University of Ottawa

uOttawa

Faculté de génie
Faculty of Engineering

Aidan O'Connor, 8666650

Malware Detection Using Deep Learning

Project Report

18th December, 2020
Advisor: Miguel Garzon
University of Ottawa

Abstract

Malware has been increasingly becoming a big problem. From fake Canadian Emergency Response Benefit (CERB) applications to malware that logs your personal data, it is a digital epidemic. Traditionally, rules were used to identify malicious files. More recently, machine learning is being used to great results. However, just as many other fields are switching to Deep Learning for many reasons, malware detection is. In this paper two avenues are explored: opcode sequence based Ngram analysis on the contents of the dex files, and Deep Belief network permission based analysis on the androidmanifest.

Contents

1 Introduction

Ensuring the safety of a platforms users has, and will always be a high priority. Android has 71.18% of the market share of mobile Operating Systems. Thus, there is heavy research done in the field of detecting malware within this operating system. Malware is any type of software created to intentionally cause malicious effects on any software system. The Android operating system, with its commanding market share, is a common target for people to put malware on. A malware detector is thus software that specializes in detecting malware. Recently, Deep Learning and Machine Learning have made breakthroughs in the detection of malicious software. However, malware is also evolving. Methods such as better code obfuscation are consistently being used and improved to get around malware detectors.

There are two main types of malware detection: static and dynamic analysis. Static analysis analyzes the apks, or software, in a static context. This means that it analyzes it while it is not running. This is a safer route, however it does not have the full scope of information a dynamic analysis has. A dynamic analysis analyzes the apk while it is running, this way it can know what types of things the software is currently doing. Static analysis generally looks at the source code and AndroidManifest, which includes permissions given to the software. Dynamic analysis gives the code actually executed. Both are good, and more efficient in some ways.

To tackle this problem, various methods were tested. However, due to resource limitations, various methods had to be cut out after being programmed. Originally, the intent was to create a two-pronged network: a Highway Bi-Long Short Term Memory Network, used to analyze Ngrams of the opcode, and a Deep Belief Network used to analyze permissions.

2 Background

2.1 Structure of an APK

An Android Package(APK) is the compiled and packaged program used for the operating system Android. It contains all code relating to the program, and consists of four primary parts: the code (.dex files), resources, assets and the manifest. The manifest includes all permissions the program requests, as well as a general overview of the program.

2.2 Opcode

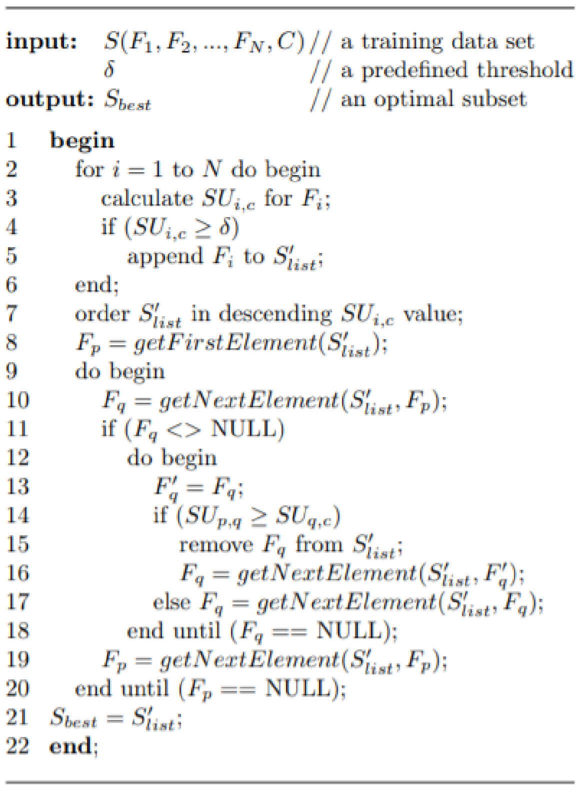
Opcodes, also known as machine code, is the portion of a machine language that specifies the instruction to be performed. In the context of Androids, it is the code (.dex files), converted from .dex -> .smali ->.opcodeseq.

2.3 Permissions

Permissions, contained within the AndroidManifest.xml file. Android permissions can give access to any part of your operating device, and thus are extremely important.

