Revolutionary Advances in Computing and Electronics (RACE): An International Journal
Volume 01, Issue 02, pp.26-35, Oct-Dec 2025
ISSN (Online): 3107-8540

Received: 17/12/2025, Accepted: 31/12/2025

# Risk-Oriented Taxonomy of Android App Assessment Models

Anuradha Dahiya[1], Sukhdip Singh[2], Gulshan Shrivastava[3]

[1,2]Department of CSE, Deenbandhu Chhotu Ram University of Science and Technology, Murthal, India

[3]School of Computer Science Engineering and Technology, Bennett University, Greater Noida, UP, India

anudahiya3973@gmail.com, sukhdeepsingh.cse@dcrustm.org, gulshanstv@gmail.com

## ABSTRACT

Android application security research has been dominated by the malware-detection paradigm, in which either benign or malicious labels are assigned to applications. Although effective in terms of large-scale screening, this binary perspective is not effective to capture the diverse risks brought about by modern Android applications, such as privacy leakage, financial abuse, intrusive tracking, and misuse of system resources. This work provides a structured review of the research efforts on Android app assessment models from a risk-oriented perspective. Rather than proposing new techniques for detection, this review synthesizes the existing literature and organizes prior work through a multi-dimensional taxonomy with respect to risk modelling granularity, risk semantics, evidence sources, learning paradigms, and interpretability. This paper reveals important trends, constraints, and unresolved issues by analysing how current approaches conceptualise and depict risk. The purpose of the article is to outline the shift from malware detection to thorough application risk assessment and to provide a comprehensive reference for future Android security research.

**Keywords**: *Android security, Malicious activities, User privacy, Application Trust & Risk, Application metadata, Risk assessment.*

## 1. Introduction

Due to Android's widespread use, a vast ecosystem of apps has appeared from entities with diverse security policies and motivations. The diverse apps reached users through the official Play Store and third-party app stores. Security experts have noted that with diversity, the different versions of malicious programs have increased continuously, and new families of malware are occurring at an alarming rate [1]. The previous malware basically concentrated on premium SMS fraud and basic identity theft, but today's threat environment has enhanced. Today's threat strategy includes ransomware, highly targeted spyware, banking trojans, cryptocurrency miners. Detection is avoided by evasion tactics that adapt to timing triggers or device conditions. An enhanced detection challenge is presented by dynamic code loading, code obfuscation, encryption of malicious payloads, and polymorphic behaviour [2]. Especially the proliferation of "logic bombs" threats, which do not trigger during security evaluations but start maliciously working when certain conditions are met in real-world circumstances, has raised security concerns.

Another obstacle is threats that can misuse legitimate platform functions as weaponry to commit malicious conduct. The best example of this is an overlay attack, which can exploit the platform's screen rendering feature to create phishing pages that steal user credentials. These types of attacks mainly exploit authorised system functionalities and make themselves harder to detect [3]. Moreover,

*Corresponding author: Anuradha Dahiya, Department of CSE, Deenbandhu Chhotu Ram University of Science and Technology, Murthal, India. (anudahiya3973@gmail.com) increasing use of mobile apps, IoT devices, cloud services, and enterprise services enhanced the complexity of threat identification by progressively increasing the threats. Figure 1 highlights some devices from a wide range of devices powered by the Android platform, such as mobile phones, wearables, tablets, ATMs, GPS devices and other smart devices. The deployment of Android across heterogeneous devices increases the surface area for attackers, enabling propagation of security threats introduced by mobile applications across multiple endpoints. Malicious activities can likely affect the privacy of the country and its users, organisational infrastructure, and various imperative domains such as the healthcare, education, and finance sectors.

**Figure 1:** Some Android-powered devices

The famous malware incidents, such as FluBot, Xenomorph, and Joker, demonstrate how malware can access official app stores by evading detection mechanisms and cause significant security harm. Several methods based on static and dynamic analysis and machine learning have been proposed to analyse Android applications for protection [4]. Most of these works have been framed as malware detection mechanisms, with the primary objective of distinguishing malicious applications from benign ones. This work analysed existing Android app assessment research from a risk-oriented perspective and reached a determination. The primary objective is to structure the risk assessment field, highlight its limitations, and foster a clearer understanding of how risk is currently assessed in Android applications.

