2023 IEEE 6th International Conference on Cloud Computing and Artificial Intelligence: Technologies and Applications (CloudTech) | 979-8-3503-0306-3/23/$31.00 ©2023 IEEE | DOI: 10.1109/CLOUDTECH58737.2023.10366167

Towards a DeepMalOb improvement in the use of formal security risk analysis methods

Zakaria SAWADOGO
Laboratory LANI
Gaston Berger University
Saint Louis, Senegal
sawadogo.zakaria@ugb.edu.sn

Muhammad Taimoor Khan
Cyber Assurance Lab
University of Greenwich
London, UK
M.Khan@greenwich.ac.uk

Jean Marie DEMBELLE
Laboratory LANI
Gaston Berger University
Saint Louis, Senegal
jean-marie.dembele@ugb.edu.sn

Gervais MENDY
Laboratory LITA
University Cheikh Anta Diop of Dakar
Dakar,Senegal
gervais.mendy@ucad.edu.sn

Samuel OUYA
Laboratory LITA
University Cheikh Anta Diop of Dakar
Dakar,Senegal
samuel.ouya@ucad.edu.sn

***Abstract*—Researchers are concerned about the detection of obfuscated Android malware, and multiple studies have been proposed to address certain obfuscation techniques. However, the comprehensive consideration of all obfuscation techniques remains a critical cybersecurity challenge due to their mutations. To tackle this issue, we developed the DeepMalOb approach, which utilizes memory dumping and deep learning with MLP to detect obfuscated malicious applications. Although the approach has yielded satisfactory results, we acknowledge potential security risks associated with MLPs, such as adversarial attacks, model inversion attacks, overfitting, and model biases, which may impact the accuracy and robustness of the MLP model and render it vulnerable to obfuscated malware. To improve the DeepMalOb approach, we propose the use of formal security risk analysis methods with MLP to detect hidden malware in Android by analyzing the security risks associated with the MLP model and the input features used for training.**

***Index Terms*—Android malware detection, Obfuscation techniques, Deep learning, Cyber-security,Memory dump, formal method.**

I. Introduction

To safeguard their software from cyber attacks, application developers employed obfuscation methods, primarily aimed at thwarting hacking attempts and preventing typical forms of attacks, such as code injection, reverse engineering, and the manipulation of users' confidential data [1]. Currently, scholars are actively researching obfuscation techniques, as they have become a popular tool among creators of malicious Android apps to circumvent detection models used by researchers and evade anti-malware products [2] [3]. This is confirmed by Bit-defender's report that malicious Android app developers have been using COVID-19 related keywords to mask data leakage apps. The report states that more than 85% [4] [5] tracking apps are leaking data. Bit-defender's report [4]reveals that developers of malicious Android apps are resorting to COVID-19 related keywords as a smokescreen to conceal data leakage apps. According to the report, over 85% of tracking apps are found to be leaking data. There are various obfuscation techniques described in literature, such as Rename obfuscation, data obfuscation, control flow obfuscation, encryption, and more. These techniques are widely cited and include several sub-techniques, as mentioned by Zhang [2]. While many authors base their approaches on one or more of these techniques, a majority of these approaches tend to overlook certain obfuscation techniques. The literature review highlights numerous challenges associated with detecting obfuscation malware. Despite various proposed detection approaches, many of them focus only on specific obfuscation techniques, thereby limiting their effectiveness. Additionally, obfuscated malicious applications are increasingly evading detection and antivirus approaches, and the constant evolution of obfuscation techniques renders existing detection models obsolete.

Our research focuses on using memory dumping for obfuscated malware detection, as memory analysis has proven effective in detecting unconventional malware [6]. We employed debug mode for the memory dump process due to the volatility of memory, allowing us to more accurately represent the user's actions at the moment of the malware attack. Memory analysis is a promising technique for malware detection [7], providing a better understanding of malicious code.

