POSTER: Developing Secured Android Applications by Mitigating Code Vulnerabilities with Machine Learning

Janaka Senanayake*
j.senanayake@rgu.ac.uk
Robert Gordon University
Aberdeen, Scotland, UK

Harsha Kalutarage
h.kalutarage@rgu.ac.uk
Robert Gordon University
Aberdeen, Scotland, UK

Mhd Omar Al-Kadri
omar.alkadri@bcu.ac.uk
Birmingham City University
Birmingham, England, UK

Andrei Petrovski
a.petrovski@rgu.ac.uk
Robert Gordon University
Aberdeen, Scotland, UK

Luca Piras
l.piras@rgu.ac.uk
Robert Gordon University
Aberdeen, Scotland, UK

ABSTRACT

Mobile application developers sometimes might not be serious about source code security and publish apps to the marketplaces. Therefore, it is essential to have a fully automated security solutions generator to integrate security-by-design into the development practices, especially for the Android platform. This research proposes a Machine Learning (ML) based highly accurate method to detect Android source code vulnerabilities. A new labelled dataset containing Android source code vulnerability samples was generated initially. The dataset was used to train binary and multi-class classification based ML models, to identify code issues by following a static analysis approach. The proposed model can detect code vulnerabilities with a 0.90 F1-Score and vulnerability categories (CWE) with a 0.96 F1-Score. By integrating this with the Android development environment, app developers can analyse source code and identify security vulnerabilities in real-time. The proposed framework can be extended to suggest suitable patches to overcome the source code issues by providing real-time fixes in future.

CCS CONCEPTS

• **Security and privacy → Software and application security**; • **Human-centered computing → Mobile computing**; • **Computing methodologies → Machine learning**.

KEYWORDS

android, code vulnerability detection, static analysis, vulnerability dataset, machine learning, secure mobile apps

ACM Reference Format:
Janaka Senanayake, Harsha Kalutarage, Mhd Omar Al-Kadri, Andrei Petrovski, and Luca Piras. 2022. POSTER: Developing Secured Android Applications by Mitigating Code Vulnerabilities with Machine Learning. In *Proceedings of the 2022 ACM Asia Conference on Computer and Communications Security (ASIA CCS '22), May 30-June 3, 2022, Nagasaki, Japan.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3488932.3527290

ASIA CCS '22, May 30-June 3, 2022, Nagasaki, Japan

ACM ISBN 978-1-4503-9140-5/22/05.
https://doi.org/10.1145/3488932.3527290

1 INTRODUCTION

With the increased use of smartphones, device manufacturers and Operating System (OS) vendors try to increase their market shares. As of February 2022, Google Android leads with a 70.97% market share, while Apple iOS has a 28.27% market share [12]. Sometimes, mobile application developers try to develop applications and release them to the market in a rush, without having much concern about the security-by-design concepts. Therefore, some applications contain vulnerabilities that may lead to severe problems. However, the application developers and mobile OS vendors are responsible for securing users from vulnerabilities. Though Android has the highest market share, the applications are not thoroughly verified for security aspects, like iOS [3]. The security of these applications is not guaranteed since they might not comply with extensive security protocols. According to the security development lifecycle, [6], it is better to follow the security first development practices simultaneously as the code is being written [11], without waiting until the application is completely developed. Therefore, a proper vulnerability detection mechanism is required, which can be used to enhance their security when developing Android apps.

Researchers used conventional and ML-based m ethods to detect and mitigate vulnerabilities with the three analysis methods: static, dynamic and hybrid [8]. Due to the increasing popularity of ML, many studies have used ML methods compared to conventional methods [4]. Nevertheless, many of these studies conducted their experiments to identify vulnerabilities in software written in Java, C++, Python. Few works have been conducted specifically for Android applications [8], but they were not comprehensive.

2 BACKGROUND AND RELATED WORK

There are two ways of analysing Android applications. The first way is to analyse the code by reverse-engineering the developed Android Application Packages (APKs). It is the most popular and the easiest way, but a pre-build application is required to follow this approach [1]. The second way is to analyse the source code simultaneously as the code is being written, which is more valuable to the developers. Researchers have developed several automated tools to identify Android app vulnerabilities [4, 9] with various scanning methods. However, these tools require deployment-ready APKs to perform the analysis, which is the main limitation of those methods.

