e-ISJN: A4372-3114
p-ISJN: A4372-3115

ISSN: 2321-7782 (Online)
ISSN: 2347-1778 (Print)
Impact Factor: 6.012

Volume 12, Issue 3, March 2024

International Journal of Advance Research in Computer Science and Management Studies

Research Article / Survey Paper / Case Study

Available online at: www.ijarcsms.com

Advancements in Android Malware Detection: A Comprehensive Review of Graph Neural Network Approaches

Pallavi Papankar[1]
G. H. Raisoni University,
Amravati, India.

Nitin Mandaogade[2]
G. H. Raisoni University,
Amravati, India.

DOI: https://doi.org/10.61161/ijarcsms.v12i3.2

Abstract: ***Within the rapidly evolving realm of mobile technology, the imperative for robust security measures against Android malware is increasingly apparent. Traditional detection methods, grappling with the intricacies of contemporary threats, highlight the need for innovative strategies in malware identification and analysis. This review critically evaluates existing methodologies, spotlighting their limitations in accurately representing the intricate structure of Android applications (APKs) and effectively detecting malwares. These methodologies encounter challenges in addressing the dynamic and complex nature of APK components, leading to suboptimal detection rates and a lack of transparency in decision-making processes. Furthermore, scalability concerns persist as the Android application universe continues to expand. In response to these challenges, this review explores a pioneering framework based on Graph Neural Networks (GNNs) designed to revolutionize Android malware detection. The proposed model excels in feature extraction and the representation of APKs as graphs, capturing nuanced relationships among components such as permissions, API calls, and code segments. The core of this framework lies in its robust malware detection model, distinguishing between benign and malicious applications with unprecedented precision. Notably, the model prioritizes transparency and interpretability, offering insights into its decision-making process. Additionally, the review delves into the scalability and efficiency of the system, optimized for real-time analysis to handle the continuous influx of new Android applications. While empirical results are not included in this review, the discussion underscores the potential of the proposed architecture. This work sets a new standard in Android malware detection, providing a more accurate, transparent, and scalable solution. The review not only delivers a comprehensive overview of the current state of GNN-based approaches for Android malware detection but also lays the foundation for future research by proposing an architecture awaiting validation in subsequent studies.***

Keywords: ***Graph Neural Networks, Android Malware Detection, Feature Extraction, Explainable AI, Scalable Security Systems.***

I. INTRODUCTION

The rise in mobile computing technology has brought about a widespread adoption of Android applications (APKs), accompanied by a corresponding surge in security threats, particularly from malware. Android's popularity and open nature make it an attractive target for malicious activities. Addressing these evolving threats requires advanced methodologies for effective malware detection and analysis.

Traditional approaches to malware detection, relying on signature-based and heuristic methods, often fall short when dealing with sophisticated and evolving malware types. These methods encounter challenges in identifying new malware that deviates from known patterns, resulting in a notable rate of false negatives. Moreover, the increasing complexity of Android applications, characterized by intricate networks of permissions, API calls, and code segments, poses a challenge for current systems, limiting their capacity for comprehensive analyses.

In response, the field has shifted towards machine learning-based solutions, with a particular focus on Graph Neural Networks (GNNs). GNNs have demonstrated promising results in various domains by effectively capturing relational data and dependencies between components. In the domain of Android malware detection, GNNs introduce a novel approach by representing APKs as graphs, enabling a more detailed analysis of structural and behavioral patterns.

Despite these advancements, certain gaps persist. Primarily, the overlook of explainability and interpretability in these models hinders their practical application. In a domain where understanding the reasoning behind a classification is crucial, the absence of insights into why an APK is labeled as malicious poses a challenge for security analysts and decision-makers.

Secondly, the scalability and real-time analysis capabilities are imperative for practical malware detection systems. Given the continuously expanding volume of Android apps, models must efficiently analyze large datasets. Existing solutions often encounter difficulties in meeting this demand, leading to processing bottlenecks and delayed responses to emerging threats.

Conventional strategies for identifying Android malware can be broadly classified into three primary categories [16] static analysis techniques, dynamic analysis techniques, and hybrid analysis techniques. Here, we highlight a range of conventional technologies commonly employed.

In the realm of static analysis techniques, Feng et al. [17] have devised an efficient method that utilizes a blend of static taint analysis and an innovative program representation to identify Android applications exhibiting specific control- and data-flow properties. Faruki et al. [18] introduce a resilient approach that constructs a signature by extracting statistically improbable features to detect malicious Android apps. This method proves effective against code obfuscation and repackaging, tactics frequently employed to elude antivirus signatures and propagate unseen variants of known malware.

