ESANN 2021 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Online event, 6-8 October 2021, i6doc.com publ., ISBN 978287587082-7.
Available from http://www.i6doc.com/en/.

Validating static call graph-based malware signatures using community detection methods

Attila Mester[1,2] and Zalán Bodó[1]

1- Babeş–Bolyai University - Faculty of Mathematics and Computer Science
Cluj-Napoca - Romania

2- Bitdefender
Cluj-Napoca - Romania

Abstract. Due to the increasing number of new malware appearing daily, it is impossible to manually inspect each sample. By applying data mining techniques to analyze the program code, we can help manual processing. In this paper we propose a method to extract signatures from the executable binary of a malware, in order to query the local neighborhood in real time. The method is validated by applying community detection algorithms on the common fingerprint-based malware graph to identify families, and assessing these with evaluation metrics used in the field (e.g. modularity, family majority, etc.). The signatures are obtained via static code analysis, using function call n-grams and applying locality-sensitive hashing techniques to enable the match between functions with highly similar instruction lists.

1 Introduction

The number of new malicious programs implies the need to automate malware analysis. This paper presents a method to extract signatures (or fingerprints) of a malicious sample. These features can be used in an indexing engine to query the neighborhood of a given sample in real time. In order to demonstrate the capability of the signatures to cluster malware families, we build a common fingerprint-based malware graph by applying community detection algorithms, evaluating it using modularity, coverage, performance, etc. metrics. The signatures are extracted based on the function call graph [1].

The novel idea in this work is the way the information is extracted from local subroutines, without training a model, enabling the continuous analysis of the daily incoming new samples. These local subroutines are represented using a locality-sensitive hashing (LSH) method, based on their instruction n-gram distribution. The final fingerprints consist of n-grams of such LSH codes.

The paper is structured as follows. In Section 2 we briefly overview the existing ideas and research results in the field, indicating our decision to work with call graphs. Section 3 describes the process of obtaining the call graphs, the extraction of the fingerprints, building the malware graph and detecting communities in it. The paper ends with Section 4, presenting the results, concluding the experiments and discussing future directions.

ESANN 2021 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Online event, 6-8 October 2021, i6doc.com publ., ISBN 978287587082-7.
Available from http://www.i6doc.com/en/.

2 Related work

Applying machine learning methods to analyze malware has been extensively studied in the literature. The authors of [2] categorize the different approaches considering the objective (family/category/metamorphic variant selection and similarity detection), type of the analysis (static, based on the binary file, e.g. byte sequences, or dynamic, examining the malware's behaviour in a sandbox environment), and the algorithms applied. Investigating the bibliography of the survey, the following conclusions can be drawn: the most prevalent approach is to apply a supervised learning method, while the most frequently used features are API/syscalls, byte sequences and API call graphs. Using the API graph features, it is common to apply graph matching algorithms or calculate graph edit distances (GED) [3]. Another approach is to build a feature vector of the graph based on n-grams [4].

Convolutional neural networks are also successfully applied to detect patterns in opcode and (dynamic) syscall sequences for identifying malicious code [5, 6]. Malware clustering based on locality-sensitive hashing was already experimented in [7]. In [1], LSH is applied on local subroutines, using minhash signatures to approximate similarity between the instruction sets of two subprograms.

In this paper we extract multiple fingerprints from the call graph, enabling the continuous processing and clustering of new incoming samples, without the need to retrain the underlying model; the nearest neighbors of a sample can be determined based on the common fingerprints. The proposed signatures are validated applying community detection algorithms on the common fingerprint-based malware graph. Another key aspect of this paper is the validation of the industrially used features of [8] as well.

3 N-grams for malware clustering

3.1 Generating the fingerprints

The fingerprints are extracted from the static call graph, obtained by IDA Pro 6 disassembler, combining the outputs of two APIs: *GenCallGdl* and *GenFuncGdl*. In this paper we use the term *call graph* to name the result of processing the outputs of these two APIs (details in [9]): a node is a function name (API/sys, or local subroutine – containing its instruction list too), a link is a caller—callee relation. A fingerprint is an n-gram in the resulting call graph: unigram meaning a codeword for a subroutine, while bigram standing for a caller—callee codeword pair, where possible combinations are local subroutine—local sub., local sub.—DLL function, and local sub.—statically linked function. Only in case of DLLs the function names themselves are used as codewords.

