Industrial Engineering Journal
ISSN: 0970-2555
Volume : 54, Issue 6, No.3, June : 2025

LEVERAGING LONG SHORT-TERM MEMORY NETWORKS FOR MALICIOUS CODE DETECTION IN EXECUTABLE FILES

Dr. M. Deepa, Assistant Professor, Dept.Of Computer Science, Pavai Arts and Science College for Women, Namakkal.
Dr. D. Selvanayagi, Guest Lecturer, Dept.Of Computer Science, Government Arts and Science College, Modakkurichi.
Dr. R. Anubhama, Guest Lecturer, Dept.Of Computer Science, Government Arts and Science College, Modakkurichi.

ABSTRACT
Due to the widespread use of information and communication technology (ICT) applications in our daily lives, malicious software threats and their detection are becoming a crucial aspect of information security. One of the most challenging issues in the design and development of antimalware systems is malware identification. In order to detect polymorphic and metamorphic malware quickly, dynamic analysis methods must be developed. demonstrates how to use Long Short-Term Memory (LSTM) to analyze trace data and find dangerous code. Models were developed for the execution traces of both malicious and benign Portable Executable (PE) files. We constructed our first dataset using the execution trace output obtained from dynamic analysis of the PE file. The suggested solution is more than 98% accurate, according to extensive testing with a data set that includes both benign and malicious programmers.

Keywords:
Deep Learning, LSTM, Malwares, Artificial intelligence.

I. Introduction
Malware is software designed to carry out destructive tasks, such as obtaining root access, stealing private data, and rendering the target machine inoperable. Meanwhile, the rapid growth of the software industry and the Internet has led to the emergence of a broad variety of malware. More than 774 million malware samples have been found in the previous three quarters, representing a nearly 34% rise. Over time, malware—also referred to as malware—increases. Malware detection is therefore a significant and intriguing subject. Malware detection techniques have been thoroughly researched. Static anti-virus software that relies on signatures is frequently employed to detect malware because of its limited capacity to recognize novel threats. If malware has been encrypted, disguised, or packed to evade detection, it can readily evade detection by signature-based security methods.
Zero-day malware can get beyond this detection mechanism. Scanning of the system in real time The malware detection program lurenjie17@mails.ucas.ac.c is more effective than cloaking techniques. A safe and regulated environment, including virtual computers, emulators, sandboxes, etc., is necessary for dynamic behavior-based malware detection techniques [4] [5]. Conducting behavioral analysis using information gathered from interactions with the environment, such as API and DLL calls, is the next step. Despite their significant research, these methods are useless when used to large data sets [6]. It takes a lot of effort and time to prevent dynamic behavior-based malware detection technologies from compromising the operating system. Machine learning-based methods for detecting malware have been developed in recent years. The first publication of the data mining-based malware detection technique was in ref [7]. It employs three different kinds of static features: A text string, a byte sequence, and a PE header are used to spot malware. Kolter and Maloof [8] evaluated the effectiveness of naive Bayes, decision trees, and support vector machines for virus identification using n-grams as opposed to byte sequences. In recent years [9,10], malware has also been detected using artificial neural networks [9]. There are also new methods for detecting malware.

