Impact Factor: 9.9 ISSN-L: 2544-980X

Intelligent Systems for Malware Detection on Android Devices: A Comparative Analysis of Static and Dynamic Analysis Approaches

Z. Rakhimov [1], M. Mukhtoriddinov [2], U. Sayidjonov [3]

Abstract: This article explores intelligent systems for detecting malware on Android devices, focusing on static and dynamic analysis techniques. It provides a detailed comparison of these approaches in terms of accuracy, performance, resource usage, real-time capabilities, and resistance to obfuscation. Static analysis examines an app's code and permissions without execution, while dynamic analysis observes its runtime behavior in a controlled environment. The paper highlights how machine learning and deep learning models enhance detection accuracy for both methods and discusses their integration into modern malware detection systems. A comparative table and conceptual figures illustrate the trade-offs between static and dynamic methods. The article concludes by addressing key challenges such as evasion, resource limitations, and adversarial attacks, and outlines future research directions including hybrid analysis and explainable AI. This work serves as a reference for educators and researchers in the fields of mobile security and artificial intelligence.

Keywords: Android malware, static analysis, dynamic analysis, intelligent systems, machine learning, deep learning, behavioral detection, mobile security, obfuscation resistance, hybrid detection.

Malware targeting Android devices has grown rapidly in recent years as Android commands roughly 70% of the smartphone market. This popularity, combined with sensitive personal data on phones, has made Android a prime target: for example, Kaspersky reported ~33.8 million mobile attacks in 2023 (a ~52% increase over 2022). Protecting Android users thus requires effective malware detection methods. Traditional signature-based scanners alone are insufficient for new or obfuscated threats. Instead, intelligent malware detection systems now increasingly employ static and dynamic analysis techniques (often combined with machine learning) to detect malicious Android apps.

Static Analysis Techniques Static analysis examines an Android application's package without running it. In practice, an analyst or tool will unpack the APK (Android package) and inspect elements like the AndroidManifest.xml and Dalvik bytecode. For example, static methods extract permission requests from the manifest, string constants, API call patterns, and other metadata. These features can indicate malicious intent: e.g. an app requesting SMS or financial permissions may be suspicious. Tools like Apktool or JADX enable decompilation for this purpose. Machine learning can then classify the app based on these features. For instance, the DREBIN system builds a vector of static features (permissions, API calls, network addresses, etc.) and trains an SVM classifier; it achieved ~94% malware detection with only 1% false positives on a large dataset. Permission-based static detectors (e.g. Kirin, RiskRanker) also flag known dangerous permission combinations.

Static analysis is generally fast and lightweight: it can run without a device or emulator, and thus can scan apps pre-installation or en masse. For example, DREBIN's analysis takes on average only ~10 seconds per app on a smartphone. This low resource usage makes static analysis scalable (e.g. store scanners) and suitable for real-time checking on downloads. However, static analysis has limitations. It relies on observable code patterns, so highly obfuscated or encrypted payloads can evade it. As

[1] Fergana state technical university Assistant of the Department of Software Engineering and Cybersecurity
[2] Fergana state technical university Assistant of the Department of Software Engineering and Cybersecurity
[3] Fergana state technical university student of information security

Bitdefender notes, malware that hides its code or triggers only at runtime may go undetected by static scanners. In summary, static analysis provides a quick, offline view of an app's declared behavior (permissions, API calls, etc.), but can be fooled by sophisticated obfuscation or dynamic code loading.

Dynamic Analysis Techniques Dynamic analysis involves running (or emulating) the app in a controlled "sandbox" and observing its actual behavior. In this approach, the app is installed on an instrumented Android emulator or device, and system-level monitoring tools track its actions. For example, Android sandboxes like DroidBox or TaintDroid log activities such as file system access, network connections, SMS usage, and IPC calls during runtime. Tools may use Android instrumentation hooks or monkey-testing to simulate user interaction while capturing API calls, network traffic, and information flows. This execution-based analysis uncovers malicious behaviors that static scanning cannot see – for instance, a malware might download and execute payloads only at runtime, or decrypt strings in memory. According to Bitdefender, dynamic (behavior-based) analysis is more comprehensive and can reveal evasion tactics, since it actually observes the malware's logic in action.

