Android Malware Detection From APK File-LITERATURE SURVEY

Sandhya S, Preethi CS, Preethi JL, Dhanushree HT

1. Android Malware Detection Using Static Analysis of APK Files

1.1 Abstract

One of the most common Android malware detection methods is the use of the static analysis method since it analyzes the internal architecture of an APK file without running it. This is done by scanning permissions, manifest attributes, code instruction, and API invocation to determine that there are evil patterns within the application. Machine learning models can effectively identify benign and malicious apps by transforming extracted features into numbers. Compared to dynamic execution, static analysis is safer and faster hence can be used in large scales to screen malware. Its accuracy however decreases in the presence of highly obfuscated code, encrypted payloads or malware that only triggers malicious behaviour once it is in execution.

1.2 Approach

1. Manifest and authorization Analysis : AndroidManifest.xml is uprooted to identify the use of warrants, exported factors and intent pollutants. gratuitous or suspicious high- threat authorization is a sign of ill intent.

2. Bytecode and API Call Examination : Decompiling the classes.dex train and examining smali law shows that API calls are being made to SMS transferring, data theft, background services, and network communication functionality that are employed by malware.

3. Opcode Sequence Extraction : Opcode Sequence birth Dalvik opcodes are restated to sequences or n-grams. These malware family groups are grounded on instruction- position geste using these opcode patterns.

4. Generation of point vectors authorization, frequence of API, opcode patterns, structural features are formed into a point vectors that's the input to machine literacy algorithms.

5. Machine Learning Bracket SVM or Random Forest, or Neural Networks are exemplifications of classifiers, which are trained so that they can distinguish between vicious and benign apps, using uprooted static features.

1.3 Contributions

1. Speedy and huge : Be it scanning through millions of apps on a play store or checking a full company network it can get an entire job done within minutes-it doesn't have to be put on the bus.

2. No reason to actually open the app : Scans the virus without opening or launching it. Zero risk, straight-up safe.

3. Peeks right into the code : Displays the precise situation of what is happening under the hood in the code, the images, settings, all crystal clear, nothing remains hidden.

4. Old-style malware families can be detected immediately : Viruses that continue to re-appear with a new disguise (repackaged viruses) are caught the instant their mechanisms are detected (when they fit into an existing pattern).

5. Very light on phone/laptop : Operates so efficiently that even basic antivirus software or an in-house scanner can scan it and not run it like the engine - no battery loss, no drama.

1.4 Limitations

1. Obfuscation Vulnerability Malware written in sophisticated code obfuscation or packing can become hard to analyze in a static form.

2. Absent Visibility of runtime geste : It's possible to avoid malware that does not produce vicious geste until it's executed.

3. Issues in Dynamic Code Loading : During static examination, it is not possible to descry which law is downloaded or deciphered on demand.

4. Risk of High False Positive Risk: Because of permissions or manifest features only, benign applications are likely to get misclassified.

5. Poor Detection of Zero-day Attacks: Dynamic signature and feature sets can not work with new and obscure malware variants.

2. Android Malware Detection Using Dynamic Analysis

2.1 Abstract

Dynamic analysis detects Android malware by executing an operation in a controlled sandbox terrain and observing its real-time geste. This includes covering system calls, train operations, network

communication, and sensitive data access. Unlike static analysis, dynamic ways can descry blurred or translated malware since vicious law must ultimately execute. Behavioural logs and runtime traces are anatomized to identify abnormal patterns reflective of vicious exertion. Although dynamic analysis offers deeper behavioural sapience, it requires further computational coffers and may fail if malware uses anti-analysis ways.

2.2 Approach

1. Sandbox Activity : APK is injected on such tools as CuckooDroid or DroidBox to trace the behavior of running without any harm to a real device.

2. System Call Monitoring : System There are system level operations like process creation, file writes, and network connections that can be monitored to identify unauthorised or suspicious activity.

3. Network Traffic Inspection : Traffic in and out of the networks is inspected to understand traffic with malicious servers or attempts of data exfiltration.

4. Taint and Information Flow Tracking : TaintDroid, etc. Trackers such as TaintDroid trace the flow of sensitive data within the application, and find privacy violations.

5. Behavioural Sequence Analysis : Malicious actions is classified by ML models grounded on runtime API call patterns, event sequences, resource operation.

2.3 Contributions

1. Resistant to Obfuscation: It doesn't matter how much the malware tries to hide or disguise itself while it's running – the sandbox still catches what it's really doing the moment it launches.

2. Reveals Hidden Payloads: Spots sneaky payloads that only wake up under certain conditions (like when it thinks it's on a real victim's machine instead of inside an analysis tool).

3. Detailed Behavioural Insights: Gives you super-clear logs that basically walk you through exactly what the malware is trying to do, step by step, so you can see the whole attack playbook.

4. Improved Detection of Zero-day Malware: Great at catching brand-new, never-seen-before threats because it flags anything that behaves weird, even if there's no signature for it yet.

5. Effective for Advanced Malware Types : Works really well against the nasty stuff—ransomware that locks your files, spyware that spies on you, banking trojans that steal your money, all of it.

2.4 Limitations

1. The tackle cost is significant ; Emulating apps in a secure sandbox eats up a lot of computational coffers, taking important and precious machines.

2. Malware frequently evades discovery : numerous ultramodern pitfalls are designed to fete when they are being run in an impersonator and will shut down their vicious conditioning to avoid analysis.

3. We can not catch everything : Without genuine stoner commerce — like tapping, codifying, or swiping some corridor of the app's law simply no way run, creating eyeless spots.

