Comparative Analysis of Malware Detection Response Times Across Android Versions: An Emphasis on the "Hoverwatch" Application

Chiemela Ndukwe, Dr. Elaheh Homayounvala*, Prof. Hassan Kazemian, Istteffanny Araujo

Intelligent Systems Research Group, School of Computing and Digital Media, London Metropolitan University, 166-220 Holloway Rd, London N7 8DB, UK

cndukwe1@londonmet.ac.uk; e.homayounvala@londonmet.ac.uk; h.kazemian@londonmet.ac.uk; s.araujo@londonmet.ac.uk

Abstract. As Android-based smartphones become increasingly prevalent in our digital age, they also become inviting targets for malicious applications. Our research demystifies malware detection, with a special focus on response times across multiple Android versions. We began by examining the vast influence of Android and the need for robust malware detection in today's market. To understand the threat landscape, we investigated application-based threats and their impact on users and provided an encompassing review of the current malware detection methodologies, which include static, dynamic, and hybrid techniques. Our study centres around comprehensive tests conducted on distinct Android emulators, with the intent to measure malware detection response time. We used a known malicious app, "Hoverwatch," for this experiment. Our findings reveal notable disparities in detection times, emphasizing the need for constant advancements in defence systems and in a number of test cases the need for improvements in real-time detection capabilities. This research offers insights into the effectiveness of current Android malware detection methods, stressing response times. We underscore the necessity for regular updates and system enhancements to combat the evolving threat environment.

Keywords: Security, Malicious Mobile Applications, Malware, Malware Detection, Response time; Android operating system; Mobile Device Emulator.

1. Introduction.

Android, Google's open-source operating system (OS), has greatly impacted the mobile industry through its flexibility, compatibility, and user-friendly design with its 75% global market share (Statista, 2020). Its widespread adoption results from its universality, operating seamlessly across diverse devices, and cost-effectiveness, appealing to both premium and budget-conscious market segments.

Android's open-source characteristic allows developers free rein in application development and customization, fostering a diverse application ecosystem and fuelling Android's popularity. The Google Play Store, Android's official app marketplace, offers millions of applications to cater to a myriad of user needs. Nonetheless, Android's success and ubiquity render it a prime target for hackers. The high user concentration entices malicious entities, leading to an uptick in harmful applications designed specifically for Android (Tam et al., 2017)(Chang and Wang, 2016). Consequently, the necessity for efficient detection techniques for malicious applications on Android has escalated. While most research studies concentrate on increasing the performance of malware detection in terms of accuracy few have addressed the performance of such applications in terms of response time.

Subsequent sections will explore various combatant measures against these security threats, assessing their effectiveness and potential for refinement as well as reporting the comparative analysis of malware detection response time across different versions of Android operating system.

2. Related work

The Android platform's open-source nature and substantial user base make it a prime target for cybercriminals, who exploit apps, such as seemingly harmless flashlight tools, to access user data and compromise device security (Al Ali et al., 2017) (Tam et al., 2017). Malware from third-party stores can steal various personal information and perform unauthorised actions, highlighting the need for advanced malware detection methods, including static, dynamic, hybrid, and machine learning techniques. These methods are crucial for identifying and neutralising such threats. Malicious applications on Android, disguising themselves as or embedding within legitimate apps, pose a significant risk to user data and device performance. Once installed, they can exploit personal data for identity theft, sell it on the dark web, or use it for ransom, while also manipulating device functionalities for fraudulent activities or DDoS attacks (Al Ali et al., 2017) (Tam et al., 2017). Originating mainly from third-party app stores, these apps highlight the urgent need for effective detection and prevention techniques to safeguard users, businesses, and the broader digital ecosystem.

2.1 Static, dynamic and hybrid analysis for detecting malicious Android applications

Static analysis, a crucial method for detecting malicious Android applications, involves examining an app’s code without execution, focusing on the .apk file (Tao et al., 2019) (Kim et al., 2018). This non-runtime method scrutinises two essential files within the APK: AndroidManifest.xml, which declares app components and their interactions, and classes.dex, a compiled version of the application code. By analysing these files, static analysis provides significant insights into the app's behaviour and intentions, aiding in the identification of potential threats.

