Graph Convolutional Network for Malware Detection

Yun Da, Tsai
b04920103@ntu.edu.tw
National Taiwan University
Taipei, Taiwan

Cheng Syuan, Lee
r07922055@ntu.edu.tw
National Taiwan University
Taipei, Taiwan

Chun Hsiang, Wang
b06902006@ntu.edu.tw
National Taiwan University
Taipei, Taiwan

ABSTRACT

Malware detection has been a long-lived issue for such a long time. In recent years, new concepts have been introduced as more and more machine learning and deep learning findings were figured out. In our work, we focused on a graph-based malware detecting techniques leveraging graph convolutional network when given a raw binary program. Our model is strongly based on graph convolutional network and word2vec algorithm. We first extract data from raw binary program. By these extracted data, we get multiple control blocks and instruction codes which can be viewed as a control flow graph. Since instruction codes can be viewed as sequence of words, we treat the codes as sequence of words which can be trained by word2vec algorithm. Combining both techniques, we are able to build our model and train successfully. However, we've also encountered some limitations which lower the efficiency and the accuracy of our result. Eventually, we compared to four different prevalent malware detection methods to evaluate our work.

KEYWORDS

graph neural networks, malware detection

ACM Reference Format:
Yun Da, Tsai, Cheng Syuan, Lee, and Chun Hsiang, Wang. 2019. Graph Convolutional Network for Malware Detection. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION (15%)

Nowadays, malware has become a big threat to computer systems. Most of the malwares intrude users' systems as they download files over Internet. When people gain access to those infected files, it is likely to bring up such a big harm to their operating system, network devices and so on. There are several common malware types existing through the years including Trojans, Adware and Botnet.

Therefore, malware detection has become a significant issue recently. Malware detection can be classified into two categories: anomaly-based detection and signature based detection. The former identifies what is considered normal in benign programs and distinguishes whether the testing program meets the benign one; while the latter emphasizes what is considered malicious to inspect the similarity between the feature of the testing program and the malicious program.Commonly used malware detection features include string signatures, byte sequence n-grams, control flow graphs, and so on. Signatures are manually introduced in a database by experts but it cannot recognize new malware. To solve this problem, machine learning techniques are used for malware detection to automatically learn from malware samples.

Machine learning-based static PE malware detectors have been used since at least 2001 [21], and owing largely to the structured file format and backwards-compatibility requirements. Current approaches requires feature engineering to convert information contained in file properties, raw binary, and pe file headers into representations that can be used by machine learning algorithms.

In recent years, deep learning techniques such as deep neural networks, convolutional neural networks and recurrent neural networks are widely applied and achieve significant performance in machine learning problems. Driven by the success of deep learning, researcher have borrowed ideas from convolution neural networks, recurrent neural networks, and deep autoencoders to design the architecture of graph neural networks which aims to handle the complexity of graph data. Graph convolutional network is a kind of graph neural network inspired by convolutional networks. It re-defines the notation of convolution for graph data and handle the sophisticated structure of graph.

Our main purpose is to implement a graph convolutional network for malware detection. Graph-based features such as control flow graph or function call graph are strong features in malware detection. It provides the ability to model deeper information such as binary graph geographic structures and instruction sequences in basic blocks. With advanced deep graph neural network, we can make end-to-end training and automatically extract features from graph structure data and direct similarity compare. This helps to release the limitations of massive labeled data which requires extensive human effort with domain knowledge.

We summarize our contribution as follow:

(1) We improved the graph-based detection process with automatic feature extraction from graph structure data and make it into an end-to-end manner for a single malware binary instead of a single malicious function code inspired by the work of [24] . It dramatically reduce human effort with domain knowledge and training time.
(2) We improved the feature engineering process with the work of [25] which uses neural machine translation techniques to model cross-architecture instruction sequences.

2 PROBLEM DEFINITION (10%)

Given a raw binary program, we need to determine whether it is a malicious program or a benign program for malware detection.

