
Article:
HaddadPajouh, H., Dehghantanha, A, Khayami, R. et al. (2018) A Deep Recurrent Neural Network Based Approach for Internet of Things Malware Threat Hunting. Future Generation Computer Systems, 85. pp. 88-96. ISSN: 0167-739X

https://doi.org/10.1016/j.future.2018.03.007

A Deep Recurrent Neural Network Based Approach for Internet of Things Malware Threat Hunting

Hamed HaddadPajouh[a], Ali Dehghantanha[b], Raouf Khayami[a], Kim-Kwang Raymond Choo[c]

[a]*Department of Computer Engineering and Information, Technology, Shiraz University of Technology, Iran*
[b] *Department of computer science, University of Sheffield, UK*
[c]*Department of Information Systems and Cyber Security and Department of Electrical and Computer Engineering, The University of Texas at San Antonio, San Antonio, TX 78249, USA*
School of Information Technology & Mathematical Sciences, University of South Australia, Adelaide, SA 5095, Australia

Abstract

Internet of Things (IoT) devices are increasingly deployed in different industries and for different purposes (e.g. sensing/collecting of environmental data in both civilian and military settings). The increasing presence in a broad range of applications, and their increasing computing and processing capabilities make them a valuable attack target, such as malware designed to compromise specific IoT devices. In this paper, we explore the potential of using Recurrent Neural Network (RNN) deep learning in detecting IoT malware. Specifically, our approach uses RNN to analyze ARM-based IoT applications' execution operation codes (OpCodes). To train our models, we use an IoT application dataset comprising 281 malware and 270 benign ware. Then, we evaluate the trained model using 100 new IoT malware samples (i.e. not previously exposed to the model) with three different Long Short Term Memory (LSTM) configurations. Findings of the 10-fold cross validation analysis show that the second configuration with 2-layer neurons has the highest accuracy (98.18%) in the detection of new malware samples. A comparative summary with other machine learning classifiers also

Email addresses: hp@sutech.ac.ir (Hamed HaddadPajouh), alid@alid.info (Ali Dehghantanha), khayami@sutech.ac.ir (Raouf Khayami), raymond.choo@fulbrightmail.org (Kim-Kwang Raymond Choo)

demonstrate that the LSTM approach delivers the best possible outcome.

Keywords: ARM-based IoT Malware Detection, IoT Malware Detection, Long Short Term Memory, Machine Learning, OpCodes Analysis, Deep Learning Threat Hunting

1. Introduction

Threats from malware are not new, although malware or cyber threat hunting remains an ongoing challenge. For example, with the increasing popularity of Internet of Things (IoT) devices [7] and the general lack of security protection for such devices, IoT devices can be vulnerable to malware attacks [24]. According to Kaspersky Lab, in 2016 the majority of IoT devices examined were insecure, in the sense that these devices had either default password or unpatched vulnerabilities. In other words, these devices can be easily compromised using malware such as Hijme [31] and Mirai [19]. Previous literature have suggested the potential of leveraging machine learning in static and dynamic malware analysis techniques to enhance malware hunting [8, 13, 38], but it is not practical to simply integrate machine learning in static and dynamic malware analysis techniques due to the wide variety and distribution of IoT devices, particularly for (inexpensive) IoT devices with limited processing power.

Existing machine learning-based IoT malware hunting approaches have focused on energy consumption patterns [3] and OpCode [2]. This is not surprising, as system calls and OpCodes are two common features in malware hunting [23]. For example, in [32], the authors proposed a method to classify variants of known malware families based on OpCodes' frequency. The authors in [30] also built a similarity graph based on application's OpCodes to detect metamorphic malware. In [26], SVM classifier and n-gram techniques were used to evaluate OpCodes and identify optimum feature for malware detection. The authors in [33] proposed a method based on the frequency of appearance of OpCodes sequences under different machine learning classifiers and reportedly obtained over an accuracy rate of 96%. Using a text mining method that utilizes n-gram technique, the authors claimed that their approach in detecting malicious software has an accuracy rate of 75% when N=3 and N=4 [25]. In [9], n-gram technique was utilized to extract application's OpCodes sequence, and the findings indicated an accuracy rate of 99.88%. The authors in [16] used the application programming inter-

face (API) sequence as a feature for classification, and the longest common sequence (LCS) technique and sequence analysis to detect malware. An accuracy of 99% was reported in their 70% (training)-30% (testing) dataset split evaluation.

In recent years, deep learning methods have also been used in malware analysis and detection. In [39], for example, the authors used over 200 features extracted from static and dynamic analysis of Android malware to build a model based on deep belief networks. The reported detection accuracy of this approach is 96%. Saxe and Berlin [34] proposed a model based on the deep feed-forward neural network that extracts features from over 40,000 Windows application binary files, and reported a detection rate of 95% with a 0.1% false positive rate. Kolosnjaji *et al.* [20] combined convolutional neural network (CNN) and recurrent neural network (RNN) to perform hierarchical feature extraction, and used N-gram technique to select appropriate OpCodes for malware detection. The authors reportedly had a 89% detection accuracy. More recently in 2017, Rhode, Burnap and Jones [29] presented an approach using RNN and long short term memory (LSTM) for OpCodes-based malware detection and reportedly obtained a 98% detection accuracy with a 1.41% false alarm rate. The authors in [40] utilized deep belief network (DBF) to achieve a 98% accuracy rate in detecting malware based on OpCodes sequences.

