Enhancing Endpoint Security in Android Devices Using Machine Learning

Rushikesh Jagtap[1], Raviraj Jagtap[2], Anurag Tapkir[3], Tejas Gholap[4], Siddhi Waghchoude[5], Jai Girawale[6]
[1,2,3,4] *Student, Department of Design Analytics and Cyber Security, MIT ACSC, Alandi, Pune*
[5,6] *Faculty, Department of Design Analytics and Cyber Security, MIT ACSC, Alandi, Pune*

***Abstract*—The Android operating system commands the largest share of the global mobile device market, making it a prime target for an ever-growing variety of malware and cyber security threats. Traditional endpoint protection methods predominantly rely on signature-based detection, which are limited in identifying zero-day attacks, polymorphic malware, and advanced persistent threats. This research proposes a hybrid machine learning (ML) based approach for enhancing endpoint security on Android devices by combining static and dynamic analysis of application behavior. The framework is designed to be lightweight, enabling efficient on-device execution while preserving user privacy. Our approach demonstrates the feasibility and benefits of integrating ML techniques in mobile endpoint protection, offering improved detection rates and adaptability to evolving threat landscapes.**

***Index Terms*—Android Security, Endpoint Protection, Hybrid Machine Learning, Malware Detection, Privacy Preservation**

I. INTRODUCTION

Android smartphones are everywhere, deeply part of people's daily lives, both at home and work.
They help with money transactions, communication, health tracking, and business tasks, which means they hold a lot of personal and important information. Because of how common they are and how much data they store, they are a big target for hackers who want to steal, spy, or mess things up.
Older security tools for phones rely on signature-based detection.
These tools look for known bad software by checking against a list of known harmful patterns. They work well for things we already know are bad, but they can’t catch new or hidden threats, like the ones that haven’t been seen before. Plus, these lists need to be updated often, which is hard to do in real time on phones that might not always be connected to the internet.
Machine learning offers a better way to fight these threats.
Unlike old methods, machine learning can learn what normal behavior looks like and spot anything that doesn't fit. It can identify bad apps even if we haven't seen them before. By studying a lot of examples, ML models can catch more subtle signs of danger that static methods miss.
This research is about creating a new system that uses machine learning for Android phone security.
It combines both the way apps look (static features) and how they behave (dynamic features) to detect malware more effectively. We also deal with challenges like the limited power of phones, the need to keep user information private, and tricks that hackers use to avoid being caught. Our system is designed to work directly on the phone, so it doesn’t rely too much on the internet, and it still keeps accuracy and privacy high.
The rest of this paper is organized like this: Section 2 talks about what we already know and what others have done, Section 3 explains how we're doing it, Section 4 discusses how to stop hackers from fooling our system, Section 5 covers real-world issues and problems we might face, and Section 6 shares ideas for future work.

II. LITERATURE REVIEW

• Static Analysis: Examines permissions, APIs, and manifest files; efficient but vulnerable to obfuscation and unable to capture runtime behavior.
• Dynamic Analysis: Observes execution behavior such as system calls and network activity; robust but resource-heavy.
• Hybrid Approaches: Combine static and dynamic

indicators, offering higher accuracy by mitigating weaknesses of each method.
• Adversarial ML Threats: Attackers craft evasion samples; mitigated using adversarial training and anomaly detection.
• Mobile Constraints: Android devices require lightweight and power-efficient models.
• Research Gap: Lack of hybrid ML solutions that are privacy-preserving, adversarial-resistant, and practical for real-time endpoint protection.

III. RESEARCH QUESTIONS AND HYPOTHESES

Research Questions:
1. How can hybrid ML models improve Android malware detection compared to traditional methods?
2. Which static and dynamic features are most effective for lightweight, real-time detection?
3. How can blocking and quarantine mechanisms enhance endpoint security without harming usability?
4. What strategies can preserve privacy and resist adversarial attacks in ML-based systems?

Hypotheses:
H1: Hybrid ML approaches achieve higher detection accuracy than purely static or dynamic methods.
H2: Lightweight on-device models enable real-time malware detection with minimal resource overhead.
H3: Blocking and quarantine mechanisms strengthen endpoint protection by reducing user risk.

IV. METHODOLOGY

A. Data Collection and Feature Extraction

The proposed system collects a diverse range of features from Android applications, combining static properties of the app with dynamic behaviour observed during execution.
Static Indicators:

- Permissions: Identifying sensitive system permissions requested by the application (e.g., SMS or location access) provides early insight into potential misuse.
- API Calls: Examining invoked APIs helps detect risky actions such as file manipulation or unauthorized communication.
- Intent Filters: Declared intents are analyzed to understand interaction pathways between the app, system services, and other applications.
- Code Metrics: Attributes like program size, control flow, and complexity provide additional static context regarding the app's structure.