Shifting to dynamic analysis, Xiao et al. [19] adopt a method to differentiate malware based on system call sequences. Employing two distinct feature models, namely the frequency vector and the co-occurrence matrix, features are extracted from the system call sequence.

Hybrid analysis, an amalgamation of static and dynamic approaches, offers a more comprehensive perspective. Feng et al. [20] present a two-layer method for detecting malware in Android apps. The first layer employs static malware detection based on permission, intent, and component information. In the second layer, a novel method called CACNN, combining Convolutional Neural Network (CNN) and AutoEncoder, is employed to detect malware through network traffic features of apps. While hybrid analysis methods generally exhibit superior detection and recognition performance compared to static and dynamic analyses, it is important to note that they also incur higher time, hardware, and model complexity.

Static methods are generally recognized for their efficiency compared to dynamic approaches, yet they are susceptible to obfuscation techniques, such as the alteration of function names within code. Consequently, the Function Call Graph (FCG) emerges as a pivotal element in the static characterization of Android applications. It goes beyond merely memorizing the form of malware code, playing a crucial role in capturing contextual dependencies and semantic meaning of functions during runtime [4–6].

In the era of machine learning, there is a growing emphasis among researchers on leveraging machine learning techniques for Android malware detection. Qiao et al. [21] explore various machine learning methods, such as Support Vector Machines, Random Forest, and Neural Networks, to detect malware by extracting and analyzing patterns within Permissions and API Function Calls used by Android apps. Zhao et al. [22] adopt a decision tree classifier and kNN classifier to devise an Android malicious detection scheme based on sensitive API calls. Their approach involves measuring the correlation between specific API calls and malware using mutual information, generating a set of sensitive API calls for detection. MalScan [4] treats FCGs as social networks and employs centrality analysis to derive structural features of APK samples. These features are then fed into downstream classifiers for effective classification.

However, one challenge lies in the scale of vertices within FCGs, making it challenging for traditional machine learning methods to directly harness the structural information embedded in FCGs.

The deep graph neural network (GNN) approach has proven highly advantageous for tackling large-scale graph-level classification tasks, making it particularly well-suited for the challenges inherent in android malware detection leveraging Function Call Graphs (FCGs). In a noteworthy study, Cai et al. [11] employed the Continuous Bag of Words (CBOW) algorithm from natural language processing to generate embedding features for functions (vertices). Subsequently, they applied graph neural networks to extract graph embedding features from FCGs in android applications. These features were then integrated with downstream classifiers for effective malware detection.

In a similar vein, Gdroid [12] pioneered the construction of a heterogeneous graph connecting APKs and APIs. They then leveraged word2vec and clustering algorithms [13] to establish neighborhood relationships between APIs. The subsequent integration of graph convolutional neural networks (GCN) facilitated the learning of representations for APKs. The final step involved passing these representations to downstream classifiers for robust detection and classification. Despite their innovative approach, it's worth noting that Gdroid's construction of heterogeneous graphs still necessitated domain knowledge.

In contrast to the methodology outlined in [12], a more direct and streamlined approach involves combining the Function Call Graph (FCG) with the extracted function features using graph neural networks. A notable example is the work by Lo et al. [1], where FCGs were utilized to derive centrality features of functions (vertices). The authors seamlessly integrated well-established graph neural networks like GCN, GIN, and GraphSAGE for graph-level classification tasks. The model's representation was further enhanced through the implementation of the Jumping Knowledge technique, resulting in exceptional detection accuracy.

In a similar vein, Vinayaka et al. [14] focused on extracting properties directly from functions within the FCG. These properties encompassed aspects such as whether functions were declared by public or static descriptors, whether the function belonged to the Android API, the size of the function bytecode, and the opcodes contained within the function. These extracted properties encapsulate essential functional characteristics. The FCG, enriched with these features, was fed into a graph convolutional neural network for representation learning. Subsequently, the graph embedding features obtained from the Readout function were supplied to a downstream classifier for effective classification.

Ms-Droid [15] adopted a distinctive approach by segregating local subgraphs surrounding sensitive APIs from FCGs. They employed a graph neural network to classify these subgraphs. In the context of an APK, the identification of a local subgraph as malicious led to the classification of the entire APK as a malicious application.

