Archives available at journals.mriindia.com

International Journal on Advanced Computer Engineering and Communication Technology

ISSN: 2278-5140

Volume 14 Issue 03s, 2025

Advanced Risk Analyzer for Android Apps

[1]Shrushti S. Barhate, [2]Vedanti N. Dhage, [3]Tanisha S. Teware, [4]Prof. Neha Zade

[1,2,3,4] Computer Science and Engineering (Cybersecurity), St. Vincent Pallotti College of Engineering and Technology, Nagpur, India

Email: [1]shrushtibarhate344@gmail.com, [2]vedantidhage72@gmail.com, [3]tanishateware23@gmail.com, [4]nehazade@stvincentngp.edu.in

Peer Review Information

Submission: 05 Nov 2025

Revision: 25 Nov 2025

Acceptance: 17 Dec 2025

Keywords

Android Security, APK Analysis, Malware Detection, Static Analysis, Machine Learning, Mobile Privacy, Compliance.

Abstract

The rapid growth of Android applications has changed mobile technology by making it accessible, innovative, and convenient for billions of users around the world. However, this popularity has also led to an increase in malicious apps that threaten user privacy, exploit sensitive data, and pose serious security risks. Traditional detection methods, like signature- based antivirus solutions, often fail against zero-day malware, obfuscated code, or overly permissive applications that seem safe but misuse data.

To tackle these issues, we present AndroidRiskCheck, a hybrid framework that combines static program analysis with machine learning (ML) to evaluate risks in Android applications. The system extracts permissions and metadata directly from APK files using a Python-based pipeline, removing Java dependencies and ensuring light cross-platform performance. A smart XML parsing mechanism is used to manage both human-readable and binary AndroidManifest files, ensuring reliability across different APK structures. The features extracted are converted into numerical vectors and classified with an ensemble of three ML models: Multinomial Naive Bayes, Logistic Regression, and Gradient Boosting. This categorizes applications as Safe, Risky, or Malicious. A risk scoring system, permission categorization, and compliance checks with GDPR, CCPA, and DPDP regula- tions further improve the system's clarity and practical use.

The tool includes a Flask-based web interface to provide an easy upload-and-analysis process, along with visualization dashboards for risk scores and tracker detection. Additionally, the system allows optional extensions such as dynamic analysis (using Frida and mitmproxy), threat intelligence integration (like the VirusTotal API), and blockchain-based report storage for auditability and integrity.

By combining lightweight static analysis, strong ML-driven classification, and privacy compliance checks, AndroidRiskCheck offers an effective, accessible, and adaptable solution for evalu- ating Android security.

Introduction

Android is the most used operating system for phones. There are over 3.5 million apps on the Google Play Store, and billions of phones use Android [[1]]. It's open, easy for developers to use, and has grown very quickly, changing the phone world by making things easy, convenient, and advanced for people everywhere.

But, being so open also causes security and privacy prob- lems. Bad apps can get into the store and trick people. These apps can steal your info, commit fraud, watch you, or put secret ways to get into your phone [[2]]. One common trick is asking for too many permissions. Seemingly safe apps might ask for access to your contacts, microphone, texts, or location, and then use that info for bad stuff [[3]]. Also, bad programmers often hide their bad code to make it harder to spot. New attacks that have never been seen before also make things complicated because they can get past normal security [[4]].

Current ways to spot bad apps aren't good enough. Normal antivirus programs check apps against a list of known bad ones. This works for known stuff but not for new or changed bad programs [[1]]. Other services use online databases, but those don't work if you're not online or if you want to change things for research or business use. Some tools watch apps while they run, which can spot things that normal checks miss, but these tools use lots of resources, are slow, and aren't good for checking many apps [[5]]. Some systems check app permissions and code and can tell if an app is bad, but these aren't strong, they don't work well with all files, and they don't explain things to normal users [[6], [7]]. Because of these problems, we need something that is easy to use and strong, can spot known and hidden bad apps, is easy to understand, and follows privacy rules [[8]].

To fix these problems, we have AndroidRiskCheck, which is a system that uses both code checking and machine learning to guess the risks in Android apps. It takes permissions and info from app files using a system that doesn't need Java, which makes it work on different systems, light, and easy to keep up. A clever system makes sure it can read all types of files, which helps it avoid tricks. The info is turned into organized pieces that are checked by a group of machine-learning models. These models tell you if apps are safe, risky, or bad and give a risk score to show how likely it is to be harmful. Using machine learning helps AndroidRiskCheck spot new and unknown attacks [[9], [10]].