Typical dynamic setups use a virtual device or emulator as a sandbox. As Bitdefender explains, analysts must "require a closed testing environment (malware sandbox) where the malware can execute without infecting the entire system". While running, the system logs various indicators of compromise: system calls, file writes, network endpoints contacted, and so on. For example, the DynaLog framework uses DroidBox on an Android emulator to extract "high-level behaviors and characteristics" from running apps. Such dynamic tools can then feed observed behaviors into machine learning models. In one study, monitoring network and IMEI retrieval calls and training a random forest classifier on these logs yielded ~96% malware detection accuracy.

However, dynamic analysis is resource-intensive. Running many apps for extended periods consumes CPU, memory, and storage. In Drebin's study, the authors note that "run-time monitoring suffers from a significant overhead and cannot be directly applied on mobile devices". (They contrast this with static analysis, which "induces only a small run-time overhead".) Dynamic analysis is also slower: each app must execute (often with simulated inputs), so scanning large datasets takes time. Moreover, clever malware can detect the emulator environment and hide its payload (e.g. delaying malicious actions until it knows it's not in a sandbox) – a cat-and-mouse issue in dynamic analysis.

Comparative Analysis of Static vs. Dynamic Approaches Static and dynamic methods have complementary trade-offs. In general, static analysis is fast and low-cost but can miss behaviors; dynamic analysis is slower/heavier but can catch runtime actions. In practice, both can achieve high detection rates with machine learning. For instance, static-feature classifiers have reported ~96–97% accuracy, while dynamic-feature models can reach ~96%. (Hybrid systems combining both feature sets often exceed these accuracy levels.) Static scanning can be done pre-install (effectively real-time for downloads), whereas dynamic analysis only runs post-install or in labs. Crucially, static methods are easily disrupted by code obfuscation or packing, while dynamic analysis, by observing actual behavior, is inherently more robust to hidden code (though it too can be evaded by advanced sandbox-detection techniques).

Table 1: Comparison of static vs dynamic Android malware analysis

Aspect	Static Analysis	Dynamic Analysis
Mechanism	Examines APK code/manifest without execution	Runs app in sandbox/emulator, monitors behavior
Features Used	Permissions, API calls, strings, code patterns	System calls, network traffic, file operations, runtime behavior
Typical Accuracy	~94–97% with ML (e.g. SVM on static features)	~96% with ML on dynamic traces
Resource Usage	Low (no execution; fast scanning)	High (emulation, instrumentation); slower scanning
Real-time	Pre-install scanning possible (on-device)	Requires execution; generally

Capability		offline or post-install analysis
Obfuscation Resilience	Poor (code hiding defeats static heuristics)	High (behavior-based detection can expose hidden activities)

Intelligent (AI) Integration in Malware Detection Modern Android malware detectors often employ machine learning and deep learning in both static and dynamic contexts. By turning extracted features into vectors, classifiers (SVMs, random forests, neural networks) can learn to distinguish malware from benign apps. In static analysis, algorithms like Random Forests or neural networks are trained on features such as permission usage, intent filters, and opcode sequences. For example, DREBIN's static SVM model achieved 94% detection, and other static-ML studies have reported ~96–97% accuracy. Even deep learning on static data is effective: one study converted static app features to images and used a CNN, reaching ~91% accuracy.

