Early-Stage Malware Prediction Using Deep Learning

Shivam Pandit
pandit@clemson.edu

Christin Wilson
cwils28@clemson.edu

Ebuka Johnbosco Okpala
eokpala@clemson.edu

I. Abstract

Most anti-virus systems use static analysis in analyzing an executable file by comparing the characteristics of a file to a list of known malicious files, because static malware analysis is fast and users of the system does not have to wait for several minutes for analysis to complete. While static analysis can be conducted quickly, it is easy to obscure a code such that it can't be detected by systems that employ it. The behaviour of a file on a system during its execution can be used to overcome the shortcomings of static analysis, but using behavioural data takes time and by the time the analysis is done, the malicious payload of the file might have been delivered before anything can be done about it.

In this paper we predict if a file is malicious or not using data extracted from both static and dynamic analysis. We find that CART and Random Forest achieved high accuracy of 94.95% and 94.69% respectively. We also find that an ensemble of recurrent neural networks were able to give a prediction accuracy of 91%. While this is lower than other machine learning algorithms, this could as a result of our data not being a sequential time series data that RNNs are very good at learning. To the best of our knowledge, there is currently no paper on Linux malware prediction using deep learning and other machine learning algorithms and this project can serve as a stepping stone for Linux malware prediction using deep learning.

II. Introduction

The anti-virus industry and academia has focused more on developing both software and tools used in analyzing and detecting malware specifically windows malware. This is likely because of the popularity of windows operating system and its market share [1]. The rapid rise of internet of things is causing the malware detection landscape to move towards Linux based systems because most of these embedded devices that has the ability to communicate with each other over the internet is based on the Linux operating system.

The companies that are manufacturing these devices are always in a rush to be the first to release their products to the public, this leads to the company ignoring the security of these devices. Attackers are generating and customizing new malware for these devices because of the obvious vulnerabilities found in them and for the fact that they can be turned into bot-nets to carryout denial of service attacks. Malicious Linux programs has been the least concern for the anti-virus industry, and it is not until 2014 that VirusTotal a free tool that runs files through 60 anti-virus engines recognized Linux malware as a growing concern [2]. While Linux malware is still not as complex as its windows counterpart, there is need for more automatic malware detection tools to process the rapid rise of Linux malware. The common way which most anti-virus systems detect malware is to look at the features of the code of an incoming sample and compare it to a list of known malware signatures. This type of detection using static data is not robust to code obfuscation, malware authors can obscure their code to enable their programs avoid being detected by these anti-virus systems.

Previous research has shown that behavioural analysis, the execution of a file in a virtual environment and examining its activity on the machine as it executes could be used to fill the gap of static analysis. That malware cannot avoid to leave a trace of its activity because of the actions needed for it to achieve its goal of infecting a system, is the assumption of behavioural analysis method. However, executing the sample in a virtual environment takes time and it is simply not convenient for a user to wait for several minutes for a file to execute, the malicious payload may have been delivered before it can be prevented. Even though detection models based on behavioural analysis is more precise that static analysis [3] [4], because of its time penalty, most of the anti-virus systems don't use behavioural data.

There are different approaches that have been employed to avoid waiting during a files execution in a virtual environment, for this project we based and trained our model using Rhode et al. [5] methodology, and the dataset from Cozzi et al. [6]. In this project, we used the behavioral data extracted during a samples execution in an analysis pipeline, and an ensemble of recurrent neural network and machine learning algorithms to predict if a Linux executable is malicious or not.
The main contributions of this project are:

1) To demonstrate that use recurrent neural networks can predict malicious executable Linux files using machine activity data with good accuracy
2) To demonstrate that Linux executable can be predicted as malicious or not with high level of accuracy using machine learning algorithms.

III. Related work

Despite many efforts in this area, malware detection with around 100% accuracy is still a challenge. Automatic malware detection models use signature based or behaviour based features to classify a sample as malicious or benign. Research has shown that using static analysis is faster but more susceptible to code obfuscation. Though it is faster, it cannot detect new samples with high accuracy. Grosse et al. [7] showed an accuracy of around 95% for a model that was trained using statically collected data but its accuracy dropped to just 20% for repacked malware samples. To overcome that, dynamic data [2] which is more robust and a better detector of malware samples can be used as it doesnt only rely on malware signatures but also on behavioral signatures, which malicious files is expected to exhibit during its execution. Huang and Stokes [8] used system API calls and features derived from that to achieve 97% accuracy. First, we make a dataset csv file from statically and dynamically collected data for our model. Then train recurrent neural networks which are good at detecting malware using machine activity and compare the results to other machine learning algorithms like Support Vector Machines(SVM), K-nearest Neighbors(KNN), Classification and Regression Trees(CART), and Random Forest. We use confusion matrix to analyze the performance of the machine learning algorithms.