Malware can be detected by image processing in [11] and [12]. However, the previous attempt was successful enough in terms of malware detection. The machine learning classifier is trained by manually analyzing malicious code and comparing it with features extracted from the code itself. An innovative and efficient approach to determine if a Windows executable is malware has been proposed in this study to reduce engineering costs for artificial features. The assembly format files of the executables need to be recovered by splitting them first with IDA Pro. We need a method to take the opcode string out of each file in assembly format. Then, word embedding techniques [13] and long-term memory (LSTM) [14] are used to comprehend the feature vector representation of the opcode and automatically learn the opcode sequence patterns of the malware. After the second LSTM layer, we add an average aggregation layer to improve the immutability of the local feature representation. We ran a series of tests on a dataset consisting of 969 malicious files and 123 benign files to see if our strategy worked. MalwareXiv: 1906.04593v1 [cs.CR] 10 June 2019 Detection performance was evaluated during the experimental period and comprehensive performance comparison with other similar studies was conducted. The assessment results demonstrate that our suggested method can identify malware with an average AUC of 0.99 and classify malware with an average AUC of 0.987.
Malware analysis techniques are usually categorised into two major groups: static and dynamic. In static analysis, an application is observed for malicious patterns without execution. These data files or applications are decrypted and disassembled into feature vectors. Feature vectors characterise the essential features and format information of the file potentially containing the malicious pattern. Anti-malware solutions detect malicious patterns by analysing these feature vectors. Conventional malware detectors use a signature-based detection method. These detectors are stored with a vast database of malware signatures (malicious code patterns). They decode the suspicious file and match its extracted static features with the stored malware signatures. Malware variants can easily undermine conventional anti-malware solutions.

II. Literature Review

Unlike static analysis, dynamic analysis-based malware analysis techniques are more resistant to obfuscation. Dynamic analysis was used to classify API requests that took less than five minutes in [1]. The AUC, a measure of quality, was calculated using 170 samples and yielded a score of 0.96. Separately collected samples of benign and malicious software were used to build a response network using the API call feature set. It performs well compared to previous methods, but lacks research on execution speed, which is essential for real-time implementations. ESN and RNN tests were performed in [3] to learn the language of the malware. ESN performed better than RNN in the majority of studies. Tests were performed in [4] to determine when to stop running viruses on network traffic, as shown in [5]. Conventional procedures require 67% more time than this method. With long-chain API calls as a feature, the RNN and its long-term short-term memory (LSTM) and CNN versions were used for malware classification in [6]. The main problem with current methods is that they take a long time to test the behavior of the system during operation. It has been used in [7] to classify malware using a system call sequence in the form of a hybrid CNN and RNN. SVM and Hidden Markov Model were previously used to obtain these system calls, but dynamic analysis was used to obtain them and it was found to be more efficient (HMM). The biggest problem, however, is the lack of discussion about the relevance of runtime in real-time virus detection. Using RNN and two datasets, [13] proposes a technique. In addition, they tested the performance of several wellknown classical machine learning classifiers. With a run time of 5 seconds, they claimed an accuracy of 94%. Analysis-based static, dynamic, and mixed malware detection methods have been the subject of several investigations. HMM has been used for both static and dynamic feature set analysis and to compare detection rates for a large number of malware types in [8]. Overall, they found that dynamic analysis had the best detection rate WindowsDynamic-Brain-Droid (WDBD) was the model we developed to compare and contrast several traditional machine learning algorithms (MLA) and deep learning architecture to determine

which technique is best for Windows Malware Classification. The number of malware and benign patterns in our dataset varies with runtime, which is why we used two separate datasets.

III. Proposed Methodology

The main goal of Long-Term Memory (LSTM), a specialised RNN architecture, is to resolve the leak slope problem or at the very least lessen the impact of the gradability problem. performance of computers is a leak slope. Nodes in the LSTM neural network have concealed state from the preceding phase, similar to RNN. The node, a typical LSTM unit, has a better structure than an RNN, which is crucial for supplying long-term memory by lessening the impact of the leak gradient. A standard LSTM unit generates an output value from an input value. The produced output value of the current cell and the prior cell's cell state value—which will be detailed in more detail in the coming paragraphs—are both used during this process. The following three functions can be carried out by an LSTM unit. Erasing undesired information from the tile's present state by using the Forgotten Gate Through the front door, add new details to the current state of the cell. Output the condition of the current cell through the output port.