At the time of this research, there has been no existing work that uses deep learning in IoT malware detection. Therefore, in this paper, we propose using RNN to detect IoT malware by analyzing IoT application's OpCodes. Our approach does not require the modification of OpCodes representation. The latter is particularly attractive for real-world malware threat hunting. The focus of this paper is on ARM-based IoT applications since the majority of Unix System-V IoT devices use ARM processors [5]. We then evaluate its utility by comparing its performance against those of using conventional machine learning classifiers – Support Vector Machine (SVM), K-Nearest Neighbor, Nave Bayes, Decision Tree, and Random Forest, as well as AdaBoost (an ensemble learning technique).

In the next section, we will present our proposed approach.

2. Proposed Approach

The proposed IoT malware hunting approach comprises three stages, as presented in Figure 1. In the first stage, we collected IoT malware and

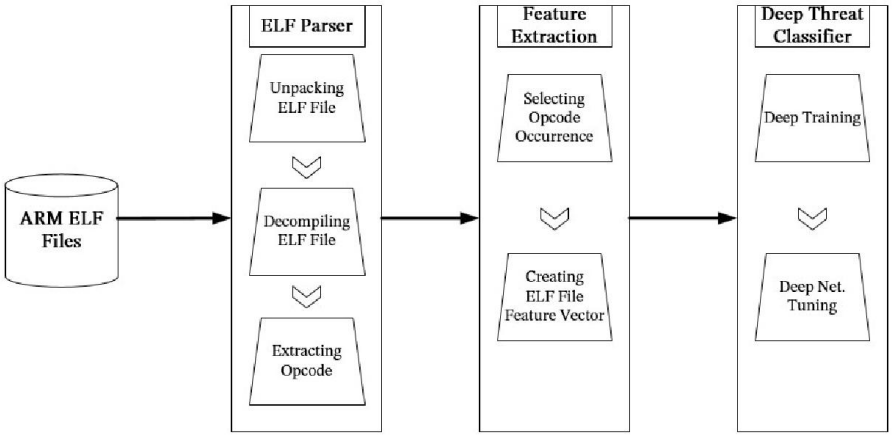

Figure 1: Proposed deep IoT threat hunting approach

benignware samples to build our dataset and extract the OpCodes. A feature vector file based on the OpCodes was then created for each sample. The final stage utilized vectored data for deep neural network training and evaluation, and finally tuning for optimum results.

2.1. Dataset Creation and OpCodes Extraction

As previously discussed, the focus of this research is on ARM-based IoT applications. Since Unix System-V is Debian-based, our benign samples were collected from the Linux Debian package repositories ("Linux Packages Search - `https://pkgs.org/`") of applications compatible with Raspberry Pie II. ARM processors have been widely used in cloud edge devices, and the Raspberry Pi II can also be considered as an IoT cloud edge device [28]. There are two major types of ARM processors, as follows:

1. Application processors (e.g. Arm Cortex-A processor) are generally used in systems with a full-featured operating system (OS), such as Linux distributions and Windows RT, and on smart mobile devices, servers, etc.
2. Embedded processors can be found in a number of microcontroller products, and embedded systems. The Arm Cortex-M processor family, for example, is one of the market leaders in the microcontroller market, and the Cortex-R processor family is typically used in specialized controllers such hard disk drives.

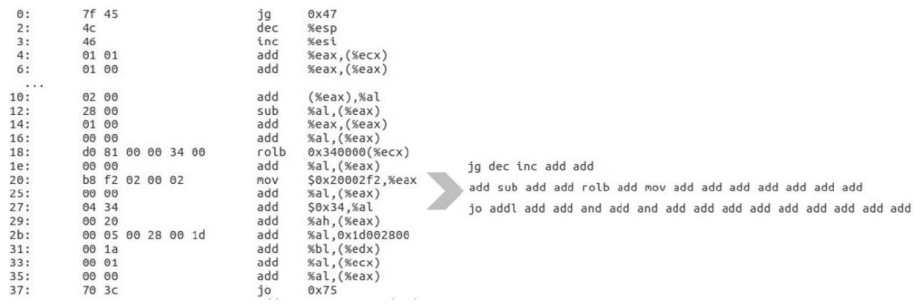

Figure 2: Pruning sequential OpCodes from decompiled IoT applications.

The malware samples were collected by searching for available 32-bit ARM-based malware in the Virus Total Threat Intelligence platform as of September 30th, 2017 [37]. The collected dataset consisted of 280 malware and 271 benign files. All files were unpacked using Debian installer bundle and then Object-Dump tool was used to decompile all samples. We wrote a Linux bash script for the dataset samples' OpCodes. First, the script extracted each Debian package files (deb file), then searched for ELF files from the extracted materials, and finally feeding the object-dump tool to decompile the ELF files. The decompiled codes were then pruned to extract the sequence of OpCodes in each sample. As observed from Figure 2, the output of object dump tool consists of irrelevant data such as operands and line of codes. Thus, our batch script was applied on each output in order to obtain a pruned file, which listed the sample OpCodes in a sequential order. In terms of the instruction set in these type of microprocessors, Cortex-A has the largest instruction set (OpCodes). Since Raspberry Pie II devices is based on Cortex-A, the complete set of Opcodes obtained will increase detection date (in comparsion to, say the Cortex M families since memory management instruction set is not provided).