The codewords used for representing a subroutine are obtained via an LSH algorithm [10], mapping the data points to hashes, such that similar points are likely to be put in the same (or nearby) hash bucket, significantly reducing the number of computations for finding the neighbors. In the present experiments the random hyperplane-based method of [11] is used, that generates a

ESANN 2021 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Online event, 6-8 October 2021, i6doc.com publ., ISBN 978287587082-7.
Available from http://www.i6doc.com/en/.

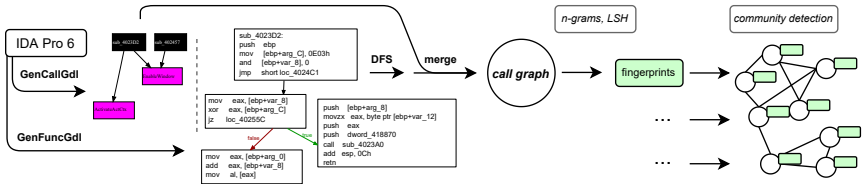

Fig. 1: Pipeline of generating the fingerprints and validating these by applying community detection methods on the malware graph.

low-dimensional binary embedding of the data, in a way that the probability of collision is "closely related" to the cosine similarity of the points. Our goal is to reduce the dimensionality of the n-gram-based representation of the subroutines, allowing slight differences in the instruction sequences of two functions. Therefore, two programs can have common fingerprints even if they share only a fraction of their instruction n-grams. Thus, some of the precision is sacrificed for getting a higher recall [12], as we consider recall important in this scenario.

3.2 The malware graph

In order to measure the usefulness of the generated signatures, community detection methods are applied on the common fingerprint-based malware graph – where nodes represent the malware, edges indicate common fingerprints (Fig. 1).

We consider the following requirements: (i) the malware graph should consist of dense components, with few edges between them – meaning the separation of the malware groups from each other; (ii) ideally, the nodes of a component should share the same family label. These dense (and homogeneous) components are called communities in a network. The most prevalent community detection algorithms are Clauset–Newman–Moore (CNM), Label propagation algorithm (LPA), Louvain and Infomap [13], therefore we chose these for validating our approach. In order to evaluate the topological properties of the obtained malware graph, we calculate *modularity*, *coverage* and *performance* scores [13]. Since the family distribution of the clusters is similarly important, we also measure the majority percentage of the labels within each cluster.

4 Experimental results and discussion

In the experiments we used the following technologies and libraries: Python 3.6 (*networkX* library), IDA Pro 6, Graphviz, Gephi 0.9.2. We carried out experiments considering the following: subroutine instruction n-grams (1–3-grams, 2–3-grams or trigrams); number of random hyperplanes (8 or 16); projection partition (by projecting a vector onto a hyperplane and taking the sign of the projection to generate the hash codewords, the distance information is lost – enabling this parameter sets distance intervals of $[0, 10]$, $(10, 100]$, $(100, 1000]$ and > 1000, and symmetrically with negative signs, to use a finer partition of

ESANN 2021 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Online event, 6-8 October 2021, i6doc.com publ., ISBN 978287587082-7.
Available from http://www.i6doc.com/en/.

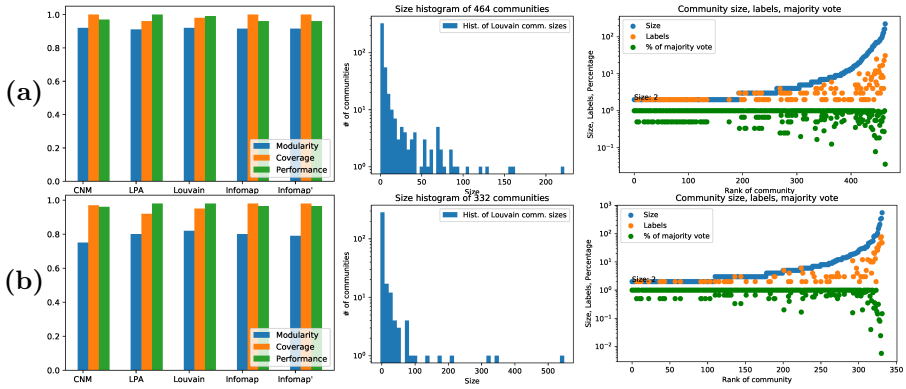

Fig. 2: Comparison of modularity, coverage, performance (1st column), size distribution (2nd column) and majority percentage (3rd column) of the labels within communities in different malware graphs (**(a)** and **(b)**).