Both static and dynamic analysis offer unique advantages for detecting malicious applications but also have inherent limitations. A hybrid approach combines these methods, enhancing detection efficacy and providing a comprehensive evaluation of an app's behaviour (Du et al., 2015). Typically, static analysis is performed first, examining the application's code and metadata, followed by dynamic analysis, which observes runtime behaviour. This sequential integration helps to identify discrepancies

between expected and actual behaviour, often indicating malicious activities. The hybrid approach is particularly effective against advanced malicious applications designed to evade standalone static or dynamic analysis methods.

Hybrid analysis merges techniques from static and dynamic analysis, providing an exhaustive examination of applications. This approach includes essential elements like network traffic analysis to spot anomalous activities, API and system call monitoring to observe interactions with the operating system, and dependency graphs to understand app structure and behavior. It also involves function call monitoring, information flow analysis to detect unauthorised data access, and inter-process communication analysis to reveal potential exploitation of vulnerabilities. Additionally, hardware analysis and meta-data analysis assess interactions with device components and app credibility, respectively. By integrating these strategies, hybrid analysis effectively identifies and mitigates a broad spectrum of malicious behaviours, showcasing its prowess in combating malicious applications.

2.2 Machine Learning Techniques in Malware Detection.

Machine learning assumes a critical function in the detection of malicious applications. As suggested by its nomenclature, machine learning enables a system to assimilate from data, thereby enhancing its efficacy over time. In relation to malicious application detection, machine learning methodologies leverage historical data, comprising both benign and malicious applications, for the classification of new applications. Machine learning algorithms discern patterns and interdependencies within the data and subsequently generate a model capable of predicting an application's nature—benign or malicious—based on its attributes (Wang et al., 2018)(Garcia et al., 2018)(Badhani and Muttoo, 2018). This automation substantially augments the efficiency and precision of malicious application detection, particularly in light of the escalating quantity and sophistication of emerging malware.

Machine learning application in the detection of malicious apps conventionally comprises two modules: training and testing. In the training module, machine learning algorithms are educated on a dataset inclusive of both benign and malicious applications. These algorithms discern the unique characteristics of each type, thereby formulating a model capable of differentiating between them. The testing module sees these trained algorithms examined against a separate dataset to gauge their accuracy (Gheorghe et al., 2015)(Painter and Kadhiwala, 2018) (Chen et al., 2016). The purpose of this phase is to evaluate the algorithms' ability to classify new applications effectively, leveraging the knowledge they have acquired. Upon satisfactory performance, these algorithms can be employed to identify malicious applications in practical scenarios.

2.3 Accuracy versus Response Time

As we delve into this pivotal phase of our study, it is crucial to elucidate the link connecting the two prongs of our investigation: accuracy in malware detection mechanisms and their response time. These distinct yet intertwined facets underpin a comprehensive examination of malware detection within Android ecosystems. The first facet, accuracy, gauges the system's ability to correctly discern malicious applications without erroneously flagging benign ones. While high accuracy underscores the

sophistication of implemented detection mechanisms, it isn't sufficient in isolation, given the real-world ramifications. Hence, the role of speed—specifically, the response time in malware detection—is equally vital. Android devices, a significant segment of the mobile market, are appealing targets for malicious apps (Statista, 2022). This underscores the imperative of swift and accurate malware detection. Swift detection truncates the potential damage window of malicious apps—every unnoticed minute signifies a window for malicious activities such as data theft, system impairment, or monetary losses (Statista, 2022)(McAfee, 2020).

Additionally, swift detection mitigates the broader network threats effectively. Given the interconnectivity of contemporary digital devices, a compromised device could potentially catalyse large-scale attacks (Symantec, 2021). Speedy identification and mitigation of such breaches curtail malware incursions. From a user perspective, expedited malware detection cultivates a sense of safety, fostering trust in the Android environment (Google, 2022). In summation, rapid and accurate malware detection is paramount for fortifying Android system security and user safety. As digital interconnectivity burgeons, the urgency for prompt and precise malware detection amplifies.

