BotBucket - Static Analysis and Classification of Botnets and Malwares

Shubhangi Kishore, Sayli Karnik, Binayak Ranjan Das, Atharva Urdhwareshe

Abstract—

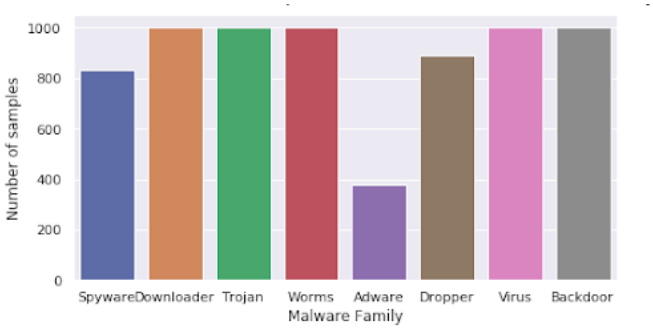
The use of operating system API calls is a promising task in the detection of PE-type malware in the Windows operating system. We leveraged The Mal-API-2019 Dataset [1] published by Catak and Yazi in 2019. We performed exploratory analysis of the dataset and created a baseline model for the dataset. This labeled dataset consists of Windows operating system API calls of various malware. We're using using PyTorch neural network framework for training the many to one RNN model to classify the malwares into 8 families (Trojan, Backdoor, Downloader, Worms, Spyware Adware, Dropper, Virus). The dataset contains about 7107 rows, each row consists of many binaries separated by space. We first converted the space separated file to a csv file. Corresponding to each row a label is associated which gives the malware. Our approach classified each row into a vector. After classifying each vector, we classified each vector which consists of the comma separated binaries. We used the LSTM algorithm as a RNN approach to classify and train the dataset and then test it to the labels associated with it.

1 Introduction

Malware and Bots have been a pressing threat to the internet and computational systems. With the rapidly growing amount of software in use around the world, it has become impossible to manually audit it all for vulnerabilities, so it is imperative that we develop efficient systems to automate this process. Due to the extensive use of the binary obfuscation, it is very difficult to analyze and classify the bots. Various techniques including intrusion detection uses algorithms that helps to detect and classify these bots and eradicate them. The ability of Machine learning to extract meaningful decision-making based on wide ranging features of data has been applied to wide ranging fields. Malware is constantly adapting in order to avoid detection. There are a large number of malwares and despite the ever increasing number of Malware, there is a 7 percent decline in new malwares. Signature based malware detection is the most commonly used method in industry and organizations. This completely fails at detecting the new malware or variants of the existing malware. The traditional machine learning algorithms rely on the features. They require manual intervention for identifying new malware or variants of existing malware. More than 1 million malwares, a significant chunk, are Windows Malwares. Hence, our work is on detecting/classifying Windows Malwares. Unless the malware itself is installed in the kernel of the operating system, the malware will have to use system calls to function. With this in mind, it is possible to trace system calls and analyze them for any discernible patterns. We develop a classification system, 'BotBucket' which successfully trains and classifies the obfuscated bot binaries using the RNN - LSTM machine learning models. BotBucket will be designed to effectively deal with obfuscated adversarial bot binaries. We highlight the key issues with existing approaches to this problem. It is difficult to analyse packed malware that contains unpacking code with static analysis. For dynamic analysis that involves running the program and studying execution flow (function calls, branches etc.) using a debugger (e.g. GDB), binary instrumentation tools (e.g. Frida), or by emulating instructions (e.g. in symbolic execution), in order to study all possible execution paths, current approaches either fork a new process at every branch point to study

both branches (which causes an exponential increase in memory/resource consumption), or study each possible branch sequentially which is slower. Behavioural analysis in the sense of network bandwidth consumption / memory consumption etc. means classifying binaries based on patterns in how they access system resources. But this is prone to generating false positives/negatives. e.g. a binary which sends a lot of data from the system over the network could either be a malware, or a benign application like a web server.