2.2. Feature Selection

Here, we used text mining to obtain the features vector from the pruned OpCodes. We compiled a word dictionary from all unique OpCodes from our dataset. The final vector comprised 681 possible Opcode indices for each sample. The resulted featured vectors were also parsed using different metrics, namely: binary encoding (takes a value for each OpCode index that exists in the given sample, otherwise takes the value of 0), Term FrequencyInverse Document Frequency (TF-IDF) [15], and count of occurrences of

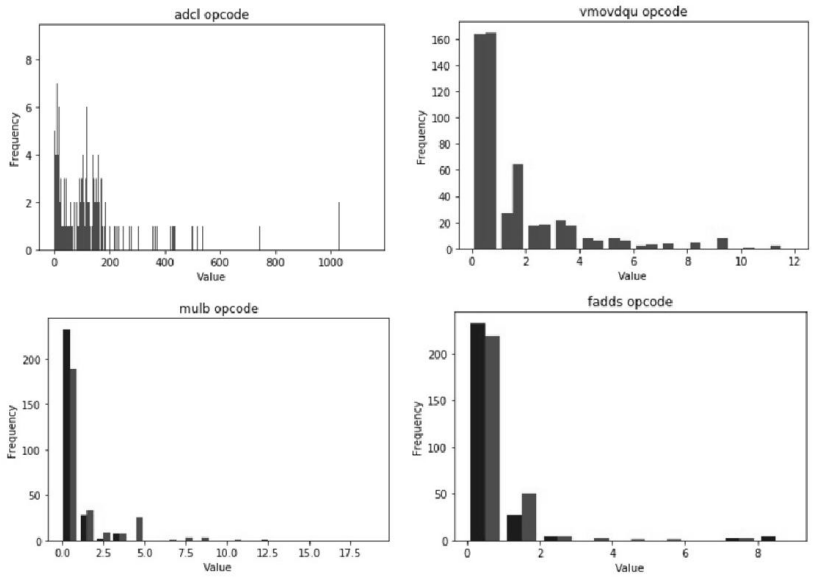

Figure 3: Frequency distributions of selected prominent OpCodes based on IG feature selector

each feature in each application. Figure 3 shows the distribution frequency of selected OpCodes in both malware and benign class using Information Gain (IG) [22] feature selector.

In Equation 1, f denotes the given OpCode in dataset D, c is the number of classes in the training set (and we had two classes, namely: malicious and benign), D_v is the OpCode stream where feature f exists, and wi is the proportion of D_v to class i.

$$IG(D, f) = \sum_{i=1}^{c} -p_i \ln p_i - \sum_{V \in \{0,1\}} \frac{|D_V|}{|D|} \sum_{i=1}^{c} -w_i \ln w_i \tag{1}$$

After obtaining each IG, we sorted the values in decreasing order to identify the most prominent features required in the setting of a threshold ($\alpha > 0.3$) –See Table1.

Since not every sample consists of all OpCodes in their feature vectors, features may have a zero value. Therefore, we used the word embedding technique [4] to transform each sample to a numeric sequence representation. One of the main challenges to solve natural language processing problem is the "curse of dimensionality" [14]. As we had 681 possible feature values for

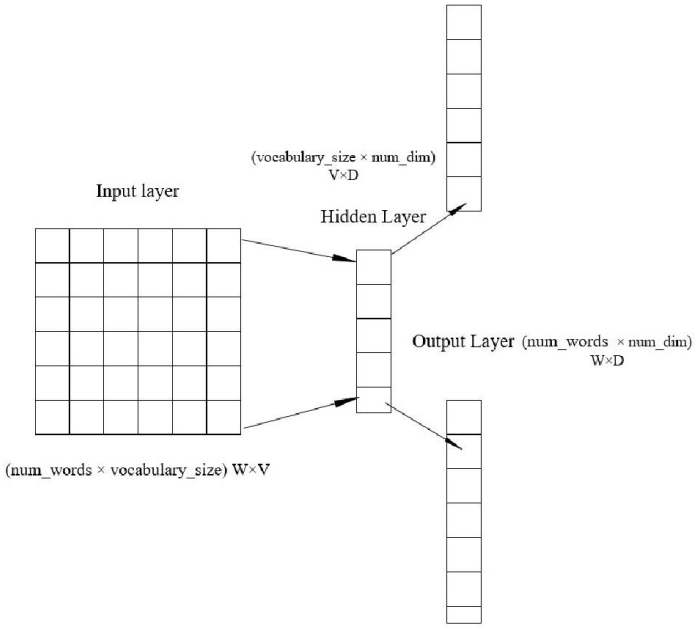

Figure 4: Implementation of embedding layer to mitigate curse of dimensionality

each sample, we utilized Principle Component Analysis (PCA) to mitigate this issue in our dataset. Also, most of our dataset vectors consisted of zero value, features sparsity is another issue that we need to address. We did so by embedding a hidden layer of neurons in the proposed model to reduce feature space of dataset – see Figure 4.

We also compared each selected OpCode in both malicious and benign dataset sample, in terms of their occurrence in dataset to obtain a forensic insight into the analysis of these IoT malwares – see Figure 5. As it can be observed, the "add" OpCode is most frequently found in both malware and normal applications. This Opcode along with "xor, mov, sub and pop" have a high frequency pattern in our dataset samples.