2.1 Motivation of the Research

As per our findings, there is no proper method to detect vulnerabilities with high accuracy at the Android application development time. Many of the studies only consider detection methods rather than mitigation methods. Furthermore, a properly labelled dataset that can be used to identify Android-specific vulnerabilities is also required to train an ML model to predict the vulnerabilities in each source code. This work proposes a method to detect Android source code vulnerabilities using ML with static analysis techniques. The proposed model can detect vulnerabilities in Android source code with an F1-Score of 0.90 in the binary classification and 0.96 in the multi-class classification when trained with the XGBoost algorithm. This work also proposes a comprehensive source code vulnerability dataset for Android, which the research community can use for further experiments. The current dataset contains vulnerable and non-vulnerable source code samples from over 1000 applications, along with attributes such as the standard identifiers, levels of severity, and descriptions. Furthermore, a thorough analysis of existing application reverse engineering and source code scanning methods is also introduced, used as the base method for this dataset generation stage.

3 METHODOLOGY

This research proposes a method for Android source code vulnerability mitigation using ML. There were two main stages in this approach: 1) Training dataset generation, 2) Pre-processing and ML model training to identify the accuracy of dataset generation and prediction model accuracy. Figure 1 illustrates the overall approach.

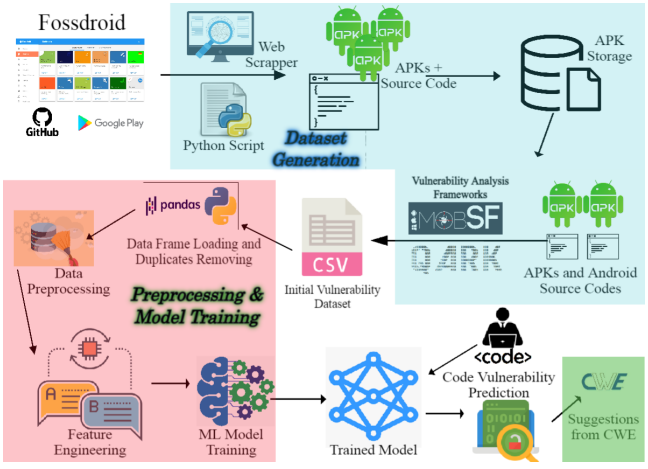

Figure 1: Overall approach

3.1 Training Dataset Generation

By identifying the research gap through related studies and existing methods, it is identified that the lack of benchmark dataset to train an ML model for Android code vulnerability detection. Therefore, as the proof of concept, a novel dataset was created at the initial stage. The dataset generation process has two main steps: a) Scrapping APKs and source code and b) Scanning APKs and generating the vulnerability dataset.

Scrapping APKs and Source Code (Data Collection). The initial step of the dataset generation process was to scrape APKs and their source code from application repositories, including Google Play. With the use of python script, APKs were downloaded along with their source code from GitHub repositories. This study considered only the free and open-source Android applications since the source code can also be compared with the reverse-engineered APKs in subsequent studies to review the accuracy of the reverse engineering methods. Since the Fossdroid project [10] provides APKs and their GitHub repositories, it was scrapped to create a list of trending Android apps using the Python BeautifulSoup library. 1008 APKs and their source codes were successfully scrapped and downloaded by performing these operations. If necessary, it is possible to run the same script to increase the sample size of the APKs and source codes.

Scanning APKs for Vulnerabilities (Data Labelling). The second step was to scan the downloaded APKs to identify their vulnerabilities. To accomplish this, the APKs need to be re-engineered and then scanned. MobSF [7], and Qark [5] were used in this research as vulnerability scanning tools since they can provide high accuracy in identifying Android vulnerabilities. When using the combination of these tools, it is identified that the accuracy of the proposed model can be increased by avoiding the dependence of just one method. However, none of these tools can perform real-time source code vulnerability detection, requiring pre-build applications.