3. Material and Methods

The experiment was executed using the Android Studio Electric Eel version, an expansive environment for Android application development. To recreate varied user circumstances and device environments, six virtual mobile devices were established as emulators with unique device profiles. The configurations of these emulators are as detailed: Google Pixel 4-API 33, Google Pixel 3-API 24, Google Pixel 3a-API 33, Google Pixel 2-API 24, Nexus 5X-API 24, Nexus 5-API 33. These configurations exhibited diversity in device model, Android API level, screen resolution, CPU cores, RAM size, among other hardware specifics such as the enablement of GPS or dPad. Google Play was activated on all devices, and network capabilities were incorporated. The malicious application chosen for the study was "Hoverwatch". This application was installed across all six emulators to facilitate comprehensive examination under varied device environments.

The experiment progressed with the observation of the malicious application's behaviour across each emulated environment once the emulators were prepared and the app installed. The procedure enacted can be outlined as (1) Initialization: Initiated each emulator, confirmed network connections were functional, and verified all hardware features, such as GPS, were operating appropriately. (2) Application Installation: The malicious app, "Hoverwatch", was installed on each emulator, ensuring a successful installation process and anticipated start-up of the app. (3) Monitoring and Data Collection: The behaviours of the Hoverwatch app were meticulously observed across each environment, concentrating on system calls, dynamic code loading, network information flow, and other elements highlighted in the static, dynamic, and hybrid analysis methods. The app's interaction with hardware components and user data received special scrutiny, as did any irregular activities indicating possible malicious intent. (4) Analysis: The gathered data was subjected to machine learning techniques delineated in the literature review. SVM and other prominent machine learning algorithms were employed to categorize the app's behaviour as benign or malicious. (5) Results Interpretation: The

results across varying emulators were compared to comprehend the impact of hardware, software, and Android API level variations on the app's behaviour and ensuing categorization.

4. Results

The process aimed to comprehend and catalogue the behaviour of the malicious app across different Android environments and assess the efficacy of machine learning techniques in detecting such apps accurately. The outcomes from each stage were meticulously recorded for subsequent analysis and examination. The experiment described in this study offers a thorough examination of malware detection in various Android contexts, with a specific emphasis on the Hoverwatch virus. The experiment incorporates several strategies. The aforementioned strategies can be categorised into two main groups: behavioural analysis and cross-version analysis. The behavioural analysis took into account the following. (1) Dynamic analysis incorporates the execution of an application within a controlled environment to observe and analyse its behaviour in real-time. This could shed light on any harmful activity that the software may have started. (2) Heuristic analysis is a useful tool for spotting anomalous or dubious behaviour patterns that may be signs of malware. The cross-version analysis took into account the following: (1) Compatibility Testing: Assess the app's behaviour across several Android versions in order to find any vulnerabilities or behaviours unique to a particular version. (2) Update Impact Assessment: Analyse the influence of Android updates on the effectiveness of malware detection techniques.

4.1 Test Case 1: Google Pixel 4 Emulator Running Android API 33

Parameter	Values
avd.ini.encoding	UTF-8
disk.dataPartition.size	6442450944
hw.cpu.ncore	4
hw.device.manufacturer	Google
hw.device.name	pixel_4
hw.dPad	No
hw.gps	Yes
hw.gpu.enabled	Yes
hw.gpu.mode	Auto
hw.lcd.density	440
hw.lcd.height	2280
hw.lcd.width	1080
hw.ramSize	1536
hw.sdCard	Yes
image.androidVersion.api	33
PlayStore.enabled	TRUE
runtime.network.latency	None
runtime.network.speed	Full
tag.display	Google Play
tag.id	google_apis_playstore
vm.heapSize	228

Table 1: Configuration Details of the Android Virtual Device (AVD); Emulator 1

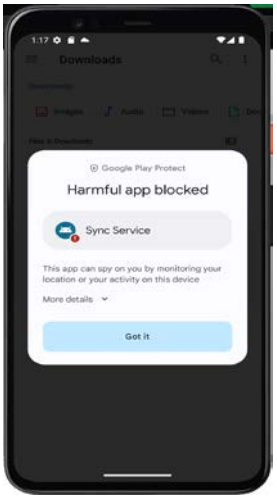

Fig. 1. Screenshot of Emulator 1; Test Case 1

Number Of trials	Installed	Detection time
First trial	Yes	11 minutes 50 seconds
Second trial	No	Instantly

