This hampers the functioning of the models. So for better performance, the vital features were extracted using either Random Forest Classifier or Extra Trees Classifier [19].

Random Forest Classifier: A hundred thousand decision trees are constructed in Random Forest. Each decision tree is characterized by randomly selecting variables and data. For predicting a particular tree in a forest we use the remaining dataset. Each tree yields a final answer, yes or no. Finally, the decision of the majority of trees is chosen as the final decision [20].

Extra Trees Classifier: This method is similar to Random Forest Classifying method. In this method, in each tree, a random set of k features from the set of features is given at each of the nodes. The only difference between a random forest classifier and an extra tree classifier is that all the features we select for the split may or may not be the best features of the split since the point to split is randomly chosen. This leads to more diversified trees and less evaluation of splitters at each node. Thus it is preferred over Random Forest Classifier because it takes less time [21].

Implementing Reinforcement Learning: Reinforcement Learning iteratively trains the agent and increases the learning experience of the agent which distinguishes it from other supervised learning algorithms. Reinforcement Learning is defined by a tuple of state, reward, and action. The agent interacts with the environment for a particular set of states and rewards. It performs an optimal action and the environment gives a reward or a penalty. Environment iteratively returns a new set of states and rewards to the agent. We aim to maximize the reward and gradually filter the bad actions and provide a set of suitable actions for a particular state.

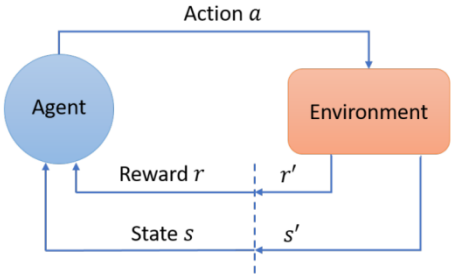

Figure 3 Reinforcement Learning Environment

Figure 3 depicts a Reinforcement Learning environment, where an agent interacts for a particular set of states and actions. The agent takes an optimal action on the state and gives a reward. The environment then returns the agent a new set of states and rewards. The agent again takes a new optimal action and gives a reward, this process takes place iteratively until a terminal state is reached [22].

Markov Decision Processes MDP's: The mathematical formulation of Reinforcement Learning can be described using Markov Decision Processes (MDP's) which consists of (equation 2):

$$(S, A, T, R, \square)$$

where S: a set of states, A: a set of actions, $T(s_{t+1}|s_t, a_t)$: maps a state-action pair at time t onto a distribution of states at time t+1, $R(s_t, a_t, s_{t+1})$: reward function, $\square$: the discount factor, between 0 and 1: this quantifies the difference in importance between immediate rewards and future rewards. Memorylessness: once the current state is known, the history of the previous states can be erased because the current Markov state contains all useful information from history.

For our use case:

S: each state is a tuple of possible combinations of feature values.

A: actions defined are either benign or malicious.

T: next state is defined as the next tuple in the dataset.

R: if predicted true reward of +1 else a penalty of -

discount factor is chosen as 0.95.

Summing across all time steps t (equation 3). For $\square$ = 1,

r(x, a) is a reward function. For state ‘x’ and action ‘a’, it gives the reward associated with taking that action at state ‘x’. We’re trying to maximize the sum of future rewards by taking the best action in each state. Using Markov Decision Processes we set up our reinforcement learning problem and formalize the goal.

Classification using Q-Learning: Q learning improves the behavior of a learning agent iteratively by using Q-values Q(s, a). Where Q(s, a) is the estimation of how good it is to take an action at a state ‘s’. Every time any action is performed, a reward or a penalty is awarded until it reaches a terminating state where it is said to complete one episode. Actions are chosen based on an □-greedy policy: either take an action with maximum q value or perform a random action.

$$Q_t(s, a) = Q_{t-1}(s, a) + \square(R(s, a) + \square max_{a'} Q(s', a') - Q_{t-1}(s, a)) \quad (4)$$

Bellman’s Equation for calculation Q value at a state ‘s’ on taking an action ‘a’, where Q(s, a) is the old q value, □ is the learning rate, R(s, a) is the reward at state ‘s’ and action ‘a’, □ is a discount factor and max Q(s′, a′) is the estimate of optimal future value (equation 4). We maintain a Q table to store the q value of each state-action pair. Figure 4 depicts the flowchart or process of Q-Learning for training the model.