DeepMalOb [8] approach is unique in that it can detect malicious applications without specifically targeting a particular obfuscation technique, thus accounting for the mutation of various obfuscation types used by developers of malicious applications. Our contributions include an efficient approach to detecting obfuscated malicious applications independent of obfuscation techniques and an excellent evaluation of our approach based on the supervised neural network algorithm, multi-layer perceptron.

While the approach has produced satisfactory outcomes, we recognize the possible security threats linked to MLPs, such as overfitting, model biases, adversarial attacks, and model inversion attacks. These vulnerabilities may compromise the

979-8-3503-0306-3/23/$31.00 ©2023 IEEE

MLP model's accuracy and resilience, leaving it susceptible to obfuscated malware. We propose enhancing the DeepMalOb method by utilizing formal security risk analysis approaches in conjunction with MLP for detecting hidden malware in Android. This entails incorporating formal techniques to analyze security risks associated with MLP and using these insights to improve the detection of obfuscated malware.. This involves evaluating the security risks associated with the MLP model and the input features employed during training. By undertaking a thorough analysis, we can ensure the MLP model's robustness and efficacy in detecting obfuscated malware.

We organized our paper as follows: We start with an introduction in section 1. In section 2, we present the state of the art on obfuscated Android malware detection. In section 3, we describe DeepMalOb, we presented formal security risk analysis approaches in Section 4. In section 5, we discusse. The last section concludes the paper.

II. Related work

Mirzaei et al. [9] proposed AndrODet, a mechanism to detect three types of obfuscation in Android applications: ID renaming, string encryption and control flow obfuscation.

Alireza et al. [10]They found that there is a methodological problem in the experimental evaluation of the ability of the AndrODet system to detect the string encryption reported in [9].

According to Alireza et al, AndrODet's ability to detect string encryption was evaluated with an unreliable dataset. In fact, the authors of AndrODet did not take into account that many samples are very similar. And this could introduce a risk that the model derived from these samples may not be generalizable to all types of string encryption detection. For them, the AndrODet approach instead learns to classify samples according to the features of each malware family.

Li et al. [11] proposed Obfusifier, a system based on obfuscation-resistant features extracted from non-obfuscated applications, the system is effective in detecting obfuscated malware. They report that their system can obtain accuracy, recall and F-measurement greater than 95%.

According to them, the features on which their model is trained are extracted from unobfuscated applications, but they do not say in the paper how these features can characterize obfuscated malicious applications.

Guo et al. 2020 [12]proposed an approach based on dynamic extraction of the entire layout tree, called WALTDroid. They claimed that their approach can resist better the obfuscation effect.

Zhang et al. [3] conducted a study in which they summarized which obfuscation approaches are most popular for protecting their software against code theft and tampering. But also those that are used to bypass anti-malware products and detection systems.

Alessandro et al. [13] showed through a survey how the application of several obfuscation techniques affects the effectiveness of widely used machine learning based malware detection approaches coupled respectively with static and dynamic analysis .

Yusheng et al. [14],worked on a malware classification method based on virtual memory dumping. Their classification method focuses on backdoor malware detection, leaving other types of malware.

Sihag et al. [15] proposed BLADE, an obfuscation-resistant malware detection system based on opcode segments. They proposed a system that allows the characterization of features that are resistant to obfuscation techniques through opcode segments.

Sihwail et al. [7], investigated the types of malware and detection methods. They also reviewed three types of malware analysis techniques: static, dynamic and hybrid. Their study found that in the face of obfuscation techniques used by malware to evade detection, memory analysis can be used to track the activities taking place in the system.

In light of the literature review on the detection of obfuscated applications, some challenges exist, as we indicated in section I. It is in this context that we propose an approach to detect obfuscated malware without targeting a particular obfuscation technique.