the space, thus obtaining finer hash codes); call graph n-grams (unigrams or bigrams, as described in Section 3.1). This resulted a total of 13 runs – discarding some of the 16 hyperplanes option due to being computationally too expensive, except the following: instruction 1–3-grams, projection partition disabled, call graph bigrams. On the final graph, the following filters are applied: frequency of fingerprints ($[2, 80]$, $[2, 100]$, $[3, 100]$, $[10, 100]$ or $[10, 400]$); edge weight (minimum 5, 10, 100, 200 or 1000). These parametrizations resulted in a total of 260 runs, some of them being discarded, as we chose a minimum requirement of 3000 nodes for the results to be comparable between each other.

4.1 Dataset used

The private Bitdefender dataset consists of 7977 samples from 254 families, a family having 30.3 ± 41.96 samples on average – the mean and variance demonstrate its imbalanced nature, typical in this domain. The number of subroutines per sample is on average 1163 (median of 354 and maximum of 65 060) – summing up to a total number of 9 284 242 subroutines with 650 225 unique mnemonic sequences (discarding instruction parameters like registers, addresses). The number of unique mnemonic n-grams in the dataset is as follows: 1783 unigrams, 61 910 bigrams, 559 835 trigrams, resulting in a 623 528 long sparse vector for each subroutine (having 58 ± 70.64 nonzero values, on average).

4.2 Results and discussion

We calculated the above-mentioned evaluation metrics for different malware graphs, based on the following fingerprints: **(a)** internal, industrially used fingerprints [8], **(b)** fingerprints described in this paper: instruction 2–3-grams, 8 hyperplanes, projection partition, call graph bigrams, $[2, 100]$ frequency filtering and min. weight of 100 – yielding one of the best results in our experiments.

ESANN 2021 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Online event, 6-8 October 2021, i6doc.com publ., ISBN 978287587082-7.
Available from http://www.i6doc.com/en/.

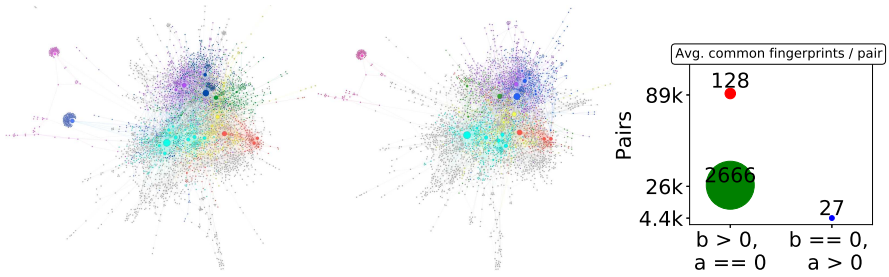

Fig. 3: Comparison of two call graphs – (x) and (y) – from the green bubble scenario of (z) (the extra function group on the left graph shows a handful of *winsock2* functions).

In Fig. 2 we compare different metrics for the above configurations. The first bar plot compares the topological properties (modularity, coverage, performance) of the malware graph's partition, using different community detection algorithms. The second plot is the size histogram of the Louvain communities, while the rightmost plot shows the majority label percentage of each community, ranked according to their sizes. The blue, orange and green series represent the size of the communities, number of different families, and the majority percentage across the families – plotted on a logarithmic scale. Ideally, the yellow and green dots should remain close to 1, regardless of the growth of the blue series.

Based on the results depicted in Fig. 2, the following conclusions can be drawn: *(i)* the industrially used fingerprints can cluster families with high modularity and majority label percentage scores; *(ii)* configuration **(b)** provides similarly good results as **(a)**; *(iii)* instruction n-grams, sequences yield better representation than simple unigrams; not only the code of the subroutines, but also the sequentiality of these subroutines can characterize malware families – the best configurations resulted from selecting instruction bigrams and trigrams, and bigrams from the call graph.