This can be viewed as a traditional binary classification for machine learning, so it is often solved by machine-learning based techniques. Various kinds of features such as pe header, byte histogram, opcode sequences, etc can be extracted from raw binary files. With these features and pre-defined labels (malicious or benign) for each raw binary program, a machine learning model can be trained and utilized to predict a new-incoming raw binary file when its features being extracted. Graph-based features such as control flow graph is also a kind of feature model the graph structure of raw bianry program. Graph convolutional networks which handle the structure of graph data is a ideal methodology for this task give control flow graph as input features. We modified the original graph convolutional networks and incorporated neural machine translation techniques to build our graph convolutional network for malware detection. Some classical malware detection methods such as PE header + random forest, LSH + KNN, EMBER, Malconv and are also implemented for comparison.

3 RELATED WORK (10%)

Static malware analysis mostly based on information in binary file such as API calls analysis, header information, opcode sequences and other general characteristics.

Graph based approaches for detecting malware is an active area of research for malware targeted at PCs [4, 10, 13, 14, 16] as well as mobiles [6, 17, 26]. A popular approach is to build Program Representation Graphs [17], i.e. graphs that capture semantic representations derived from static analysis, e.g. function call graphs, control and data flow graphs, and data and program dependency graphs [3, 6, 9, 11, 12, 19]. In these related works, different graph sturcture data are used such as function call graph, control flos graph, data dependecy graph, etc. The thing in common is that all of them utilize a graph embedding framework Struc2Vec [7] to make similarity learning and graph representation. The main disadvantage is that the struc2vec framework actually learns a node embedding for each vertex. This means that every single malware sample needs to be optimized respectively which will cost many time and effort. Furthermore, all related works use a aggregator by performing weighted sum to all node embeddings to represent the entire graph. Such linear combination would cause high-dimensional information loss since each embedding contains the node's geometric properties such as proximity, k-degree, clustering properties, etc. These drawbacks are the main reason that motivates us for making improvements in this work.

In our survey, most related works extract graph based features such as betweenness centrality, closeness centrality, degree, shortest path length, density, number of edges and nodes, which would loss many informations. In contrast, we would like to make end-to-end model with automatically features extracting from graph structure data and preserve the entire graph information including informations such as API calls and opcode sequences and thus will outperform the each of them respectively. In addition, we would like to examine how the graph matching networks for learning the similarity of graph structured objects can be used in detecting malware.

Methods	classification algorithn	feature
PE header	random forest	header information
LSH	KNN	whole file
Malconv	CNN	byte sequence
EMBER	gradient boosting tree	domain features

Table 1: Baseline Model

4 RESULTS (50%)

4.1 Baseline

We implemented four different baseline models. Each of them are machine learning based detectors which are very commonly in use.

(1) PE header + random forest : PE header contains very useful information. Several works [Learning the PE Header, Malware Detection with Minimal Domain] have focus on detecting malware by merely header information and is proven that information from the PE header alone can work well in detecting malware.
(2) LSH + KNN : Locality-Sensitive Hashing (LSH) is an algorithmic technique that hashes similar input items into the same "buckets" with high probability. The technique can be seen as a way to reduce the dimensionality of high-dimensional data. By turning binaries into a hash value, we can conduct efficient O(1) online search for similar hash value and make use of the k-nearest neighbors classification algorithm (k-NN) to detect malware. The advantages of this method is that it is very fast and does not require domain knowledge.
(3) EMBER [5] : EMBER is a recent released open source project that provides useful feature extraction parsers and aims to help the machine learning community in malware detection. It uses gradient boosting trees as classifiers and mainly uses features extracted from domain experts.
(4) Malconv [18] : MalConv, a recently published end-to-end (featureless) deep learning model for malware detection. This approach utilize CNN for automatic feature extraction and end-to-end classifications. Mainly aims to reduce domain knowledge and make the classification data driven.

