

Malware detection and analysis in Android applications with mobile security framework (MobsF) using a permission-based approach.

Mark Kizito, Dr. Elaheh Homayounvala, Proffesor. Hassan Kazemian, Dr Istteffanny Araujo,
Intelligent Systems Research Group, School of Computing and Digital Media,
London Metropolitan University, 166-220 Holloway Rd, London N7 8DB, UK

Mak2347@my.londonmet.ac.uk; e.homayounvala@londonmet.ac.uk;
h.kazemian@londonmet.ac.uk; s.araujo@londonmet.ac.uk

Abstract. This paper examines security concerns in Android applications arising from violations to the principle of least privilege. Using mobile security framework (MobsF), static and dynamic analysis techniques were used in evaluating how non-essential permission groups that are not core to application functions greatly increase risks of unauthorized system access by attackers, data breaches and privacy concerns, enlarged attack surfaces, and hijacking system resources among others. A permission-based approach systematically identifies malware and highlights dangerous permission groups that magnify security risks. The findings of this research study, emphasis meticulous auditing of permissions to prevent security threats in Android applications and to safeguard user-sensitive data. The conducted experiments compared essential core permissions to non-essential ones regarding application functionality and overall system security. It was also noted that background permissions potentially pose a great deal of risk to security when granted without proper rationale as this increases attack surfaces within the application itself.

Keywords: Permissions, mobile security framework (MobsF), Principle of least privilege, malware, Android package kit (APK).

1 Introduction

Android platforms are targets for malicious activities with hackers engineering code to penetrate and infect user systems which necessitates advanced malware detection techniques. The idea behind this project is to comprehensively carry out malware detection and analysis in Android applications utilising static as shown in **Fig. 1** and dynamic analysis focusing on permissions. Mobile security framework (MobsF) is an open-source automated penetration testing tool for Android, iPhone operating systems and window applications. It can perform static and dynamic analysis to identify malicious behavior and patterns within these applications through code reviews and real-time monitoring (LaMalva and Schmeelk, 2022). Static analysis involves

examining the resources of an application without executing it like “Androidmanifest.xml”, “Class.dex” while dynamic analysis involves examining application behavior by executing it in a controlled environment like a sandbox (Chao et al., 2020). The permission-based approach involves the identification of malware through application requests at installation or runtime and carrying out risk assessments. This approach to malware detection and analysis is a simplified and efficient method because it is less resource intensive as compared to dynamic analysis. Furthermore, the permission-based approach complements the static analysis technique which makes it more user friendly and scalable for large scale application screening. This paper explains the benefits of a combined approach of static analysis, dynamic analysis, and permission-based approach to malware detection to mitigate potential mobile threats and improve security on these platforms.

2 Related Work

Malware detection and analysis in Android applications starts with a comprehensive understanding of their behavior to evade in-place detection systems. These measures can range from obfuscation to zero-day exploits which malicious actors use to exploit Android platforms. Furthermore, a complete analysis of permissions changing trends is key to identifying potentially vulnerable applications.

2.1 Analysis of Malware Behavioral Patterns

Malware detection has become quite complicated as attackers and code developers keep finding new ways to bypass detection. Malware code is modified and re-engineered almost regularly to increase its dynamicity, behavior, and code signatures. Developed algorithms in place, might not catch their modifications because the malware becomes metamorphic and self-generating. Through iterations, they modify their attributes like size and syntax but maintain constant behavior defined in their semantics. Polymorphic malware changes its code structures into a process known as mutation and encrypts their payloads through an internal generator. All these changes make malware detection extremely challenging for security experts to detect (Mehra et al., 2015).

Packaging techniques increase the efficiency of malware in avoiding detection through encryption. The encryption standards used transform malware code into cyphertext making it unreadable although still executable when called upon by the application a method known as obfuscation. It can be done in several ways like fake code insertions or changing the execution order of application processes which compresses the file contents and conceals potential threats from being detected. However, malware disguised in these applications still requests excessive or unusual permissions for malicious execution of its activities like data theft. Through

examining these permissions, potentially dangerous behaviour can be flagged (Sun et al., 2020).