An typical LSTM unit's interior is straightforward and practical, as seen in Figure 1. By disregarding the input of this current cell (Xt) and its output (ht1) of the preceding cell (ht1), the sigmoid function may be utilised to create an output between 0 and 1 on the left side of a cell. The current cell state is updated and created by multiplying this value, ft, by the cell state that was previously displayed, Ct1. A value that travels across cells to transfer information between them is essentially what a cell state is. The forgetting gate is a component of the unit that uses a multiplier operation to determine which information will be forgotten and how much will be remembered in succeeding cells [26]. In the centre of the cell, there are two sigmoid functions, a tan h function, and their output is multiplied together. The output of the previous cell, ht1, as well as the input from the current cell, Xt, are both used as inputs for the sigmoid function in this. The output value of this sigmoid, in contrast to the sigmoid function used in the monitoring process, will be used to signify what new value should be added to the existing state of the cell. An array of possible values is created by the tan h function, which may or may not be added to the cell's present state in the future. By dividing the output of the cell's sigmoid by the output of tan h C t, one may determine the values that should be added to the cell's present state. The output of the sigmoid it is multiplied by the output of the tan h C t to achieve this. The final current cell state is produced by updating the prior cell state Ct1, which was altered by the forget gate, with fresh data from the input via an add operation. The front gate is the name given to this portion of the LSTM unit as a result [26].

An array of possible values is created by the tan h function, which may or may not be added to the cell's present state in the future. By dividing the output of the cell's sigmoid by the output of tan h C t, one may determine the values that should be added to the cell's present state. The output of the sigmoid it is multiplied by the output of the tan h C t to achieve this. The final current cell state is produced by updating the prior cell state Ct1, which was altered by the forget gate, with fresh data from the input via an add operation.

Industrial Engineering Journal
ISSN: 0970-2555
Volume : 54, Issue 6, No.3, June : 2025

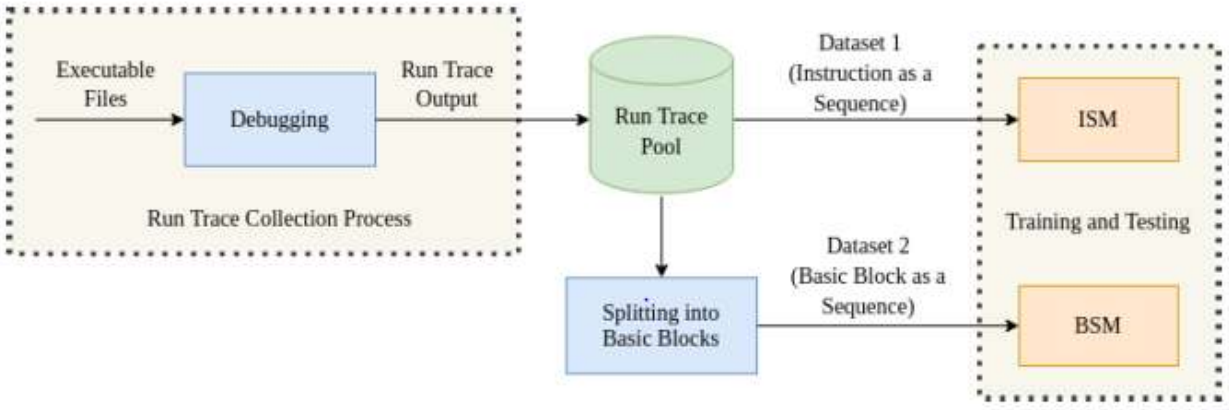

Fig 1: Data processing Pipeline

The front gate is the name given to this portion of the LSTM unit as a result [26]. In general, the current LSTM cell creates the output by updating the cell state of the previous cell while also receiving the output and cell state of the previous cell in addition to the current cell's input. The sequential internal nature of the LSTM architecture gives higher efficiency when handling data that comprises prolonged sequences of events than the basic RNN architecture.