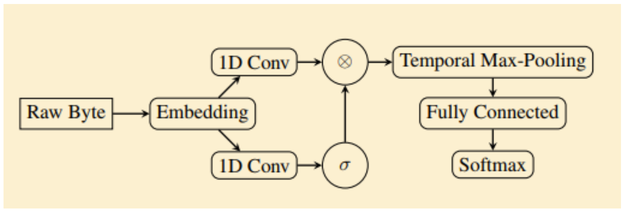

Figure 1: Malconv architecure

4.2 Methodology

Our model is based on graph convolutional network and word2vec algorithm.

- **Graph Convolutional Network** is a kind of graph neural network which its convolutional structure is inspired by standard 2D convolution for traditional data (images or grids). Its main idea is to learn a function f to generate a node representation by aggregating its own features and neighbors' features. It plays a important role in building up many other complex graph neural network models such as auto-encoder-based models, generative models, and spatial-temporal networks, etc.

- **Word2vec** is one of the most popular word embedding techniques based on two main algorithms : Skip-gram or continuous bag-of-words. The learned representations for words and phrases exhibit linear structure such as vector('Paris') - vector('France') + vector('Italy') results in a vector that is very close to vector('Rome'). With such charaetestics, it is capable of capturing context of a word in a document, semantic and syntactic similarity, relation with other words and widely used in many Natural Language Processing tasks.

Control flow graph extracted from raw binary program can be used as graph data for training graph convoltional networks. Multiple code blocks containing a sequence of instruction codes form such control flow graph can be viewed as nodes in the graph. A sequence of instruction code in a code block is similar to a sequence of word in a sentence. Treating sequences of instruction codes as sequences of words, we can derive instruction embedding just like word embedding by training word2vec algorithm on sequence corpus. Instead of using traditional node features, we use average of a sequence of instruction embedding in each code block as node features. Combining graph convolutional network and node features deriving from word2vec algorithm, our model can be built and trained.

4.3 Experiment

4.3.1 Datasets. We collect malicious and benign programs from Virustotal [23], Virusshare [2], MalShare [1]. There are totally 39405 programs. 19491 of them are malicious programs and 19914 of them are benign programs. Despite the huge amount of malware samples, many of the malware sample way be in the format of NE (new executable), pe embedded in html, dos scripts, etc.

4.3.2 Preprocessing. Preprocessing the malware sample is the main bottleneck in the entire workflow. Constructing control flow graph fro a single malware sample may need several minutes. Assume that each sample needs 3 minutes for preprocessing in average, 10000 samples would need 500 hours which is about three weeks. In order to solve the problem, we run the pre-process work with 16 cores and on 3 different machines for about a week.

To construct control flow graph for each program, we use Angr [22] to extract its control flow graph and store the graph structure in pickle format via Networkx [8]. To be noticed that many malware sample are actually malformed and cause the parser in angr cannot work correctly. We made huge effort to clean this up and track codes in angr to deal with this problem.

To meet the requirements for training graph convolutional network, we need to assign a graph id for the graph and a node id for each node in the graph. Gensim [20], which implements the word2vec algorithms, is used for training instruction embedding for deriving feature vector for each node. A graph is labeled as positive if extracted from malicious program, as negative if extracted from benign program for graph label. Each graph with graph id and graph label, which contains a set of nodes with node id and node feature vector, can be viewed as a sample when training graph convolutional network. The major problem encountered in this step is that graph structure data requires much time to operate and need huge memory to fit in. It also implies that the training process can only be accelerate by GPU servers equipped with huge memory.

4.3.3 Setup. Although we collect many programs from virus scanner websites, not all of them are included in our experiments and we will discuss the reasons detailed later. Finally, there are totally 1425 samples which their graph structures are successfully extracted from binary program are ued, 614 of them are positive samples and 811 of them are negative samples. We assign neuron size of 32, 64, 16 for each layer. Dropout rate is set for 0.5 and earlystopping is used to avoid overfitting.