Analysing malware behavioral characteristics is key to determining and selecting a suited approach for detection. Most malware research studies focus on static analysis techniques or signature-based matching because of their advantages like less demand for computing resources, and easy method of deployment among others. If static approaches are solely used in the detection process, the dynamicity of malware during runtime is not checked due to the absence of dynamic analysis. Runtime vulnerabilities pose a threat to host systems and combining both static and dynamic analysis in a hybrid approach gives a better understanding of malware behavior (Wang et al., 2022).

Malware research has been constrained by their samples in the past due to legal and ethical concerns. Security experts come up with inconclusive results due to the lack of malware samples meaning that they can't analyse the malicious behavior needed to compile datasets. The existing production level of malware samples is commercially based not purely academic which leaves a knowledge gap in their analysis. Limited datasets mean poor quality research findings and conclusions due to a certain degree of speculation and generalisation, biased results, low detection capabilities due to evolving malware (Chen et al., 2010).

Zero-day exploits can be defined as bugs in software or applications which don't have quick fixes or software patches in the current existing time frame. Vulnerabilities in software can be exploited by attackers leaving static detecting algorithms like signature-based matching at a disadvantage because they reference specific patterns. Zero-day exploits can exhibit obfuscation methods, dynamic and runtime behavior observable when the application is executed. Constant research into application of software upgrades, potential threat assessments can mitigate zero-day exploits impact levels since permission analysis is set at an intermediary level. Application permissions requests are examined to identify malicious intent although a layered security approach is recommended (Sara and Hossain, 2023).

..

2.2 Critical analysis of android permissions

When analysing Android application permissions at a granular level, much emphasis is shifted to user privacy, hardware compatibility and functionality to a lesser extent. Granular analysis is all about proper management of system resources, limiting its wide Application programming interfaces (API) key range which is based on a Linux kernel system and protection of user data. This is done through granting or denying application requests rather than granting all permissions as it potentially violates the principle of least privilege (PLP) and impacts system resources like battery life. Benign and malicious applications could indulge in explicit permission overclaim, implicit permission overclaim or AD Library permission overclaim. Although Android uses access control lists in a permission-based model, user data and privacy are still vulnerable to potential malware attacks (Tang et al., 2017).

Contextual permission awareness and analysis refers to the lack of dynamicity when dealing with application requests. Generally, a static approach is suited to avoid misinterpretation of these requests needed for device functionality. There is an existing user static policy in place where once permission is granted, it will not change without user intervention. If a contextual awareness approach existed, permissions would only access needed resources when executed not when running in the background or during the installation. This would enable users to understand the necessity of requested permissions and allow them to access system resources based on their current conditional states (Olejnik et al., 2017).

Cross-platform compatibility impacts the accuracy level of malware detection using a permission-based approach. Existing variations in platform security architectures lead to different results. For example, the Android security model has a permission-based system, and the iPhone operating system has a secure boot chain directed by Apple manufacturers and a complex permission system for users to accept or deny permissions. Security policies run differently on these models providing unbalanced data on application requests. The more data provision, the higher the chances of making secure user decisions limiting violations to the principle of least privilege (PLP). Default applications are also executed differently on these platforms with at least one requiring user input (Squires and Centonze, 2016).

Permission-based analysis examines overprivileged applications that make more requests than others flagging their behavior as suspicious. This behavior violates the principle of least privilege and by utilising static analysis, a working baseline for basic permissions is needed to establish application functionality. Appropriate permission levels may be contextually based or subjective depending on the current system configuration. Mapping application programming interface (API) calls to requested permissions identifies overprivileged applications deviating from the established baseline which is also a tool that analyses malicious behavior (Wu et al., 2019).

3 Methodology

This session will focus on the methods of detection used to carry out tests which are static analysis, dynamic analysis and permission-based approach to malware detection and analysis in malicious Android applications.