2.4 Fast Correlation Based Filter

Fast Correlation Based Filters (FCBF), by Lei Yu and Huan Liu [40], is a method of data reduction. FCBF combines both Information Gain (IG) and Symmetrical Uncertainty (SU) to reduce the data using both IG and correlation to the rest of the data. Here is the pseudo-code:

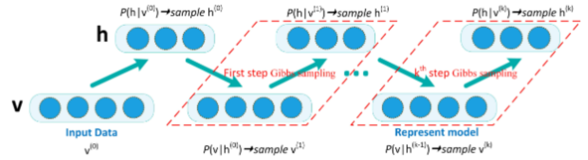


```
input:   $S(F_1, F_2, ..., F_N, C)$ // a training data set
         $\delta$                      // a predefined threshold
output:  $S_{best}$                    // an optimal subset
begin
  for $i = 1$ to $N$ do begin
    calculate $SU_{i,c}$ for $F_i$;
    if ($SU_{i,c} \geq \delta$)
      append $F_i$ to $S'_{list}$;
  end;
  order $S'_{list}$ in descending $SU_{i,c}$ value;
  $F_p = getFirstElement(S'_{list})$;
  do begin
    $F_q = getNextElement(S'_{list}, F_p)$;
    if ($F_q$ <> NULL)
      do begin
        $F'_q = F_q$;
        if ($SU_{p,q} \geq SU_{q,c}$)
          remove $F_q$ from $S'_{list}$;
          $F_q = getNextElement(S'_{list}, F'_q)$;
        else $F_q = getNextElement(S'_{list}, F_q)$;
      end until ($F_q$ == NULL);
    $F_p = getNextElement(S'_{list}, F_p)$;
  end until ($F_p$ == NULL);
  $S_{best} = S'_{list}$;
end;
```

Figure 1. FCBF Algorithm

2.5 Restricted Boltzmann Machine (RBM)

Restricted Boltzmann Machine (RBM) are a two-layered artificial neural network with generative capabilities. They have the ability to learn a probability distribution over its set of input. It is used as layers to a Deep Belief Network.


$P(h|v^{(0)}) \rightarrow sample\ h^{(0)}$
$P(h|v^{(1)}) \rightarrow sample\ h^{(1)}$
$P(h|v^{(k)}) \rightarrow sample\ h^{(k)}$
h
v
First step Gibbs sampling
k^{th} step Gibbs sampling
Input Data
$v^{(0)}$
$P(v|h^{(0)}) \rightarrow sample\ v^{(1)}$
Represent model
$P(v|h^{(k-1)}) \rightarrow sample\ v^{(k)}$

This is an RBM. Each layer is fully connected to itself, inputs and outputs.

3 Related Work

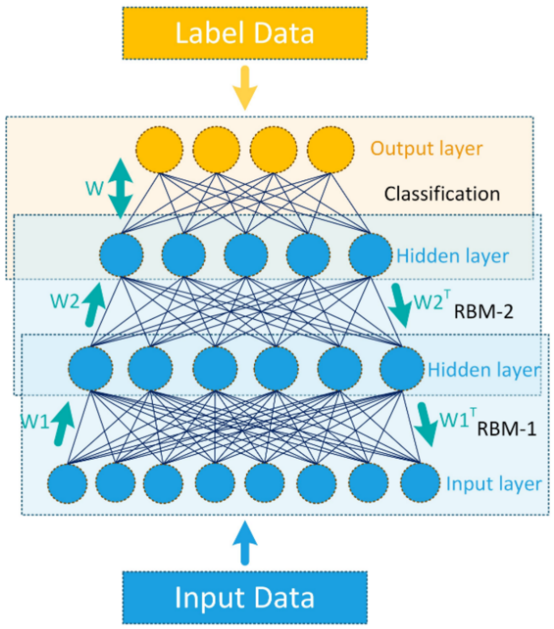
In this section I review related work in malware detection. To begin, both dynamic [12][14-15], and static [5-9] malware detection methods are commonly used. For example, the authors of [2] proposed an efficient dynamic malware-detection method based on API similarity. In 2020, [10] performed a systematic literature review of all static analyses. They found that the most commonly used static technique is Android characteristic, in-the-lab datasets such as Drebin and Genome are the most used datasets, most studies use Apktool, Information Gain (IG) is the most common data reduction technique, machine learning is the most popular, and accuracy is the most applied performance measure. [8] uses code obfuscation, usually to achieve a 99.82% accuracy with *zero* false positives. This is even more impressive considering [13] performed a review of code obfuscation and it's effects on static and dynamic analysis and concluded that static analysis is completely useless against code obfuscation.