IV. Result & Discussion

We designed and developed two datasets for the study: the sequences of instructions (for the ISM model) and the basic blocks (for the BSM model). We obtained native x86 PE (Portable Executable) files from Windows Operating Systems (Microsoft Windows 8.1 Pro (OS Build 9600), Microsoft Windows 10 Pro 19.09 (OS Build 18363.418), and Commando VM v-2.0 [30]). Malicious executables were downloaded from the VirusShare website [31]. Since we aim to detect malware, we randomly chose the malicious samples, including various malware types, such as virus, worm, and trojan. We conducted a total of 16 experiments for the first model, ISM by manipulating 4 values for the sequence length, 3 values for dropout rate, 5 values for optimizer and 4 values for the number of LSTM nodes. The resulting number of correctly and incorrectly classified samples are shown in a confusion matrix (Table 2). The number of true negatives TN in the confusion matrix refers to correctly recognized instructions as benign instructions. In contrast, the number of true positives TP refers to correctly recognized instructions as malicious instructions. The number of false positives FP shows benign instructions recognized as malicious, whereas the number of false negatives FN shows malicious instructions recognized as benign. The true positive rate TPR is calculated by (1) as 92.19% and the false positive rate FPR is calculated by (2) as 18.34%. Accuracy rate ACC is calculated by (3) as 87.51%.

Here fig-2representsthe precision of the distinction between benign and malicious samples. Additionally, the graph contrasts the proposed LSTM model with the existing ANN model. The ANN model is unable to accurately differentiate between dangerous and benign components. Because malicious code consists of a series of operations, ANN is unable to remember code sequences. However, because the LSTM model has a memory unit, it can provide improved accuracy as the number of epochs increases. While the number of ages is increased, ANN fails to offer improved accuracy at the same time. The dataset (Drebin-215) also contains 215 functionalities from 15,036 app samples, of which 9476 were malware samples from the Drebin project and the remaining 5560 were safe samples [4]. The dataset (Drebin-215) also contains 215 data points from 15,036 app specimens, of which 9476 were deemed benign and the remaining 5560 were deemed malicious. The public can access the Drebin samples for free and they are frequently used by scientists

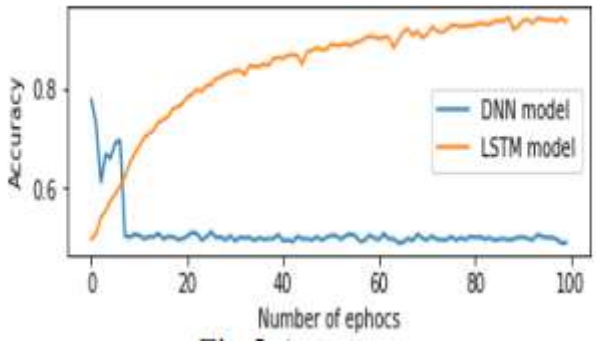

Fig 2: Accuracy Measurement

Precision is defined as TP/(TP+FP). Figure 3 depicts the classification of harmful samples and good samples. The graph also compares the proposed LSTM model to the current ANN model. The ANN method falls short of offering a more precise classification of damaging and beneficial cases. ANN is unable to recall harmful code sequences since it comprises of a number of operations. However, because the LSTM model has a memory component, it can provide better precession as the number of epochs rises. At the same time, as the number of epochs rises, ANN fails to provide. In [27] and [26], two LSTM architectures were reported, which used two and four hidden LSTM layers, respectively. The proposed ISM method achieved similar accuracy rates with those two models. Our BSM method improved the accuracy rate by approximately %10 by achieving %99.26. Our LSTM architecture included a single hidden LSTM layer, which made the architecture less complicated. Finally, we developed a novel method to detect malicious code by extending previously proposed methods in several aspects. Our method is capable of detecting malware using obfuscation techniques by employing dynamic analysis since run trace outputs are dynamically generated. The ability of neural networks to adapt to data changes makes our method robust against malware obfuscation methods. In addition, working on assembly code without preprocessing in ISM and with small preprocessing in BSM keeps the time minimum spent for preprocessing. Also, using a single LSTM layer reduces the need for resources required for training the model.