While the aforementioned studies have made notable strides in the realm of Android malware detection, it becomes apparent that relying solely on the caller–callee relationship between functions may not fully tap into the rich semantic information available. A promising avenue for further research involves leveraging hypergraphs to reconstruct Function Call Graphs (FCGs) or constructing hypergraphs based on properties like common permissions and descriptors shared between functions. Subsequently, employing hypergraph neural networks for representation learning stands out as a meaningful and potentially transformative research direction.

In the realm of smartphones, particularly those running on the Android system, there is a growing apprehension surrounding malware – those cunning and detrimental programs capable of causing havoc to your device. Traditional methods of identifying these rogue apps often fall short when dealing with the more sophisticated and elusive ones.

To combat this challenge, innovators are employing a technology known as Graph Neural Networks (GNNs). Picture GNNs as digital investigators scrutinizing how various elements within an Android app link and engage, including permissions, API calls, and code segments. They transform this intricate information into a visual map, aptly named a graph.

Here's the intriguing aspect: GNNs excel at comprehending and interpreting these maps. They function like superhero detectors, adept at uncovering unusual patterns and behaviors in Android apps. When detecting something suspicious, they sound the alarm, signaling, 'Hey, this app might be engaging in malicious activities!'

Nevertheless, challenges persist. Ensuring these detective models can transparently explain why they deem an app harmful is one hurdle, akin to understanding why a friend dislikes a movie – clear reasons are sought. Another challenge involves the need for these detectors to operate at lightning speed, given the constant influx of new Android apps each day.

In essence, employing Graph Neural Networks for Android malware detection mirrors having cutting-edge detectives examining how different components of apps interconnect, identifying dubious activities, and swiftly forewarning users about potential threats.

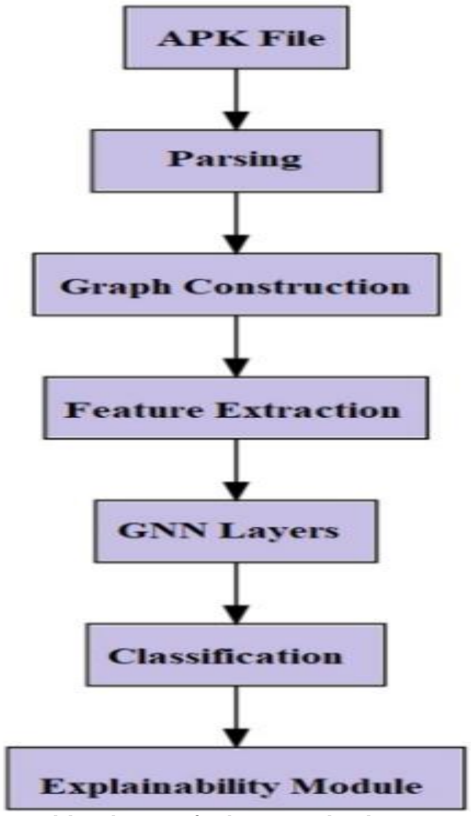

Figure 1. Model Architecture for the Proposed Malware Detection Process

Innovating the Android malware detection methodology initiates with Step 1, where the Android Application Package (APK) file stands as the primary input. Progressing to Step 2, APK Parsing involves an in-depth dissection of the file to extract pivotal information encompassing code, permissions, and various elements. Step 3 embarks on the construction of a graph representation for the APK, featuring nodes representing distinct components and edges denoting relationships between these elements. These relationships encapsulate permissions, API calls, and code segments. Advancing to Step 4, Feature Extraction unfolds, extracting meaningful characteristics from the graph, such as the frequency of specific permissions and structural properties. Sequentially, Step 5 introduces the application of Graph Neural Networks (GNNs) Layers to the graph, fostering the learning and comprehension of intricate relationships within the data. Moving forward to Step 6, the model undergoes Classification, acquiring the ability to distinguish between benign and malicious apps based on learned patterns. Culminating in Step 7, an innovative Explainability Module is introduced, amplifying transparency by providing insights into the decision-making process of the model. This module becomes instrumental in deciphering why the model classifies an app as benign or malicious, delivering valuable information for security analysts and decision-makers. In unison, these distinctive steps weave a forward-thinking methodology, harnessing the power of Graph Neural Networks for highly effective Android malware detection.

Step 1: Android APK File

Begin with the Android Application Package (APK) file, which is essentially the file format used to distribute and install applications on Android devices.

Step 2: APK Parsing

Parsing involves breaking down the APK file into its various components to extract essential information. This includes dissecting the code, permissions, and other elements within the APK.

Step 3: Graph Construction

Construct a graph representation of the APK. This involves creating nodes for different components of the app, such as permissions, API calls, and code segments. The relationships or connections between these nodes are established based on how these components interact with each other.