## 2. Background

In risk analysis, different types of weaknesses are identified, such as permission misuse in the app, misuse of legitimate platform APIs in the app, and privacy leakage in the app. From the perspective of risk assessment, Table 1 lists the primary threats to the Android platform, their corresponding mechanisms, the significant risk impact, and the entities that are impacted. The information Applications use rather than their initial intention raises privacy concerns. Financial losses are mostly caused by illegal transactions, stolen banking credentials, and the misuse of premium SMS services. Phishing overlays and forged calls are pointing to user interface risks. Misuse of accessibility services, abuse of background execution permissions, and abuse of inter-component communication are examples of platform dangers.

Android apps can act as gateways for lateral movement and even for the integration of compromised cloud services, posing organisational hazards in the corporate world. Risk-oriented techniques usually produce a risk score, rating, or category that represents the likelihood and severity of potential harm rather than a single output, such as a yes or no verdict. In situations where prioritisation and contextual decision-making are necessary, such as app marketplace screening, enterprise mobile management, and regulatory compliance, this distinction is very crucial. This work thus focuses on risk assessment models for Android apps that have been proposed in the literature utilising classical machine learning, deep learning, and rule-based risk modelling strategies in addition to static, dynamic, and hybrid analysis techniques.

**Table 1:** Risk-Oriented Taxonomy of Android Apps Threats

| **Threat Type** | **Common Mechanisms** | **Impact** | **Affected Entities** |
|---|---|---|---|
| Financial fraud | Premium services abuse, billing API exploitation | Monetary loss | Users, banks |
| Privacy abuse | Sensor misuse, background tracking | Sensitive data leakage | End users and platform providers |
| User interface deception | Overlay attacks, phishing interfaces, fraudulent login screens, clickjacking | Credential compromise | End users |
| Platform capability abuse | Misuse of accessibility services, privilege escalation mechanisms | System integrity compromise | Device ecosystem |
| Enterprise-oriented threats | Credential harvesting and reuse, unauthorized service access | Network and organizational breach | Enterprises |
| Resource abuse | Cryptomining, excessive background execution | Device degradation and energy drain | End users |
| Communication abuse | Covert network channels, data exfiltration over encrypted traffic | Information leakage and control loss | Users, organizations |

## 3. Analysis Mechanisms for Android App Risk Assessment

The main analysis techniques used in current Android app risk assessment models are examined in this section. Static, dynamic, and hybrid approaches can be used to broadly classify malicious activities, each of which provides varying degrees of insight into application behaviour and related risk factors. Further analysis approaches considered miscellaneous perspectives for protection.

### *3.1 Static Analysis-Based Risk Assessment*

Static analysis is the most commonly used analysis approach, evaluating apps without executing them and conducting app package, manifest, and source- or bytecode-level reviews to identify issues. Static analysis is used to infer an app’s capabilities and intent, which can indicate potential risk even in the absence of observed malicious behaviour that can be identified by running the app. The most popular features of an app that can be identified in static analysis are static app permissions, declared components, intent filters, app metadata, and API calls [5]. The app's requests for excessive or sensitive permissions, such as those for messaging, accessibility services, overlays, or background execution, are often seen as signs of increased platform, financial, or privacy risk. Additionally, the usage of crypto APIs in apps, reflected calls, and dynamic class loading indicates malicious app activity, such as attempts to hide behaviour or permit remote command execution. These concerns complicate the identification of threats through static analysis [6]. The application's attack surface is also expanded by manifest-level attributes, such as exported components lacking proper access controls, which are commonly incorporated into risk scoring models [7]. Static analysis is a suitable method for early risk screening in large-scale situations. Although static analysis-based methods produced satisfactory results, they struggled to handle environment-specific logic, identify conditional execution paths, and capture runtime behaviour. The reliability of static indicators can be seriously compromised by many tactics, including obfuscation, reflection, and encrypted payloads.