2 Dataset

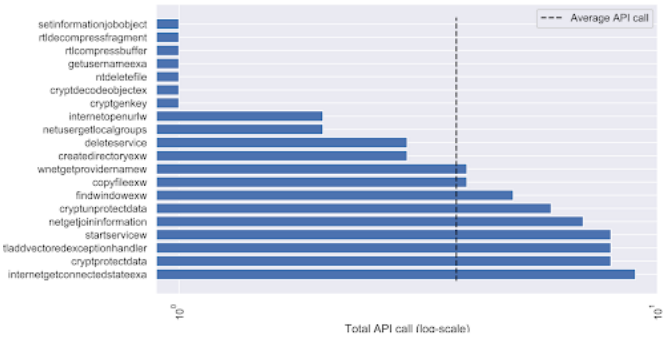


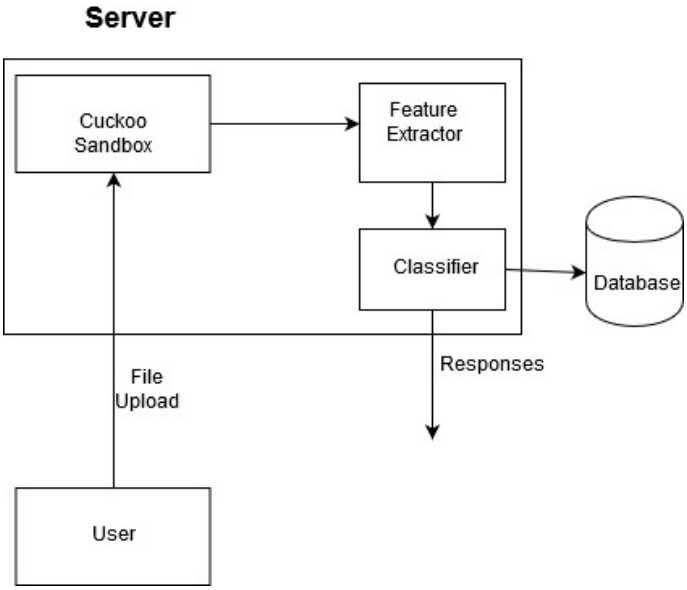
We leveraged 'The Mal-API-2019' Dataset [1] published by Ferhat Ozgur Catak, Ahmet Faruk Yazi in 2019. This labeled dataset consists of Windows operating system API calls of various malware. The malwares are classified into 8 families (Trojan, Backdoor, Downloader, Worms, Spyware Adware, Dropper, Virus) i.e there are 8 types of labels in the labels file. The dataset contains about 7107 rows, each row consists of many API calls separated by space. Corresponding to each row a label is associated which gives the malware. The authors obtained the MD5 hash values of the malware collected from Github. They searched these hash values using the VirusTotal API, and obtained the families of these malicious software from the reports of 67 different antivirus software in VirusTotal. The use of operating system API calls is a promising task in the detection of PE(Portable Executable)-type malware in the Windows operating system.

3 Approach

3.1 Cuckoo Sandbox

We provide motivation for a system that runs Cuckoo Sandbox 2012, open source instance pro-





vided with a full fledged interface through which people can upload files to be analysed and the analysis by cuckoo is passed through multiple classifiers. If the first binary Classifier suggests that the file contains suspicious malware, we categorise the malware into known malware families. This system not only would perform the classification task but would also collect and annotate data for future work. In our proposed setup designed for automated malware detection and profiling the sandbox runs the executable and traces the various system API calls and passes this information to the Feature Extractor module.

3.2 Feature Extractor

This module gets the relevant SYS API calls from the sequence provided by sandbox in below mentioned way.

We used a total of 300 Windows system API calls which were found in our dataset and they are numbered from 1 to 300.

Each sequence of system call is first converted into a stream of integers and Run-length encoding is performed. For example, if a system call X is repeated 10 times , we presented that information as (X,10+some integer > 300) in the sequence.

Adversarial inputs could potentially be crafted to exploit vulnerabilities in the classifier. To prevent adversaries from manipulating the input data to exploit specific vulnerabilities of a classifier we excluded commonly used system calls that convey little information about malicious function of process (such as NtAllocateVirtualMemory).

3.3 Classifier

Layer (type)	Output Shape	Param #
embedding_8 (Embedding)	(None, 23500, 128)	384000
conv1d_8 (Conv1D)	(None, 23500, 32)	16416
max_pooling1d_8 (MaxPooling1	(None, 5875, 32)	0
lstm_8 (LSTM)	(None, 200)	186400
dense_8 (Dense)	(None, 1)	201

Sequence classification is a predictive modeling problem in which some sequence of inputs over space or time is assigned a category. What makes this problem difficult is that the sequences can vary in length, be comprised of a very large vocabulary of input symbols and may require the model to learn the long-term context or dependencies between symbols in the input sequence. LSTM is very powerful when dealing with time series kind of information, simply because it can store information about previous values and exploit the time dependencies between the samples. In our setup, each integer is the index of an individual system call event. This sequence of integers is then input into our classification system that uses RNN. At each time step, the hidden control state ht is provided as input to a second LSTM layer. After viewing all the frames, the second LSTM layer learns to decode this state into a sequence of system calls. The LSTM model that we used in our system consists of an Embedding layer, 32 4*4 filters with padding to keep the input in same size and 4*4 max pooling layer. We have used RELU as our activation function and Adam optimizer. Each sequence and its corresponding category can be uploaded to the database for annotating and created a larger dataset for future research.