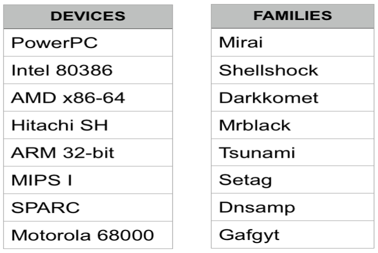
IV. Dataset

To the best of our knowledge, there is no proper organized data-set available for Linux malware. The resources that are available are the Linux samples as such. These files are binaries and are all over the place. Websites like VirusShare provide a repository of these Linux malware samples. To get data from these samples, a researcher has to run these samples in their suitable environments by setting up a virtual environment that replicates its working environment. Due to the heterogeneity of platforms available, each sample will require a new environment to be setup for its execution. In addition to this, there is also the added risk of not setting up the sandbox environment securely and the Linux malware actually affecting the system running the sandbox environment. This can lead to loss of information and other problems. So it is a necessity that a data-set containing these analyzed reports are available to researchers doing research in this field.

Our work involves filling this gap and creating a csv file containing important values obtained from the analysis of the Linux samples. An online platform for multi-architecture ELF analysis called Padawan [10] was used.

The platform already had analysis reports of several linux samples. These reports were in JSON format and was available as webpages that were accessed by using a hash value in a URL. The hash values and the devices that the analysis was performed on is available on the site. We processed this list to get each hashvalue, which we used along with the wget command to download all the json files from their respective webpages. We obtained 10548 json files containing the analysis reports.

The JSON files had several nestings in their structure and thus when converted to a csv file resulted in the creation of different csv files when the python library funcion was made use of for the conversion. A script was written in R to do this conversion instead. A few JSON files contained more than 80000 lines which resulted in the creation of csv file that had 80000 columns. Also the runtime required for this would have been a lot. Therefore we had to tune the feature set required for the final dataset which had all the required important features while being small at the same time. The final dataset took almost half a day to be generated. It had 57 features and 10548 samples.The dataset for linux malware was thus created. Families of some samples and devices from which the samples were obtained as shown in the table in fig. 1 below.



DEVICES	FAMILIES
PowerPC	Mirai
Intel 80386	Shellshock
AMD x86-64	Darkkomet
Hitachi SH	Mrblack
ARM 32-bit	Tsunami
MIPS I	Setag
SPARC	Dnsamp
Motorola 68000	Gafgyt

Fig. 1. Families of some samples and Devices from which the samples are obtained

V. Implementation and Results

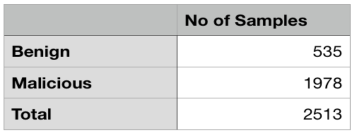
A. Dataset

After making the data-set from the JSON files, it initially had 10547 samples with 57 variables. We scanned the data-set for missing values and removed rows that contained NA values. After cleaning the dataset, we checked for the most significant predictive variables using forward stepwise regression method which gave 40 most important variables. Out of the 40 variables, 37 were used in further analysis for training our machine learning and deep learning models. The features in this dataset are elf.nsegments, elf.stripped_sections, elf.e_shoff, funcover.idapro.max_basic_blocks, bytes.entropy, funcover.idapro.average_bytes_func, funcover.idapro.avg_loc, libide.lstrings.libc.matches, funcover.idapro.branch_instr, funcover.idapro.avg_basic_blocks, bytes.common_bytes5, elf.e_shstrndx, funcover.idapro.max_cyclomatic_complexity, elf.e_shnum, funcover.findcrypt.total, bytes.rarest_bytes5, funcover.idapro.indirect_call_instr, funcover.idapro.nfuncs, bytes.printable, funcover.idapro.avg_cyclomatic_complexity, bytes.rarest_bytes6, elf.stripped, funcover.idapro.badstack, bytes.rarest_bytes1, bytes.null_bytes, bytes.rarest_bytes2, bytes.common_bytes3, funcover.idapro.overlapped_instr, funcover.idapro.call_instr, bytes.min_entropy, funcover.idapro.loc, elf.nsections, bytes.longest_sequence.length, funcover.idapro.indirect_branch_instr, bytes.common_bytes4,

For the machine learning techniques we are using, we required both benign and malicious samples. To get the benign samples we tried to use Padawan to analyze the binary files from our linux systems. But Padawan was incapable of doing

this since it could only analyze ELF binaries.
To obtain benign samples, we obtained the samples from our dataset with vt.positives = 1 and labelled them as benign.The vt.positives value is the number of engines in Virus Total that classified the sample as malicious. Rhode et al. [9] detected Windows malware used a similar approach where they actually removed samples with vt.positives less than or equal to 5 as they are contentious in terms of whether they are malicious or not.