### *3.2 Dynamic Analysis-Based Risk Assessment*

Dynamic analysis involves basically monitoring the app's behaviour by running it in a controlled environment, such as a sandbox. Dynamic analysis identifies actual app usage by monitoring app actual events, providing insights into risks that may occur during use. Dynamic analysis techniques mostly concentrate on monitoring system calls, network usage, file system interaction, and user interface usage [8]. Dynamic analysis is useful for the detection of data exfiltration attacks, inappropriate network messaging, the usage of premium services, and UI attacks such as the creation of phishing overlays. Dynamic analysis is quite important in the analysis of logic bombs and trigger-style malicious code, which behave harmlessly at the time of static analysis. Although dynamic analysis provides strong positive points for risk analysis by observing the runtime behaviour of apps, it may face significant challenges in risk assessment. Enhanced modern threats generally exhibit environment-awareness. These threats alter behaviour in response to timing conditions, leading to missed external triggers or sandbox behaviour detection [9]. The detection of delayed behaviours through dynamic analysis is affected by limited execution time in the control environment and poor input coverage. So, dynamic analysis observations may underestimate the true risk posed by an application when malicious actions are designed to activate only under real-world conditions. Additionally, dynamic analysis is inappropriate at large scale due to the time and resource constraints.

### *3.3 Hybrid Analysis-Based Risk Assessment*

Many studies used hybrid analysis approaches that overcome complementary limitations of both static and dynamic methodologies by combining features from both. The primary objective of hybrid analysis is to enhance risk assessment by integrating actual usage information from dynamic analysis with the scalability and completeness of static analysis. In hybrid risk assessment models, dynamic information features, such as network activity, behavioural data, and user interface activity, are usually combined with static features, permissions, APIs, and components [10]. By doing so, one can better cover threat types to examine both latent and visible malicious activity. In particular, hybrid analysis is very helpful for assessing platform capability abuse and enterprise threats, for which static information indicates possible paths to abuse, while dynamic information indicates abused behaviours [11]. While hybrid analysis offers improved expressiveness and robustness, it introduces additional complexity in feature integration, system design, and computational overhead. The increased cost of analysis can limit scalability, especially in large-scale app vetting scenarios.

From a standpoint, no single analysis technique provides complete visibility into all threat dimensions. Table 2 compares different state-of-the-art approaches used in the literature for Android apps risk assessment. Static analysis is very effective in identifying risks, particularly in the early phases, but lacks behavioural evidence, and the dynamic analysis that supports it is susceptible to evasive malware and mixed code coverage. The hybrid approach attempts to strike a middle ground between these two and to offer a wide range of risk-related information about the apps, but it increases complexity. This comparative viewpoint emphasises that the desired balance between scalability, depth of analysis, and tolerance for uncertainty in risk estimate ultimately determines the choice of analytic technique.

**Table 2:** Overview and Comparison of Prior Android App Risk Assessment Approaches

| **References** | **Approach used** | **Used Features** | **Risk Evaluation** | **Outcomes** |
|---|---|---|---|---|
| Arif et al. (2021) [12] | Evaluated apps using static analysis-based fuzzy analytical hierarchy process risk assessment. | Permission | Classified apps in four risk levels. | Increase awareness in granting any permission request. |
| Amin et al. (2019) [13] | Hybrid analyses (static and dynamic) used to detect vulnerabilities in Android apps. | Manifest, source code | Vulnerabilities identified and | Detect information leaks and insecure network requests. |