3.1 Static analysis

Static analysis refers to examining an application's written code without executing it, trying to determine suspicious and malicious behavioral patterns. This process is independent of runtime environments enabling less complicated testing environments. Static analysis involves decompilation of the application file in a mobile security framework architecture which reverse engineers it to reconstruct its source code and displays other defects like malware, network security, encryption standards in a report format as shown in Figure 2. It is a more scalable and efficient analysis method since it requires fewer computing resources than its counterpart dynamic analysis (Moghaddam and Abbaspour, 2014) (Gyamfi and Owusu, 2018).

Below is a diagram describing the static approach to malware detection.

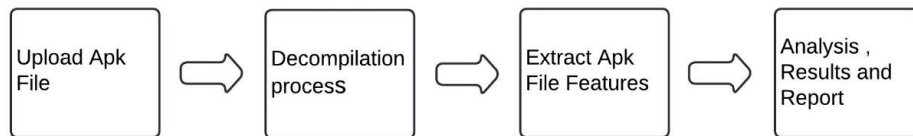


Figure 1: Static analysis processes.

Fig. 1 describes the static analysis process performed on an application file in a mobile security framework (MobsF) starting with file upload. After successful upload, the file is decompiled and its contents such as source code are examined for file packages like "AndroidManifest.XML", and "Dex file". Features contained in the APK file are extracted and categorised like permission groups, API calls, modules that include activities and services. These classifications are cross-checked against the vulnerability settings in MobsF to generate analysis reports.

3.2 Dynamic analysis

Dynamic analysis refers to the process of examining an application while being executed in a controlled environment. This is done by observing its functional application programming interface (API) calls, tracking its network traffic and data streams, examining its function parameters. While performing dynamic analysis, controlled environments like sandboxes or virtual machines are used for investigations securing the application in question in these virtual domains. The controlled environmental setup prevents potential malware spillage and propagation to the local host system. This method has an impressive resilience to evasion, code obfuscation and malware polymorphism surpassing static analysis. However, dynamic

analysis depends on triggering malicious behavioral conditions during execution or they will remain dormant and undetected. Some malwares may only activate in instances like specific locations, particular times and dates, external intervention limiting its ability to catch these malwares. (Moghaddam and Abbaspour, 2014) (Gyamfi and Owusu, 2018).

3.3 Permission-based approach

A permission-based model is the core of Android security which manages access to system resources and user data. Its security analysis is based on application requests to access system resources outside controlled environments determining whether they exhibit malicious or normal behavior. When emphasising a permission-based approach, screening dangerous, normal and signature permissions is key as shown in Table 2. Permissions in Android applications are stored in the “AndroidManifest.xml” located in the root folder where they can be granted or denied access to system resources. Android supports three protection levels and in its base permission structure, they are normal, dangerous and signature protection levels. A logical categorization of an application's permissions is known as a permission group as shown in Table 1 and applications bear permission trees for their encoding mechanisms. Permission enforcement is checked according to API level and SDK version contrasting between Android 6+ and Android versions below 5.1 or API 23 (Almomani and Khayer, 2020).

4 Tests

This section includes two types of testing carried out first to determine core permissions needed for application functionality potentially discarding unnecessary ones and second, to compare requested permission groups against their functionality.

4.1 Permission group analysis

A set of 50 applications was randomly selected in categories of Shopping applications, video and photo applications, banking applications, music applications, fitness tracking applications, navigational applications, and social media applications to study the permission groups that appeared the most and flagged dangerous. Although ‘android.permission.INTERNET’ normally connects applications to the internet, it has high potential for abuse. The selected Apk’s in this study had various permissions directly associated with the Apps functional requirements and use cases. For example, location access used for navigational Apps, banking Apps among others. The Apk files in different categories were obtained from apkpure.com and APKMirror and they included: -

Table 1: Randomly selected applications.