In section III, we present our approach called **DeepMalOb** which is a system that allows to detect malicious obfuscated applications from the data retrieved from memory dump. This approach doesn't target any particular obfuscation technique. Thus, we take up the challenge of one of the limitations that has been raised in the literature which is that the proposed detection approaches are mostly focused on some obfuscation technique, which limits their performance.

These studies, among others, highlight the potential of memory dumping and related techniques in detecting malicious applications and enhancing cybersecurity. Our proposed approach aims to enhance the performance of DeepMalOb in detecting obfuscated Android malware by leveraging the potential of memory dumping and formal security risk analysis techniques. The memory dumping approach involves extracting the runtime memory of a mobile device and analyzing it to identify potentially malicious code. By incorporating this technique into the DeepMalOb approach, we can increase the chances of detecting obfuscated malware that may be hidden or disguised within the device's memory. In addition, we propose utilizing formal security risk analysis techniques to evaluate the security risks associated with the DeepMalOb approach. This involves identifying potential vulnerabilities in the model architecture and input features and mitigating them to improve the model's robustness and efficacy in detecting obfuscated malware. By combining the memory dumping and formal security risk analysis techniques, our proposed approach can significantly enhance the performance of DeepMalOb in detecting obfuscated Android malware, reducing the risk of successful cyber attacks on mobile devices.

III. Proposal to improve DeepMalOb

In this section, we first present our approach to obfuscated malware detection, and then we explain the supervised neural

network model we used.

A. DeepMalOb approach

The DeepMalOb approach can be described in seven steps: (Figure 1).

- In step 1, we create a data-set composed of real-life scenarios simulations of the memory dump process. This data-set is composed of malicious and benign applications. The malwares comprising the data-set are some of the most widespread in the real world.
- In step 2, we clean the data impurities, we put the data on the same scale. At the end of this step, we have features ready to be used for the training . This data-set will be split in two: training data-set and test data-set. We do not select features in this step because we are using a deep learning algorithm.
- In step 3, we implement a supervised neural network model (more details in subsection III-C) to build our obfuscated malware detection approach. This step will train our model from the training data. And provide us with a model capable of obfuscated malware detection.
- In step 4, we test our model with the test data to obtain its score.
- In steps 5 and 6, we prepare non-training and non-test data through none labeled data cleaning up.
- In step 7, we evaluate our model with data from previous steps and compare it with the performance obtained with the test data in step 4.

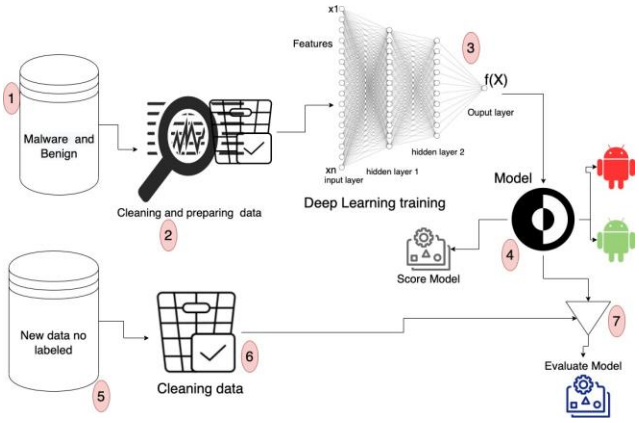

Fig. 1. DeepMalOb Approach [8]

Our team successfully tested the DeepMalOb system using the CIC-MalMem-2022 dataset [16], a newly curated repository of obfuscated malware. The system demonstrated exceptional accuracy, achieving 99%. However, these impressive results may prompt concerns about the system's robustness in the face of potential risks, such as adversarial attacks, overfitting, model biases, and model inversion attacks. To address these potential security concerns, we propose an enhancement to incorporate a formal method of security risk analysis. This would help to identify and mitigate any potential security risks associated with the DeepMalOb system.