Table 2: Detection Times based on Installation Trials for Test Case 1

Table 1 displays configuration details of the Android Virtual Device (AVD). Figure 1 and Table 2 display a screenshot of test case 1 running on the emulator and detection times on installation trials. Experiment for Test Case 1, Google Pixel 4 Emulator Running Android API 33 can be summarised as follows: In the first trial, the malicious app was successfully installed. Google Play Protect did not immediately identify the malware app. It was only after initiating the Play Protect Scan that it detected the app and prompted for its uninstallation, taking approximately 11 minutes and 50 seconds for detection. In contrast, during the second trial, Google Play Protect immediately blocked the installation of the same app, as evidenced by the screenshot.

4.2. Test Case 2: Google Pixel 3 emulator running Android API 24

Parameter	Values
avd.ini.encoding	UTF-8
disk.dataPartition.size	2G
hw.cpu.ncore	4
hw.device.manufacturer	Google
hw.device.name	pixel_3
hw.dPad	no
hw.gps	yes
hw.gpu.enabled	yes
hw.gpu.mode	auto
hw.lcd.density	440
hw.lcd.height	2160
hw.lcd.width	1080
hw.ramSize	1536
hw.sdCard	yes
image.androidVersion.api	24
PlayStore.enabled	TRUE
runtime.network.latency	none
runtime.network.speed	full
tag.display	Google Play
tag.id	google_apis_playstore
vm.heapSize	228

Table 3: Configuration Details of the Android Virtual Device (AVD); Emulator 2

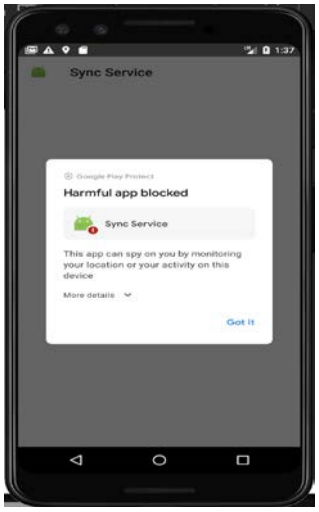

Fig. 2. Screenshot of Emulator 2; Test Case 2

Number Of Trials	Installed	Detection Time
First trial	No	Instantly

Table 4: Detection Times based on Installation Trials for Test Case 2

Experiment summary for Test Case 2, Google Pixel 3 emulator running Android API 24, can be summarised as follows: On the first attempt to install the app on the emulator, Google Play Protect immediately blocked the installation, as evidenced by the screenshot in Fig. 2 and Table 4. Table 3 presents configuration details of the Android Virtual Device (AVD) in Emulator 2.

4.3. Test Case 3: Google Pixel 3a emulator running Android API 33

Parameter	Values
avd.ini.encoding	UTF-8
disk.dataPartition.size	6442450944
hw.cpu.ncore	4
hw.device.manufacturer	Google
hw.device.name	pixel_3a
hw.dPad	no
hw.gps	yes
hw.gpu.enabled	yes
hw.gpu.mode	auto
hw.lcd.density	440
hw.lcd.height	2220
hw.lcd.width	1080
hw.ramSize	1536
hw.sdCard	yes
image.androidVersion.api	33
PlayStore.enabled	TRUE
runtime.network.latency	none
runtime.network.speed	full
tag.display	Google Play
tag.id	google_apis_playstore
vm.heapSize	228

Table 5: Configuration Details of the Android Virtual Device (AVD); Emulator 3

Number Of Tries	Installed	Detection Time
First Trial	Yes	6 minutes 59 seconds
Second Trial	Yes	50 seconds
Third Trial	No	Instantly