Step 4: Feature Extraction

Extract meaningful features from the constructed graph. Features are characteristics or properties that help the Graph Neural Network (GNN) understand and learn patterns. Features might include the frequency of specific permissions, the types of API calls, or the structural properties of the graph.

Step 5: Graph Neural Networks Layers

Apply Graph Neural Networks layers to the constructed graph. GNNs have layers that process information from the graph's nodes and edges, allowing the model to understand the relationships and dependencies between different components.

Step 6: Classification

Train the model for classification. This involves feeding the features extracted from the graph into the GNN and training it to distinguish between benign and malicious apps. The model learns to recognize patterns indicative of malware during this phase.

Step 7: Explainability Module

Implement an explainability module to provide insights into the model's decision-making process. This module helps interpret why the GNN classifies an app as either benign or malicious. It enhances transparency and helps security analysts understand the rationale behind the model's predictions.

proposed methodology starts with the APK file, parses it to understand its components, constructs a graph to represent the relationships between these components, extracts relevant features, applies Graph Neural Networks layers for learning, classifies the app as benign or malicious, and incorporates an explain ability module for better understanding the model's decisions. This comprehensive approach leverages the power of GNNs to enhance Android malware detection.

II. Conclusion

The arena of Android malware detection is undergoing a significant transformation with the introduction of Graph Neural Networks (GNNs). Conventional methods, grappling with the complexities of evolving threats, frequently fall short in accurately depicting the intricate structures of Android applications (APKs). The examined methodologies shed light on deficiencies in detection rates and transparency, underscoring the need for inventive strategies. The envisioned GNN-based framework stands out as a pioneering solution, demonstrating excellence in both feature extraction and graph representation. Its robust malware detection model, with a pronounced emphasis on transparency, establishes a novel benchmark. While acknowledging the critical importance of scalability and real-time analysis, the proposed architecture not only exhibits promise but also lays the groundwork for subsequent research endeavors. The outlined methodology, as illustrated in Figure 1, intricately navigates through steps ranging from APK parsing to GNN-based classification, culminating in an explainability module. This approach effectively leverages GNNs for proficient and transparent Android malware detection, marking a noteworthy advancement in the field.