4 Results

Our model achieved an accuracy of 0.62 on our dataset. We used cross entropy loss as our loss function here. The classes - Downloaders and Virus were not accurately predicted. A probable reason for this could have been the longer system calls call sequences. Spyware and Adware were correctly predicted with good accuracy. This accuracy can be attributed to the vectorized representation of System Call sequences that maintains the sequence information and at the same time ignores the most frequently occurring non- malicious call. We are able to preserve locality and hence provide a suitable input the LSTM network. The same vector input when provided to other models like SVM results in poor accuracy of 0.30. Previous studies use varying different types of dataset, and while some outperform out model, they report the test result on on their custom datasets which are much smaller.

5 Future Work

The applicability and accuracy of our model also extends for predicting properties like the malicious and benign nature of a windows executable. We remark that the Botbucket model does not analyse the complex paths the malware could have taken as we would in a thorough dynamic analysis where we observe the parameters passed

to each system call. An example of this is where say normal program would write its data, a self replicating program would write its own code in that place. We would also like to explore the classification of malwares that use low resources and perform operations like time check. A possible approach for this problem is to patch the file to enable early activation.

The final question we would like to address is the feasibility of utilizing the Botbucket system. This requires the assessment of the BotBucket model on a much larger data set ofMalwares. In summary, we have devised and applied a systematic hierarchy of efficient models to estimate malware families and malicious nature for a system call sequences of Malware. The developed BoB model is quite successful for our categories, hinting that it could also be extended to study model features resistant to manipulations by attackers to enhance the system security.

This system also involves continuously annotating new data for better future classification and identifying common/repeated sequences that could map to specific functions. This would greatly improve the performance of the model. We propose the use of control flow graphs from static analysis to identify sequences of system calls. For added resistance to other adversarial attacks we propose patching malware to remove time delays.

6 Related Work

The analysis and classification of malicious software was introduced by Lo et al. (1995). Malware binaries are manually analyzed, characteristic features for malicious binaries can be extracted and later applied for detection of other malware samples. This approach has been further enhanced by Christodorescu and Jha (2005). They proposed an architecture to detect malicious activities that are rigid to common obfuscation techniques.

There are two types of techniques which are used to classify the obfuscated binary into malware. First is static analysis of malware binaries. This technique examines malware without running it(e.g., Linn and Debray,2003; Szor, 2005; Popov et al., 2007; Ferrie, 2008, 2009).

Recently, most of the systems have been proposed to commonly unpack malware samples (e.g., Royal et al., 2006; Martignoni22et al., 2007; Dinaburg et al., 2008; Sharif et al., 2009). The common idea in all these systems is to execute malware binaries and decide dynamically, when the samples are unpacked. It executes the malware in a monitored environment to analyze its behavior. As discussed above these approaches provide knowledge of both static analysis and dynamic analysis, but we did our work entirely on dynamic analysis of malware behavior. Dynamic analysis of malware bot binaries is a largely used technique in the research community.

In 2009, Jian Li et al came up with an algorithm to detect malware obfuscation using maximal patterns which are subsequences in malware's runtime system call sequence, which frequently appear in program execution, and can be used to describe the program specific behavior. While this approach was resilient to code obfuscation technologies,Obfuscation can relocate instructions in a bot binary and also introduce extra system calls into a call sequence. Both of these can negatively affect the classification accuracy.

In 2011, Konrad and others proposed a framework for automatically identifying novel classes of malware with similar behavior (clustering) and assigning unknown malware to these discovered classes (classification) that reduced the run-time overhead of current analysis methods, while providing accurate discovery and discrimination of novel malware variants. However, the framework focuses on a single environment to classify the malware binaries. It does not provide any results for the type of behavior of malware in different conditions. L Natraj et al visualized Malware binaries as gray-scale images with k-nearest neighbors approach with the observation that for many malware families, the images belonging to the same family appear very similar in layout and texture. Adversaries can relocate sections in a binary or adding vast amount of redundant data as countermeasures to the method.