Social Apps	Navigational	Health Apps	Video/Photo	Shopping Apps	Banking Apps	Music Apps
Google Meet	RouteRunner	HealthHero	PicPerfect	eBay	We Bull	Music
BeFriend	Waze	Peloton	Google clips	thredUp	Yahoo Finance	Music Player
ChatterBox	Citymapper	HomeWorkout	Clips Maker	Carvana	Peralending	Music Tiles
TagTalk	Petal Maps	Idle Workout	Snap Studio	CARFAX	Cash app	Radio garden
SocialSphere	Google Maps	Fitness	Photo Studio	EcoShop	Mitrade	Radio Online
WeChat	Maps.me	Google Fit	Movie Maker	PriceNotify	Chime	radio.net
VibeVideo	AllTrails	Freelectics	PicCollage	Price Right	Turnkey lender	Radio
SocialSphere						

Table 1 shows applications selected for the study in their respective categories limited to a sample size of 50 which ensures a manageable and focused scope. This study size ensured in depth understanding how applications request permissions based on their functional requirements.

Out of the selected apps, the results were the following: -

Table 2: Permissions based on group analysis.

Permission	Description	occurrences
android.permission.INTERNET	Enables internet access by opening web sockets for data exchange	50
android.permission.ACCESS_COARSE_LOCATION	Approximates location data using cell towers and WIFI connections.	30
android.permission.ACCESS_FINE_LOCATION	Obtains precise location from GPS network connections.	45
android.permission.READ_CONTACTS	Reads user contact data crucial for app's managing phone address books	30
android.permission.CAMERA	Accesses phones camera to capture photos and videos	30
android.permission.RECORD_AUDIO	Accesses phones microphone to record audio messages	36
android.permission.ACCESS_WIFI_STATE	Accesses information about current WIFI network connections	22
android.permission.WRITE_EXTERNAL_STORAGE	writes to device external storage crucial for saving documents and constant data sharing	43
android.permission.READ_PHONE_STATE	accesses device status like network information, ongoing calls, phone numbers	29

Table 2 shows android permissions classified in categories of shopping applications, video and photo applications, banking applications, music applications, fitness tracking applications, and navigational applications.

Social media applications had INTERNET, READ_EXTERNAL_STORAGE, WRITE_EXTERNAL_STORAGE which are core permissions. CAMERA, RECORD_AUDIO, ACCESS_FINE_LOCATION, ACCESS_COARSE_LOCATION enhance its functionality but READ_SMS, CALL_PHONE are potentially unnecessary.

Navigational applications had INTERNET, ACCESS_FINE_LOCATION, ACCESS_COARSE_LOCATION, READ_EXTERNAL_STORAGE, WRITE_EXTERNAL_STORAGE needed for core functionality but READ_CONTACTS, RECORD_AUDIO are potentially unnecessary.

Banking applications had INTERNET, READ_PHONE_STATE, ACCESS_FINE_LOCATION as core permissions but CAMERA, RECORD_AUDIO are potentially unnecessary.

Shopping applications had INTERNET, ACCESS_FINE_LOCATION, ACCESS_WIFI_STATE as core permissions but READ_PHONE_STATE, RECORD_AUDIO, READ_SMS are potentially unnecessary.

Music, radio and podcast applications have INTERNET, RECORD_AUDIO, WRITE_EXTERNAL_STORAGE as core permissions but ACCESS_FINE_LOCATION, READ_CONTACTS are potentially unnecessary.

Health and fitness tracking applications have INTERNET, ACCESS_FINE_LOCATION, WRITE_EXTERNAL_STORAGE as core permissions but READ_CONTACTS, CAMERA are potentially unnecessary.

Video and photo applications have INTERNET, CAMERA, RECORD_AUDIO, READ_EXTERNAL_STORAGE, WRITE_EXTERNAL_STORAGE as core permissions but READ_SMS, ACCESS_FINE_LOCATION are potentially unnecessary.

The above results displayed from Table 1 show the most potentially abused Android permission groups in various applications that could violate the principle of least privilege (PLP). When these permissions are granted without proper rationale and knowing their core functionalities, it can lead to severe consequences like unauthorized access to sensitive user data. Each category of application listed above is required to carefully distinguish and evaluate necessary core permissions from potentially unnecessary ones restricting unauthorised access. This approach protects users from data breaches and the developer's reputation.

4.2 Permission group comparison against functionality

The diagram below shows a decompiled APK file used in permission group comparison regarding functionality.