4.4 Evaluation

For these collected samples, we split 80 percent of the data for training our model and 20 percent of the data for testing our model performance. AUC metric is calculated for comparing baseline model and our graph convolutional model.

As metioned in section, the preprocess step took too much time so that we only got about 18000 each for malicious and benign data for training baseline models. All of the baseline models achieved 95% in AUC which is a reasonable result according to our survey. Due to the huge usage of memory of our proposed model and time constraints, we only conduct the experiments with 800 training data. The result of the experiment is around 80%. Although the result does not meet our expectation, the result is reasonable and under estimated. Not only because the disparity gap between the amount of training data, it is also because our proposed model only consider the graph geometric feature and instruction sequence but lack the most important part of the pe header. There are analysis on machine learning based malware detection [15] which found that the most critical information of malware detection is from the pe header. To sum up, by combining the reasons above, We believe that with the same amount of training data and with pe header features, our proposed model will out perform the baselines and achieve similar scores in related works.

5 CONCLUSION AND FUTURE WORK (5%)

By training graph convolutional network based on control flow graph extracted from raw binary program and instruction embedding learned from instruction sequences, we can build an end-to-end machine learning model for malware detection. Though we can not reach the performance such as baseline models, an above average performance is achieved by our model given a small amount of training samples, convincing that graph-basedd feature is a good idea to used for malware detection.

As a result, our proposed method has several potential to explore. By applying machine reasoning techniques, we can make it a automatic tool for finding suspicious binary function in never before seen binary files. It can help reducing time and effort for malware Analyst. Furthermore, this graph model can be used to craft adversarial samples efficiently since it is trained in a end-to-end manner. Adversarial samples for binary files are also an active topic since more and more malware detection uses machine learning.

Table 2

Methods	Classification Algorithm	Features	Training Data	Test AUC
PE header	random forest	PE header	36134	0.957
LSH	KNN	whole file	36134	0.973
Malconv	CNN	byte sequence	36134	0.985
EMBER	gradient boosting tree	All	80000	0.991
Graph (our result)	GNN	cfg + instructions	800	0.799

However, current works can only use evolutionary optimization process or reinforcement learning to craft adversarial samples due to binary file’s discrete nature. Our proposed model might be a way to overcome this problem.

