Android Malware Identification and Polymorphic Evolution Via Graph Representation Learning

Miguel Quebrado
Computer Science dept.
Boise State University
Boise, ID, USA
miguelquebrado@u.boisestate.edu

Edoardo Serra
Computer Science dept.
Boise State University
Boise, ID, USA
edoardoserra@boisestate.edu

Alfredo Cuzzocrea
iDEA Lab
University of Calabria,
Rende, Calabria, Italy
alfredo.cuzzocrea@unical.it

***Abstract*—Developing techniques to identify malware is critical. The polymorphic nature of malware makes it difficult to detect, especially if the detection is done with Hash-based based techniques. Image-based binary representations have been shown to be more robust to popular polymorphic obfuscation techniques. In contrast to image-based techniques, in this paper, we employed a graph-based technique that extracts control flow graphs from Android APK binary. To process the resulting graph, we use a procedure combining a new graph representation learning method, called Inferential SIR-GN for Graph representation, that preserves graph structural similarities, with XGboost, which is a standard machine learning model. Then, we apply this procedure to MALNET, which is a publicly available cybersecurity database that provides image and graph-based Android APK binary representations for a total 1,262,024 million Android APK binary with 47 types and 696 families. Experimental results show that this graph-based procedure is even more accurate than the image-based approach. Moreover, this paper provides a procedure that, by leveraging Inferential SIR-GN is able to create malware polymorphic evolution representations to use during the train of the XGboost that strengthens the malware classification tasks when the train and test datasets are split temporally according to the binary creation date. This means that our procedure can predict malware polymorphic evolution.**

***Index Terms*—Obfuscation, Neural Networks, Structural Graph Representation Learning, Malware Polymorphism.**

I. Introduction

The economic costs that malicious cyber activity has on the U.S. economy can be challenging to determine, but these attacks cost the economy anywhere between $57 billion and $109 billion in 2016[1]. In a data-driven business world, hackers leverage advanced techniques, technologies, and polymorphic methods to compromise networks. Cyberattacks are often highly sophisticated, targeting governments and large-scale enterprises to interrupt critical services and steal intellectual property[2].

Malware applications are one of the main reasons why such attacks are possible and successful. Identifying malware is a difficult task, but there are two common approaches to analyzing malware static code analysis and dynamic code analysis. Static analysis works by disassembling the code and exploring the control flow of the executable to look for malicious patterns without actually running the code. Dynamic analysis involves executing the code in a virtual environment; this approach is behavior-based, so the important methods can be identified.

The static analysis offers complete coverage, but it usually suffers from code obfuscation. The executable has to be unpacked and decrypted before analysis, and even then, the analysis can be hindered by problems of intractable complexity. The dynamic analysis does not need the executable to be unpacked or decrypted. Unfortunately, as noted in [3], dynamic analysis can still be time-intensive and resource-consuming. Moreover, some malicious behaviors might be unobserved because the environment does not satisfy the triggering condition [3]. For windows and android malware, the industry has turned to image-based malware presentations as they are quick to generate, require no feature engineering, and are resilient to some common obfuscation techniques (e.g., section encryption [3]).

However, in the specific context of Android OS, static analysis is effective, and extracting the control flow graph is doable. Moreover, similar to the image, once the graphs are produced, they do not need any specific future engineering process since the well-established field of graph representation learning automatically creates the feature representing the graph.

Graph representation learning methods have emerged across many scientific fields and are driving the development of representation learning techniques. Graph representation learning techniques encode structured information into low dimensional space for a variety of important downstream tasks (e.g., toxic molecule detection, community clustering, malware detection)[4].

Graph representation learning methods are divided into methods preserving the connectivity information of the nodes and the methods preserving nodes' structural information. While there are a lot of works that focus on preserving node connectivity, only a few works focus on preserving nodes' structure. Properly encoding nodes' structural information is fundamental for many real-world applications as it has been demonstrated that this information can be leveraged to successfully solve many tasks where connectivity-based methods usually fail [5]. Malware analysis through the extraction of control flow graphs is another field where the structural pattern of the graph can distinguish malicious from benign activities.