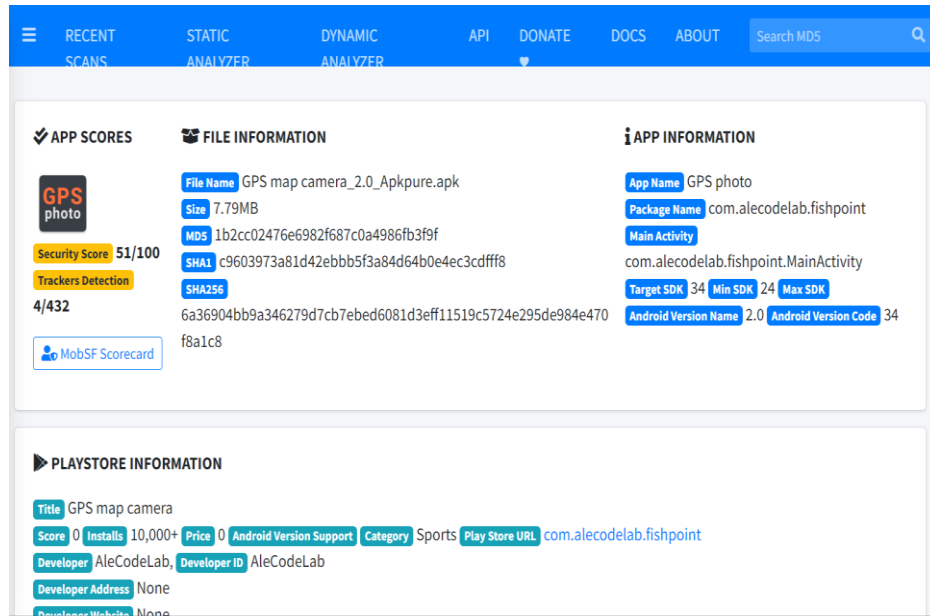


Figure 2: Decompiled GPS application.

Fig. 2. depicts a decompiled GPS application with the file name “GPS map camera_2.0” and its attributes such as size, encryption standards, application name, and compatible SDK among others downloaded from the “apkpure” website. The main aim of decompiling the file above was to analyse its “AndroidManifest.xml” to determine its core permissions, dangerous permissions, and potentially unnecessary permissions and compare application functionality to available documentation highlighting violations to the principle of least privilege (PLP) as shown in the table below: -

Table 3: Permission group comparisons

android.permission.WRITE_EXTERNAL_STORAGE	Writes to device external storage needed to save large data amounts yet deemed dangerous.
android.permission.READ_EXTERNAL_STORAGE	Allows the application to read from external storage such as sd card classified as dangerous.
android.permission.ACCESS_FINE_LOCATION	Obtains phones precise location from GPS network connections deemed exploitable and dangerous.
android.permission.INTERNET	Enables internet access and data exchange by opening up web sockets classified as normal permission.
com.google.android.gms.permission.AD_ID	Deals with advertising id thats unique and user resettable provided by google play services.
android.permission.ACCESS_NETWORK_STATE	Allows access to information about network connections classified as a normal permission granted at install time not runtime.
android.permission.WAKE_LOCK	Prevents the phone from going into sleep mode when an application is running and still needs system resources like cpu.
android.permission.RECEIVE_BOOT_COMPLETED	This enables the application to start an activity automatically after the phone boots by recieving system message.
android.permission.FOREGROUND_SERVICE	It allows apps tto continuously inform users of ongoing system processes by displaying notifications although they have prior knowledge of operations.
com.google.android.finsky.permission.BIND_GET_INSTALL_REFERRER_SERVICE	This enables developers of different apps to track usage through referral APIs that retrieve data from google play store.

