Classification on Strelized Binary Malware Representation by Using LSTM

Fatih Can Kurnaz
Computer Engineering
Middle East Technical University, Turkey
Email: fatih@ceng.metu.edu.tr

Doguhan Yeke
Computer Engineering
Middle East Technical University, Turkey
Email: doguhan.yeke@metu.edu.tr

***Abstract*—In todays world, it is almost impossible to not encounter a malware file that tries to access sensitive information. Cyber-attackers are everywhere to capture user data, and trace user activities. They are not only nightmares of ordinary web users. Big companies also suffer from those attacks and lose information, money and reputation. Therefore, there are many counter measures taken against these attacks ,which vary from inspecting file before opening it (static) to running it in some safe environment(dynamic) to see if it is benign or not. However, todays malwares are smart enough to escape from static approaches by packing itself, and unpacking when it is time to run. On the other hand, the dynamic approaches can not be automated well, and some files can understand that it is running on VM and hides their suspicious actions. Our approach is different than these types of approaches and works with both packed and unpacked files. We get the malware file itself and construct decimal valued one dimensional vector that represent the whole file. After that, with a deep LSTM network model we classify malicious files. With this model we got an accuracy of 86% in a dataset with 9 different malware type.**

1. Introduction

Deep learning methods proved themselves for being state of art classification solutions. After the training part, most of the time they give fast results on the given data. Therefore, it is understandable to use it on one of the most crowded area of the computer society, malware detection and classification.

Malware intrusion is as old as the internet and computer technologies itself[1]. During these times many technologies from different areas are used to detect malwares. However, detecting them is never enough, because in order to save the computer from the malware, you should have an idea of what is the malware that infected the computer. Therefore, malware classification is another important aspect of computer security, thanks to its importance taking countermeasures against the malwares.

In the early attempts against malicious files, one of the main approaches was to let the possible malware program run sometime in an honeypot environment and observe its API calls to operating system, and detect whether it is accessing sensitive information or not. However, this process requires running the possible malware which in itself is a long and costly action. Another approach was to look into what kind of dlls the program using and decide on a risk factor, so that it could be possible to decide whether this program is malicious or not. From the mentioned approaches, the former one is an example to dynamic approach to classify the malware and latter one is a static approach example. In general dynamic approach lets the possible malicious code run for some time, and by observing its actions decides whether it is malicious or not. In static approach, possible malicious code is observed in different manners with relation to different aspects of it like dll etc., but it is not run on any system. By looking at those information detector decides on whether the code belongs to any malware class or it is benign.

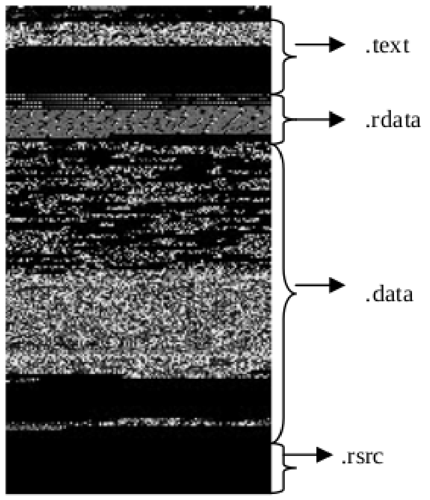


Figure 1: shows section fields of sample[6].

However, todays malwares are usually obfuscated and resolving it to a state that cannot be traced easily. Tracing the data section shown in Figure1 is hard. There is also no certainty that the malicious file can be converted to unobfuscated state in lossless format.That is why studies on already obfuscated file is drawn a lot more attention.

In this work, we are combining the old malware classification problem and state of art deep learning methods to classify malware binary files in a static manner. Our approach just observes the limited amount of the binary malware files. Also thoso files does not contain PE header, which is actually the most informative part of a binary malware file. We tried to look into just pure binary code information, and saw that we can classify our malware files quite successfully by only using that information.

2. Related Work

Recurrent Neural Network: Recurrent networks have been used in many different fields since their invention in 1980. Thanks to their internal memory, they can memorize certain time dependent features of the given data. Language and sound process are some of the fields that contain time relavelant information in their data. They became more successful and more popular with the LSTM model.

Malware Classification: R. Pascanu[2] used echo state networks(ESNs) and recurrent neural networks(RNNs) to extract features from malware files. They tried different architectures for recurrent model, non-linear sampling and final classification. 250,000 files were used to train classification model and each file is labeled as benign or malware. As the accuracy, they succeeded with true positive rate upto 98% and false positive rate 1%.

S. Tobiyama[3] proposed a way to detect and classify malware files by observing process behaviour with deep neural networks. In their work they first trained a recurrent neural network(RNN), and after that they act as if features extracted from RNN are images. They apply convolutional neural network(CNN) over that data. Their data contains header information in each binary malware file.Their dataset contains close to 40000 binary malware files. With that system they acquired AUC close to 0.96.

T.H Huang and H.Kao[4] proposed a RGB color representation of Android app binaries and used a fixed size image representation for their convolutional neural network(CNN) process. They had collected 1 million malware and 1 million benign samples to apply CNN. However, their R2-D2 project works on only Android files.