Many have begun doing both dynamic and static, hybrid [1-4][11-12]. [1] achieved an astonishing 99.7% accuracy rate over 2794 malicious applications, and a negligible false positive rate tested on over 10 000 applications.

Recently using opcodes in Internet of Things (IoT) has popularized [16-17]. Afterwards, researchers have begun using opcodes in primarily static context to detect malware [19-20]. Then, using n-gram analysis popularize in the Natural Language Processing (NLP) field [21-24]. Then, N-opcode analysis, which is an n-gram analysis of opcodes, popularized [25-26], yielding decent rules. [26] achieve 97% accuracy with 10-opcode. Some even use opcodes in a dynamic context [18], although it is not popular. Notably, opcodes bypass obfuscation very well.

On the other side, permission-based malware detection has always been popular [27-30]. Without amazing results, as only focusing on the permissions is only one side of a many-faceted program, most recent research includes additional information.

Deep Belief Networks (DBN) are extremely versatile networks used in a large range of things, such as hyperspectral imaging [30], protein models [31] and ant colony optimization [32]. Recently, DBN's have been getting usage within malware classification [33-35]. While none of these are only permission-based, [35] is based on api call blocks.



4 The Models

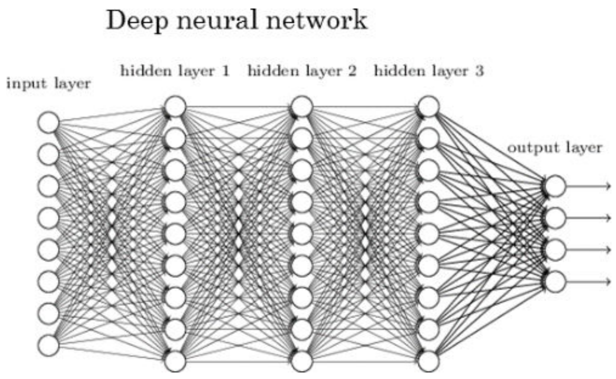
There were two models in total used in my experiments. A Deep Belief Network (DBN), and a Deep Neural Network (DNN).

4.1 Deep Belief Network

A DBN is formed by stacking RBMs, and every two hidden layers (including the input layer) form an RBM. Each RBM consists of a visible layer and a hidden layer, and is trained according to the CD-k algorithm. RBMs are trained one by one and stacked together to form a DBN. Then, add the output layer to the last layer of RBM to form DBN. After trained can still fine-tune.

4.2 Deep Neural Network

A DNN is a basic neural network, it is simply a neural network with multiple layers between the input and output. My DNN layers had the following dimensions (note Linear means Dense): (2048, 1024)->(1024, 64)->(64, 128)->(128, 1). All layers were activated via ReLU, and there was dropouts of 0.3 between the second and third layer and between the third and output. The output was a sigmoid function.



5 Data

5.1 Data Collection

Two datasets were used in this project. The first is the Drebin [38] dataset, which consists of 5, 560 malware applications. Next is the Androzoo [39] dataset, which contains 13,970,710 applications, both malware and benign.

I took all 5, 560 malware from the Drebin dataset and took 5000 random benign applications from the Androzoo dataset.

5.2 Data Preprocessing

5.2.1 Apktool

First, apktool was used to convert all apk folder into a .smali folder.

5.2.2 N-gram Opcodes

Next, all the .dex files were converted into opcodes. This was done by manually going into each .dex file and converting the instructions into the equivalent Dalvik Bytecode.

Then, 2-grams were made out of the opcode instructions. An example of a 2-opcode is 00a5.

5.2.3 Feature Selection

Finally, Fast Correlation Based Filters was used to converge onto 2048 features. This was the input

5.2.4 Permissions

On the other side, the AndroidManifest was extracted from the .smali folder. The permissions were then extracted from the AndroidManifest. If, out of every single possible permission, the permissions was within the AndroidManifest, the value was set to 1. Otherwise it was set to 0.