| | | | | |
|---|---|---|---|---|
| | | based and runtime | classified into three category. | |
| Shrivastava and Kumar (2019) [14] | Static analysis is used to evaluate efficiency of intents and permissions as a differentiating feature. | Intent, Permission | Classified apps in three categories. | Assisting how risky an app is before installation. |
| Dhalaria and Gandotra (2022) [15] | An artificial neural network model is developed to estimate class probabilities. | Permission | Apps classified into four risk factors. | A graphical user interface is provided to evaluate risky apps. |
| Razak et al. (2019) [16] | Applied Analytic Hierarchy Process as a decision factor with static analysis to calculate risk value. | Permission | Classified apps in four different risk levels. | Provides effective risk evaluation with increased user awareness. |
| Sharma and Gupta (2018) [17] | Identified risk factors using static analysis with usage of permissions in dataset samples. | Permission | Classified apps in four different risk levels. | Quickly identified app's intention by permission requests |
| Deypir and Horri (2018) [18] | Introduced a risk estimation metric that uses previously known malicious and non-malicious app instances. | Instance based | Risk score calculated for apps. | Effectively assign high risk score to malware apps. |
| Xiao et al. (2020) [19] | Combined collaborative filtering with static analysis to find the minimum consents needed for an app. | Permission, API usage | Risk value identified for apps. | Evaluates over privilege risk of apps by examining app’s additional privileges. |
| Malleswari et al. (2017) [20] | Combined static permission analysis with different factors to estimate the risk score. | Permission | Risk score calculated for apps. | Increasing awareness about privacy risk. |
| Son et al. (2022) [21] | A hybrid code analysis is used to determine whether app requests more than required and whether collected data is used locally or externally. | Static and dynamic code features | Classified into five different risk levels. | Encouraging evaluation results with two different groups (users and crowd workers). |
| Yoo et al. (2019) [22] | Introduced a visual analytics system that enables observation, mitigation, and assessment of the effect of security risks. | Permission | Risk score assigned to apps. | Easily recognise unintended security activities using lifelog risk views. |
| Merlo and Georgiu (2017) [23] | Applied different machine learning techniques for app risk analysis. | Permission | Provides reliable risk index value | Provide reliable estimation using multiple permission usage patterns |
| Sanna et al. (2024) [24] | Provided the approach for checking the vulnerabilities of native code in apps. | Static native code components inspection | Risk score assigned to apps. | Aid researchers and developers in mitigating immediate risks. |

| | | | | |
|---|---|---|---|---|
| Dahiya et al. [25] | Static analysis based demeanour study of apps. | Permission | Classified apps in five different risk levels. | Prior to installing an app, determine the level of danger. |

Existing studies typically use static, dynamic, or hybrid analysis techniques to extract security-relevant indicators (Figure 2), which are then processed using rule-based or learning-based models to produce risk scores, rankings, or categorical risk levels rather than binary decisions. In the current threat analysis of Android applications, risk assessment methods primarily focus on single analysis parameters. To obtain a comprehensive understanding of application risk profiles, an integrated model that combines code-level vulnerabilities, authorisation requirements, runtime behaviour, application purpose, reputation, and marketplace placement is required. This multifaceted assessment approach can guide security analysts in making more informed decisions to protect users and promote innovation in the Android ecosystem.

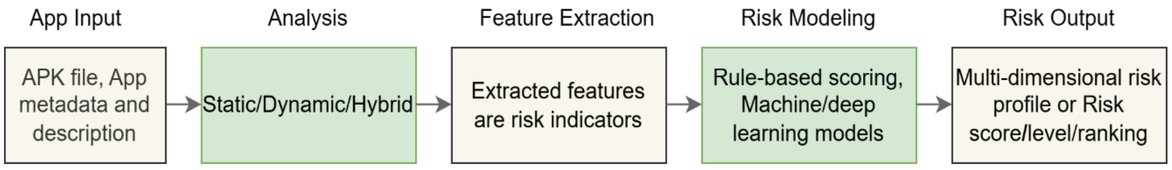
App Input
Analysis
Feature Extraction
Risk Modeling
Risk Output
APK file, App metadata and description
Static/Dynamic/Hybrid
Extracted features are risk indicators
Rule-based scoring, Machine/deep learning models
Multi-dimensional risk profile or Risk score/level/ranking

**Figure 2:** General workflow of risk assessment that follow in the literature

## 4. Analysis of Feature-Level Risk Indicators

The literature review of this work has been completed, with a representative analysis that identifies empirical observations discussed here. A sufficient number of AndroZoo [26] dataset app samples have been analysed to examine how commonly used application features relate to application risk. Various machine learning classifiers are used for classification with static features derived from requested permissions and invoked API calls. A CatBoost [27] classifier yielded the best results among these. The features that play the most important role in the classifier's decision-making are identified along with their importance values. This analysis identifies features that consistently exhibit strong associations with risk-related behaviour. The classifier's feature importance scores are used to rank permissions and API calls by their contribution to the classification outcome, and the top 20 features in each category are shown in Figures 3 and 4, which visualise their relative influence. To put these findings into context from a risk assessment perspective, the risk impact of these highly influential features, which can affect user devices, is shown in Table 3, taking into account privacy disclosure risks, financial fraud risks, and platform capability risks. This table shows that misuse of these highly influential features can affect users and devices in different ways, such as an application requesting read phone state permission without a clear need, pointing to suspicious activity that can impact anonymity, and leading to data and privacy leakage. An app whose main task is related to the calendar but that asks for storage and device state permissions indicates suspicious behaviour.