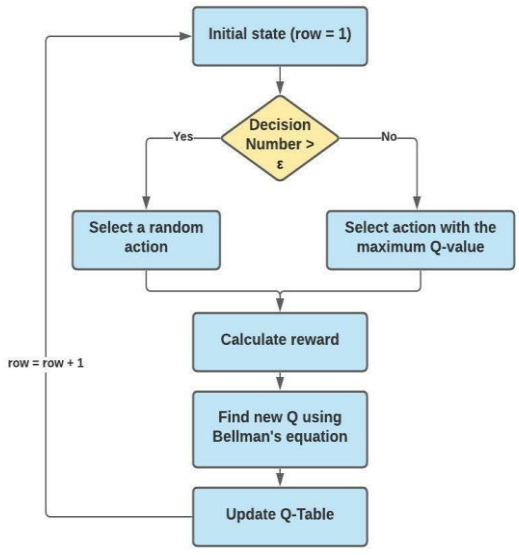

Figure 4 Flowchart of Q Learning algorithm implementation

5. Experimental Results

The focus of this paper was to apply the Reinforcement Learning algorithm (Q-learning) in the field of cybersecurity, as major work/research has not been done in this field. The accuracy of the model varied with varying values of learning rate.

We utilized feature selection using both Random Forest and Extra Trees classifiers at several learning rates and obtained the following accuracies and F1 score for our model.

Q-Learning with Random Forest analyses: As demonstrated in table 2, the Random Forest Classifier, accuracy decreased with an increased learning rate. It performed the best at a learning rate of 0.00039 with an accuracy of 87.652%.

Learning Rate	Accuracy	F1-Score
2.5e-05	87.19%	0.791
0.00015	86.56%	0.877
0.00039	**87.65%**	**0.899**
0.095	86.14%	0.769
1.490	57.17%	0.349
3.725	34.43%	0.027

Table 2 F1-Score and accuracy along with the learning rate for Q-Learning on applying Random Forest Classifier

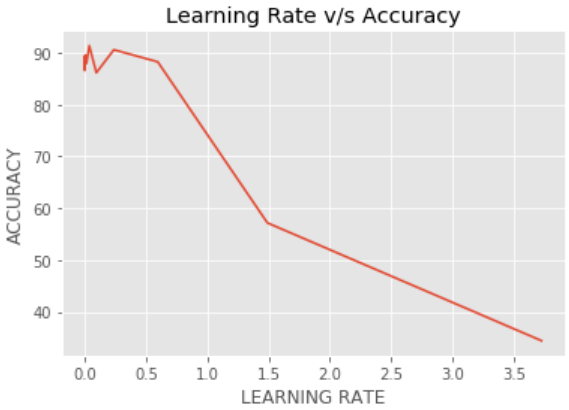

Figure 5 shows the learning rate v/s accuracy for Random Forest Classifier with Q-Learning

Q-Learning with Extra Trees Classifiers analyses: Table 3 shows the variation of accuracy and f-score along with the learning rate. In the Extra trees Classifier, the accuracy decreased with an increased learning rate. The performance was the best at a learning rate of 0.00039

with an accuracy of 94.30%.

Learning Rate	Accuracy	F1-Score
2.5e-05	88.51%	0.922
0.00015	88.56%	0.923
0.00039	**94.30%**	**0.932**
0.095	84.86%	0.852
1.490	56.70%	0.561
3.725	29.31%	0.133

Table 3 F1-Score and accuracy along with the learning rate for Q-Learning on applying Random Forest Classifier

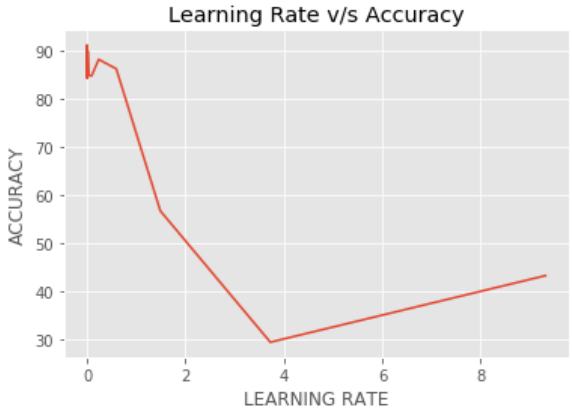

Figure 6 shows the learning rate v/s accuracy for Extra Trees Classifier with Q-Learning

The results differed if we used different feature selection methods, as depicted in figure 5 and figure 6. Extra tree classifiers performed better at a learning rate of 0.00039 with an accuracy of 94.30%, whereas 87.65% using Random Forest Classifiers. Therefore, we used the Extra Trees Classifier for feature selection and our model showed an accuracy of 94.30% at a learning rate of 0.00039 and f-score of 0.873.