REFERENCES

[1] [n. d.]. MalShare. https://malshare.com/
[2] [n. d.]. VirusShare.com. http://virusshare.com/
[3] Hisham Alasmary, Aminollah Khormali, Afsah Anwar, Jeman Park, Jinchun Choi, DaeHun Nyang, and Aziz Mohaisen. 2019. Analyzing, Comparing, and Detecting Emerging Malware: A Graph-based Approach. *arXiv preprint arXiv:1902.03955* (2019).
[4] Blake Anderson, Daniel Quist, Joshua Neil, Curtis Storlie, and Terran Lane. 2011. Graph-based malware detection using dynamic analysis. *Journal in computer Virology* 7, 4 (2011), 247–258.
[5] Hyrum S Anderson and Phil Roth. 2018. Ember: an open dataset for training static PE malware machine learning models. *arXiv preprint arXiv:1804.04637* (2018).
[6] The Dam and Tayssir Touili. 2017. Malware Detection based on Graph Classification. 455–463. https://doi.org/10.5220/0006209504550463
[7] Daniel R. Figueiredo, Leonardo Filipe Rodrigues Ribeiro, and Pedro H. P. Saverese. 2017. struc2vec: Learning Node Representations from Structural Identity. *CoRR* abs/1704.03165 (2017). arXiv:1704.03165 http://arxiv.org/abs/1704.03165
[8] Aric Hagberg, Pieter Swart, and Daniel S Chult. 2008. *Exploring network structure, dynamics, and function using NetworkX*. Technical Report. Los Alamos National Lab.(LANL), Los Alamos, NM (United States).
[9] William Hardy, Lingwei Chen, Shifu Hou, Yanfang Ye, and Xin Li. 2016. DL4MD: A deep learning framework for intelligent malware detection. In *Proceedings of the International Conference on Data Mining (DMIN)*. The Steering Committee of The World Congress in Computer Science, Computer âĂę, 61.
[10] Xin Hu, Tzi-cker Chiueh, and Kang G Shin. 2009. Large-scale malware indexing using function-call graphs. In *Proceedings of the 16th ACM conference on Computer and communications security*. ACM, 611–620.
[11] Jae-wook Jang, Jiyoung Woo, Aziz Mohaisen, Jaesung Yun, and Huy Kang Kim. 2015. Mal-netminer: Malware classification approach based on social network analysis of system call graph. *Mathematical Problems in Engineering* 2015 (2015).
[12] Haodi Jiang, Turki Turki, and Jason TL Wang. 2018. DLGraph: Malware Detection Using Deep Learning and Graph Embedding. In *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 1029–1033.
[13] Chanhyun Kang, Noseong Park, B Aditya Prakash, Edoardo Serra, and VS Subrahmanian. 2016. Ensemble models for data-driven prediction of malware infections. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*. ACM, 583–592.
[14] Bum Jun Kwon, Jayanta Mondal, Jiyong Jang, Leyla Bilge, and Tudor Dumitraş. 2015. The dropper effect: Insights into malware distribution with downloader graph analytics. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1118–1129.
[15] Demetrio Luca, Battista Biggio, Lagorio Giovanni, Fabio Roli, and Armando Alessandro. 2019. Explaining Vulnerabilities of Deep Learning to Adversarial Malware Binaries. In *ITASEC19*.
[16] Carey Nachenberg, Jeffrey Wilhelm, Adam Wright, and Christos Faloutsos. 2010. Polonium: Tera-scale graph mining for malware detection. (2010).
[17] Annamalai Narayanan, Guozhu Meng, Liu Yang, Jinliang Liu, and Lihui Chen. 2016. Contextual Weisfeiler-Lehman graph kernel for malware detection. In *2016 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 4701–4708.
[18] Edward Raff, Jon Barker, Jared Sylvester, Robert Brandon, Bryan Catanzaro, and Charles K Nicholas. 2018. Malware detection by eating a whole exe. In *Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*.
[19] Deebiga Rajeswaran, Fabio Di Troia, Thomas H Austin, and Mark Stamp. 2018. Function call graphs versus machine learning for malware detection. In *Guide to Vulnerability Analysis for Computer Networks and Systems*. Springer, 259–279.
[20] Radim Řehřek and Petr Sojka. 2011. GensimâĂŤstatistical semantics in python. *statistical semantics; gensim; Python; LDA; SVD* (2011).
[21] Matthew G Schultz, Eleazar Eskin, F Zadok, and Salvatore J Stolfo. 2000. Data mining methods for detection of new malicious executables. In *Proceedings 2001 IEEE Symposium on Security and Privacy. S&P 2001*. IEEE, 38–49.
[22] Yan Shoshitaishvili, Ruoyu Wang, Christopher Salls, Nick Stephens, Mario Polino, Audrey Dutcher, John Grosen, Siji Feng, Christophe Hauser, Christopher Kruegel, and Giovanni Vigna. 2016. SoK: (State of) The Art of War: Offensive Techniques in Binary Analysis. In *IEEE Symposium on Security and Privacy*.
[23] Gaurav Sood. 2017. *virustotal: R Client for the virustotal API*. R package version 0.2.1.
[24] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. 2018. An end-to-end deep learning architecture for graph classification. In *Thirty-Second AAAI Conference on Artificial Intelligence*.
[25] Fei Zuo, Xiaopeng Li, Zhexin Zhang, Patrick Young, Lannan Luo, and Qiang Zeng. 2018. Neural machine translation inspired binary code similarity comparison beyond function pairs. *arXiv preprint arXiv:1808.04706* (2018).