2.3. Deep Malware Threat Classifier

We utilized the Long Short Term Memory (LSTM) [10], a RNN structure, to build the deep learning structure and detect IoT malware samples based on their sequences of OpCodes. This is an approach suggested by Keras [6] in Weka 3.9 [12]. We also used Google Tensor Flow [1] as the

Table 1: Top 10 selected (OpCodes) features by their IG score.

opcode	**gainValue**
pushl	0.599
fildll	0.548
fcos	0.53
andl	0.527
movups	0.505
fdivrp	0.466
ret	0.472
incl	0.465
cmp	0.459
xor	0.441

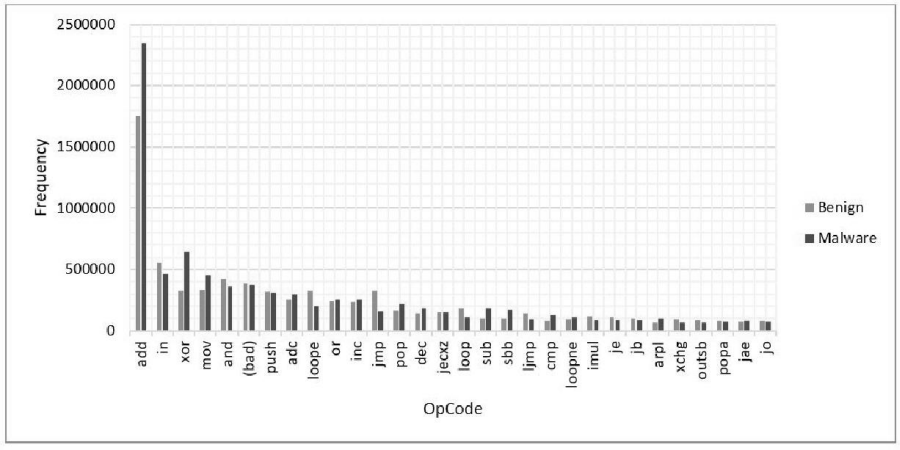

Figure 5: Frequency of occurrences for top 30 OpCodes in collected dataset.

backend structure of the deep learning approach and Scikit-learn [27] as the machine learning library to perform model evaluation tasks. Since LSTM structure is capable of learning dependencies between given data, it can be used in OpCodes sequence based learning too. Although LSTM structure is a repeating blockchain similar to the RNN architecture, it only has four neural networks [11]. In the LSTM structure, each memory block contains the following equations:

$$i_i = (U_i h_{(t-1)} + W_i x_t + b_i) \tag{2}$$
$$f_i = (U_f h_{(t-1)} + W_f x_t + b_f) \tag{3}$$
$$o_t = (U_o h_{(t-1)} + W_o x_t + b_o) \tag{4}$$
$$c_t = f_i * c_{(i-1)} + i_t * tanh(U_c h_{(t-1)} + W_c x_t + b_c) \tag{5}$$
$$h_t = o_t * tanh(c_t) \tag{6}$$

In the above equations, i_i, f_i and o_i denote i^{th} input, forget, output gates, respectively within $(n \times d)$ vector. c_t is the $(n \times d)$ cell state in t^{th} timestamp. h_t is the $(n \times d)$ activation of hidden unit in t time in Equations 4 and 5. x_t is $(l \times d)$ vector, tanh denotes the hyperbolic tangent function and $*$ operator is the point-wise (Hadamard) multiplication. U and W are the respective weight matrix in each cell, and b is the bias parameter.

We also used bidirectional neural networks (BNN) [35] instead of regular RNN neuron structure. BNN basically splits the regular RNN into two directions as follows: the forward states are used for positive time direction and the other direction (i.e. backward states) is used for negative time direction. BNN structure can be trained as a regular RNN due to the lack of interactions between the two existing directions. However, in the back propagation, additional computations are required to update neuron weights - see Figure 6.

3. Findings

For the evaluation, we built three LSTM models with different configurations – see Table 2. We denoted data set D as $D = \{S_1, S_2, S_3, , S_n\}$, and each sample exists in the dataset defined as S. Every sample has many sets of Opcode, $S = \{o_1, o_2, o_3, , o_n\}$. We also provided an Opcode dictionary I, where each Opcode was mapped into an integer index $I_s = \{i_1, i_2, i_3, , i_4\}$. Then, we set a window size for an OpCode sequences as $W_j = \{w_1, w_2, w_3, , w_j\}$, where

