Adding semantic to level-up graph-based Android malware detection

Roxane Cohen[1,2], Florian Yger[1], and Fabrice Rossi[2]

[1] LAMSADE, CNRS, Université Paris-Dauphine, PSL Research University
[2] CEREMADE, CNRS, Université Paris-Dauphine, PSL Research University

1 Introduction

Polymorphic malwares operate changes in their source code, while their semantics remain the same, which impair detection capabilities of static analysis methods. However, oriented Function Call Graphs (FCG), where vertices represent functions of the program and edges denote interprocedural calls, have been shown to provide reliable descriptions of their programs, see e.g. [3]. Inspired by those results, a very large database of Android FCG was released called MalNet [2]. MalNet graphs are reduced to FCG structure only, without any information about the represented functions. By recreating the tiny version of MalNet, this time keeping the semantic attributes of the graph, we can establish a comparison between structure-only based machine learning methods and techniques that combine graph structure and semantics.

2 Approaches

We study two approaches by considering two types of features from the FCG: only structural ones for the unlabelled MalNetTiny and features that combine structure and semantics for the labelled version. The first consists in various graph and nodes features, like the number of nodes, in both directed and undirected cases. The second takes advantage of the the rich available API in Android that can be used to characterize to some extent the semantics of a program (see e.g. [1]). Notice that we can only extract from the apk calls from so-called *encoded nodes* (i.e. functions whose bytecode are contained in the DEX files of an Android application) to *external nodes* (i.e. functions which are not and that in general come from the Android API). We then extract and combine structural and semantic features as follows.

We first notice that MalNetTiny uses more than 42,000 external classes, some of which come from the Android API, with vastly different uses from a program to another. To capture this variability at a reasonable computational cost, we represent each FCG by its top 3 functions/encoded nodes in terms of API usage diversity, with a counting vector over the API classes. We use PCA to reduce the dimensionality of the representation to 32 components, explaining more than 75 % of the variance. Each encoded node is then represented by the projection of its external connections to those 32 coordinates. To obtain a fixed size representation of arbitrary FCG, we use two different methods: either an average call pattern, where a FCG is represented by the average of the vector representations of all of its encoded nodes; or average call and covariance, where we simply add the flatten covariance matrix.

3 Results

We compare the quality of the above features, structural ones first and PCA-reduced based second, by using them to classify MalNetTiny into five classes of malwares. In order to do so, we either use a RandomForest as a traditional machine learning method or a Graph Neural Network (GNN) model. We split the data set into a training (80 %) and test sets. We use 10 fold cross validation on the learning set to set the hyperparameters of the classifier and report the accuracy on the test set. The process is repeated 5 times.

RandomForest combined with graph features achieves around 0.869 accuracy with a very little difference between runs. RandomForest and PCA features with both structural and semantic informations perform better, with an accuracy of 0.908 for only PCA features and 0.921 if we add covariance. Surprisingly, GNN give significantly worse results, with an accuracy of respectively 0.674 and 0.753 for node degree features in undirected and directed cases. GNN with PCA features are still worse than simpler baselines with an accuracy of 0.896. We implemented a dozen of different models and tested various types of GNN, such as GCN, GraphSAGE and GIN. The hyperparameters of the GNN are selected on a validation set which represents 25% of the training set. The weaker results for GNN can be explained by the particular structure of the reduced graphs after applying PCA. Since FCGs come from Android API, reduced graphs contain many isolated nodes, which may prevent GNN from properly propagating informations between nodes.

4 Conclusion and future work

In this work, we demonstrate that while structural features on MalNetTiny can be used to classify malwares, significant improvements of the classification rate can be obtained by representing each graph by a low dimensional feature vector extracted from external calls to the Android API. Finally, we show that simpler shallow models can have worthwhile performances and we advocate for their systematic use as baselines in graph classification tasks. There are several possible ways to improve this work by validating hyperparameters or, since we only consider external class API calls, by increasing the feature granularity and taking into account external methods calls.

References

1. Aafer, Y., Du, W., Yin, H.: Droidapiminer: Mining api-level features for robust malware detection in android. In: Security and Privacy in Communication Networks (2013)
2. Freitas, S., Dong, Y., Neil, J., Chau, D.H.: A large-scale database for graph representation learning. NeurIPS (2021)
3. Hu, X., Chiueh, T.c., Shin, K.G.: Large-scale malware indexing using function-call graphs (2009)