X. Wang and S.M Yiu[5] used API calls for their classification. In their dynamic approach, they accepted each API call as sentence and used RNN-AE to learn low-representation of malware from API calls. At the end, they used decoder to make malware classification. They used different combinations for AE, and their test accuracy gives a performance in range of 98% and 99%.

L. Nataraj et al[6] used image representation of malware and got really interesting images from malware files.They used 9458 samples with 25 different malware families. At the end of their study, they reached 98% accuracy in classification part.

3. Malware classification on sterilized binary malware file

Our deep learning model is built upon LSTM layers. Our initial idea was that binary files are actually not that different from natural language. In natural language, understanding the meaning of the words require knowledge about the previous words.Sentences and paragraphs follow the same pattern. Therefore, in order to understand the whole meaning, it is required to keep the previous information on some kind of memory. Likewise, binary files carries the same structure. In order to understand the meaning of the bytes, one have to have some knowledge about previous bytes. Therefore it is logical to use some learning method that keep memory to carry information, in order to make classification on binary data. Therefore, we designed a model that will carry the information between the nodes of the same layer and by doing so it will act like a memory. Best choice for this kind of model is using LSTM layers. As a result of these informations we created our model as 4 LSTM layers, 5 Leaky Relu layers and 2 Dense layers. Complete model is given in Figure 2.

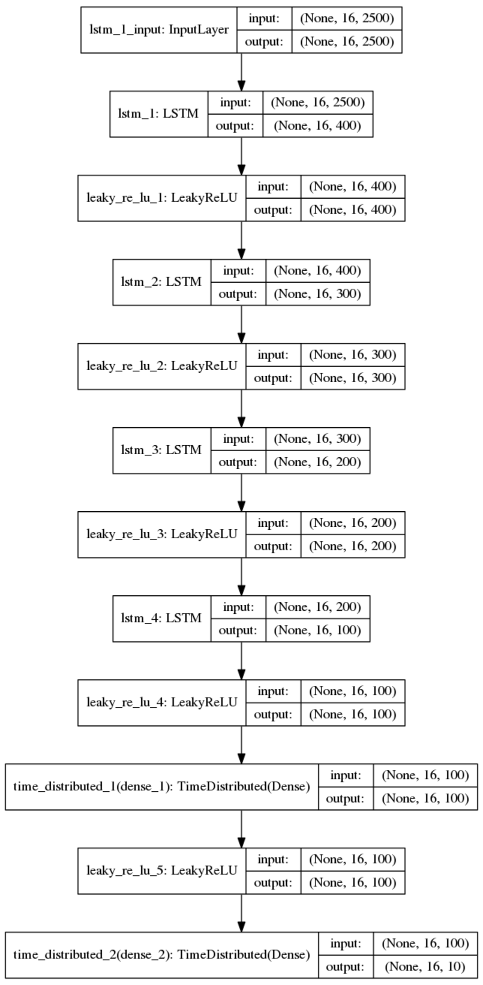
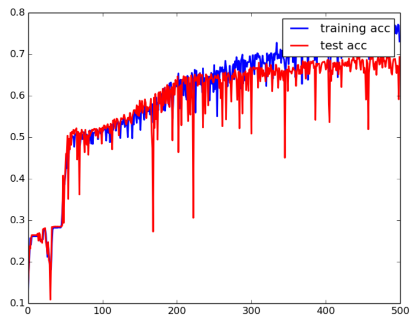
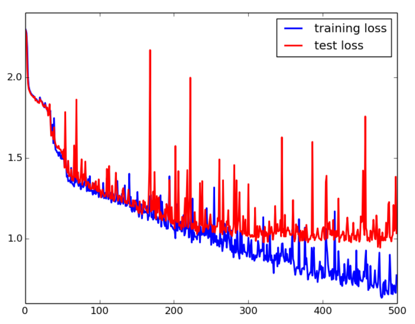


Figure 2: Our Network structure with timesteps and inputsizes.

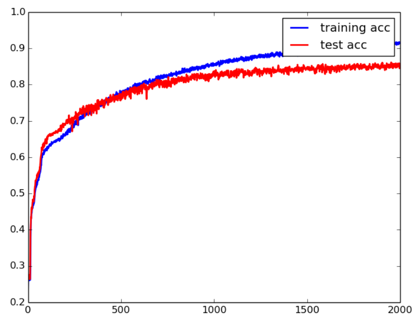


(a) Accuracy

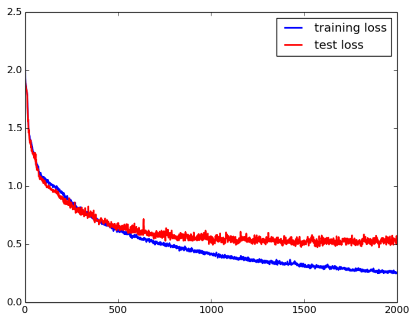


(b) Loss

Figure 3: Initial training results of the network



(a) Accuracy



(b) Loss

Figure 4: Optimized training results of the network

4. Experiment

To test the effectiveness of our classification model, we tested it with some binary malware file and recorded the accuracy of the classification.