**Table 3:** Security Implications of High-Importance Features

| Feature | Security Implications |
|---|---|
| "android_permission _read_phone_state" | Most legitimate app features do not require detailed phone state information. Misuse of this permission is commonly associated with loss of anonymity, data leakage, tracking, and privacy-invasive behaviour. |
| "android_permission _send_sms" | Few modern apps genuinely need to send SMS automatically. If an app that is not a core messaging or telecom app requests SEND_SMS, it can be associated with financial fraud, subscription abuse, spam propagation, |

| | |
|---|---|
| | account compromise, and delaying user awareness of outgoing messages when combined with other permissions. |
| “android.telephony.TelephonyManager.getDeviceId” | It exposes non-resettable persistent identifiers that do not change across app reinstalls or device reboots. From a security perspective, its misuse has serious privacy implications. |
| “android.app.ActivityManager.getRunningTasks” | It allows an app to obtain information about tasks and activities currently or recently running on the device. Misuse of this API is commonly associated with privacy and monitoring concerns, user behaviour surveillance, credential harvesting, targeted exploitation, and bypassing user awareness. |
| “android_permission_receive_boot_completed” | Legitimate apps such as alarms or system utilities may need this permission. If an app without a clear need to start at boot requests, it signals a risk of stealthy execution, battery and resource abuse, early-stage surveillance, combination attacks, and persistent behaviour. |
| “android_permission_get_tasks” | It allows an app to retrieve information about tasks and applications running on the device. Misuse of this permission is commonly associated with user activity tracking, privacy leakage, phishing attacks, silent surveillance, adaptive malware behaviour, and credential-stealing attacks. |
| “java.io.FileNotFoundException.<init>” | It is not malicious by nature, but in behavioural analysis, it can act as a weak signal of file probing or environment detection. It can be used to decide whether to activate malicious behaviour and data access attempts. |
| “java.util.zip.ZipFile.<init>” | This API is common and legitimate, but in security analysis, its context of use is important. It can be used to support hidden payload loading, evasion techniques, dynamic code execution, dropper behaviour, and obfuscation of malicious resources. |
| “android_permission_write_external_storage” | It allows an app to write data to shared storage, which is accessible by other apps or the user. If an app requests user-accessible files without a clear need to store them, it can be associated with data leakage, malware persistence, payload staging, tampering and corruption, and hiding malicious files. |
| “android.os.Parcel.writeLong” | This API is neutral by itself. It acts as a contextual feature that may indicate hidden data exchange, component communication, privilege abuse, or obfuscation of behaviour. |