Reinforcement Learning v/s other Machine Learning algorithms on Drebin dataset: In the past years, various researchers have built models to classify benign and malware files using Drebin dataset with different approaches like machine learning, deep neural networks, etc. Through the works of various researchers, SVM performed with an accuracy of 94.2%, Random Forest had an accuracy of 94.10%, Decision Tree had an accuracy of 92.8%, Gradient Boosting had an accuracy of 89.96% and LSTM had an accuracy of 70%, as depicted in Figure 7. Our model had an accuracy of 94.30 % and performed better than all other models on the Drebin dataset.

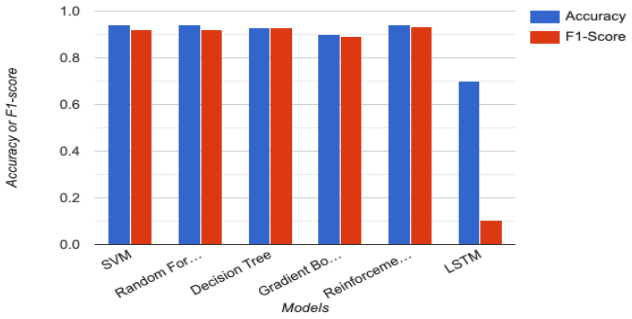

Figure 7 shows the comparison of accuracy and f1-score for Reinforcement Learning model v/s other algorithms on Drebin dataset.

Algorithm	Accuracy	F1-Score
SVM [23]	94.20%	0.92
Random Forest [24]	94.10%	0.94
Decision Tree [24]	92.87%	0.93
Gradient Boosting [24]	89.96%	0.89
LSTM [25]	70%	0.104
Reinforcement Learning	**94.30%**	**0.932**

Table 4 depicts the accuracy of each algorithm for Android malware detection using the Drebin database.

Table 4 depicts the accuracy of each algorithm for Android malware detection using the Drebin database. LSTM had the least accuracy of 70%, followed by Gradient Boosting (89.96%) and **Q-Learning** had the best accuracy of **94.30%**, and f1-score of **0.932**, and learning rate of **0.00039**.

6. Conclusion

Android being the most used and widely available open-source software, has led to a shocking increase in the number of android malware detected. To Avoid the risk due to presence of malware in Android devices, effective detection technique is an essential process. However due to the constantly changing nature of Android malware, there is a need for such a detection strategy that also inline itself accordingly. We proposed and presented the Reinforcement learning algorithm as a detection

mechanism for Android malware. Q-Learning when applied with Extra Trees Classifier provided the best results for the detection of android malware. It had an accuracy of 94.30% for segregating the samples as benign or malign (malicious). In conclusion, it can be stated that this work is like a proof of concept to formalize the Android malware detection method via Markov Decision Process. We have shown the five tuples of Markov Decision Process formulation that are state, action, transition, reward and discount and map the detection strategy into them which is used as the mathematical foundation of the Reinforcement Learning algorithm. We experimented with the Q-learning variant of Reinforcement learning algorithm and used Q-table to maintain the state-action pair. For the efficient feature extraction, we used Random forest classifier and Extra tree classifier and juxtapose our performance against competing approaches. To our best knowledge, such a mechanism of using Q-learning for the android malware has seldom been investigated before. The key purpose of exploring the q-learning for this detection is to enable the environment to retrain itself thus reducing the heavy training cost. However, more exploration can be done towards modelling the MDP formulation to get better accuracy in future.

References

[1] N. Gershenfeld, R. Krikorian, and D. Cohen, “The internet of things,” Scientific American, vol. 291, no. 4, pp. 76–81, 2004.

[2] Snell, Bruce. "Mobile threat report: What’s on the horizon for 2016." Intel Security (2016): 1-12.