The system also focuses on being easy to understand and following rules. A website lets people upload and check apps and shows the results in dashboards that show risk scores, permission use, and possible trackers. The system also checks if apps follow privacy rules like GDPR, CCPA, and India's DPDP Act by flagging apps that send data without security, ask for too many permissions, or don't have privacy policies [

[8]]. It's also made to be expanded with other tools, like live tracking and threat data [[5]].

AndroidRiskCheck is new because it combines a Python- only code checking system with a machine-learning group and also checks for privacy rule compliance and is easy to see. Instead of using only old methods or heavy live tracking, it gives a balanced way that is easy to grow for real-world uses. By dealing with both spotting bad apps and following privacy rules, AndroidRiskCheck is a good way to protect Android users from security dangers [[4], [8]].

Ease of Use

AndroidRiskCheck is great because it's simple and easy to use. Lots of malware detectors need you to set up complicated stuff like Java or Android SDKs, or they want you to use some outside online service [[1], [2]]. But this thing? It just uses Python. That means no big downloads, and it runs on Windows, Linux, and macOS without needing extra stuff. It's made so anyone—researchers, coders, students, or security people—can jump in and use it [[5]].

It has a simple web page made with Flask. You don't have to be a tech expert. Just load up your APK file. The thing looks at the code, grabs the OKs, uses some smart computer models, and gives you a risk score [[9], [10]]. Plus, it tells you why the app got marked as Safe, Risky, or Malicious. If the app has weird code that's hard to read, the system has a smart backup plan that uses some example OKs. So, it keeps working, and you don't have to do anything.

To help you see things clearly, there's a dashboard that turns all the techy stuff into charts and graphs. OKs are grouped, risk levels are shown in colors, and warnings are made easy to read. Even if you're not a security expert, you can understand what's going on [[8]]. This is really helpful for school projects, security checks at work, or when you need to explain things to people who aren't techy.

AndroidRiskCheck is all about being easy to set up, having clear results, and giving you a good experience. It takes the latest malware research and makes it something real people can use [[6], [7]]. It's not like those old research projects that are hard to use outside a lab. This is a tool that researchers can use for tests, coders can

use to check their apps, security people can use for inspections, and teachers can use to teach about mobile security.

Existing Methodology

The detection of privacy risks and harmful behaviors in Android applications has changed significantly over time. Various methods have emerged, each with its own advantages and challenges.

At first, signature-based detection was the most common method. This approach identifies known malware by using fin- gerprints, such as cryptographic hashes or specific sequences of API calls [[1]]. The main advantage of this method is its speed and simplicity, allowing for quick scanning and easy interpretation of flagged results. However, these systems can only recognize threats that have been identified before. New variants, obfuscated code, or repackaged malware often slip by undetected [[2]]. This limitation requires constant updates to the signature database, which can be costly and never-ending. Static analysis techniques, on the other hand, look at ap- plication code and metadata without executing it [[3]]. By examining elements such as app manifest files and requested permissions, static analysis can quickly spot apps asking for excessive or suspicious rights. More in-depth inspections involve scanning the code for sensitive API use and tracking potential data flows from private sources to risky endpoints, such as the internet. While static methods are scalable and useful for broad assessments, they can overestimate risks due to cautious assumptions and struggle with dynamic features like code loading or obfuscation. Advanced static methods create program graphs to represent control flow and data dependencies [[6]]. However, these graph-based analyses require significant computational resources and may miss actions within native libraries or reflective calls.

To address the limitations of static methods, dynamic analysis observes apps during actual execution. It monitors system calls, network activity, and user actions [[5]]. This live perspective uncovers behaviors that static analysis cannot predict, such as downloads that happen after installation or actions triggered after a set time. However, dynamic analysis needs substantial infrastructure and time to carry out, and its success relies on thorough user interface stimulation. Addi- tionally, sophisticated malware can detect emulators or encrypt communications, making dynamic inspection difficult.

Understanding that no single method is enough, researchers have created hybrid approaches. These combine static and dynamic data sources. By merging different perspectives like code features and runtime behavior, these systems achieve greater accuracy in detection while reducing false positives [4], [7]]. The downside is that these models become more complex, need longer processing times, and require more effort to maintain consistency among their components.

With the increase of machine learning (ML) and deep learning (DL), automated detection models have gained trac- tion. Traditional ML techniques depend on manually created features like permissions or API call frequencies, producing interpretable models that work well with limited data [[1]].