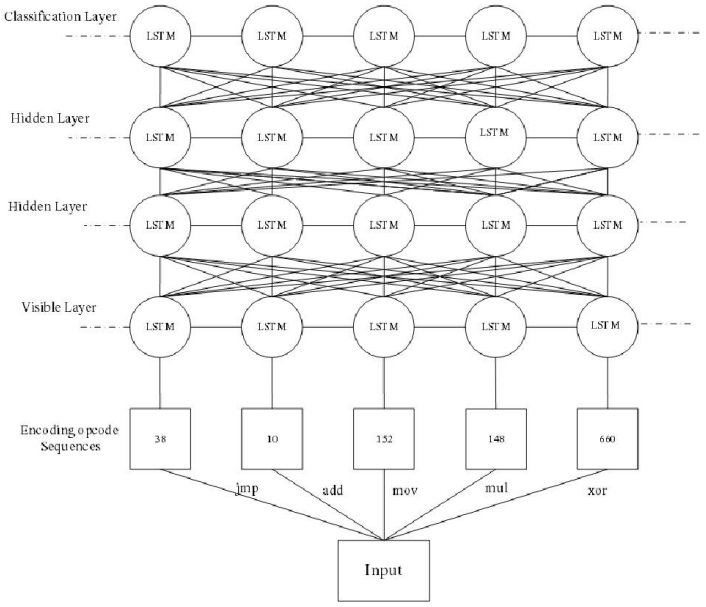

Figure 6: LSTM structure for IoT malware detection

w_j is a sequence of Opcode with length of j. We defined different window size (100-250) for each sample Opcodes sequence, as shown in Figure 8. Due to the size of our dataset, we used 1-55 batch size (min=1, max = one tenth) for feeding each model to find the optimum parameter. The batch size is the number of samples, which pass through neural network in each propagation. A very small batch size may affect the training time due to network convergence upon weight updates. On the other hand, a larger batch size may lead to over-fitting [21]. We used Adam as the weight updating algorithm [17] in our configuration. Adam is an implementation of the scholastic gradient descent that does not require tuning of its parameters [18]. To avoid overfitting (a common phenomena in deep neural network model), we applied the dropout technique. Another issue with the neural network that can result in overfitting is the limited size of training data (similar to our case) [36]. By omitting some units of the model temporarily drop from the network with a fixed probability (and in our case, the optimum parameter is 0.2), our approach achieved a 94% detection rate in unseen samples.

Finally, the following common performance indicators were used for evaluating the performance of the classifiers:

Table 2: Model configurations used in the evaluations

Hyper Parameter	LSTM-1	LSTM-2	LSTM-3
Depth	1	2	3
Bidirectional	True	True	True
No. of neurons	64	192	320
Weight updating algorithm	Adam	Adam	Adam
Dropout rate	0.2	0.2	0.3
Epochs	100	100	100
Weight regularization	None	None	None
Windows size	100	150	100
Batch size	48	46	49

Table 3: Possible parameter values for the model configurations

Hyper Parameter	Possible values
Depth	1, 2, 3
Bidirectional	True,False
No. of neurons	1-320
Weight updating algorithm	Adam
Dropout rate	0 0.5 (0.1 increments)
Epochs	1-100
Weight regularization	None
Windows size	100-250
Batch size	1-55

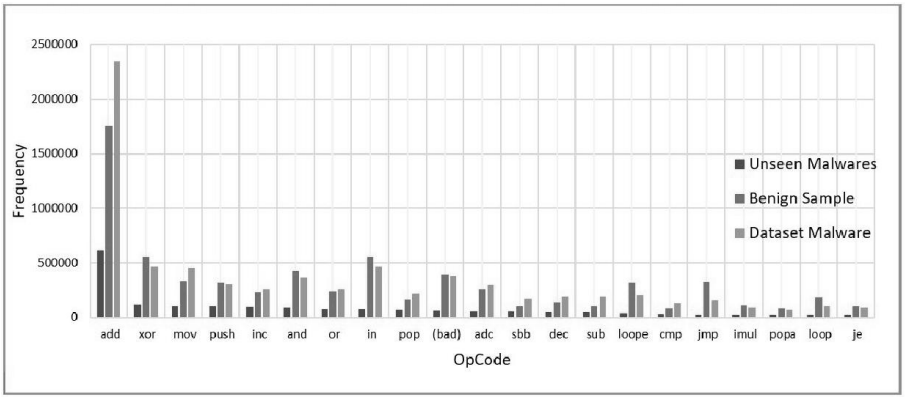

Figure 7: OpCodes frequency of unseen malwares against collected dataset samples

True Positive (TP): ratio of benign files classified as benign;
True Negative (TN): ratio of malware correctly detected as malware;
False Positive (FP): ratio of malware identified as benign; and
False Negative (FN):: ratio of benign files classified as malware.

We then computed the accuracy (ACC) using the following equation:

$$ACC = \frac{TP + TN}{FN + TP + FP + TN} \tag{7}$$

We utilized 10-fold Cross Validation (CV) on 100 epochs for each configuration and also used 100 malware samples' OpCodes not previously used in the training to evaluate the utility of our approach. The OpCodes of unseen samples and dataset samples had different distribution, and the detection findings was reasonable –see Figure 7. Findings demonstrated that the second configuration (LSTM-2) with two layers of LSTM architecture had an optimum average accuracy of 97%. Figure 9 presents the three LSTM configuration models' accuracy rate with a 10-fold CV. Figure 10 is a comparative summary, and as observed from Figure 12 and Table 5, the second configuration outperformed the other approaches (i.e. achieved 94% accuracy in classification of new malware samples). We also examined different window sizes to obtain the optimum parameter for classify the samples. Figure 11 shows the different windows size within their classification result.