5.2.5 Feature Generation

Notably, FCBL is a method of feature generation, and thus no other method was used. Additionally, a different approach was used with permission-based classifying. Thus, every single possible permission (there are 167) is used, rather than just a top amount.

6 Training

6.1 Deep Neural Network

6.1.1 Opcode-Based-Classification

The Deep Neural Network was trained at every interval of 50 [50, 500] epochs with batch size 35. It was trained using 2800 samples, 1400 malware samples and 1400 benign samples. All samples randomly taken from datasets.

6.1.2 Permission-Based-Classification

The Deep Neural Network was trained at 50, 100 and 150 epochs with batch size 5. It was trained using 2000 samples, half of which were benign and half of which were malware. Taken randomly from directory of manifests.

6.2 Deep Belief Network

The Deep Belief Network was trained for every interval of 25 [25, 1000] epochs with a batch size of 16 and 32. It was trained using 2358 samples, half of which are benign and half are malware. All samples randomly taken from datasets.

7 Testing

8 Deep Neural Network

For the opcodes, the Deep Neural Network was tested using 700 samples, 351 benign and 349 malware. These samples were randomly selected and taken away from the training data. This means no testing data was used to train the model. An accuracy of 95% was achieved, which is very impressive considering only a 2-opcode was used. Additionally, it only had a 2.5% false positive rate. It is likely a 3-opcode would boost the accuracy by a considerable margin. Another thing to consider is this Deep Neural Network is extremely fast. It takes negligible time to predict 700 samples with dual core i7.

8.1 Deep Belief Network

The Deep Belief Network was tested using 500 samples, 247 benign and 253 malware.These samples were randomly selected and taken away from the training data. This means no testing data was used to train the model. The Deep Belief Network achieved, at best, a 53% accuracy. It is likely the model could never predict anything, and it was just chance that made it that high.

9 Results and Discussion

9.1 Results

The results of the opcode-based-DNN were great, with a validation accuracy of 95% and a dataset accuracy of 97%. The results of the permission-based-DNN were satisfactory, with a validation accuracy of 90%, or 90.6% and a training accuracy of 95.2%. Malware had a 89.0% precision with 92.3% recall, while Benign had a 92.2% precision with 89.0% recall.

The results of the permission-based-DBN were as good as randomly selecting whether is malware or not, or 50%.

The permission-based classifications and its shortcomings are discussed in Problems.

9.2 Discussion

N-gram analysis of opcodes, specifically those using FCBF show great promise in the field of Deep Learning for Malware Classification. It is almost certain that with 3-opcode instead of 2-opcode, accuracy would go up a considerable margin. However, the large shortcoming of n-opcode is that n-opcode takes up considerable amounts of space. While it would be quite easy to run with more space, I only had access to 6GB. Something to consider is that while the initial n-opcode takes up space, after running FCBF on it it can become as small as needed. Additionally, this model is extremely lightweight. The model is small, and computationally extremely fast. Thus it could be implemented on a device such as a phone quite easily. This is a major point in this paper, as anyone with any device can easily check an opcodesequence for malware. To test an apk, more time is needed, but it is still extremely fast compared to other techniques.

Additionally, permission-based classification with the DNN showed promising results. With a speed of 0.0004 seconds to classify 500 inputs, this is extremely fast and practical for use. Even with only a 90% accuracy, the speed makes it practical.

Additionally, this shows that even with a new model, such as a DBN, all permissions are only so important at classifying whether a file is malware or not. It is likely that if feature selection were performed, the model would have a higher accuracy.

10 Problems

10.1 Space Issues

Space issue was a big deal. Originally a 3-opcode analysis was planned, but it took too much memory (4GB). Additionally, a more complex network was planned to be used but that also took up to much memory. Given 2TB of memory, the entire dataset could be loaded and have FCBF performed on it at once with 3-opcode. This would be ideal and likely yield greater results.

10.2 Decompiling the APK

Decompiling the APK into it's proper contents is by far the biggest bottleneck of this methods. While only decompiling the manifest takes less than 3 seconds, and thus the permission-based method is under 3 seconds to classify an apk, to decompile the entire APK takes 6-7 seconds. Thus, to do both the opcode and permission-based classification, it takes 6-7 seconds.