A python-based script was developed to automate the scanning process using the tools mentioned earlier. All the applications were scanned through this script. Scanned source codes were stored as text lines, and if there is any identified vulnerability associated with the code line, they were mapped with the CWE identifiers. A description, type of the vulnerability, severity level, CWE ID, description of CWE ID, Open Web Application Security Project (OWSAP) ID and a further reference link for the vulnerability were also recorded for the vulnerable source code lines.

3.2 Pre-processing and ML Model Training

The second stage is to pre-process the generated dataset and train ML algorithms to validate the possibility of using the dataset to train an ML model. Features to train the model were generated using 1-3 n-gram techniques.

Only the vulnerability status, CWE-ID and the source code sample attributes in the dataset were considered for further analysis in this work. The source code samples should be processed appropriately to maximise the detection accuracy of the model since it is the independent variable of the model. The string values mentioned within the source code were replaced with "user_str", except the IP address related details and encryption-related details stated as string values. These particular string values were substituted with matching templates (i.e. "0.0.0.0", "AES", "SHA1", "MD5") since they can cause vulnerabilities such as CWE-200 (exposure of sensitive information to an unauthorised actor), CWE-201 (insertion of sensitive information into sent data), and CWE-327 (use of a broken or risky cryptographic algorithm) [2]. Additionally, all the comments in the source code were removed.

Model Training and Evaluating. Once the pre-processing was completed, both binary and multi-class classification methods were used. Conversely, as the primary research goal, it is required to identify whether a given source code is vulnerable or not. If vulnerable, the CWE-IDs associated with it are required to provide suggestions for vulnerability mitigation.

Several ML algorithms, including Naive Bayes, Logistic Regression, Gradient Boosting, Random Forest and XGBoost were considered and selected XGBoost as it provides a high accuracy compared to the others in this approach. Once the model is applied to the coding environment, it is possible to detect whether a given code line is vulnerable or not. If the code line is vulnerable, suggestions to mitigate the vulnerability by referring to the CWE are also provided.

4 PRELIMINARY RESULTS

After training and testing the model, the binary and multi-class classification results are discussed in this section.

4.1 Binary Classification

By training 2,952,785 source code samples using XGBoost, the binary classification model provides an F1-Score of 0.90 as illustrated in Figure 2.

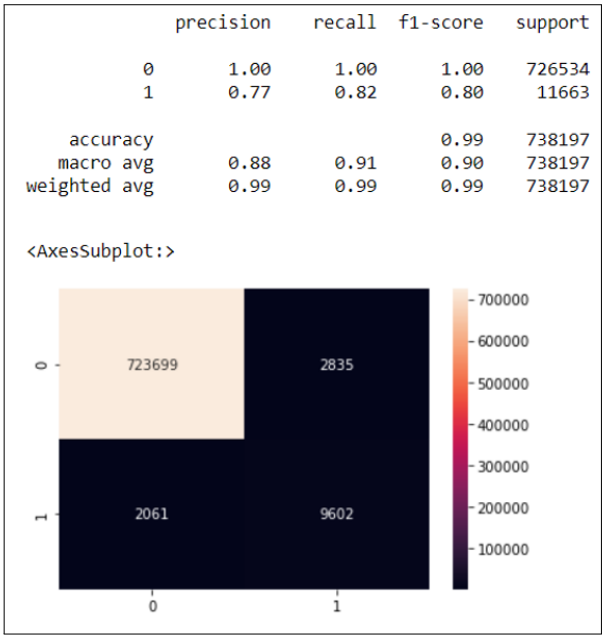

Figure 2: Binary classification results with XGBoost

4.2 Multi-Classs Classification

By training XGBoost model using a multi-class classification approach with 46,333 vulnerable code samples, 0.96 F1-Score was achieved as illustrated in Figure 3.