Table 2 highlights permissions contained in the GPS map camera_2.0 application with their explanations below where some of them are classified as core to its functionality which are: -

android.permission.READ_EXTERNAL_STORAGE (classified as dangerous, runtime), android.permission.READ_EXTERNAL_STORAGE (classified as dangerous, runtime),
 android.permission.ACCESS_FINE_LOCATION (classified as dangerous, runtime)
 android.permission.INTERNET (classified as normal granted at install time),
 android.permission.ACCESS_NETWORK_STATE (classified as normal, install time)

The application also has permissions that run in the background that are potentially unnecessary for its core functionality as listed below: -
 com.google.android.gms.permission.AD_ID, android.permission.WAKE_LOCK,
 android.permission.RECEIVE_BOOT_COMPLETED,
 android.permission.FOREGROUND_SERVICE,
 com.google.android.finsky.permission.BIND_GET_INSTALL_REFERRER_SERVICE

In summary, granting permissions that are not essential to application core functions potentially violate the principle of least privilege (PLP) which risks undermining user privacy and misuse of system resources. Core permissions are classified as both dangerous and normal with those granted during installation process replaced with runtime in android 6 and above. Background permissions are potentially unnecessary and ensuring that only necessary permissions are granted improves application

security. Furthermore, cross-checking permission requests against application documentation can determine unnecessary ones that undermine this principle.

5 Discussions and conclusions.

The study's findings indicate that some dangerous permission groups are needed for application functionality to avoid its overall degradation. Software developers should implement only essential permissions which are core to application functions, preventing unauthorised access to sensitive user data and system resources. This creates a sense of security on mobile device usage and non-violations of the principle of least privilege (PLP). Furthermore, limiting application permission requests to only essential ones needed for its core functions, it does not only reduce the risk of unnecessary access to user data and system resources but also decreases potential attack vectors and malicious activity. This targeted approach limits data leakages if the application is breached, improving security with enhanced user trust.

As a future recommendation, research into developing automated tools for permission analysis would help to solve violations to the principle of least privilege. These tools would identify unnecessary permissions requested by these applications during runtime in Android 6 and above. Minimal permissions needed for applications core functions could be identified with the help of artificial intelligence and machine learning algorithms which examine and learn its behavior. This automated approach can eliminate overprivileged applications that pose potential risk to user data through unauthorised access (Dilhara et al., 2018).

Creation of dynamic systems is another way to avoid violation of this principle. Studies could be conducted in dynamic systems which temporarily grant permissions depending on contextual analysis and be revoked anytime when not needed. This approach enhances application security and limits violations to the principle of least privilege (PLP) through analysis of user behavior and temporal permissions instead of granting permanent access. The downside to this is the complexity in developing software algorithms that will track these changes in real-time (Kumar et al., 2018).

Educational awareness programs are another recommendation not to be ignored when dealing with user awareness to the principle of least privilege (PLP). Researchers can evaluate the effectiveness of these training and study programs delivered to mobile users who download and install applications from trusted and untrusted sources almost daily. These studies create user awareness about risks involved in granting too many permissions, downloading applications from untrusted sources which could mimic original ones and are overprivileged in nature (Palmerino, 2018).

To sum it all up, a combined approach to malware detection and analysis on Android applications using static and dynamic techniques offers a comprehensive understanding of permission groups and their respective vulnerabilities. This approach highlights the relevance of granting essential permissions to control

unauthorised access to sensitive user data, and system resources. It further limits potential malware and attack vectors that exploit these permissions to harm system integrity. Finally, granting only essential permissions demonstrates a systematic approach to the development of secure Android applications streamlined to protect user privacy.

6. Acknowledgement

We extend our gratitude to London Metropolitan University for providing the necessary resources that supported our research, this significantly enabled the development of our conference paper.

Authors Contributions

MK. E.H. H.K. I.A. developed the idea of malware detection. M.K. reviewed malware behavior techniques and analysed applications violating the principle of least privilege conceptualising the research.

M.K. selected and verified the use of appropriate methods needed to carry out tests. M.K. documented test results with clear explanations. E.H. H.K. I.A. supervised the research implemented by M.K. providing clarity on key sessions of the paper.

M.K. discussed the effects of these results on user privacy and overall application security. M.K. made conclusions on responsible permission usage.