Our final data-set contained 2513 samples in total out of which 1978 were malicious and 535 were benign samples as shown in the table in fig. 2 below.



	No of Samples
Benign	535
Malicious	1978
Total	2513

Fig. 2. Dataset used for Prediction Implementation

B. *Machine Learning Algorithms*

We compared RNN to different machine learning algorithms. The training and test data is same for all machine learning algorithms to accurately compare results. Confusion matrix is used to get parameters like accuracy, sensitivity and specificity. The results of the various algorithms is described below.

1) Using K-Nearest Neighbors(K-NN): K-nearest neighbors is among the simplest machine learning algorithms where the function is approximated locally and computation deferred till classification is performed. Fig. 3 shows performance analysis of K-NN.

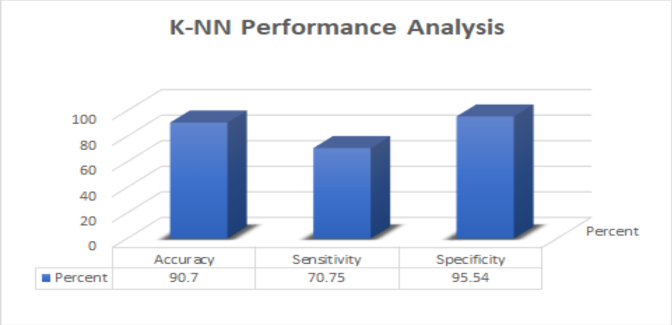


Fig. 3. K-NN Performance Analysis

2) Using Random Forest: Random Forest is also a easy and flexible supervised machine learning algorithm that performs even great without hyperparameter tuning unlike RNN. This is among the most popular algorithms because it is simple and performs tasks of classification as well as regression. Fig. 4 shows performance analysis of Random Forest.

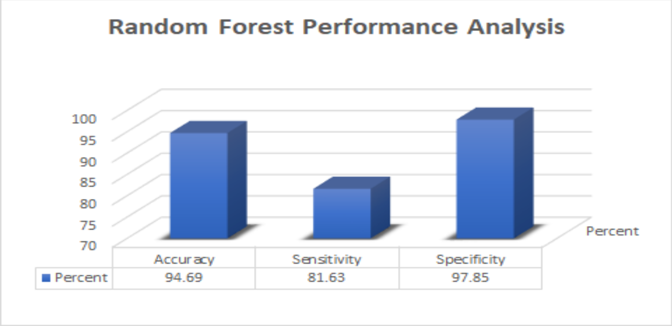


Fig. 4. Random Forest Performance Analysis

3) Using Support Vector Machines(SVM): A Support vector machine(SVM) is a supervised learning method that can perform tasks of both classification and regression. It maps sorted data into one of two categories keeping them as far as possible. Fig. 5 shows performance analysis of SVM.

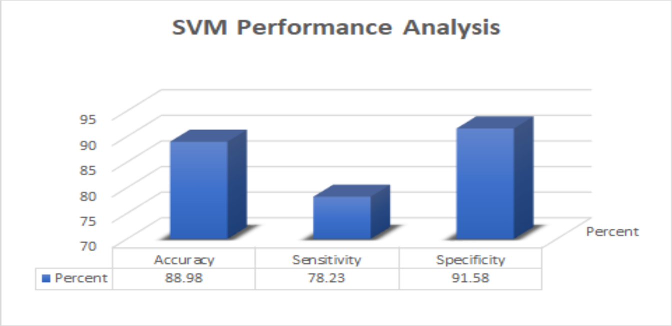


Fig. 5. SVM Performance Analysis

4) Using Classification and Regression Trees(CART): Classification and Regression Trees(CART) uses decision trees to perform predictive machine learning. This algorithm also can perform both classification and regression tasks. Fig. 6 shows performance analysis of CART.