Table 6: Detection Times based on Installation Trials for Test Case 3

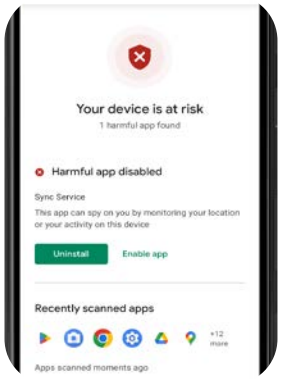

Fig. 3. Emulator 3, screenshot 1; Test Case 3

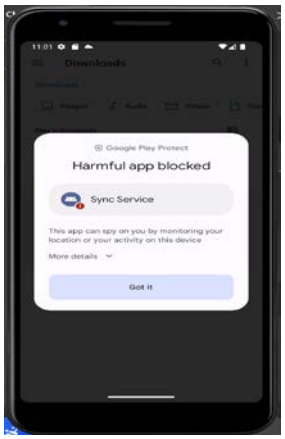

Fig. 4. Emulator 3, screenshot 2; Test Case 3

Table 5 presents configuration details of the Android Virtual Device (AVD) in Emulator 3. In the experiment for Test Case 3, Google Pixel 3a emulator running Android API 33, as displayed in Table 6 and Fig 3 and 4, both the first and second installations of the app were successful. However, Google Play Protect only detected the app after the Play Protect Scan was initiated, with detection times of approximately 6 minutes 59 seconds and 50 seconds respectively. By the third trial, Google Play Protect immediately blocked the installation, as shown in the screenshot of Fig. 4.

4.4. Test Case 4: Google Pixel 2 emulator running Android API 24

Parameter	Values
avd.ini.encoding	UTF-8
disk.dataPartition.size	2G
hw.cpu.ncore	4
hw.device.manufacturer	Google
hw.device.name	pixel_2
hw.dPad	no
hw.gps	yes
hw.gpu.enabled	yes
hw.gpu.mode	auto
hw.lcd.density	420
hw.lcd.height	1920
hw.lcd.width	1080
hw.ramSize	1536
hw.sdCard	yes
image.androidVersion.api	24
PlayStore.enabled	TRUE
runtime.network.latency	none
runtime.network.speed	full
tag.display	Google Play
tag.id	google_apis_playstore
vm.heapSize	228

Table 7: Configuration Details of the Android Virtual Device (AVD); Emulator 4

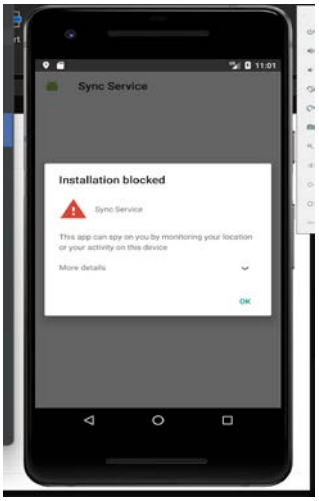

Fig. 5. Screenshot of Emulator 4; Test Case 4

Number Of Tries	Installed	Detection Time
First Trial	No	Instantly

Table 8: Detection Times based on Installation Trials for Test Case 4

In the experiment for Test Case 4: Google Pixel 2 emulator running Android API 24, on the first attempt to install the app on the emulator, Google Play Protect instantly blocked it, as shown in the screenshot in Fig 5 and Table 8. Table 7 presents configuration details of the Android Virtual Device (AVD) in Emulator 4.

4.5. Test Case 5: Nexus 5X emulator running Android API 24

Parameter	Values
avd.ini.encoding	UTF-8
disk.dataPartition.size	2G
hw.cpu.ncore	4
hw.device.manufacturer	Google
hw.device.name	Nexus 5X
hw.dPad	no
hw.gps	yes
hw.gpu.enabled	yes
hw.gpu.mode	auto
hw.lcd.density	420
hw.lcd.height	1920
hw.lcd.width	1080
hw.ramSize	1536
hw.sdCard	yes
image.androidVersion.api	24
PlayStore.enabled	TRUE
runtime.network.latency	none
runtime.network.speed	full
tag.display	Google Play
tag.id	google_apis_playstore
vm.heapSize	228