B. Formal security risk analysis approaches

Formal security risk analysis approaches refer to a set of structured and systematic methods used to identify and evaluate potential security risks associated with a particular system or process [17]. These methods involve analyzing the system's architecture, design, and implementation to identify vulnerabilities that could be exploited by attackers. Formal security risk analysis techniques use mathematical models, formal languages, and logical reasoning to assess the security risks and potential impacts of security breaches. These approaches are often employed in critical systems, such as military, aerospace, and financial systems, where the consequences of security breaches can be severe. Formal security risk analysis techniques are essential for ensuring the reliability and effectiveness of cybersecurity systems. By employing rigorous analysis techniques, potential vulnerabilities and weaknesses in the system can be identified and addressed, reducing the risk of successful cyber attacks [18] [19].

In the context of malware detection, formal security risk analysis techniques can be used to evaluate the robustness of machine learning models and identify potential weaknesses that can be exploited by attackers to evade detection. By analyzing the input features and model architecture, security risks associated with the model can be identified and mitigated to improve the model's performance and reduce the risk of false negatives or false positives [20].

C. Using formal security risk analysis methods with DeepMalOb

Our aim is to develop a comprehensive methodology to detect obfuscated Android malware and identify potential system vulnerabilities by combining formal security risk analysis methods with machine learning. This approach entails utilizing formal security risk analysis to identify potential attack paths that could be exploited by malware, while also leveraging the powerful pattern detection capabilities of the MLP algorithm to analyze the data. The resulting detection system would be highly accurate and robust, enabling it to effectively protect Android devices from a wide range of malware threats.

The utilization of formal security risk analysis methods with the multilayer perceptron (MLP) to help detect obfuscated Android malware. This is done by examining the security risks associated with the MLP model and the input features used to train it. We propose the following steps for our case:

1) Identification of security risks associated with MLP: Here we identify potential security risks associated with MLP, such as adversarial attacks, model reversal attacks, overfitting, and model bias.
2) Definition of the threat model and attacker model: By defining the threat model and attacker model, potential vulnerabilities and risks can be identified and factored into the design of the MLP model, improving its ability to resist attacks and detect obfuscated Android malware.
3) Selection of a formal method of security risk analysis: We are thinking of using attack trees, threat modeling,

or formal verification, which is adapted to the MLP and the threat and attacker models.

4) Formalize the MLP model and input features: We formalize the MLP model and input features using a formal specification language Z . This involves defining the structure and parameters of the MLP model, the input features and their properties, and the learning and inference procedures.
5) Analyze security risks using the formal method: We use our formal security risk analysis method to analyze the security risks associated with the MLP and input features. This involves identifying the attack paths and vulnerabilities that an attacker can exploit to bypass the MLP model, and evaluating the impact and likelihood of each attack.
6) Mitigate security risks: Based on the results of the analysis, apply appropriate mitigation measures to reduce the security risks associated with the MLP and the input features. This involves the use of techniques such as adversarial training, feature selection, model regularization, or input data cleaning.
7) Evaluate the effectiveness of mitigation measures: We will evaluate the effectiveness of mitigation measures using a formal evaluation method, such as model-based testing or simulation. This involves generating test cases that simulate the attack scenarios identified in the analysis, and measuring the accuracy and robustness of the MLP model under these scenarios.

IV. Discussion

Indeed, there is a significant body of research on memory dumping and the detection of malicious applications. This research has produced several noteworthy contributions to the field of cybersecurity.

For example, Lim et al. [21] proposed a system called Mal-Xtract that employs emulation-based memory analysis and image similarity algorithms to detect the end of the unpacking routine in malware samples, achieving a performance of 97% accuracy. Dai et al. [14] introduced a classification technique that relies on the visual memory dump of malware. Their research specifically focused on backdoor malware and employed classification algorithms such as K-NN and random forest, resulting in an accuracy of 95%. Ahmet et al. [22] developed a malware detection approach based on memory dumping and computer vision. Their method utilized various learning and dimension reduction techniques, as well as algorithms such as Random Forest, linear SVM, and XGBoost, achieving an accuracy of 96.39%.It should be noted, however, that this technique has not yet been employed to detect obfuscated applications or Android malware.