More advanced deep learning models use sequences of API calls, bytecode images, or graph neural networks that encode program structures [[9], [10]]. This allows for better gener- alization and captures complex relationships. However, these models often need large datasets, can be hard to interpret, and are susceptible to adversarial attacks. To overcome issues with data shortages and privacy, semi-supervised and federated learning approaches have been developed. These allow models to learn using unlabeled data or from distributed devices with- out sharing sensitive information. Although these approaches show promise, they bring challenges like communication dif- ficulties and the need to adapt to new threats.

A key focus now is explainability. Stakeholders want clear insights into why an app is marked as risky or malicious, rather than just binary labels [[8]]. Explainable systems offer global feature importance and local attributions and may even provide ”what-if” analyses to help developers reduce risks. However, creating stable and efficient explanations remains a challenge, especially for deep models that rely on complex program analyses.

Meanwhile, attackers keep improving their evasion tech- niques by using code obfuscation, packing, native code ex- ecution, environment checks, and adversarial inputs designed to bypass detection. Defenders respond with de-obfuscation, behavior-focused analysis that is less sensitive to syntactic changes, adversarial training, and hardware-backed sandbox- ing to trick emulator checks [[5]]. This ongoing back-and- forth enhances the arms race between malware developers and security researchers, often increasing detection delays and computational costs.

Finally, the quality of datasets and evaluation methods is crucial for valid research. Strong

studies use a variety of malware families and benign apps, remove duplicates, and test on newer samples to reflect real-world changes. Proper eval- uation metrics, including precision and recall within realistic false positive limits, are essential to avoid overly optimistic outcomes [[6], [7]]. Many prior studies suffer from issues like label noise, overfitting to public datasets, or problematic train-test splits, which limit their practical use.

In conclusion, static analysis provides scalability and speed, but it is vulnerable to evasion. Dynamic analysis gives richer behavioral insights, but it comes at a higher cost and has lim- ited coverage. Hybrid models that integrate data from multiple sources and use advanced learning techniques show the most promise but require considerable engineering resources [[4]]. Interpretability and robustness remain critical challenges [[8]]. As Android apps and malware continue to evolve, ongoing innovation in detection methods is vital for protecting user privacy and security.

Proposed Methodology

This research proposes creating a privacy risk analyzer tailored for Android applications (APKs). The main goal is to develop a robust system that effectively identifies privacy and security risks by combining multiple analysis techniques with machine learning [[4], [5]]. This approach aims to offer a practical and scalable solution for evaluating app safety in the complex world of Android.

Our method starts with static analysis, which examines the APK without running it. By using tools like Androguard, we extract important metadata, such as the app's requested permissions, manifest file details, and API calls related to sensitive data access (e.g., contacts, location, camera) [[3],
[2]]. Static analysis enables fast processing of a large number of applications and helps identify apps that request excessive or abnormal permissions [[1]].

Table I: Feature Summary Table

Feature Category	Feature Name	Description
Static Analysis	Permissions Requested	Permissions the app declares
	Sensitive API Calls	Use of APIs accessing private data
Dynamic Analysis	Network Traffic Anomalies	Unusual network connections
	File Access Patterns	Suspicious file system operations
Compliance Checks	Privacy Policy Presence	Whether app has privacy policy

However, static analysis alone cannot capture behaviors that only emerge during execution, especially if apps use dynamic code loading or obfuscation. To address this, our system includes an optional dynamic analysis module, which executes the app in an instrumented environment such as an emulator or real device [[5]].

Tools like Frida and mitmproxy are used to intercept and monitor behaviors during execution. These include observing network traffic, file system access, and interactions with the OS. Dynamic analysis reveals hidden or conditional behaviors and significantly improves threat detection [[4]].

Following the analysis, features extracted from both static and dynamic sources are input into machine learning models that classify apps into one of three risk categories: Safe, Risky, or Malicious [[6], [7]]. The following table outlines the evaluation metrics used to assess our classifier's performance:

Table 2: Evaluation Metrics for Classification Models

Metric	Description
Accuracy	Overall correctness of classification
Precision	Fraction of predicted positives that are true
Recall (Sensitivity)	Fraction of true positives detected
F1-Score	Harmonic mean of precision and recall
ROC-AUC	Area under the ROC curve

The models are trained on diverse datasets to improve generalization across different malware families [[1], [4]]. The classification function is formally defined by:

$$R = f(S, D) \quad (1)$$

$$R = w_1 \cdot S_f + w_2 \cdot D_f + w_3 \cdot C_f \quad (2)$$

where R is the final risk score, S_f, D_f, and C_f are feature vectors from static, dynamic, and compliance checks, respec- tively, and w_i are learned weights for each component.