V. Conclusion

Malware detection methods have been evolving since the advent of information technologies that profoundly affected people's lives. Faster virus detection methods must be developed due to the information technology sector's exponential expansion and more accurate. Malware writers' anti-detection tactics, such as obfuscation methods, also necessitate the use of complex and completely automated malware detection systems. Artificial intelligence (AI)-based technologies are the most promising options for developing more reliable malware detection methods in light of these requirements. In early AI-based research, machine learning (ML) classification algorithms were widely used to differentiate between data generated by beneficial and dangerous software. However, ML classification algorithms do not provide completely automated techniques as feature extraction and selection need time and effort.

References

[1] Tobiyama, S., Yamaguchi, Y., Shimada, H., Ikuse, T., & Yagi, T. (2016, June). Malware detection with adeepneuralnetworkusingprocessbehavior.InComputerSoftwareandApplicationsConference(COMPSAC),2016IEEE40thAnnual (Vol.2, pp.577-582). IEEE.

[2] Huang, W., & Stokes, J. W. (2016, July). MtNet: a multi-task neural network for dynamic malwareclassification.InInternationalConferenceonDetectionofIntrusionsandMalware,andVulnerabilityAssessment(pp. 399-418).Springer, Cham.

[3] Pascanu,R.,Stokes,J.W.,Sanossian,H.,Marinescu,M.,&Thomas,A.(2015,April).Malwareclassification with recurrent networks. In Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEEInternationalConference on(pp. 1916-1920).IEEE.

[4] Shibahara, T., Yagi, T., Akiyama, M., Chiba, D., & Yada, T. (2016, December). Efficient dynamicmalware analysis based on network behavior using deep learning. In Global Communications Conference(GLOBECOM),2016IEEE(pp. 1-7). IEEE.

[5] Kolosnjaji, B., Zarras, A., Webster, G., & Eckert, C. (2016, December). Deep learning for classificationof malware system call sequences. In Australasian Joint Conference on Artificial Intelligence (pp. 137-149).Springer,Cham.

[6] Raff,E.,Zak,R.,Cox,R.,Sylvester,J.,Yacci,P.,Ward,R.,...&Nicholas,C.(2018).Aninvestigationof byte n-gram features for malware classification. Journal of Computer Virology and Hacking Techniques,14(1), 1-20.

[7] Anderson, H. S., & Roth, P. (2018). EMBER: An Open Dataset for Training Static PE Malware MachineLearningModels. arXivpreprintarXiv:1804.04637.

[8] Damodaran, A., Di Troia, F., Visaggio, C. A., Austin, T. H., & Stamp, M. (2017). A comparison of static,dynamic, and hybrid analysis for malware detection. Journal of Computer Virology and Hacking Techniques,13(1), 1-12.

[9] Nataraj, L. (2015). A signal processing approach to malware analysis. University of California, SantaBarbara.

[10] Nataraj, L., Kirat, D., Manjunath, B. S., & Vigna, G. (2013, December). Sarvam: Search and retrieval ofmalware.InProceedingsoftheAnnualComputerSecurityConference(ACSAC)WorkshoponNext GenerationMalwareAttacksandDefense (NGMAD).

[11] Nataraj, L., Yegneswaran, V., Porras, P., & Zhang, J. (2011, October). A comparative assessment ofmalware classification using binary texture analysis and dynamic analysis. In Proceedings of the 4th ACMWorkshoponSecurityandArtificialIntelligence(pp.21-30).ACM.[12]Nataraj,L.,Jacob,G.,&Manjunath,B.S.(2010). Detectingpacked executablesbased onrawbinarydata. Technical report.

[12] Farrokhmanesh, M., & Hamzeh, A. (2016, April). A novel method for malware detection using audiosignalprocessingtechniques.InArtificialIntelligenceandRobotics(IRANOPEN),2016(pp.85-91).IEEE.

[13] Nataraj, L., Karthikeyan, S., Jacob, G., & Manjunath, B. S. (2011, July). Malware images: visualizationand automatic classification. In Proceedings of the 8th international symposium on visualization for cybersecurity(p. 4). ACM.

[14] Nataraj, L., & Manjunath, B. S. (2016). SPAM: signal processing to analyze malware. arXiv preprintarXiv:1605.05280.