Dynamic Indicators:

- System Calls: Monitoring frequency and sequencing of system calls during runtime highlights suspicious operational patterns.
- Network Activity: Analysis of data transmission may reveal hidden connections to remote servers or command-and-control channels.
- Resource Consumption: Abnormal CPU and memory usage can indicate malicious payload activation.

Sensor Usage: Unexpected access to hardware sensors (e.g., camera or microphone) signals possible privacy intrusion

B. Data Preprocessing

Raw datasets often contain unnecessary or redundant information that may lower model accuracy and increase computation costs. To address this, the framework applies:

- Normalization: Standardizing feature scales to stabilize learning.
- Feature Selection: Employing measures such as Mutual Information or Information Gain to retain the most informative attributes.
- Dimensionality Reduction: Applying Principal Component Analysis (PCA) to condense the feature space while maintaining variance.

These steps ensure the final dataset is both compact and suitable for efficient, real-time inference on mobile devices.

C. ML Framework Architecture

The architecture consists of three major modules:

- Feature Extractor: Runs in the background, collecting both static descriptors and runtime features without causing noticeable disruption to the user.
- Inference Engine: Deploys optimized ML models directly on the device, providing near real-time classification with minimal CPU and memory overhead.

- Alert & Response Unit: Issues warnings when suspicious activity is identified and triggers automated countermeasures such as blocking or isolating malicious apps.

D. Blocking and Quarantine

To contain threats after detection, the framework provides two response strategies:

- Blocking: Immediately restricts harmful apps by revoking sensitive permissions and disabling dangerous components.
- Quarantine: Moves the suspicious application into a controlled state where it cannot access core resources, while still being preserved for user or administrator review.

These mechanisms are adaptable, allowing customization based on user preferences or organizational policies, thus balancing strong protection with usability.

E. Privacy-Preserving Design

The framework is designed with privacy as a priority. All security analysis is performed locally on the device, minimizing data transfer. Information collected is anonymized and limited strictly to behavioral features required for detection.

Future extensions will explore federated learning, enabling decentralized model training across multiple devices without sharing raw user data, thereby improving both privacy and scalability.

V. DEFENSE AGAINST ADVERSARIAL ATTACKS

A. Challenges to ML-Driven Security on Android

Malicious actors often engineer apps to subtly modify characteristics so as to evade detection. These adversarial manipulations exploit weaknesses in the model's decision-making boundaries. In the context of Android security, attackers tamper features while retaining functional behavior, making traditional classifiers vulnerable.

B. Integrating Adversarially Hardened Training

To counter adversarial strategies, this framework expands its training dataset to include crafted examples that mimic evasion tactics. By training on both regular and adversarially perturbed samples, the model learns to distinguish malicious variants more reliably—even when attackers attempt to disguise them.

C. Runtime Statistical Sentinel Layer

Beyond training-time defenses, we include a runtime monitoring layer that tracks statistical shifts in feature distributions. Should an app's behavioral patterns deviate beyond expected thresholds, the system generates an alert—providing a secondary line of defense that helps detect novel evasion tries in real time.

VI. BLOCKING AND QUARANTINE INTEGRATION

A. Purpose

Malware detection alone is insufficient; security systems must also prevent further impact. Blocking and quarantine provide containment while allowing investigation and user control.

B. Blocking Mechanism

When an app is identified as malicious:

- Android APIs (e.g., PackageManager, DevicePolicyManager) are used to disable execution.
- Sensitive permissions are revoked.
- Optional network filtering is applied through a local firewall.
- The user is alerted and can choose to remove the app or move it into quarantine.

This prevents access to system resources without heavily disrupting normal device use.

C. Quarantine Design

If deletion is not desirable, the app is placed in a restricted state:

- It cannot be launched.
- Access to storage and personal data is suspended.
- Network communication is fully blocked.
- Any attempt to bypass restrictions is logged.
- The app remains on the system but inactive until reviewed by the user or administrator.

This ensures safe containment without risking data or device compromise.

D. Implementation Aspects

- Metadata (package name, ID, timestamp, detection reason) is recorded securely and stored in encrypted form.