10.3 Permission Issues

Permissions are quite simple, and unsuitable for Deep Belief Networks. This is likely because the hidden layers in DBN's are unsupervised learning. Supervised learning is used in RBMs to deconstruct the input, usually in order to reconstruct. However, in this case, it is used to classify if it is malware or not. Likely, a different probabilistic model would need to be used. Deep Belief Networks could, however, be used to create training datasets for these other models.

11 Conclusion

In this paper, I investigated and analyzed various ways of malware classification. The two principal ways, through n-opcode analysis and permission extraction, revealed the pros and cons of each. The pros of n-opcode analysis with FCBF is it is fairly fast and very accurate even with an extremely simple model. The cons are how much space the initial dataset takes up, before FCBF is performed. THe pros of permission extraction is it is extremely lightweight and fast. The cons is no results were obtained. To continue the permission-based study, I would recommend adding additional features, such as API calls, to the extraction.

12 Future Work

To begin, the n-opcode FCBL could be done with more RAM. This would increase the accuracy by quite a large amount. Next, binary opcodes, rather than hexidecimal opcodes, could be used, as in the literature review, binary opcodes have been tested to yield greated accuracy. Finally, if you get the native code and convert that as well, the model would have greater information to make a decision. Next, I would recommend performing feature selection on permission, and also expanding the model to use more than just permissions. Additionally, Deep Neural Network could output a [1, 2] list of softmaxes, and then that be put into another model. This way, both permissions and opcode sequences could be used to classify an apk.

13 Reference

[1] Martinelli, Fabio, et al. “BRIDEMAID: An Hybrid Tool for Accurate Detection of Android Malware.” Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, Association for Computing Machinery, 2017, pp. 899–901. ACM Digital Library, doi:10.1145/3052973.3055156.

[2] Martinelli, F., et al. “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. 129–36. IEEE Xplore, doi:10.1109/PST.2016.7906947.

[3] Sugunan, Krishna, et al. “Static and Dynamic Analysis for Android Malware Detection.” Advances in Big Data and Cloud Computing, edited by Elijah Blessing Rajsingh et al., Springer, 2018, pp. 147–55. Springer Link, doi:10.1007/978-981-10-7200-0_13.

[4] Kapratwar, Ankita. “Static and Dynamic Analysis for Android Malware Detection.” Master’s Projects, June 2016, doi:https://doi.org/10.31979/etd.za5p-mqce.

[5] Fereidooni, H., et al. “ANASTASIA: ANdroid MAlware Detection Using STatic AnalySIs of Applications.” 2016 8th IFIP International Conference on New Technologies, Mobility and Security (NTMS), 2016, pp. 1–5. IEEE Xplore, doi:10.1109/NTMS.2016.7792435.

[6] Şahın, D. Ö., et al. “New Results on Permission Based Static Analysis for Android Malware.” 2018 6th International Symposium on Digital Forensic and Security (ISDFS), 2018, pp. 1–4. IEEE Xplore, doi:10.1109/ISDFS.2018.8355377.

[7] Vinayakumar, R., et al. “Detecting Android Malware Using Long Short-Term Memory (LSTM).” Journal of Intelligent Fuzzy Systems, vol. 34, no. 3, Jan. 2018, pp. 1277–88. content.iospress.com, doi:10.3233/JIFS-169424.

[8] Suarez-Tangil, Guillermo, et al. “DroidSieve: Fast and Accurate Classification of Obfuscated Android Malware.” Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy, Association for Computing Machinery, 2017, pp. 309–320. ACM Digital Library, doi:10.1145/3029806.3029825.

[9] Firdaus, Ahmad, et al. “Discovering Optimal Features Using Static Analysis and a Genetic Search Based Method for Android Malware Detection.” Frontiers of Information Technology Electronic Engineering, vol. 19, no. 6, June 2018, pp. 712–36. Springer Link, doi:10.1631/FITEE.1601491.

[10] Pan, Y., et al. “A Systematic Literature Review of Android Malware Detection Using Static Analysis.” IEEE Access, vol. 8, 2020, pp. 116363–79. IEEE Xplore, doi:10.1109/ACCESS.2020.3002842.