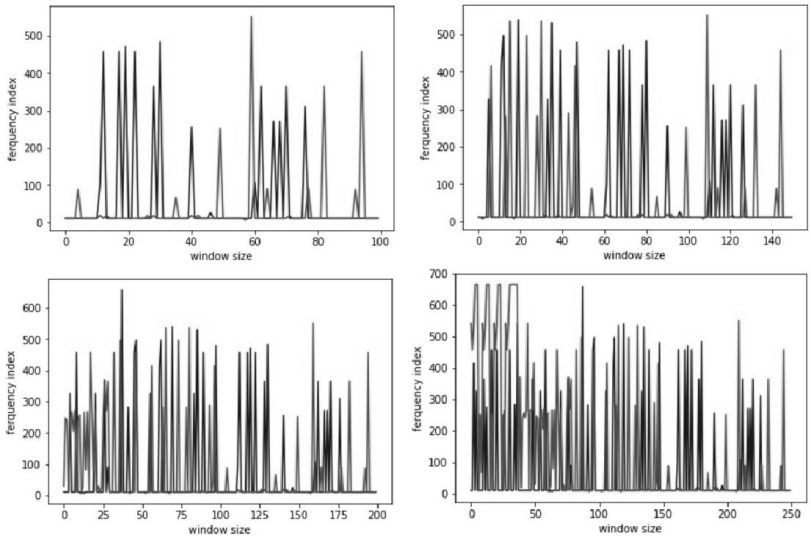

Figure 8: Two different input sequences windows

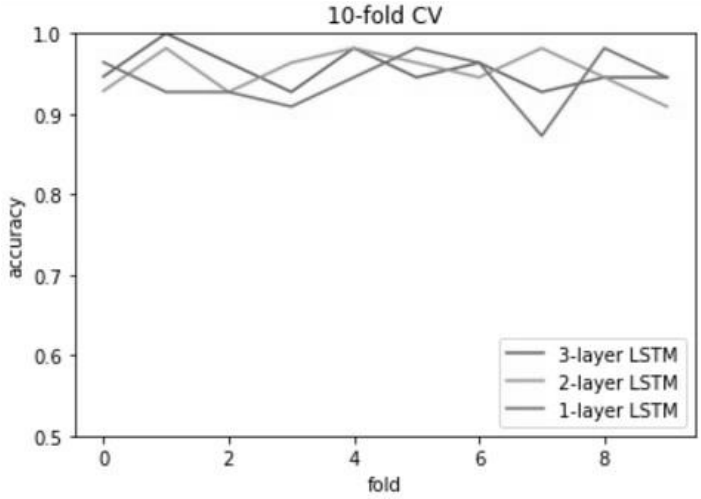

Figure 9: Accuracy of three LSTM models configurations

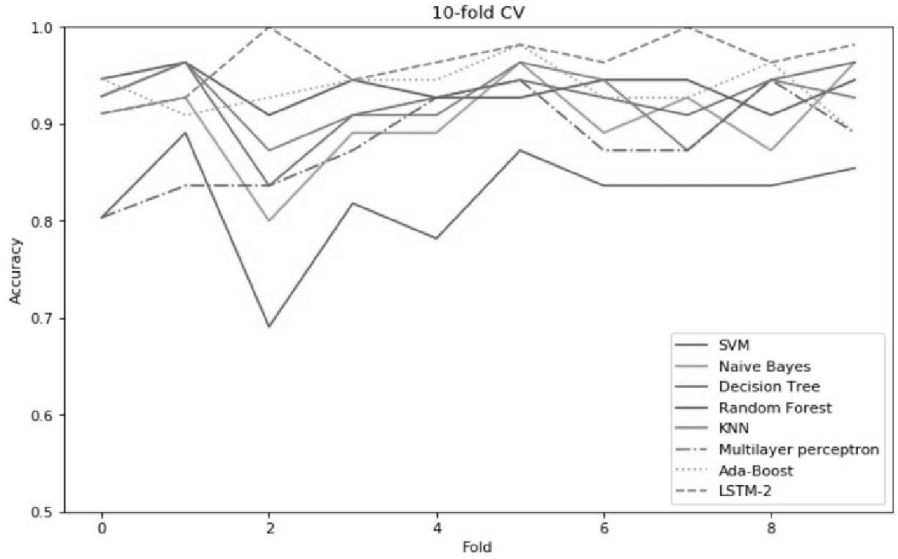

Figure 10: Conventional machine learning classifiers vs the second LSTM threat hunting configuration

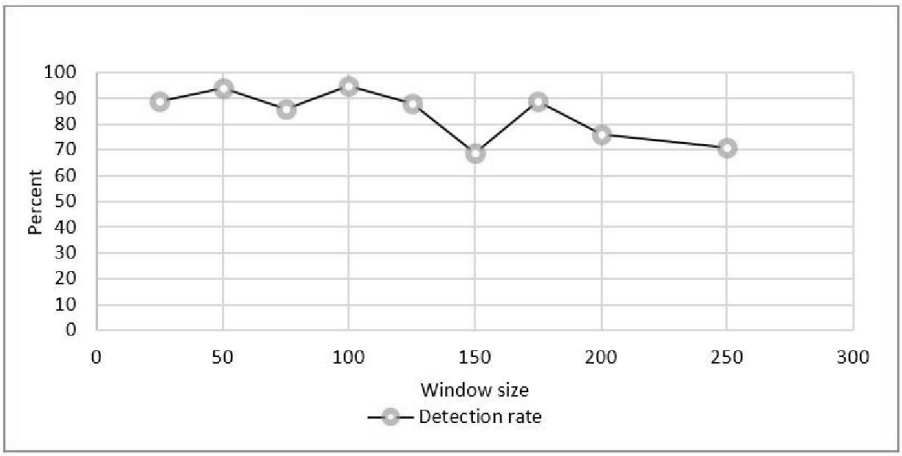

Figure 11: Detection rate of the optimum model on different windows sizes

Table 4: Accuracy of conventional machine learning classifiers and LSTM models by 10-fold CV: A comparative summary

Classifier	Accuracy (%)
RandomForest	92.37
SVM	82.21
NaiveBayes	90.37
MLP	88
KNN	94
AdaBoost	93.64
DecisionTree	92.36
LSTM-1	94.54
LSTM-2	**98.18**
LSTM-3	96.36

Table 5: Accuracy of conventional machine learning classifiers and LSTM models on new malware: A comparative summary