Table 9: Configuration Details of the Android Virtual Device (AVD); Emulator 5

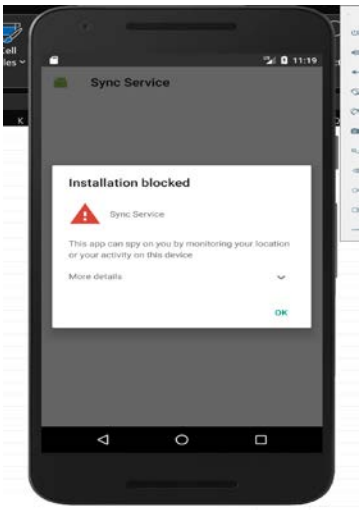

Fig. 6. Screenshot of Emulator 5; Test Case 5

Number Of Tries	Installed	Detection Time
First Trial	No	Instantly

Table 10: Detection Times based on Installation Trials for Test Case 5

Table 9 presents configuration details of the Android Virtual Device (AVD) in Emulator 5. In the experiment summary for Test Case 5: Nexus 5X emulator running Android API 24, on the first attempt to install the app on the emulator, Google Play Protect instantly blocked it, as evidenced by the screenshot in Fig 6 and Table 10.

4.6. Test Case 6: Nexus 5 emulator running Android API 33

Parameter	Values
avd.ini.encoding	UTF-8
disk.dataPartition.size	2G
hw.cpu.ncore	4
hw.device.manufacturer	Google
hw.device.name	Nexus 5
hw.dPad	no
hw.gps	yes
hw.gpu.enabled	yes
hw.gpu.mode	auto
hw.keyboard	yes
hw.lcd.density	480
hw.lcd.height	1920
hw.lcd.width	1080
hw.ramSize	1536
hw.sdCard	yes
hw.trackBall	no
image.androidVersion.api	33
PlayStore.enabled	TRUE
runtime.network.latency	none
tag.display	Google Play
tag.id	google_apis_playstore
vm.heapSize	128

Table 11: Configuration Details of the Android Virtual Device (AVD); Emulator 6

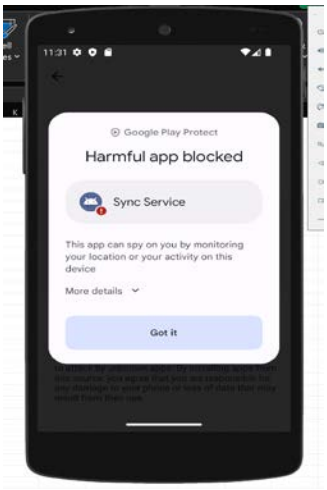

Fig. 7. Screenshot of Emulator 6; Test Case 6

Number Of Tries	Installed	Detection Time
First Trial	No	Instantly

Table 12: Detection Times Based on Installation Trials for Test Case 6

In the experiment for Test Case 6: Nexus 5X Emulator Running Android API 24, on the initial attempt to install the app on the emulator, Google Play Protect instantly blocked it, as depicted in the screenshot of Fig. 7 and Table 12. Table 11 presents configuration details of the Android Virtual Device (AVD) in Emulator 6.

4.7. Comparative Analysis of Test cases

Table 13 presents a summary of all the experiment outcomes and response times for different Android versions.

Number of Test Cases	Number of Trials	Successful Installation of Hoverwatch Malware	Response Time
1	2	1	11 minutes 50 seconds
2	1	0	0 seconds
3	3	2	7 minutes 49 seconds
4	1	0	0 seconds
5	1	0	0 seconds
6	1	0	0 seconds

Table 13: Comparative Analysis of Malware Detection Response Times Across Android Versions

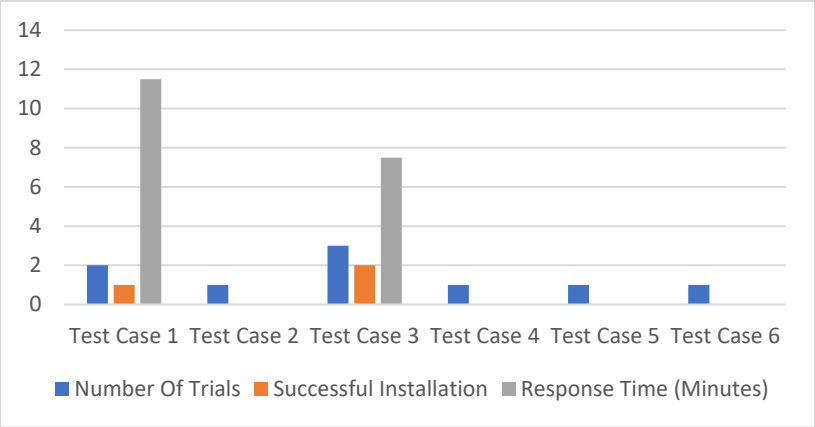

Fig. 8. Comparison of number of trials, successful installation, and response time across six test cases