To conclude our experiments, we describe a case of two samples having the same family within the same community, (x) and (y) (both of size of $\approx$ 800 kB). In the call graph of (x) and (y) appear the exact same 384 DLL functions; (x) has 2884 statically imported library functions, while (y) has 2522, of which 2345 are identical; in (x) there are 809 subroutines, while in (y) 749, sharing the exact same code in 734 of these. Their similarity is reflected by the topology of their call graphs (Fig. 3), and the number of common fingerprints too: (x) having 7698, (y) having 6942 fingerprints (based on **(b)**), of which 4878 are shared. This is just one of thousands of cases where **(a)** does not yield common fingerprints, while **(b)** does – the analysis of such cases is depicted in Fig. 3(z). Here we show – regarding **(a)** and **(b)** – the number of virus pairs having the same family, where one method does not yield common fingerprints, while the other does, listing the avg. common fingerprints is such cases. The first column depicts two

ESANN 2021 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Online event, 6-8 October 2021, i6doc.com publ., ISBN 978287587082-7.
Available from http://www.i6doc.com/en/.

cases: the upper bubble representing inter-community pairs, the lower showing samples from identical communities. The second column refers to pairs from different communities where **(b)** does not yield common fingerprints, while **(a)** does – significantly smaller number than vice-versa. This plot demonstrates the capability of our method to capture new, relevant information from the samples.

In this paper we present a novel method to extract fingerprints from static analysis of a malicious sample, by applying LSH on its subroutines and gathering n-grams from its call graph. After testing the proposed method we also evaluate the usefulness of the industrially used fingerprints regarding their ability to cluster families. Possible future directions include but are not limited to a more comprehensive analysis of the call graph, revealing its topological structure, or using convolutional networks for finding useful patterns.

Acknowledgments

This project was supported by Bitdefender. I would like to thank my colleagues and managers, A. Mihalca, C. Oprişa, G. Cabău, O. Ardelean for their support. I am grateful for the academic guidance of my supervisor, prof. dr. A. Andreica.

References

[1] M. Hassen and P. K. Chan. Scalable function call graph-based malware classification. In *CODASPY*, pages 239–248, Scottsdale, AZ, USA, 2017. ACM.

[2] D. Ucci, L. Aniello, and R. Baldoni. Survey of machine learning techniques for malware analysis. *Computers & Security*, 81:123–147, 2019.

[3] Y. Park, D. Reeves, V. Mulukutla, and B. Sundaravel. Fast malware classification by automated behavioral graph matching. In *CSIIRW*, pages 1–4, Oak Ridge, TN, USA, 2010. ACM.

[4] G. E. Dahl, J. W. Stokes, L. Deng, and D. Yu. Large-scale malware classification using random projections and neural networks. In *ICASSP*, pages 3422–3426, Vancouver, BC, Canada, 2013. IEEE.

[5] N. McLaughlin, J. Martinez del Rincon, B. Kang, et al. Deep Android malware detection. In *CODASPY*, pages 301–308, Scottsdale, AZ, USA, 2017. ACM.

[6] F. Martinelli, F. Marulli, and F. Mercaldo. Evaluating convolutional neural network for effective mobile malware detection. *Procedia Computer Science*, 112:2372–2381, 2017.

[7] C. Oprişa, M. Checicheş, and A. Nădrean. Locality-sensitive hashing optimizations for fast malware clustering. In *ICCP*, pages 97–104, Cluj-Napoca, Romania, 2014. IEEE.

[8] V. I. Topan, S. V. Dudea, and V. D. Canja. Fuzzy whitelisting anti-malware systems and methods, 2013. US Patent 8,584,235.

[9] A. Mester. Scalable, real-time malware clustering based on signatures of static call graph features. Master's thesis, Babeş–Bolyai University, Faculty of Mathematics and Computer Science, Cluj-Napoca, Romania, 2020.

[10] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *STOC*, pages 604–613, Dallas, Texas, USA, 1998. ACM.

[11] M. S. Charikar. Similarity estimation techniques from rounding algorithms. In *STOC*, pages 380–388, Montréal, Québec, Canada, 2002. ACM.

[12] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to information retrieval*. Cambridge University Press, Cambridge, UK, 2008.

[13] S. Fortunato. Community detection in graphs. *Physics Reports*, 486(3-5):75–174, 2010.