This paper presents DeepMalOb, which is a novel approach that utilizes a supervised neural network algorithm for malware detection. Our approach achieves an impressive 99% performance for evaluation metrics such as Accuracy, Precision, Recall, and F1-score, outperforming several previous works in the literature. Notably, our approach is not limited to a single obfuscation technique, making it versatile and effective against a broad range of malware samples.

In order to further improve the efficacy of the DeepMalOb approach, we propose incorporating formal security risk analysis techniques in conjunction with a Multilayer Perceptron (MLP) for detecting hidden malware in Android. This approach involves evaluating the security risks associated with the MLP model and the input features used during training. By conducting a comprehensive analysis, we can enhance the MLP model's robustness and ensure its effectiveness in detecting obfuscated malware. Utilizing formal techniques to assess security risks can provide valuable insights into potential vulnerabilities or weaknesses in the model and help mitigate these issues to improve the overall performance of the system.

V. Conclusion

Our proposed improvement to DeepMalOb aims to increase its effectiveness in detecting obfuscated malicious applications. Although DeepMalOb has already shown promising results in identifying obfuscated android malware, we propose incorporating formal security risk analysis approaches to assess the security risks associated with the MLP model and the input features used in its training. Using formal techniques, we can discover hidden malware in Android and use this knowledge to improve obfuscated malware detection.

This extensive analysis ensures the robustness and efficiency of the MLP algorithm used by DeepMalOb to develop the model for identifying obfuscated Android malware. Our contribution aims to enhance the performance of DeepMalOb, allowing a better understanding of obfuscated application behavior without focusing on a specific obfuscation technique. We plan to revisit the implementation of DeepMalOb to compare the results obtained with and without our proposed improvement.

Acknowledgment

Our work was sponsored by the Partnership for Skills in Applied Science, Engineering and Technology - Regional Scholarship and Innovation Fund (PASET-RSIF).

References

[1] Shouki A. Ebad, Abdulbasit A. Darem, and Jemal H. Abawajy. Measuring software obfuscation quality–a systematic literature review. *IEEE Access*, 9:99024–99038, 2021.

[2] Xiaolu Zhang, Frank Breitinger, Engelbert Luechinger, and Stephen O'Shaughnessy. Android application forensics: A survey of obfuscation, obfuscation detection and deobfuscation techniques and their impact on investigations. *Forensic Science International: Digital Investigation*, 39:301285, 2021.

[3] Xiaolu Zhang, Frank Breitinger, Engelbert Luechinger, and Stephen O'Shaughnessy. Android application forensics: A survey of obfuscation, obfuscation detection and deobfuscation techniques and their impact on investigations. *Forensic Science International: Digital Investigation*, 39:301285, 2021.

[4] Malicious android apps capitalizing on covid-19 promon. https://promon.co/security-news/malicious-android-apps-are-capitalizing-on-covid-19/. Accessed: 2020-12-15.

[5] Zakaria Sawadogo, Gervais Mendy, Jean Marie Dembele, and Samuel Ouya. Android malware classification: Updating features through incremental learning approach(ufila). In *2022 24th International Conference on Advanced Communication Technology (ICACT)*, pages 544–550, 2022.

[6] Christopher Hargreaves and Howard Chivers. Recovery of encryption keys from memory using a linear scan. In *2008 Third International Conference on Availability, Reliability and Security*, pages 1369–1376, 2008.

[7] Rami Sihwail, Khairuddin Omar, and Khairul Akram Zainol Ariffin. International journal of advanced science, engineering and information technology IJASEIT. *International Journal on Advanced Science, Engineering and Information Technology*, 8(4-2):1662–1671, 2018.