References

1. Almomani, Almomani, I.M. & Khayer, A.A. (2020) A Comprehensive Analysis of the Android Permissions System. *IEEE Access*, 8, pp.216671-88.
2. Chao, W. et al. (2020) An Android Application Vulnerability Mining Method Based On Static and Dynamic Analysis. In *IEEE, ed. 2020 IEEE 5th Information Technology and Mechatronics Engineering Conference (ITOEC)*. Chongqing, China, 2020. IEEE.
3. Chen, J.-M., Feng, Y.-M. & Mei, Y.-M. (2010) Automatically identifying technology in malware based on mass samples. In *2010 International Conference on Machine Learning and Cybernetics*. Qingdao, China, 2010. IEEE.
4. Dilhara, M., Cai, H. & Jenkins, J. (2018) Automated Detection and Repair of Incompatible Uses of Runtime Permissions in Android Apps. In *2018 IEEE/ACM 5th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*. Gothenburg, Sweden, 2018. IEEE.
5. Gyamfi, N.K. & Owusu, E. (2018) Survey of Mobile Malware Analysis, Detection Techniques and Tool. In *IEEE, ed. 2018 IEEE 9th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*. Vancouver, BC, Canada, 2018. IEEE.
6. Kumar, S., Shanker, R. & Verma, S. (2018) Context Aware Dynamic Permission Model: A Retrospect of Privacy and Security in Android System. In *2018 International Conference on Intelligent Circuits and Systems (ICICS)*. Phagwara, India, 2018. IEEE.
7. LaMalva, G. & Schmeelk, S. (2022) MobSF: Mobile Health Care Android Applications Through The Lens of Open Source Static Analysis. In *IEEE, ed. 2020 IEEE MIT*

Undergraduate Research Technology Conference (URTC). Cambridge, MA, USA, 2022. IEEE.

8. Mehra, V., Jain, V. & Uppal, D. (2015) DaCoMM: Detection and Classification of Metamorphic Malware. In 2015 Fifth International Conference on Communication Systems and Network Technologies. Gwalior, India, 2015. IEEE.
9. Moghaddam, S.H. & Abbaspour, M. (2014) Sensitivity analysis of static features for Android malware detection. In 2014 22nd Iranian Conference on Electrical Engineering (ICEE). Tehran, Iran, 2014. IEEE.
10. Olejnik, K. et al. (2017) SmarPer: Context-Aware and Automatic Runtime-Permissions for Mobile Devices. In IEEE, ed. 2017 IEEE Symposium on Security and Privacy (SP). CA, USA, 2017. IEEE.
11. Palmerino, J. (2018) Improving Android Permissions Models for Increased User Awareness and Security. In 2018 IEEE/ACM 5th International Conference on Mobile Software Engineering and Systems (MOBILESoft). Gothenburg, Sweden, 2018. IEEE.
12. Sara, J.J. & Hossain, S. (2023) Static Analysis Based Malware Detection for Zero-Day Attacks in Android Applications. In 2023 International Conference on Information and Communication Technology for Sustainable Development (ICICT4SD). Dhaka, Bangladesh, 2023. IEEE.
13. Squires, W. & Centonze, P. (2016) Cross-Platform Access-Rights Analysis of Mobile Applications. In 2016 IEEE/ACM International Conference on Mobile Software Engineering and Systems (MOBILESoft). Austin, TX, USA, 2016. IEEE.
14. Sun, C. et al. (2020) DroidPDF: The Obfuscation Resilient Packer Detection Framework for Android Apps. 8, pp.pp. 167460-167474.
15. Tang, J. et al. (2017) Detecting Permission Over-claim of Android Applications with Static and Semantic Analysis Approach. In 2017 IEEE Trustcom/BigDataSE/ICSS. Sydney, NSW, Australia, 2017. IEEE.
16. Wang, S. et al. (2022) APER: Evolution-Aware Runtime Permission Misuse Detection for Android Apps. In 2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE). Pittsburgh, PA, USA, 2022. IEEE.
17. Wu, S., Liu, J. & Chen, X. (2019) Overprivileged Permission Detection for Android Applications. In ICC 2019 - 2019 IEEE International Conference on Communications (ICC). Shanghai, China, 2019. IEEE.