	precision	recall	f1-score	support
CWE-200	0.94	0.99	0.97	587
CWE-250	0.92	0.73	0.81	15
CWE-276	0.99	0.99	0.99	668
CWE-295	0.94	0.96	0.95	48
CWE-312	0.98	0.97	0.97	669
CWE-327	1.00	1.00	1.00	113
CWE-330	1.00	0.89	0.94	9
CWE-532	1.00	1.00	1.00	8416
CWE-649	1.00	1.00	1.00	7
CWE-749	0.96	1.00	0.98	25
CWE-89	0.98	1.00	0.99	161
CWE-919	0.99	0.97	0.98	680
NA	0.97	0.92	0.95	186
accuracy			0.99	11584
macro avg	0.97	0.95	0.96	11584
weighted avg	0.99	0.99	0.99	11584

Figure 3: Multi-class classification results with XGBoost

5 CONCLUSION AND FUTURE WORK

Based on the initial experiments for building a highly accurate Android vulnerability detection method, it is possible to see promising results for binary and multi-class classification approaches with the XGBoost algorithm. Therefore, the proposed model can be integrated with an Android development environment after training with more data. Furthermore, explainable artificial intelligence is planned to be integrated to identify the reasons for vulnerabilities.

ACKNOWLEDGMENTS

We thank Robert Gordon University — UK, the University of Kelaniya and the AHEAD grant — Sri Lanka for their support.

REFERENCES

[1] Ashwag Albakri, Huda Fatima, Maram Mohammed, Aisha Ahmed, Aisha Ali, Asala Ali, and Nahla Mohammed Elzein. 2022. Survey on Reverse-Engineering Tools for Android Mobile Devices. *Mathematical Problems in Engineering* 2022 (2022), 4908134. https://doi.org/10.1155/2022/4908134

[2] The MITRE Corporation. 2022. CWE - Common Weakness Enumeration. https://cwe.mitre.org/ Accessed: 2022-01-02.

[3] Shivi Garg and Niyati Baliyan. 2021. Comparative analysis of Android and iOS from security viewpoint. *Computer Science Review* 40 (2021), 100372. https://doi.org/10.1016/j.cosrev.2021.100372

[4] Seyed Mohammad Ghaffarian and Hamid Reza Shahriari. 2017. Software Vulnerability Analysis and Discovery Using Machine-Learning and Data-Mining Techniques: A Survey. *ACM Comput. Surv.* 50, 4, Article 56 (aug 2017), 36 pages. https://doi.org/10.1145/3092566

[5] LinkedIn. 2015. Quick Android Review Kit (QARK). https://github.com/linkedin/qark/ Accessed: 2022-01-02.

[6] Nana Onumah, Sam Attwood, and Rupak Kharel. 2020. Towards Secure Application Development: A Cyber Security Centred Holistic Approach. In *2020 12th International Symposium on Communication Systems, Networks and Digital Signal Processing (CSNDSP)*. IEEE, Porto, Portugal, 1–6. https://doi.org/10.1109/CSNDSP49049.2020.9249631

[7] OpenSecurity. 2015. Mobile Security Framework (MobSF). https://github.com/MobSF/Mobile-Security-Framework-MobSF Accessed: 2022-01-02.

[8] Janaka Senanayake, Harsha Kalutarage, and Mhd Omar Al-Kadri. 2021. Android Mobile Malware Detection Using Machine Learning: A Systematic Review. *Electronics* 10, 13 (2021), 1606. https://doi.org/10.3390/electronics10131606

[9] Faysal Hossain Shezan, Syeda Farzia Afroze, and Anindya Iqbal. 2017. Vulnerability detection in recent Android apps: An empirical study. In *2017 International Conference on Networking, Systems and Security (NSysS)*. IEEE, Dhaka, Bangladesh, 55–63. https://doi.org/10.1109/NSysS.2017.7885802

[10] Daniele Simonin. 2022. Fossdroid. https://nvd.nist.gov/vuln Accessed: 2022-01-02.

[11] Murugiah Souppaya, Karen Scarfone, and Donna Dodson. 2021. *Secure Software Development Framework (SSDF) Version 1.1: Recommendations for Mitigating the Risk of Software Vulnerabilities.* Technical Report. NIST.

[12] Statcounter. 2022. Mobile Operating System Market Share Worldwide. https://gs.statcounter.com/os-market-share/mobile/worldwide/ Accessed: 2022-03-02.