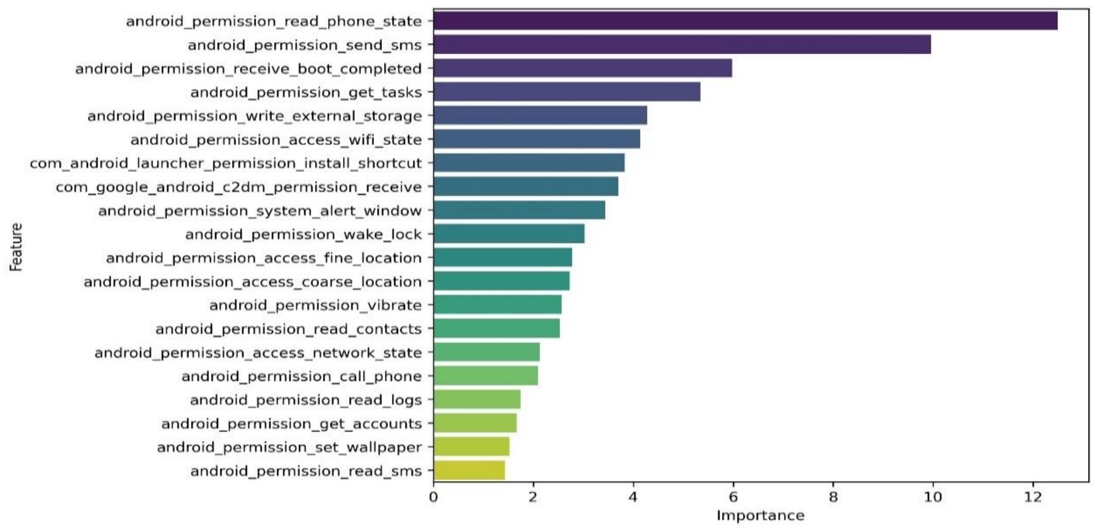

**Figure 3:** Most Influential Permissions for Classification

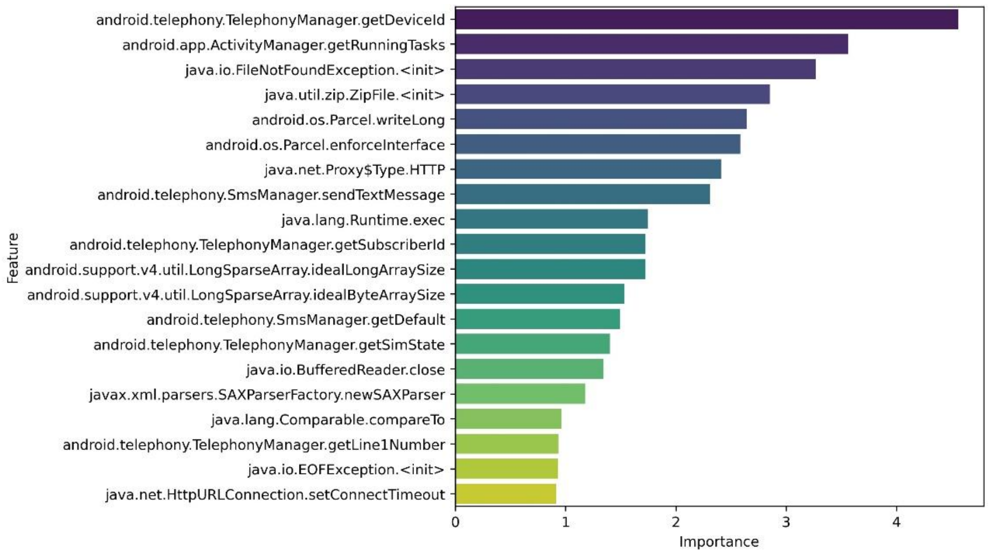

**Figure 4:** Most Influential API Calls for Classification

## 5. Future work

The analysis indicates that, though static, dynamic, and hybrid approaches remain very popular, each has its own trade-offs regarding scalability, behaviour coverage, and resistance to evasion attacks. Learning models, such as machine learning and deep learning, have been shown to improve the ability to combine indicators into risk scores or rankings, but they rely on datasets that may lack temporal coverage and risk labelling. Existing solutions struggle to model the behaviour of delayed, context-dependent actions, especially those related to UI deception, logic bombs, and the misuse of legitimate services for malicious purposes. Moreover, the solution, which impairs transparency of explanations, makes it difficult to implement within decision-making mechanisms involving consumers, organisations, or regulators. Furthermore, no standardised benchmark or evaluation metric is defined in risk assessments. In the future, the research should aim to develop context-aware, explainable risk assessment systems that articulate user, device, and deployment contexts. Lightweight risk analysis pipelines are needed to enable large-scale app risk scanning. The longitudinal data sources and standardised risk definition will enable the research to evaluate the study in the future. Finally, building bridges between the existing mobile and cloud security ecosystems and the Android app risk evaluation environment may be a good way to address the new cross-platform threats.

## 6. Conclusions

This paper conducted an appraisal of existing approaches to Android app risk evaluation models within a risk-based taxonomy, as defined by the existing literature. In this, it showcased how contemporary research has moved from simple binary malicious-code detection to a much more complex understanding of application-related risks, taking into account privacy abuse, financial threats, deceptive user interface, platform abuse, and business impact. This paper is able to point out the efforts put forth by existing work, aside from malware identification, towards risk representation, their strength, as well as existing challenges.