[3] Koli, J. D. "RanDroid: Android malware detection using random machine learning classifiers." 2018 Technologies for Smart-City Energy Security and Power (ICSESP). IEEE, 2018.

[4] Sahs, Justin, and Latifur Khan. "A machine learning approach to android malware detection." 2012 European Intelligence and Security Informatics Conference. IEEE, 2012.

[5] Lopez, Christian Camilo Urcuqui, and Andres Navarro Cadavid. "Machine learning classifiers for android malware analysis." 2016 IEEE Colombian Conference on Communications and Computing (COLCOM). IEEE, 2016.

[6] Sahs, Justin, and Latifur Khan. "A machine learning approach to android malware detection." 2012 European Intelligence and Security Informatics Conference. IEEE, 2012.

[7] Chaba, Sanya, et al. "Malware detection approach for android systems using system call logs." arXiv preprint arXiv:1709.08805 (2017).

[8] Shahidi, Seyed Mehdi, Hassan Shakeri, and Mehrdad Jalali. "A semantic malware detection model based on the GMDH neural networks." Computers & Electrical Engineering 91 (2021): 107099.

[9] Pierazzi, Fabio, et al. "A data-driven characterization of modern Android spyware." ACM Transactions on Management Information Systems (TMIS) 11.1 (2020): 1-38.

[10] Supriya, Yamiala, et al. "Malware Detection Techniques: A Survey." 2020 Sixth International Conference on Parallel, Distributed and Grid Computing (PDGC). IEEE, 2020.

[11] Mohata, Vinit B., Dhananjay M. Dakhane, and Ravindra L. Pardhi. "Mobile malware detection techniques." Int J Comput Sci Eng Technol (IJCSET) 4.04 (2013): 2229-3345.

[12] F. Martinelli, F. Mercaldo, A. Saracino and C. A. Visaggio, "I find your behavior disturbing: Static and dynamic app behavioral analysis for detection of Android malware," 2016 14th Annual Conference on Privacy, Security and Trust (PST), 2016, pp. 129-136, doi: 10.1109/PST.2016.7906947.

[13] Vinod, P., Akka Zemmari, and Mauro Conti. "A machine learning based approach to detect malicious android apps using discriminant system calls." Future Generation Computer Systems 94 (2019): 333-350.

[14] Ham, Hyo-Sik, et al. "Linear SVM-based android malware detection for reliable IoT services." Journal of Applied Mathematics 2014 (2014).

[15] Raj Samani, G. D. McAfee Mobile Threat Report tech. rep. (2821 Mission College Blvd.Santa Clara, CA, 2019) (cit. on pp. 1, 4, 5).

[16] Arp, Daniel, et al. "Drebin: Effective and explainable detection of android malware in your pocket." Ndss. Vol. 14. 2014.

[17] Borkin, Dmitrii, et al."Impact of Data Normalization on Classification Model Accuracy." Research Papers Faculty of Materials Science and Technology Slovak University of Technology 27.45 (2019): 79-84.

[18] Rahm, Erhard, and Hong Hai Do. "Data cleaning: Problems and current approaches." IEEE Data Eng. Bull.

23.4 (2000): 3-13.

[19] Bennasar, Mohamed, Yulia Hicks, and Rossitza Setchi. "Feature selection using joint mutual information maximisation." Expert Systems with Applications 42.22 (2015): 8520-8532.

[20] Babatunde, Oluleye H., et al."A genetic algorithm-based feature selection." (2014).

[21] Musunuru, Sreethi, et al. "A Comparative Study using Feature Selection to Predict the Behaviour of Bank Customers." E3S Web of Conferences. Vol. 184. EDP Sciences, 2020.

[22] Thanh Thi Nguyen and Vijay Janapa Reddi. Deep Re - inforcement Learning for Cyber Security

[23] Ayoubianzadeh, Zahra, and Vali Derhami. "Android malware detection using a combination of Support vector machines and fuzzy logic."

[24] Rana, Md Shohel, Sheikh Shah Mohammad Motiur Rahman, and Andrew H. Sung. "Evaluation of tree based machine learning classifiers for android malware detection." International Conference on Computational Collective Intelligence. Springer, Cham, 2018.

[25] Hota, Abhilash, and Paul Irolla. "Deep Neural Networks for Android Malware Detection." ICISSP. 2019.