[8] Zakaria Sawadogo, Jean-Marie Dembele, Attoumane Tahar, Gervais Mendy, and Samuel Ouya. DeepMalOb: Deep Detection of Obfuscated Android Malware. In Telex Magloire Ngatched Nkouatchah, Isaac Woungang, Jules-Raymond Tapamo, and Serestina Viriri, editors, *Pan-African Artificial Intelligence and Smart Systems*, pages 307–318, Cham, 2023. Springer Nature Switzerland.

[9] O. Mirzaei, J. M. de Fuentes, J. Tapiador, and L. Gonzalez-Manzano. ANDRODET: An adaptive Android obfuscation detector. *Future Generation Computer Systems*, 90:240–261, 2019.

[10] Alireza Mohammadinodooshan, Ulf Karge´n, and Nahid Shahmehri. Comment on ”AndrODet: An adaptive Android obfuscation detector”. 2019.

[11] Zhiqiang Li, Jun Sun, Qiben Yan, Witawas Srisa-An, and Yutaka Tsutano. Obfusifier: obfuscation-resistant android malware detection system. *Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, LNICST*, 304 LNICST:214–234, 2019.

[12] Junxia Guo, Dongdong Liu, Rilian Zhao, and Zheng Li. WLTDroid: Repackaging Detection Approach for Android Applications. In Guojun Wang, Xuemin Lin, James Hendler, Wei Song, Zhuoming Xu, and Genggeng Liu, editors, *Web Information Systems and Applications*, pages 579–591. Springer International Publishing, 2020.

[13] Alessandro Bacci., Alberto Bartoli., Fabio Martinelli., Eric Medvet., Francesco Mercaldo., and Corrado Aaron Visaggio. Impact of code obfuscation on android malware detection based on static and dynamic analysis. In *Proceedings of the 4th International Conference on Information Systems Security and Privacy - ICISSP,*, pages 379–385. INSTICC, SciTePress, 2018.

[14] Yusheng Dai, Hui Li, Yekui Qian, and Xidong Lu. A malware classification method based on memory dump grayscale image. *Digital Investigation*, 27:30–37, 2018.

[15] Vikas Sihag, Manu Vardhan, and Pradeep Singh. BLADE: Robust malware detection against obfuscation in android. *Forensic Science International: Digital Investigation*, 38:301176, 2021.

[16] Tristan Carrier, Princy Victor, Ali Tekeoglu, and Arash Habibi Lashkari. Detecting obfuscated malware using memory feature engineering. In Paolo Mori, Gabriele Lenzini, and Steven Furnell, editors, *Proceedings of the 8th International Conference on Information Systems Security and Privacy, ICISSP 2022, Online Streaming, February 9-11, 2022*, pages 177–188. SCITEPRESS, 2022.

[17] Muhammad Taimoor Khan. Towards Practical and Formal Security Risk Analysis of IoT (Internet of Things) Applications. *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA*, 2022-September, 2022.

[18] A Comparative Study on Information Security Risk Analysis Methods.

[19] Dongkui Liang, Limin Shen, Zhen Chen, Chuan Ma, and Jiayin Feng. A formal method for description and decision of android apps behavior based on process algebra. *IEEE Access*, 10:108668–108683, 2022.

[20] Giacomo Iadarola, Fabio Martinelli, Francesco Mercaldo, and Antonella Santone. Formal methods for android banking malware analysis and detection. In *2019 Sixth International Conference on Internet of Things: Systems, Management and Security (IOTSMS)*, pages 331–336, 2019.

[21] Shashank N. Kane, Ashutosh Mishra, and Anup K. Dutta. Preface: International Conference on Recent Trends in Physics (ICRTP 2016). *Journal of Physics: Conference Series*, 755(1), 2016.

[22] Ahmet Selman Bozkir, Ersan Tahillioglu, Murat Aydos, and Ilker Kara. Catch them alive: A malware detection approach through memory forensics, manifold learning and computer vision. *Computers Security*, 103:102166, 2021.