4.1. Data Preprocessing

Before we were able to use our deep learning model we had to preprocess our data, because of both our computational limits, and our raw data being not trainable . Our binary files contain memory location information and hexadecimal byte values. We removed the memory location info and turned hexadecimal values to decimal values. Another problem with our dataset was the sizes of binary files. Some of those binary files contained more than 1000000 lines, which were almost impossible for us to run it with our limited computing power. Also Keras were required to give same sized input to LSTM, so we had to truncate our data. Therefore, we have truncated our binary files to desired sizes and after that we manipulated them to work with the desired timesteps size. Resultant data would be in the shape of

$$\left[\text{number of samples},\quad \text{timestep},\quad \text{data length}\right]$$

4.2. Experimental Configuration

In our project we worked on Microsoft Malware Challenge(BIG 2015) which is on Kaggle[7]. This dataset contains 9 different malware classes and more than 10000 binary of malwares for training and 10000 for testing and evaluation. In this dataset, only training data contains labels and you are expected to send your results to Kaggle in order to get a score. Therefore, test data were not usable in our case. Also, Kaggle did not provide PE header to provide sterility.PE headers carry a lot of information about the files that we are analyzing. Also we have no access to API calls, which give certain information about sequential actions of the malicious files while they are running. We implemented our network design on Keras with Tensorflow as backend and run it on GTX GeForce 1050Ti 4 GB GPU. This GPU is really limited for this kind of problem and model. As result of limitations on computational power of our GPU, we could not run more than 4000 training and 1000 test sample files. Also we were limited with 10000 and 2500 data length for 1 and 16 timestep setup respectively. Therefore, we were only able to access to first 40000 bytes of the binary files.

4.3. Experimental Results and Discussion

In our system, we first tried different types of loss functions and optimizers, but we achieved highest amount of accuracy with sparse categorical cross-entropy loss function and adadelta optimizer. We used the default Keras implementations of those functions. Figure 3 shows the one of the first results of our model. It can be easily seen that there is overfitting, and it has a mediocre accuracy over the test set. Therefore, we had to do regularization on our model by adding dropout and recurrent dropout to LSTM layers. After regularization, we worked on parameter optimization. Kerass manual advises not to change the default learning rate of adadelta function. Therefore, we only changed the other parameters like dropout rate, epoch number, batch size, LSTM node number. After that, we were able to acquire 86% accuracy with given dataset of 4000 training and 1000 testing samples on 2500 data length with 16 timesteps. This is the highest amount we have reached with 2000 epochs.

What we can do to improve our system is to train network with more data so that it can learn more about the relations of sequences and information they carry. Since we use 5000 samples as our input data, our network can not adjust its weights properly. We also could not make our model train with more timestep and longer datalength because of the natural limits of our GPU.

5. Conclusion

Malware classification is quite challenging problem. In the literature, there exist some very good applications that give really good results. They use different kinds of features in both static and dynamic approaches. In static approaches, they used PE header to get the information about code sections and carry this information to their network models.But this can work well on only unpacked files, while most of today's malwares are packed. In dynamic approaches, they log API calls and then work on these calls to have an idea over the class of the malware. This approach takes lots of time. In literature, there is also an image representation approach[6] that uses CNN to classify malicious files.In our case, we do not have access PE headers, and also could not log the API calls, because of sterilization of our training data. In our approach, we come with a LSTM solution and got an accuracy of 86% in a classification problem with 9 different classes.

Although byte representation of malware is good solution to malware classification, the attackers can develop techniques in order to hide their malicious behaviour from our detection mechanism. Since our approach is based on merely byte representation, relocating file sections can be a choice for them to deceive our classifier system. For this reasons, we can try different techniques like max pooling to lesser the effect of the relocating the code sections. In future work, we can also focus on other type of representations of the malicious codes like asm files , and apply different architectures on them. After that we can combine those architectures to get a higher degree of accuracy.

References

[1] N. Milosevic, *History of Malware*, arXiv:1302.5392v3, 2013

[2] R. Pascanu, *Malware classification with recurrent networks*, Acoustics, Speech and Signal Processing (ICASSP), 2015

[3] S. Tobiyama, *Malware Detection with Deep Neural Network Using Process Behavior*, Computer Software and Applications Conference (COMPSAC), 2016

[4] T.H. Huang and H. Kao, *R2-D2: ColoR-inspired Convolutional NeuRal Network (CNN)-based AndroiD Malware Detections*, arXiv:1705.04448v4, 2017

[5] X. Wang and S.M. Yiu, *A multi-task learning model for malware classification with useful file access pattern from API call sequence*, arXiv:1610.05945v1, 2016

[6] L. Nataraj et al., *Malware Images: Visualization and Automatic Classification*, Do:10.1145/2016904.2016908 , 2011

[7] *Kaggle: Microsoft Malware Classification Challenge*, https://www.kaggle.com/c/malware-classification