Table 13 and Fig 8 provide a comparative analysis of use cases by depicting the malware detection time, the number of successful installation of malware, and how many times the various version were installed during this experiment.

5. Discussion and Conclusion

The primary focus of this research study is to present a comprehensive comparative examination of response times for malware detection across different versions of the Android operating system, with a specific emphasis on the "Hoverwatch" application. This study conducts a thorough examination of the effectiveness of malware detection, with a specific focus on the performance of "Google Play Protect" in promptly identifying and stopping the installation of dangerous applications. The evaluation is carried out through diligent experimentation using six distinct Android emulators. The findings derived from this study are crucial in elucidating the strengths and prospective areas for enhancement in the realm of malware detection time. The efficacy of Google Play Protect is demonstrated by its ability to effectively stop the Hoverwatch malware in the majority of test situations. However, the observed instances of delayed response times in Test Cases 1 and 3 highlight the need for improvements in real-time detection capabilities. The aforementioned occurrences, wherein the commencement of a Play Protect scan was necessary for the identification of malware, emphasise the urgent requirement for additional investigation and advancement in this field.

This study not only offers valuable insights into the performance of Google Play Protect but also makes a significant contribution to the wider academic discussion on Android security. The significance of attaining a standardised and consistent approach to addressing dangerous threats across all platforms is highlighted by the variation in malware detection efficacy observed among different Android versions or devices. Ensuring a standardised degree of security is of utmost importance due to the heterogeneous nature of Android devices. The findings obtained from this study indicate that Google Play Protect demonstrates a high level of resilience, while several areas for enhancement have been discovered. The praiseworthy aspect of the technology lies in its ability to prevent the installation of known harmful

applications. However, the inconsistent response times seen in various Android contexts require additional analysis and optimisation.

When considering the choice of an Android version for the purpose of security and malware detection, it becomes crucial to inquire about a number of significant factors: (1) In what manner does the Android operating system version react to identified and unidentified harmful applications? (2) Which security updates and vulnerability patches have been applied to this version so far? (3) Is there a documented record of compatibility difficulties between the Android version and certain security applications or features? (4) To what extent does the version exhibit proactive behaviour in terms of its ability to monitor and detect potential threats? (5) It is vital to consider these inquiries in order to ascertain that the selected Android version is in accordance with the security requirements and anticipations of the user or institution, thereby offering a robust safeguard against malevolent applications.

In conclusion, this study not only provides insights into the capabilities and effectiveness of Google Play Protect in various Android environments, but also initiates a discussion around the need of maintaining consistent and real-time malware detection. The findings obtained from this investigation function as a driving force for subsequent scholarly inquiries, advancements, and enhancement of security protocols for the Android operating system. This guarantees a more secure digital landscape for all individuals utilising Android devices.

6. Acknowledgment

We would like to extend our appreciation to London Metropolitan University for their assistance provided through the Transformation Funding 2022. This support has played a significant role in facilitating the execution of this research project.

References.

1. Anshul Arora, Sateesh K. Peddoju, Vikas Chouhan, Ajay Chaudhary, 2018. Proceedings of the 24th Annual International Conference on Mobile Computing and Networking. In: *Hybrid Android malware detection by combining supervised and unsupervised learning.* s.l.:s.n., pp. 798-800.

2. Google, 2022. *Google Play Protect.*. [Online]
Available at: https://play.google.com/intl/en_us/about/protect/
[Accessed 8 June 2023].