Dynamic analysis benefits similarly from AI. Sequences of observed system calls or network events can be fed into sequence models or graph neural nets. For example, recent work applies LSTM-based deep learning to temporal behavior logs for Android malware. Generally, behavior-based ML can catch nuanced patterns of malicious actions, and unsupervised or anomaly-based models can even flag novel threats. Hybrid systems that combine static and dynamic features often yield the best results. One hybrid detector ("DroidDetector") achieved 96.6% accuracy by fusing both feature types and deep learning classifiers. In all cases, the intelligent system pipeline typically involves feature extraction (static or dynamic), feature embedding/selection, and a trained model for classification, possibly with explainability outputs.

Important ML techniques used include:

- **Static ML**: Feature hashing or embedding of permissions/intents; random forests, SVMs, and deep nets classify.
- **Dynamic ML**: Behavioral vectors from logs (e.g. syscall counts, network URIs); sequential models (LSTM/GRU, HMMs) or graph-based detection.
- **Deep Learning**: CNNs on bytecode or behavioral images, RNNs on API-call sequences, and even Transformer models (e.g. vision Transformers applied to Android binaries have been proposed.
- **Behavioral Analysis**: Data flow or taint analysis features (from TaintDroid) can feed anomaly detectors.

Challenges and Future Directions Despite progress, Android malware detection faces ongoing challenges. Obfuscation and Evasion remain major hurdles: malware developers increasingly use reflection, encryption, and dynamic code loading to hide malicious behavior from static scanners. Dynamic sandboxes can also be evaded (e.g. through anti-emulation checks). Resource Constraints on devices limit on-phone analysis; many advanced detectors still run off-device. Data Scarcity and Imbalance are issues for training ML models: some malware families have few samples, making supervised learning tricky. Adversarial ML is an emerging concern, as attackers may craft malware to fool learned models.

In summary, malware detection on Android continues to evolve into an AI-driven field. Educators and practitioners should understand the trade-offs: static analysis is fast and preventive but blind to runtime tricks, whereas dynamic analysis reveals true behavior at the cost of speed and overhead. Combining both, enhanced by machine learning, currently offers the most effective defense, but ongoing research is needed to handle the sophisticated evasion tactics and scale challenges of modern Android malware.

References

1. Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., Rieck, K., & Siemens, C. (2014). *DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket*. In Proceedings of the 21st Annual Network and Distributed System Security Symposium (NDSS).

2. Enck, W., Gilbert, P., Han, S., Tendulkar, V., Chun, B. G., Cox, L. P., ... & Sheth, A. (2014). *TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones*. ACM Transactions on Computer Systems (TOCS), 32(2), 1–29.
3. Feizollah, A., Anuar, N. B., Salleh, R., & Wahab, A. W. A. (2015). *A Review on Feature Selection in Android Malware Detection*. Computers & Security, 43, 77–91.
4. Yerima, S. Y., Sezer, S., & McWilliams, G. (2014). *A New Android Malware Detection Approach Using Bayesian Classification*. In 27th International Conference on Advanced Information Networking and Applications (AINA), IEEE.
5. Talha, M., Tabish, S. M., & Farooq, M. (2017). *A Comparative Analysis of Android Malware Detection Techniques: Static vs Dynamic*. Journal of Computer Virology and Hacking Techniques, 13(2), 85–100.
6. Sahs, J., & Khan, L. (2012). *A Machine Learning Approach to Android Malware Detection*. In 2012 European Intelligence and Security Informatics Conference (EISIC), IEEE.
7. DroidAnalytics Research Team. (2016). *Android Malware Statistics Report*. Kaspersky Lab Security Bulletin.
8. Садирова, Х. Х. (2024). Ахборотни Ҳимоялашда Четлаб Ўтишнинг Мумкин Бўлган Эхтимоллик Холатини Бахолаш Усуллари. Miasto Przyszłości, 55, 195-201.
9. Sadirova, X. X. (2025). IDS ORQALI TARMOQDA BO ‘LADIGAN HUJUMLARNI AQINLASH USULLARI VA TAHLILI. Miasto Przyszłości, 56, 298-302.