**Funding source**

No funding was received for this study.

**Conflict of Interest**

There is no conflict of interest.

## References

[1] S. Maganur, Y. Jiang, J. Huang, and F. Zhong, ‘Feature-Centric Approaches to Android Malware Analysis: A Survey’, *Computers*, vol. 14, no. 11, p. 482, Nov. 2025, doi: 10.3390/computers14110482.

[2] A. Ruggia, D. Nisi, S. Dambra, A. Merlo, D. Balzarotti, and S. Aonzo, ‘Unmasking the Veiled: A Comprehensive Analysis of Android Evasive Malware’, in *Proceedings of the 19th ACM Asia Conference on Computer and Communications Security*, Singapore: ACM, Jul. 2024, pp. 383–398, doi: 10.1145/3634737.3637658.

[3] A. Kar, N. Stakhanova, and E. Branca, ‘Detecting Overlay Attacks in Android’, *Procedia Computer Science*, vol. 231, pp. 137–144, 2024, doi: 10.1016/j.procs.2023.12.185.

[4] A. Dahiya, S. Singh, and G. Shrivastava, ‘Android malware analysis and detection: A systematic review’, *Expert Systems*, p. e13488, Oct. 2023, doi: 10.1111/exsy.13488.

[5] Q. Wu, X. Zhu, and B. Liu, ‘A Survey of Android Malware Static Detection Technology Based on Machine Learning’, *Mobile Information Systems*, vol. 2021, pp. 1–18, Mar. 2021, doi: 10.1155/2021/8896013.

[6] Z. Qu, S. Alam, Y. Chen, X. Zhou, W. Hong, and R. Riley, ‘DyDroid: Measuring Dynamic Code Loading and Its Security Implications in Android Applications’, in *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, Jun. 2017, pp. 415–426, doi: 10.1109/DSN.2017.14.

[7] A. Dahiya, S. Singh, and G. Shrivastava, ‘Lightweight and Efficient Android Malware Detection Using Manifest File Analysis’, in *2025 International Conference on Networks and Cryptology (NETCRYPT)*, May 2025, pp. 1246–1251, doi: 10.1109/NETCRYPT65877.2025.11102561.

[8] T. Sutter, T. Kehrer, M. Rennhard, B. Tellenbach, and J. Klein, ‘Dynamic Security Analysis on Android: A Systematic Literature Review’, *IEEE Access*, vol. 12, pp. 57261–57287, 2024, doi: 10.1109/ACCESS.2024.3390612.

[9] B. Kondracki, B. A. Azad, N. Miramirkhani, and N. Nikiforakis, ‘The Droid is in the Details: Environment-aware Evasion of Android Sandboxes’, in *Proceedings 2022 Network and Distributed System Security Symposium*, 2022, doi: 10.14722/ndss.2022.23056.

[10] A. Dahiya, S. Singh, and G. Shrivastava, ‘Malware Detection Insights, Mechanisms and Future Perspectives for Android Applications’, in *Innovative Computing and Communications*, vol. 1021, Singapore: Springer Nature Singapore, 2024, pp. 381–403, doi: 10.1007/978-981-97-3591-4_31.

[11] M. Choudhary and B. Kishore, ‘HAAMD: Hybrid Analysis for Android Malware Detection’, in *2018 International Conference on Computer Communication and Informatics (ICCCI)*, Jan. 2018, pp. 1–4, doi: 10.1109/ICCCI.2018.8441295.

[12] J. Mohamad Arif, M. F. Ab Razak, S. R. Tuan Mat, S. Awang, N. S. N. Ismail, and A. Firdaus, ‘Android mobile malware detection using fuzzy AHP’, *Journal of Information Security and Applications*, vol. 61, p. 102929, Sep. 2021, doi: 10.1016/j.jisa.2021.102929.