Classifier	Accuracy (%)
RandomForest	87.84
SVM	72.12
NaiveBayes	87.51
MLP	59.07
KNN	94
AdaBoost	84.35
DecisionTree	89.36
LSTM-2	**94**

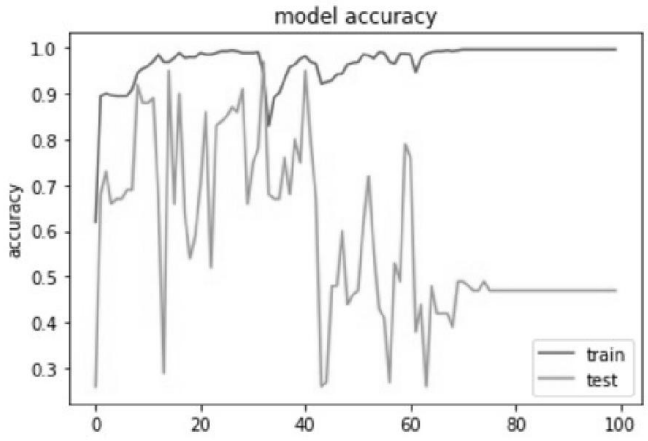

Figure 12: Accuracy of the best LSTM configuration against unseen malware

4. Conclusion

IoT-based systems will be increasingly commonplace, with the range and types of IoT devices rapidly increasing in the foreseeable future. Thus, it is important to secure such devices, say against malware.

In this paper, we proposed an approach that uses LSTM structures to hunt IoT malware based on their OpCodes sequence. We then evaluated our approach using ARM-based IoT applications' execution OpCodes, and achieved a detection accuracy of 98% against IoT malware not used in the training.

While the findings appeared promising, there are many potential extensions to this work. Firstly, the dataset we used is small in comparison to the real-world cyberthreats. Thus, future research includes implementing the proposed approach in a real-world environment and evaluating its effectiveness in identifying both known malware and new malware. We should also explore and design deep learning techniques that can be used to increase the accuracy, speed and scalability of IoT malware detection, particularly against a wider range of IoT devices with different specifications.

Acknolwedgement

The views and opinions expressed in this article are those of authors alone and not the organizations with whom authors are or have been associated or supported. We thank VirusTotal for graciously providing us with a private

API key to access their data to prepare our dataset. This work is partially supported by the European Council International Incoming Fellowship (FP7-PEOPLE-2013-IIF) grant, and the last author is supported by the Cloud Technology Endowed Professorship.

Reference

[1] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mane, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viegas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X., Mar. 2016. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. arXiv:1603.04467 [cs]ArXiv: 1603.04467.
URL `http://arxiv.org/abs/1603.04467`

[2] Azmoodeh, A., Dehghantanha, A., Conti, M., Choo, K.-K. R., Aug. 2017. Detecting crypto-ransomware in IoT networks based on energy consumption footprint. Journal of Ambient Intelligence and Humanized Computing, 1–12.
URL `https://link.springer.com/article/10.1007/s12652-017-0558-5`

[3] Azmoudeh, A., Dehghantanha, A., Choo, K. K. R., 2017. Robust Malware Detection for Internet Of (Battlefield) Things Devices Using Deep Eigenspace Learning in press.

[4] Bengio, Y., Ducharme, R., Vincent, P., Jauvin, C., 2003. A neural probabilistic language model. Journal of machine learning research 3 (Feb), 1137–1155.

[5] Brash, D., 2016. The arm architecture version 6.

[6] Chollet, F., 2015. Keras.

[7] Conti, M., Dehghantanha, A., Franke, K., Watson, S., 2018. Internet of Things security and forensics: Challenges and opportunities. Elsevier.

[8] Demontis, A., Melis, M., Biggio, B., Maiorca, D., Arp, D., Rieck, K., Corona, I., Giacinto, G., Roli, F., 2017. Yes, Machine Learning Can Be More Secure! A Case Study on Android Malware Detection. IEEE Transactions on Dependable and Secure Computing PP (99), 1–1.

[9] Ding, Y., Dai, W., Yan, S., Zhang, Y., Jul. 2014. Control flow-based opcode behavior analysis for Malware detection. Computers & Security 44 (Supplement C), 65–74.
URL http://www.sciencedirect.com/science/article/pii/S0167404814000558

[10] Gers, F. A., Schmidhuber, J., Cummins, F., 1999. Learning to forget: Continual prediction with LSTM.

[11] Greff, K., Srivastava, R. K., Koutnk, J., Steunebrink, B. R., Schmidhuber, J., Oct. 2017. LSTM: A Search Space Odyssey. IEEE Transactions on Neural Networks and Learning Systems 28 (10), 2222–2232.

[12] Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I. H., 2009. The WEKA data mining software: an update. ACM SIGKDD explorations newsletter 11 (1), 10–18.

[13] Homayoun, S., Dehghantanha, A., Ahmadzadeh, M., Hashemi, S., Khayami, R., 2017. Know Abnormal, Find Evil: Frequent Pattern Mining for Ransomware Threat Hunting and Intelligence. IEEE Transactions on Emerging Topics in Computing.

[14] Indyk, P., Motwani, R., 1998. Approximate nearest neighbors: towards removing the curse of dimensionality. In: Proceedings of the thirtieth annual ACM symposium on Theory of computing. ACM, pp. 604–613.