[11] “(PDF) Static and Dynamic Analysis of Android Malware.” ResearchGate, doi:10.5220/0006256706530662. Accessed 18 Dec. 2020

[12] “(PDF) A Family of Droids: Analyzing Behavioral Model Based Android Malware Detection via Static and Dynamic Analysis.” ResearchGate, https://www.researchgate.net/publication/323694433$_{AF}amily_of$

[13] Bacci, Alessandro, et al. Impact of Code Obfuscation on Android Malware Detection Based on Static and Dynamic Analysis. SCITEPRESS, 2018, pp. 379–85. www.scitepress.org, doi:10.5220/0006642503790385.

[14] Bhatia, T., and R. Kaushal. “Malware Detection in Android Based on Dynamic Analysis.” 2017 International Conference on Cyber Security And Protection Of Digital Services (Cyber Security), 2017, pp. 1–6. IEEE Xplore, doi:10.1109/CyberSecPODS.2017.8074847.

[15] Feng, P., et al. “A Novel Dynamic Android Malware Detection System With Ensemble Learning.” IEEE Access, vol. 6, 2018, pp. 30996–1011. IEEE Xplore, doi:10.1109/ACCESS.2018.2844349.

[16] Darabian, Hamid, et al. “An Opcode-Based Technique for Polymorphic Internet of Things Malware Detection.” Concurrency and Computation: Practice and Experience, vol. 32, no. 6, 2020, p. e5173. Wiley Online Library, doi:https://doi.org/10.1002/cpe.5173.

[17] Azmoodeh, A., et al. “Robust Malware Detection for Internet of (Battlefield) Things Devices Using Deep Eigenspace Learning.” IEEE Transactions on Sustainable Computing, vol. 4, no. 1, Jan. 2019, pp. 88–95. IEEE Xplore, doi:10.1109/TSUSC.2018.2809665.

[18] Carlin, D., et al. “The Effects of Traditional Anti-Virus Labels on Malware Detection Using Dynamic Runtime Opcodes.” IEEE Access, vol. 5, 2017, pp. 17742–52. IEEE Xplore, doi:10.1109/ACCESS.2017.2749538.

[19] Pektaş, Abdurrahman, and Tankut Acarman. “Learning to Detect Android Malware via Opcode Sequences.” Neurocomputing, vol. 396, July 2020, pp. 599–608. ScienceDirect, doi:10.1016/j.neucom.2018.09.102.

[20] Ding, Y., et al. “Application of Deep Belief Networks for Opcode Based Malware Detection.” 2016 International Joint Conference on Neural Networks (IJCNN), 2016, pp. 3901–08. IEEE Xplore, doi:10.1109/IJCNN.2016.7727705.

[21] Bhuyan, M. P., and S. K. Sarma. “An N-Gram Based Model for Predicting of Word-Formation in Assamese Language.” Journal of Information and Optimization Sciences, vol. 40, no. 2, Feb. 2019, pp. 427–40. Taylor and Francis+NEJM, doi:10.1080/02522667.2019.1580883.

[22] Yazdani, Azita, et al. “Words Prediction Based on N-Gram Model for Free-Text Entry in Electronic Health Records.” Health Information Science and Systems, vol. 7, no. 1, Feb. 2019, p. 6. Springer Link, doi:10.1007/s13755-019-0065-5.

[23] Li, Xiao, et al. “Exploring N-Gram Features in Clickstream Data for MOOC Learning Achievement Prediction.” Database Systems for Advanced Applications, edited by Zhifeng Bao et al., Springer International Publishing, 2017, pp. 328–39. Springer Link, doi:10.1007/978-3-319-55705-$2_2$6.

[24] Zhang, D., et al. “Large-Scale Point-of-Interest Category Prediction Using Natural Language Processing Models.” 2017 IEEE International Conference on Big Data (Big Data), 2017, pp. 1027–32. IEEE Xplore, doi:10.1109/BigData.2017.8258026

[25] Kang, B., et al. "N-opcode Analysis for Android Malware Classification and Categorization." 2016 International Conference on Cyber Security And Protection of Digital Services (Cyber Security), 2016.