[13] A. Amin, A. Eldessouki, M. T. Magdy, N. Abdeen, H. Hindy, and I. Hegazy, ‘AndroShield: Automated Android Applications Vulnerability Detection, a Hybrid Static and Dynamic Analysis Approach’, *Information*, vol. 10, no. 10, p. 326, Oct. 2019, doi: 10.3390/info10100326.

[14] G. Shrivastava and P. Kumar, ‘SensDroid: Analysis for Malicious Activity Risk of Android Application’, *Multimed Tools Appl*, vol. 78, no. 24, pp. 35713–35731, Dec. 2019, doi: 10.1007/s11042-019-07899-1.

[15] M. Dhalaria and E. Gandotra, ‘Risk Detection of Android Applications Using Static Permissions’, in *Advances in Data Computing, Communication and Security*, Singapore: Springer Nature, 2022, pp. 591–600, doi: 10.1007/978-981-16-8403-6_54.

[16] M. F. A. Razak, N. B. Anuar, R. Salleh, A. Firdaus, M. Faiz, and H. S. Alamri, '"Less Give More": Evaluate and zoning Android applications', *Measurement*, vol. 133, pp. 396–411, Feb. 2019, doi: 10.1016/j.measurement.2018.10.034.

[17] K. Sharma and B. B. Gupta, 'Mitigation and risk factor analysis of android applications', *Computers & Electrical Engineering*, vol. 71, pp. 416–430, Oct. 2018, doi: 10.1016/j.compeleceng.2018.08.003.

[18] M. Deypir and A. Horri, 'Instance based security risk value estimation for Android applications', *Journal of Information Security and Applications*, vol. 40, pp. 20–30, Jun. 2018, doi: 10.1016/j.jisa.2018.02.002.

[19] J. Xiao, S. Chen, Q. He, Z. Feng, and X. Xue, 'An Android application risk evaluation framework based on minimum permission set identification', *Journal of Systems and Software*, vol. 163, p. 110533, May 2020, doi: 10.1016/j.jss.2020.110533.

[20] D. Naga Malleswari, A. Dhavalya, V. Divya Sai, and K. Srikanth, 'A detailed study on risk assessment of mobile app permissions', *IJET*, vol. 7, no. 1.1, p. 297, Dec. 2017, doi: 10.14419/ijet.v7i1.1.9706.

[21] H. X. Son, B. Carminati, and E. Ferrari, 'A Risk Estimation Mechanism for Android Apps based on Hybrid Analysis', *Data Sci. Eng.*, vol. 7, no. 3, pp. 242–252, Sep. 2022, doi: 10.1007/s41019-022-00189-1.

[22] S. Yoo, H. R. Ryu, H. Yeon, T. Kwon, and Y. Jang, 'Visual analytics and visualization for android security risk', *Journal of Computer Languages*, vol. 53, pp. 9–21, Aug. 2019, doi: 10.1016/j.cola.2019.03.004.

[23] A. Merlo and G. C. Georgiu, 'RiskInDroid: Machine Learning-Based Risk Analysis on Android', in *ICT Systems Security and Privacy Protection*, vol. 502, Cham: Springer International Publishing, 2017, pp. 538–552, doi: 10.1007/978-3-319-58469-0_36.

[24] S. L. Sanna, D. Soi, D. Maiorca, G. Fumera, and G. Giacinto, 'A risk estimation study of native code vulnerabilities in Android applications', *Journal of Cybersecurity*, vol. 10, no. 1, p. tyae015, Jan. 2024, doi: 10.1093/cybsec/tyae015.

[25] A. Dahiya, S. Singh, and G. Shrivastava, 'PRAZdroid: A Novel Approach to Risk Assessment and Zoning of Android Applications based on Permissions', *SCPE*, vol. 26, no. 4, Jun. 2025, doi: 10.12694/scpe.v26i4.4439.

[26] K. Allix, T. F. Bissyandé, J. Klein, and Y. Le Traon, 'AndroZoo: collecting millions of Android apps for the research community', in *Proceedings of the 13th International Conference on Mining Software Repositories*, Austin Texas: ACM, May 2016, pp. 468–471, doi: 10.1145/2901739.2903508.

[27] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin, 'CatBoost: unbiased boosting with categorical features', in *Advances in Neural Information Processing Systems*, 2018, p.31.