[15] Joachims, T., 1996. A Probabilistic Analysis of the Rocchio Algorithm with TFIDF for Text Categorization. Tech. rep., Carnegie-mellon univ pittsburgh pa dept of computer science.
URL http://www.dtic.mil/docs/citations/ADA307731

[16] Ki, Y., Kim, E., Kim, H. K., Jun. 2015. A Novel Approach to Detect Malware Based on API Call Sequence Analysis. International Journal of Distributed Sensor Networks 11 (6), 659101.
URL https://doi.org/10.1155/2015/659101

[17] Kingma, D., Ba, J., 2014. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

[18] Kingma, D. P., Ba, J., 2014. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

[19] Kolias, C., Kambourakis, G., Stavrou, A., Voas, J., 2017. DDoS in the IoT: Mirai and Other Botnets. Computer 50 (7), 80–84.

[20] Kolosnjaji, B., Zarras, A., Webster, G., Eckert, C., Dec. 2016. Deep Learning for Classification of Malware System Call Sequences. In: AI 2016: Advances in Artificial Intelligence. Lecture Notes in Computer Science. Springer, Cham, pp. 137–149.
URL https://link.springer.com/chapter/10.1007/978-3-319-50127-7_11

[21] LeCun, Y. A., Bottou, L., Orr, G. B., Mller, K.-R., 2012. Efficient backprop. In: Neural networks: Tricks of the trade. Springer, pp. 9–48.

[22] Mitchell, T. M., 1997. Machine learning. 1997. Burr Ridge, IL: McGraw Hill 45 (37), 870–877.

[23] Narudin, F. A., Feizollah, A., Anuar, N. B., Gani, A., 2016. Evaluation of machine learning classifiers for mobile malware detection. Soft Computing 20 (1), 343–357.

[24] Nia, A. M., Jha, N. K., 2017. A Comprehensive Study of Security of Internet-of-Things. IEEE Transactions on Emerging Topics in Computing PP (99), 1–1.

[25] O'Kane, P., Sezer, S., McLaughlin, K., Jan. 2014. N-gram density based malware detection. In: 2014 World Symposium on Computer Applications Research (WSCAR). pp. 1–6.

[26] O'Kane, P., Sezer, S., McLaughlin, K., Im, E. G., Mar. 2013. SVM Training Phase Reduction Using Dataset Feature Filtering for Malware Detection. IEEE Transactions on Information Forensics and Security 8 (3), 500–509.

[27] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, d., Oct. 2011. Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research 12, 28252830.
URL http://jmlr.csail.mit.edu/papers/v12/pedregosa11a.html

[28] Puthal, D., Nepal, S., Ranjan, R., Chen, J., 2016. Threats to networking cloud and edge datacenters in the internet of things. IEEE Cloud Computing 3 (3), 64–71.

[29] Rhode, M., Burnap, P., Jones, K., Aug. 2017. Early Stage Malware Prediction Using Recurrent Neural Networks. arXiv:1708.03513 [cs]ArXiv: 1708.03513.
URL http://arxiv.org/abs/1708.03513

[30] Runwal, N., Low, R. M., Stamp, M., May 2012. Opcode graph similarity and metamorphic detection. Journal in Computer Virology 8 (1-2), 37–52.
URL https://link.springer.com/article/10.1007/s11416-012-0160-5

[31] Sam Edwards, Ioannis Profetis, Oct. 2016. Hajime: Analysis of a decentralized internet worm for IoT devices. Tech. Rep. 1, Rapidity Networks.
URL https://security.rapiditynetworks.com/publications/2016-10-16/hajime.pdf

[32] Santos, I., Brezo, F., Nieves, J., Penya, Y. K., Sanz, B., Laorden, C., Bringas, P. G., Feb. 2010. Idea: Opcode-Sequence-Based Malware Detection. In: Engineering Secure Software and Systems. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, pp. 35–43.
URL https://link.springer.com/chapter/10.1007/978-3-642-11747-3_3

[33] Santos, I., Brezo, F., Ugarte-Pedrero, X., Bringas, P. G., May 2013. Opcode sequences as representation of executables for data-mining-based unknown malware detection. Information Sciences 231 (Supplement C), 64–82.

URL http://www.sciencedirect.com/science/article/pii/S0020025511004336

[34] Saxe, J., Berlin, K., Oct. 2015. Deep neural network based malware detection using two dimensional binary program features. In: 2015 10th International Conference on Malicious and Unwanted Software (MALWARE). pp. 11–20.

[35] Schuster, M., Paliwal, K. K., 1997. Bidirectional recurrent neural networks. IEEE Transactions on Signal Processing 45 (11), 2673–2681.

[36] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R., 2014. Dropout: A simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research 15 (1), 1929–1958.

[37] Team, V., 2017. VirusTotal-Free Online Virus, Malware and URL Scanner.

[38] Xiao, L., Li, Y., Huang, X., Du, X., Oct. 2017. Cloud-Based Malware Detection Game for Mobile Devices with Offloading. IEEE Transactions on Mobile Computing 16 (10), 2742–2750.

[39] Yuan, Z., Lu, Y., Wang, Z., Xue, Y., 2014. Droid-Sec: deep learning in android malware detection. In: ACM SIGCOMM Computer Communication Review. Vol. 44. ACM, pp. 371–372.
URL http://dl.acm.org/citation.cfm?id=2631434

[40] Yuxin, D., Siyi, Z., Jul. 2017. Malware detection based on deep learning algorithm. Neural Computing and Applications, 1–12.
URL https://link.springer.com/article/10.1007/s00521-017-3077-6