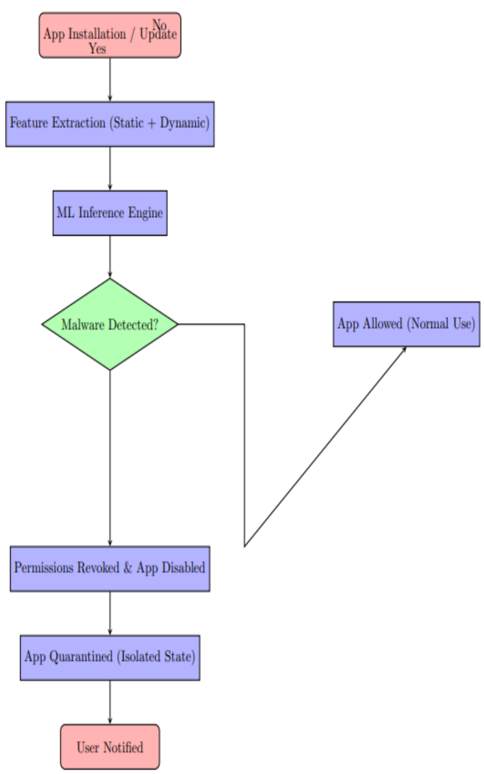
- Logs of quarantined applications are protected against tampering.
- In enterprise contexts, only authorized administrators can remove or restore quarantined apps.
- Users receive periodic notifications about quarantined items with options to act based on updated results.

E. Workflow

The system's automated response follows these steps:

1. App installation or update.
2. Feature extraction.
3. ML inference.
4. If malware is detected: permissions are revoked, the app is disabled, it is added to quarantine, and the user is notified.

This ensures rapid containment without constant user oversight.



F. Enterprise and Research Use Cases

Quarantine is particularly beneficial in:

- BYOD workplaces, where immediate removal may not align with company policy.
- Research environments, where malware must be analyzed safely.
- Incident response, where suspicious apps must be preserved for forensic study or legal evidence.

VII. EXPECTED OUTCOMES

- A fully functional lightweight hybrid machine learning framework that integrates both static and dynamic features for Android malware detection.
- Demonstrated improvement in detection accuracy, precision, and recall compared to existing signature-based and single-method approaches.
- A proof-of-concept Android prototype capable of real-time, on-device inference with minimal performance overhead.
- Effective integration of blocking and quarantine mechanisms, ensuring automatic containment of malicious applications with minimal user intervention.
- Enhanced robustness of the system against adversarial machine learning attacks, using adversarial training and anomaly detection strategies.
- Implementation of a privacy-preserving security design, leveraging federated learning and local inference to protect user data.
- Contribution of a scalable model that can be extended to enterprise mobile device management (MDM) systems for large-scale deployment.

Advancement of research knowledge in mobile cybersecurity by filling the gap in lightweight, privacy-conscious, and adversarial-resistant frameworks.

VIII. CONCLUSION

This work proposes a hybrid machine learning framework that strengthens Android endpoint security by integrating static and dynamic analysis with real-time containment strategies. By addressing privacy, efficiency, and adversarial robustness, the system advances mobile cybersecurity research and offers practical pathways for enterprise deployment.

REFERENCES

[1] B. Alzahrani et al., "Deep Learning-Based Android Malware Detection Framework," *MDPI Electronics*, 2023.

[2] M. Bouali et al., “Lightweight Android Malware Detection Using Deep Neural Networks,” *Expert Systems with Applications*, Elsevier, 2024.
[3] M. Garba et al., “Machine Learning-Based Intrusion Detection for Android Devices,” *arXiv:2412.03894*, 2024.
[4] S. Qiu et al., “Defending Against Adversarial Malware Attacks,” *arXiv:2501.13782*, 2025.
[5] CICMalDroid2020 Dataset – Canadian Institute for Cybersecurity, UNB.
[6] S. Nair, A. Thomas, “Mobile Malware Detection Using Static Analysis and ML,” *ETASR*, 2024.
[7] Y. Zhou and X. Jiang, “Dissecting Android Malware: Characterization and Evolution,” *IEEE Symposium on Security and Privacy (SP)*, pp. 95–109, 2012.
[8] M. Arp et al., “Drebin: Effective and Explainable Detection of Android Malware in Your Pocket,” *NDSS Symposium*, 2014.
[9] K. Tam, A. Feizollah, N. B. Anuar, R. Salleh, and L. Cavallaro, “The Evolution of Android Malware and Android Analysis Techniques,” *ACM Computing Surveys (CSUR)*, vol. 49, no. 4, 2017.
[10] F. Idrees, M. Rajarajan, “Investigating Machine Learning Algorithms for Android Malware Detection,” *2017 IEEE Trustcom/BigDataSE/ICESS*, pp. 1–7.
[11] S. Kim et al., “Adversarial Attacks and Defenses for Malware Detection Models,” *ACM Computing Surveys*, 2022.
[12] A. Z. M. Shahriar and M. A. Hossain, “Federated Learning for Privacy-Preserving Mobile Malware Detection,” *IEEE Access*, vol. 9, pp. 12345–12357, 2021.
[13] J. Sahs and L. Khan, “A Machine Learning Approach to Android Malware Detection,” *2012 European Intelligence and Security Informatics Conference*, IEEE, pp. 141–147.