[26] Kang, BooJoong, et al. “N-Gram Opcode Analysis for Android Malware Detection.” ArXiv:1612.01445 [Cs], Dec. 2016. arXiv.org, http://arxiv.org/abs/1612.01445.

[27] Arslan, Recep Sinan, et al. “Permission-Based Malware Detection System for Android Using Machine Learning Techniques.” International Journal of Software Engineering and Knowledge Engineering, vol. 29, no. 01, Jan. 2019, pp. 43–61. worldscientific.com (Atypon), doi:10.1142/S0218194019500037.

[28] Thiyagarajan, Janani, et al. “Improved Real-Time Permission Based Malware Detection and Clustering Approach Using Model Independent Pruning.” IET Information Security, vol. 14, no. 5, Mar. 2020, pp. 531–41. digital-library.theiet.org, doi:10.1049/iet-ifs.2019.0418.

[29] Ilham, Soussi, et al. “Permission Based Malware Detection in Android Devices.” Proceedings of the 3rd International Conference on Smart City Applications, Association for Computing Machinery, 2018, pp. 1–6. ACM Digital Library, doi:10.1145/3286606.3286860.

[30] Zhong, P., et al. “Learning to Diversify Deep Belief Networks for Hyperspectral Image Classification.” IEEE Transactions on Geoscience and Remote Sensing, vol. 55, no. 6, June 2017, pp. 3516–30. IEEE Xplore, doi:10.1109/TGRS.2017.2675902.

[31] Cao, Renzhi, et al. “DeepQA: Improving the Estimation of Single Protein Model Quality with Deep Belief Networks.” BMC Bioinformatics, vol. 17, no. 1, Dec. 2016, p. 495. Springer Link, doi:10.1186/s12859-016-1405-y.

[32] Ma, M., et al. “Discriminative Deep Belief Networks with Ant Colony Optimization for Health Status Assessment of Machine.” IEEE Transactions on Instrumentation and Measurement, vol. 66, no. 12, Dec. 2017, pp. 3115–25. IEEE Xplore, doi:10.1109/TIM.2017.2735661.

[33] Qin, Xiaoxia, et al. “MSNdroid: The Android Malware Detector Based on Multi-Class Features and Deep Belief Network.” Proceedings of the ACM Turing Celebration Conference - China, Association for Computing Machinery, 2019, pp. 1–5. ACM Digital Library, doi:10.1145/3321408.3321606.

[34] “Deep Belief Networks-Based Framework for Malware Detection in Android Systems.” Alexandria Engineering Journal, vol. 57, no. 4, Dec. 2018, pp. 4049–57. www.sciencedirect.com, doi:10.1016/j.aej.2018.10.008.

[35] Hou, Shifu, et al. “DroidDelver: An Android Malware Detection System Using Deep Belief Network Based on API Call Blocks.” Web-Age Information Management, edited by Shaoxu Song and Yongxin Tong, Springer International Publishing, 2016, pp. 54–66. Springer Link, doi:10.1007/978-3-319-47121-1_5.

[36] Hinton, G. (2007). 2007 NIPS Tutorial on: Deep Belief Nets. Retrieved November/December, 2020, from http://www.cs.toronto.edu/ hinton/nipstutorial/nipstut3.pdf

[37] “A Novel Deep Learning Based Fault Diagnosis Approach for Chemical Process with Extended Deep Belief Network.” ISA Transactions, vol. 96, Jan. 2020, pp. 457–67. www.sciencedirect.com, doi:10.1016/j.isatra.2019.07.001.

[38] “(PDF) DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket.” ResearchGate, doi:10.14722/ndss.2014.23247. Accessed 12 Dec. 2020.

[39] Allix, Kevin, et al. “AndroZoo: Collecting Millions of Android Apps for the Research Community.” Proceedings of the 13th International Conference on Mining Software Repositories, Association for

Computing Machinery, 2016, pp. 468–471. ACM Digital Library, doi:10.1145/2901739.2903508.

[40] "(PDF) Feature Selection for High-Dimensional Data: A Fast Correlation-Based Filter Solution." ResearchGate, https://www.researchgate.net/publication/$221345776_{F}eature_{S}election_{f}or_{H}igh-Dimensional_{D}ata_{AF}a$
$Based_{F}ilter_{S}olution.Accessed18Dec.$2020.