3. Harvey, B., 2021. *The Importance of Fast and Accurate Malware Detection. Cybersecurity Hub.*. [Online]
Available at: https://www.cybersecurity-insiders.com/the-importance-of-fast-and-accurate-malware-detection/
[Accessed 8 June 2023].

4. J. Garcia, M. Hammad, S. Malek, 2018. 2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE). In: *Lightweight, obfuscation-resilient detection and family identification of android malware.* s.l.:IEEE, p. 497.

5. L. Apvrille, A. Apvrille, 2015. 2015 IEEE Trustcom/BigDataSE/ISPA. In: *Identifying unknown android malware with feature extractions and classification techniques.* s.l.:IEEE, p. 182–189.

6. L. Gheorghe, B. Marin, G. Gibson, L. Mogosanu, R. Deaconescu, V.G. Voiculescu, M. Carabas, 2015. Smart malware detection on Android. In: S. C. Netw., ed. s.l.:s.n., p. 4254–4272.

7. M. Al Ali, D. Svetinovic, Z. Aung, S. Lukman, 2017. 2017 International Conference on Infocom Technologies and Unmanned Systems (Trends and Future Directions)(ICTUS). In: *Malware detection in android mobile platform using machine learning algorithms.* s.l.:IEEE, p. 763–768.

8. McAfee, 2020. *Mobile Malware: What It Is and How to Stop It. McAfee Blogs.*. [Online] Available at: https://www.mcafee.com/blogs/consumer/mobile-and-iot-security/mobile-malware/ [Accessed 8 June 2023].

9. N. Painter, B. Kadhiwala, 2018. Information and Communication Technology for Sustainable Development. In: *Machine-learning-based android malware detection techniques—A comparative analysis.* s.l.:Springer, p. 181–190.

10. S. Badhani, S.K. Muttoo, 2018. International Conference on Advances in Computing and Data Science. In: *Comparative analysis of pre-and postclassification ensemble methods for android malware detection.* s.l.:Springer, p. 442–453.

11. S. Chen, M. Xue, Z. Tang, L. Xu, H. Zhu, 2016. Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security. In: *Stormdroid: A streaminglized machine learning-based system for detecting android malware.* s.l.:s.n., pp. 377-388.

12. Statista, 2020. *Statista reports Mobile operating systems' market share worldwide from 2012 to 2019.* [Online] Available at: https://www.statista. com/statistics/272698/global-market-share-held-by-mobile-operatingsystems-since-2009/ [Accessed 14 04 2020].

13. Statista, 2022. *Android market share worldwide 2022.* [Online] Available at: https://www.statista.com/statistics/271527/android-market-share-in-the-united-states/ [Accessed 8 June 2023].

14. Symantec, 2021. *The Dangers of Delayed Detection: A Look at Advanced Persistent Threats. Symantec Enterprise Blogs.*. [Online]
Available at: https://symantec-enterprise-blogs.security.com/blogs/feature-stories/dangers-delayed-detection-look-advanced-persistent-threats
[Accessed 8 June 2023].

15. T. Kim, B. Kang, M. Rho, S. Sezer, E.G. Im, 2018. IEEE Trans. Inf. Forensics Secur.. In: 14, ed. *A multimodal deep learning method for android malware detection using various features.* s.l.:IEEE, p. 773–788.

16. Tao Lei, Zhan Qin, Zhibo Wang, Qi Li, Dengpan Ye, 2019. IEEE Internet Things J. In: 6, ed. *Evedroid: Event-Aware Android malware detection against model degrading for IoT devices.* s.l.:IEEE, p. 6668–6680.

17. W. Wang, Y. Li, X. Wang, J. Liu, X. Zhang, 2018. Detecting Android malicious apps and categorizing benign apps with ensemble of classifiers. In: s.l.:Future Gener. Comput. Syst., p. 987–994.

18. Y. Chang, S. Wang, 2016. 2016 IEEE 18th International Conference on High Performance Computing and Communications. In: *The concept of attack scenarios and its applications in Android malware detection.* s.l.:IEEE, p. 1485–1492.

19. Y. Du, X. Wang, J. Wang, 2015. A static android malicious code detection method based on multi-source fusion. In: s.l.:Secur. Commun. Netw. 8, p. 3238–3246.