References

1. Lo,W.W.; Layeghy, S.; Sarhan, M.; Gallagher, M.; Portmann, M. Graph Neural Network-Based Android Malware Classification with Jumping Knowledge. In Proceedings of the 2022 IEEE Conference on Dependable and Secure Computing (DSC), Edinburgh, UK, 22–24 June 2022; pp. 1–9.
2. F. Alswaina and K. Elleithy, ``Android malware family classi_cation and analysis: Current status and future directions,'' Electronics, vol. 9, no. 6, p. 942, Jun. 2020.
3. F. Alswaina and K. Elleithy, ``Android malware permission-based multi-class classi_cation using extremely randomized trees,'' IEEE Access, vol. 6, pp. 76217_76227, 2018, doi: 10.1109/ACCESS.2018. 2883975.
4. Wu, Y.; Li, X.; Zou, D.; Yang, W.; Zhang, X.; Jin, H. MalScan: Fast Market-Wide Mobile Malware Scanning by Social-Network Centrality Analysis. In Proceedings of the 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), San Diego, CA, USA, 11–15 November 2019; pp. 139–150.
5. Mariconti, E.; Onwuzurike, L.; Andriotis, P.; De Cristofaro, E.; Ross, G.; Stringhini, G. MaMaDroid: Detecting Android Malware by Building Markov Chains of Behavioral Models. In Proceedings of the Proceedings 2017 Network and Distributed System Security Symposium, San Diego, CA, USA, 26 February–1 March 2017; Internet Society: San Diego, CA, USA, 2017.
6. Zhang, X.; Zhang, Y.; Zhong, M.; Ding, D.; Cao, Y.; Zhang, Y.; Zhang, M.; Yang, M. Enhancing State-of-the-Art Classifiers with API Semantics to Detect Evolved Android Malware. In Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual, 9–13 November 2020; ACM: New York, NY, USA, 2020; pp. 757–770.
7. Y. Sun, Y. Chen, Y. Pan, and L. Wu, ``Android malware family classi_cation based on deep learning of code images,'' IAENG Int. J. Comput. Sci., vol. 46, no. 4, pp. 524_533, 2019.
8. D. Vasan, M. Alazab, S. Wassan, H. Naeem, B. Safaei, and Q. Zheng,``IMCFN: Image-based malware classi_cation using _ne-tuned convolutional neural network architecture,'' Comput. Netw., vol. 171, Apr. 2020, Art. no. 107138.
9. L. Taheri, A. F. A. Kadir, and A. H. Lashkari, ``Extensible Android malware detection and family classi_cation using network-_ows and APIcalls,'' in Proc. Int. Carnahan Conf. Secur. Technol. (ICCST), Oct. 2019, pp. 1_8.
10. G. Suarez-Tangil1, S. K. Dash, M. Ahmadi, J. Kinder, G. Giacinto, and L. Cavallaro, ``DroidSieve: Fast and accurate classi_cation of obfuscated Android malware,'' in Proc. 7th ACM Conf. Data Appl. Secur. Privacy,2017, pp. 309_320.
11. Cai, M.; Jiang, Y.; Gao, C.; Li, H.; Yuan, W. Learning Features from Enhanced Function Call Graphs for Android Malware Detection. Neurocomputing 2021, 423, 301–307. [CrossRef]
12. Gao, H.; Cheng, S.; Zhang, W. GDroid: Android Malware Detection and Classification with Graph Convolutional Network. Comput. Secur. 2021, 106, 102264. [CrossRef]
13. Zhang, H.; Gu, Z.; Tan, H.;Wang, L.; Zhu, Z.; Xie, Y.; Li, J. Masking and Purifying Inputs for Blocking Textual Adversarial Attacks. Inf. Sci. 2023, 648, 119501. [CrossRef]
14. Vinayaka, V.K.; Jaidhar, C.D. Android Malware Detection Using Function Call Graph with Graph Convolutional Networks. In Proceedings of the 2021 2nd International Conference on Secure Cyber Computing and Communications (ICSCCC), Jalandhar, India, 21–23 May 2021; pp. 279–287.
15. He, Y.; Li, Y.;Wu, L.; Yang, Z.; Ren, K.; Qin, Z. MsDroid: Identifying Malicious Snippets for Android Malware Detection. IEEE Trans. Dependable Secur. Comput. 2023, 20, 2025–2039. [CrossRef]
16. Liu, K.; Xu, S.; Xu, G.; Zhang, M.; Sun, D.; Liu, H. A Review of Android Malware Detection Approaches Based on Machine Learning. IEEE Access 2020, 8, 124579–124607. [CrossRef]
17. Feng, Y.; Anand, S.; Dillig, I.; Aiken, A. Apposcopy: Semantics-Based Detection of Android Malware through Static Analysis. In Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, Hong Kong SAR, China, 16–21 November 2014; ACM: New York, NY, USA, 2014; pp. 576–587.
18. Faruki, P.; Ganmoor, V.; Laxmi, V.; Gaur, M.S.; Bharmal, A. AndroSimilar: Robust Statisti-cal Feature Signature for Android Malware Detection. In Proceedings of the 6th International Conference on Security of Information and Networks, Aksaray, Turkey, 26–28 November 2013; ACM: New York, NY, USA, 2013; pp. 152–159.
19. Xiao, X.; Xiao, X.; Jiang, Y.; Liu, X.; Ye, R. Identifying Android Malware with System Call Co-occurrence Matrices. Trans. Emerg. Tel. Tech. 2016, 27, 675–684. [CrossRef]

20. Feng, J.; Shen, L.; Chen, Z.;Wang, Y.; Li, H. A Two-Layer Deep Learning Method for An-droid Malware Detection Using Network Traffic. IEEE Access 2020, 8, 125786–125796. [CrossRef]
21. Qiao, M.; Sung, A.H.; Liu, Q. Merging Permission and API Features for Android Malware Detection. In Proceedings of the 2016 5th IIAI International Congress on Advanced Ap-plied Informatics (IIAI-AAI), Kumamoto, Japan, 10–14 July 2016; pp. 566–571.
22. Zhao, C.; Zheng, W.; Gong, L.; Zhang, M.; Wang, C. Quick and Accurate Android Malware Detection Based on Sensitive APIs. In Proceedings of the 2018 IEEE International Conference on Smart Internet of Things (SmartIoT), Xi'an, China, 17–19 August 2018; pp. 143–148.

::. How to Cite this ARTICLE .::
Pallavi Papankar, Nitin Mandaogade (2024). Advancements in Android Malware Detection: A Comprehensive Review of Graph Neural Network Approaches. *International Journal of Advance Research in Computer Science and Management Studies, 12*(3), *13*-19. https://doi.org/10.61161/ijarcsms.v12i3.2