In addition to technical threat classification, our system eval- uates compliance with major privacy regulations like GDPR,

Table 3: Regulatory Compliance Checklist

Regulation	Check Description	Status	Comments
GDPR	Consent for data use	Pass	Privacy policy mentions consent
CCPA	Right to delete user data	Fail	No deletion option in app
PDPA	Secure data transmission	Pass	All network calls are encrypted

CCPA, and PDPA [[8]]. The checklist below shows the criteria used:

To enhance the analyzer's capabilities, we integrate real- time threat intelligence from platforms like VirusTotal and Exodus. These services provide up-to-date feeds on known malware signatures, suspicious domains, and trackers [[5]].

The results from all modules are compiled and visual- ized through a user-friendly dashboard using frameworks like Streamlit or Plotly. This dashboard presents risk scores, high- lights permissions and trackers, and provides interpretability tools to explain why an app was flagged [[8]].

From a system design standpoint, the backend is modular and scalable. We utilize frameworks like Flask or Django to expose RESTful APIs for APK submission and result retrieval. This structure supports large-scale batch processing and can be easily integrated into enterprise environments [[7]].

Lastly, to ensure auditability and data integrity, we incor- porate an optional blockchain-based layer that stores cryp- tographic hashes of analysis reports. This immutable ledger guarantees that once a report is issued, it cannot be altered, which is particularly useful in compliance audits and forensic investigations [[5]].

In summary, this proposed methodology integrates the scal- ability of static analysis, the depth of dynamic monitoring, the intelligence of machine learning, and the rigor of legal compliance into a unified, trustworthy framework for Android privacy risk assessment [[4], [9]]. communications, system calls, and sensor utilization can be tracked under controlled conditions to supplement static in- spection [[5]]. Merging static and dynamic evidence in a less disjointed way would yield greater strength against obfuscated malware that conceals its actual behavior until runtime [[4]]. Another promising direction is the use of sophisticated deep learning models, such as graph neural networks (GNNs) and transformer models, which can effectively capture intricate se- mantic dependencies between APIs, permissions, and control flows [[9], [10]]. These models have the potential to identify subtle patterns that are overlooked by conventional machine learning classifiers while still being adaptable to changing malware. But there should also be a focus on enhancing explainability so outputs do not become unintelligible for developers, researchers, and security auditors [[8]].

The framework can also be augmented with real-time threat intelligence integration, which will allow the tool to fetch updated malware indicators from global databases, enhancing detection precision against newer threats [[5]]. Concurrently, the inclusion of blockchain-based report storage would provide integrity and tamper resistance of security reports, making the framework more trustworthy for legal, compliance, and enterprise audit purposes [[7]].

From a usability point of view, future enhancements could involve mobile or cloud-native deployments to allow for risk analysis on mobile devices or as a service offered to organiza- tions for large-scale automated scans [[6]]. In addition, adding support for multi-platform ecosystems in addition to Android, e.g., iOS or IoT applications, would expand the reach of the system in securing various ecosystems.

Lastly, considering the growing focus on privacy, broaden- ing the compliance framework to include future regulations beyond GDPR, CCPA, and DPDP, while incorporating au- tomated analysis of privacy policies using natural language processing, would offer a complete picture of technical and legal threats [[8]]. These updates would not only enhance detection functions but also establish AndroidRiskCheck as a complete and future-proof mobile security and privacy guardian solution.

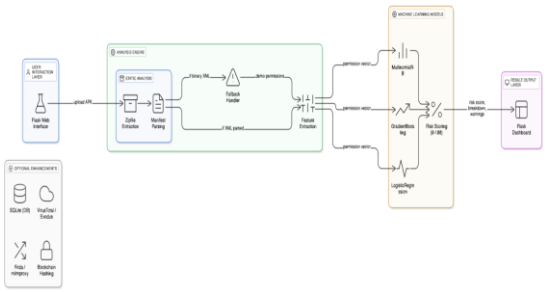

Fig. 1: Proposed system architecture of AndroidRiskCheck showing APK upload, static analysis, ML models, and result visualization.