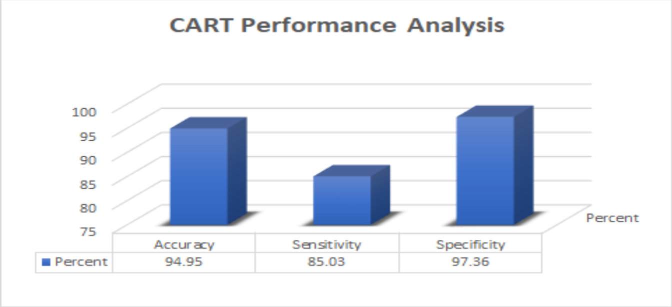
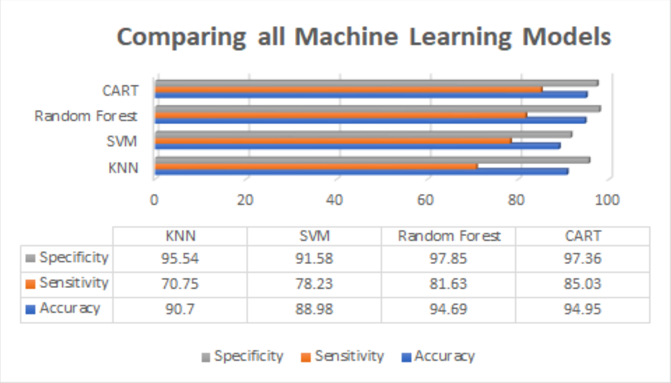


Fig. 6. CART Performance Analysis

C. Comparing all Machine learning Algorithms

In this section, results from different machine learning algorithms is compared based on accuracy, sensitivity and specificity. Fig. 7 shows the comparison of results from machine learning models.



	KNN	SVM	Random Forest	CART
▨ Specificity	95.54	91.58	97.85	97.36
▨ Sensitivity	70.75	78.23	81.63	85.03
▨ Accuracy	90.7	88.98	94.69	94.95

Fig. 7. Comparing results from all Machine Learning Algorithms

D. Recurrent Neural Network

Recurrent Neural Networks are very capable of analyzing sequential data and in predicting malware using machine activity. The data-set that we used for this project is not a sequential time-series data and might have caused the RNN to not have the level of accuracy Random Forest and CART had, it could be that the hyperparameter needs to be fine-tuned for Linux malware as the hyperparameter that we used is that of windows malware. Also, since the training data is quite similar to each other it was easier to learn for Random Forest and other machine learning algorithms that achieved high accuracy.

We used 36 machine activity data metrics as the feature inputs to the RNN model. As demonstrated in Figure 8, the data-set from [6] went through an analysis pipeline that analyzed the samples statically, and dynamically for 5 minutes inside an emulator.

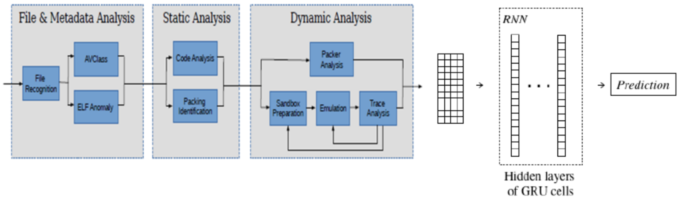


Fig. 8. High-level model overview

By using an ensemble classifier that used the best three hyperparameter configurations found during the random search space, the recurrent neural network achieved an accuracy of 91.15% with an f score of 94.65%. This is shown in Figure 9 below.

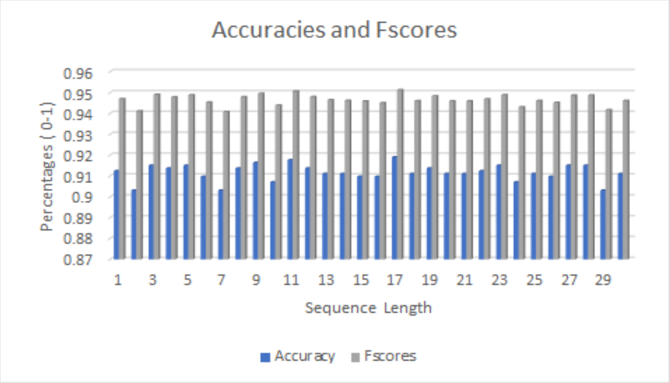


Fig. 9. RNN Accuracies and Fscores

VI. Challenges

The main challenge we faced for this project was the lack of a data-set for Linux malware. Despite the rising popularity of IoT devices, and the consequent increase in Linux malware, the absence of a data-set can slow the research happening in this area. The solution we had to take was to create the data-set by ourselves. Another challenge we faced was the runtime required for the various processes in our project. The conversion of the JSON files to a csv file took a lot of time to be processed. Also the RNN implementation required a lot of runtime even while being executed on the Palmetto cluster.