[15] Kirat,D.,Nataraj,L., Vigna,G.,&Manjunath,B.S.(2013, December).Signal:A staticsignalprocessingbasedmalwaretriage.InProceedingsofthe29thAnnualComputerSecurityApplicationsConference(p p. 89-98). ACM.

[16] Ray, P.P. Internet of Things for smart agriculture: Technologies practices and future direction. J. Ambient. Intell. Smart Environ. 2017, 9, 395–420.

[17] Kamienski, C.; Soininen, J.-P.; Taumberger, M.; Dantas, R.; Toscano, A.; Salmon Cinotti, T.; Filev Maia, R.; Torre Neto, A. Smart Water Management Platform: IoT-Based Precision Irrigation for Agriculture. Sensors 2019, 19, 276.

[18] Ojha, T.; Misra, S.; Raghuwanshi, N.S. Wireless sensor networks for agriculture: The state-of-the-art in practice and future challenges. Comput. Electron. Agric. 2015, 118, 66–84.

[19] Vijayan, T.; Sangeetha, M.; Kumaravel, A.; Karthik, B. Feature selection for Simple Color Histogram Filter based on Retinal Fundus Images for Diabetic Retinopathy recognition. IETE J. Res. 2020, 1–8.

[20] Lavanya, G.; Rani, C.; GaneshKumar, P. An automated low cost IoT based Fertilizer Intimation System for smart agriculture. Sustain. Comput. Inform. Syst. 2020, 28, 100300.
[21] Sivakumar, M.; Renuka, P.; Chitra, P.; Karthikeyan, S. IoT incorporated deep learning model combined with SmartBin technology for real-time solid waste management. Comput. Intell. 2021.
[22] Katarya, R.; Raturi, A.; Mehndiratta, A.; Thapper, A. Impact of Machine Learning Techniques in Precision Agriculture. In Proceedings of the 2020, 3rd International Conference on Emerging Technologies in Computer Engineering: Machine Learning and Internet of Things, ICETCE, Jaipur, India, 7–8 February 2020; pp. 1–6.
[23] Anitha, P.; Chakravarthy, T. Agricultural Crop Yield Prediction using Artificial Neural Network with Feed Forward Algorithm. Int. J. Comput. Sci. Eng. 2018, 6, 178–181.
[24] Anand, R.; Karthiga, R.D.; Jeevitha, T.; Mithra, J.L.; Yuvaraj, S. Blockchain-Based Agriculture Assistance. Lect. Notes Electr. Eng. 2021, 700, 477.
[25] Prasath, J.S.; Jayakumar, S.; Karthikeyan, K. Real-time implementation for secure monitoring of wastewater treatment plants using internet of things. Int. J. Innov. Technol. Explor. Eng. 2019, 9, 2997–3002.
[26] Srisruthi, S.; Swarna, N.; Ros, G.M.S.; Elizabeth, E. Sustainable agriculture using eco-friendly and energy efficient sensor technology. In Proceedings of the 2016 IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT), Bangalore, India, 20–21 May 2016; IEEE: Bangalore, India, 2016; pp. 1442–1446.
[27] Brodt, S.; Six, J.; Feenstra, G.; Ingels, C.; Campbell, D. Sustainable Agriculture. Nat. Educ. Knowl. 2011, 3, 1.
[28] Obaisi, A.I.; Adegbeye, M.J.; Elghandour, M.M.M.Y.; Barbabosa-Pliego, A.; Salem, A.Z.M. Natural Resource Management and Sustainable Agriculture. In Handbook of Climate Change Mitigation and Adaptation; Lackner, M., Sajjadi, B., Chen, W.Y., Eds.; Springer: Cham, Switzerland, 2022.
[29] Latake, P.T.; Pawar, P.; Ranveer, A.C. The Greenhouse Effect and Its Impacts on Environment. Int. J. Innov. Res. Creat. Technol. 2015, 1, 333–337.