Future Scope

The construction of AndroidRiskCheck sets a solid ground for smart and explainable Android application risk analysis, but there is still tremendous potential for future work. One key direction involves the integration of dynamic analysis at scale, whereby runtime behavior including automated network

Conclusion

The rapid growth of the Android ecosystem has created an environment where millions of applications are available to users. However, this openness also exposes them to significant security and privacy risks. Traditional detection methods, such as signature-based antivirus tools and static rule-based analy- sis, have shown they are not enough to handle zero-day threats, obfuscation techniques, and subtle misuse of permissions [[1],
[2]]. This issue highlights the urgent need for smart, flexible, and clear detection mechanisms [[4]].

In this work, we introduced AndroidRiskCheck, a hybrid framework that combines static analysis with machine learning to assess the risk levels of Android applications [[7], [5]]. The system includes a Python-only pipeline for APK inspection, which removes the need for external tools such as Java. It also uses smart XML parsing with fallback options to ensure reliable extraction of permissions, even from binary manifest files [[3]]. The extracted features are turned into vectors and analyzed using a mix of machine learning models, including Multinomial Naive Bayes, Logistic Regression, and Gradient Boosting [[6]]. This approach classifies applications as Safe, Risky, or Malicious, providing a more effective detection capability compared to traditional static or signature-based methods [[9]].

In addition to detection, AndroidRiskCheck integrates com- pliance checks for privacy regulations such as GDPR, CCPA, and DPDP, making it more consistent with modern data protection needs [[8]]. The system also includes a Flask-based web interface and interactive visualization dashboard, ensuring ease of use and accessibility for researchers, developers, and auditors [[10]]. Optional features, like dynamic behavior mon- itoring, threat intelligence integration, and blockchain-based report integrity, further enhance the framework, making it suitable for both academic research and real-world application [[5], [7]].

Overall, this solution shows that a well-designed combina- tion of static analysis and machine learning can significantly improve malware detection and privacy risk assessment in Android applications [[1], [4]]. By balancing precision, us- ability, and clarity, AndroidRiskCheck offers a step forward in creating trustworthy and user-friendly mobile security tools. With future improvements such as deep learning models, support for multiple platforms, and real-time cloud integration, the system could evolve into a strong framework for securing the mobile ecosystem against ever-changing threats [[9], [10]].

References

D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, and K. Rieck, "Drebin: Effective and explainable detection of android malware in your pocket," in *Network and Distributed System Security Symposium (NDSS)*, 2014.

Y. Aafer, W. Du, and H. Yin, "Droidapiminer: Mining api-level features for robust malware detection in android," *Proceedings of the International Conference on Security and Privacy in Communication Systems*, pp. 86–103, 2013.

Y. Feng, S. Anand, I. Dillig, and A. Aiken, "Apposcopy: Semantics- based detection of android malware through static analysis," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2014, pp. 576–589.

E. Mariconti, L. Onwuzurike, P. Andriotis, E. De Cristofaro, G. Ross, and G. Stringhini, "Mamadroid: Detecting android malware by building markov chains of behavioral models," in *Network and Distributed System Security Symposium (NDSS)*, 2017.

Y. Wu, J. Shi, P. Wang, D. Zeng, and C. Sun, "Deepcatra: Learning flow- and graph-based behaviors for android malware detection," *arXiv preprint arXiv:2201.12876*, 2022.

Sunkara, S. P. (2025). Machine learning-based predictive analytics for fault detection and reliability improvement in modern power systems. *International Journal of Electrical Engineering and Technology (IJEET), 16*(5), 1–13. https://doi.org/10.34218/IJEET_16_05_001

Y. Liu, G. Li, and Z. Jin, "Call graph based android malware detection with convolutional neural network," *Communications in Computer and Information Science (Springer)*, vol. 861, pp. 3–14, 2019.

J. Liu, P. Zhao, W. Li, and Y. Wang, "Gdroid: Android malware detection and classification with graph convolutional networks," *Computers & Security*, vol. 106, p. 102264, 2021.

M. C. Ipek and S. Sen, “Xai-droid: Explainable android malware detection and malicious code localization using graph attention,” *arXiv preprint arXiv:2503.07109*, 2025.

J. Gu, H. Zhu, Z. Han, X. Li, and J. Zhao, “Gsedroid: Gnn-based android malware detection framework using lightweight semantic embedding,” *Computers & Security*, vol. 140, p. 103807, 2024.

D. Zhang, X. Wu, E. He, X. Guo, X. Yang, R. Li, and H. Li, “Android malware detection based on hypergraph neural networks,” *Applied Sciences*, vol. 13, no. 23, p. 12629, 2023.