Kolosnjaji et al. used CNNs and RNNs to identify malware. The dataset was created by the authors using existing malware samples from Virus Share, Maltrieve and private collections and traced malware behavior using a malware zoo and Cuckoo sandbox for dynamic analysis. Calls to the kernel API were recording during their execution.

The list of call sequences to the API kernel is converted into binary vectors using one-hot encoding. One-hot encoding is a scheme for storing categorical data in form easier for machine learning. This data is used to train the DL algorithm, which consists of a CNN and RNN (consisting of an LSTM, and a softmax layer).

Recent studies generated adversarial malware samples to evade Deep Learning based malware detection.Xu et al. evaded two PDF malware classifier, PDFrate and Hidost, by modifying PDF and parsed the PDF file and changed its object structure using genetic programming. The adversarial PDF file was then packed with new objects. Another study that showcases the generation and effect of adversarial examples in malware detection is - 'Adversarial examples for malware detection' (2017).

References

[1] Mal-API-2019 Dataset Description https://github.com/ocatak/malware_api_class/blob/master/README.md

[2] https://ieeexplore.ieee.org/abstract/document/1425057

[3] Zhou, Yajin, and Xuxian Jiang. "Dissecting android malware: Characterization and evolution." 2012 IEEE symposium on security and privacy. IEEE, 2012.

[4] VirusShare. Available online: http://virusshare.com/

[5] Arp, D.; Spreitzenbarth, M.; Hubner, M.; Gascon, H.; Rieck, K.; Siemens, C.E.R.T. DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket. NDSS 2014, 14, 23–26.

[6] Microsoft Malware Classification (2015) - https://www.kaggle.com/c/malwareclassification/data

[7] Saxe, Joshua, and Konstantin Berlin. "Deep neural network based malware detection using two dimensional binary program features." 2015 10th International Conference on Malicious and Unwanted Software (MALWARE). IEEE, 2015.

[8] Yuan, Zhenlong, et al. "Droid-sec: deep learning in android malware detection." ACM SIGCOMM Computer Communication Review. Vol. 44. No. 4. ACM, 2014.

[9] Kolosnjaji, Bojan, et al. "Deep learning for classification of malware system call sequences." Australasian Joint Conference on Artificial Intelligence. Springer, Cham, 2016.

[10] Grosse, Kathrin, et al. "Adversarial examples for malware detection." European Symposium on Research in Computer Security. Springer, Cham, 2017.

[11] W. Xu, Y. Qi, and D. Evans, "Automatically evading classifiers," in Proc. Netw. Distrib. Syst. Secur. Symp. (NDSS), 2016, pp. 1–15.

[12] Brumley D., Jager I., Avgerinos T., Schwartz E.J. (2011) "BAP: A Binary Analysis Platform." In: Gopalakrishnan G., Qadeer S. (eds) Computer Aided Verification. CAV 2011. Lecture Notes in Computer Science, vol 6806. Springer, Berlin, Heidelberg.

[13] Lin, Y.-D Chiang, Y.-T Wu, Y.-S Lai, Y.-C. "Automatic analysis and classification of obfuscated bot binaries." 2014.

[14] Nataraj, L., Karthikeyan, S., Jacob, G., and Manjunath, B. S. "Malware Images: Visualization and Automatic Classification." In Proceedings of the 8th International Symposium on Visualization for Cyber Security. New York, NY, USA: ACM. https://doi.org/10.1145/2016904.2016908, (2011).

[15] "Automatic Analysis of Malware Behavior using Machine Learning". Konrad Rieck, Philipp Trinius, Carsten Willems, Thorsten Holz,Berlin Institute of Technology, Germany, University of Mannheim, Germany,Vienna University of Technology, Austria.

[16] "Three-phase behavior-based detection and classification of known and unknown malware". Ying-Dar Lin, Department of Computer Science, National Chiao Tung University, Hsinchu 300, Taiwan, Yuan-Cheng Lai, Department of Information Management, National Taiwan University of Science and Technology, Taipei 106, Taiwan, Chun-Nan Lu, Peng-Kai Hsu and Chia-Yin Lee, Information Communication Technology Laboratories, National Chiao Tung University, Hsinchu 300, Taiwan.