VII. Future work

Our results indicate that behavioural data can be used for the detection of Linux malware. According to the results we obtained, we found out that we did not get the highest accuracy for RNN. Future work can be done to implement the malware detection models using other deep learning techniques like CNN to perform the experiments to check if it can result in a better accuracy. Work can also be done to implement zero-day prediction. To achieve this, we can make use of the delivery mechanism of malware samples to predict. The data-set must contain time-stamped data in order to achieve this. An analysis tool similar to Padawan must be developed to obtain time-stamped data for the samples.

VIII. Conclusion

Static malware detection which is mostly used in anti-virus systems can be fooled into believing that an executable file is not malicious by means of code obfuscation. While Dynamic analysis can be used to over come the short-comings of static analysis it incurs a time penalty. For a prediction to be made, the sample has to be executed and its activity footprint collected, the malicious payload may have been delivered before the attack is detected and blocked.

In this project we have demonstrated that RNN can be used in predicting Linux malware with 91% accuracy which can be improved if a new random search for the best performing hyperparameter configuration is performed on the Linux dataset. A new hyperparameter configuration could increase the accuracy of the RNN by two to three percentage points, taking it to the accuracy level of Random Forest and CART. The use of a different deep learning algorithm like Convolutional neural networks, Feed-forward neural networks, and Echo state network in place of RNNs could see a huge increase in accuracy, if not surpassing the accuracy level of the best performing machine learning algorithm. This is because RNNs is not well suited for non-sequential data. The use of other deep learning algorithm could also cause the model to not perform well in zero-day malware prediction, that is predicting completely new samples it has not been previously exposed to.

IX. Team Member Contributions

We first implemented existing implementation that was based on detecting Windows Malware. This was done with efforts of all 3 team members. Next part of the project was to implement it for Linux Malware. For that our project had 3 essential components: Making a Linux malware dataset, running machine learning algorithms to get accuracies of those models and finally implementing RNN and compare accuracies of all the models. Each component had active participation of all 3 team members and we collaborated to meet project deadlines.

1) Shivam Pandit: Majorly worked on cleaning Linux malware dataset and implementing machine learning algorithms to get accuracies of all the implemented models like KNN, CART, SVM and Random Forest.

2) Christin Wilson: Majorly worked on creating and compiling the Linux malware dataset.

3) Ebuka Okpala: Majorly worked on implementing RNN for Linux malware dataset.

References

[1] StatCounter, Desktop Operating System Market Share Worldwide. http://gs.statcounter.com/os-market-share/desktop/worldwide.

[2] ZDnet, Googles VirusTotal puts Linux malware under the spotlight. http://www.zdnet.com/article/ googles-virustotal-puts-linux-malware-under-the-spotlight/.

[3] Antonakakis et al., Understanding the Mirai Botnet, in Proceedings of the USENIX Security Symposium, 2017.

[4] M. Sebastian, R. Rivera, P. Kotzias, and J. Caballero, AVclass: A Tool for Massive Malware Labeling, in RAID, 2016.

[5] Rhode, M., Burnap, P. and Jones, K., 2018. Early-stage malware prediction using recurrent neural networks. Computers Security, 77, pp.578-594.

[6] Cozzi, E., Graziano, M., Fratantonio, Y. and Balzarotti, D., 2018, May. Understanding Linux Malware. In IEEE Symposium on Security Privacy.

[7] K. Grosse, N. Papernot, P. Manoharan, M. Backes, P. D. McDaniel, Adversarial examples for malware detection, in: Computer Security - ESORICS 2017 - 22nd European Symposium on Research in Computer Security, Oslo, Norway, September 11-15, 2017, Proceedings, Part II, 2017, pp. 62

[8] W. Huang, J. W. Stokes, Mtnet: A multi-task neural network for dynamic malware classification,in: Proceedings of the 13th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment - Volume 9721, DIMVA 2016, Springer-Verlag New York, Inc., New York,

[9] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, Y. Bengio, Learning phrase representations using rnn encoder decoder for statistical machine translation, in: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Association for Computational Linguistics, Doha, Qatar, 2014, pp. 1724-1734. URL http://www.aclweb.org/anthology/D14-1179

[10] https://padawan.s3.eurecom.fr/