4. The setup is a chain : It's not a draw- and- play result. You need the right tools and a well- configured monitoring system, which demands moxie and time.

5. It's hamstrung for large volumes : When you need to check apps at the scale of an entire app store, this system is just too slow and clumsy compared to other ways.

3. Hybrid Android Malware Detection (Static + Dynamic Analysis)

3.1 Abstract

Mongrel analysis integrates both static and dynamic ways to overcome the limitations of individual approaches. While stationary analysis provides fast pre-screening of an APK through overload and law examination, dynamic analysis captures runtime behaviour that stationary styles can not reveal. By combining features similar as warrants, opcodes, API calls, system calls, and network exertion, mongrel systems achieve advanced malware discovery delicacy and better adaptability against obfuscation. This binary- subcaste model ensures that if vicious law bypasses stationary examination, it's still detected during prosecution. mongrel analysis has come a dependable system in ultramodern malware discovery fabrics due to its bettered perfection and reduced false cons.

3.2 Approach

1. We start by checking the app without running it - looking at its manifest, pulling the code apart, and seeing what permissions and sensitive APIs it uses to catch anything fishy early.

2. Then we run the app in a secure sandbox and watch exactly what it does : network traffic, system calls, file changes, and real-time API usage.

3. We combine the clues from the static check (like permissions and code patterns) with everything we saw while it was running (like sequences of system calls) into one complete picture.

4. All that data goes into powerful machine-learning models : Random Forest, Gradient Boosting, CNN-LSTM, or a mix of several models - which decide if the app is malicious or clean, usually with very high accuracy.

5. To be extra sure, anything flagged as suspicious gets a second, independent check from another layer of the system so we don't have false alarms.

3.3 Contributions

1. Way higher accuracy : Looking at both the code and what it actually does when running catches the sneakiest malware that one method alone would miss.

2. Far fewer false positives : Something might look scary in the file, but if it behaves normally when executed, you avoid crying wolf over harmless stuff.

3. Laughs at obfuscation : Packers, encryptors, weird code tricks — doesn't matter. Once the malware runs, it can't hide what it's really trying to do.

4. Great against zero-days : You don't need a signature for a threat no one's seen before; if it acts like malware, you catch it based on behavior.

5. True defence in depth : Anything that sneaks past the static scan almost always gets exposed the moment it starts moving in a sandbox.

3.4 Limitations

1. It's a resource hog : Doing both static scanning and actually running the file chews through CPU, RAM, and time way more than just one approach.

2. The whole setup gets messy fast : You're basically stitching together a bunch of different tools, sandboxes, monitors, and data pipelines — it turns into a Frankenstein system real quick.

3. Forget about running this properly on phones : Most mobiles simply don't have the horsepower or battery life to do full-on hybrid analysis without choking.

4. Smart malware can still play hide-and-seek : A lot of newer stuff checks if it's inside a sandbox or VM and just sits there looking innocent until it thinks it's safe.

5. Large Dataset Requirements: Combining static and dynamic features demands extensive labelled datasets for training.

4. Deep Learning–Based Android Malware Detection

4.1 Abstract

Deep Literacy ways automate point birth and learn complex patterns from raw APK data similar as opcode sequences, API calls, bytecode images, and call graphs. Models like Convolutional Neural Networks(CNN), intermittent Neural Networks(RNN), LSTMs, and Graph Neural Networks(GNN) can capture nonlinear connections in malware gest that traditional ML algorithms may miss. Deep literacy enhances discovery delicacy and conception, making it suitable for relating zero- day pitfalls. still, these models bear large datasets, high computational coffers, and are frequently delicate to interpret, limiting their on- device deployment.

4.2 Approach

1. Point Transformation : Bytecode, opcodes, or API sequences are converted into embeddings, vectors, or images to prepare them for neural network processing.

2. Neural Network Architecture : Selection CNNs dissect opcode images, LSTMs process API or opcode sequences, and GNNs model call graph structures for malware bracket.

3. Training on Large Datasets : Malware and benign samples from datasets like Drebin, AndroZoo, and Virus share are used to train deep models.

4. Model Optimization : Model Optimization ways similar as powerhouse, normalization, early stopping, and learning rate scheduling help ameliorate model delicacy and help overfitting.

5. Vaticination and Deployment : The trained model classifies new APKs, relating vicious patterns through deep point representations.

4.3 Contributions

1. Automatic point birth : Eliminates homemade point engineering, perfecting discovery of complex malware.

2. High Accuracy & Robust Pattern Recognition : Captures deep connections in law that traditional ML can not descry.

3. Effective Zero- Day Discovery : Learns generalized patterns that help descry unseen malware families.

4. Protean Model operations : Supports image- grounded, sequence- grounded, and graph- grounded malware analysis.

5. Scalable for Large Datasets : Performs well with massive quantities of APK data.

4.4 Limitations

1. It's a total power hog : Training (and even running) these things basically demands a beefy GPU or TPU — your regular laptop will cry and die.

2. Nobody has a clue why it says "malware" : The model just spits out a number and shrugs. Good luck explaining to your boss or a regulator exactly why it flagged something.

3. Feed it garbage data, get garbage results : If your dataset is tiny or lopsided (like 95% benign apps), performance tanks hard. It needs thousands and thousands of properly labelled samples to not look stupid.

4. It memorizes the training set instead of learning real patterns : Without a ton of regularization tricks, the model starts treating random quirks in your data as "malware features" and completely flops on anything new.

5. Forget running this on a phone : Even the "light" versions are still way too fat for cheap or old Android devices - battery drains in minutes and the thing crawls.

5. Android Malware Detection Using Network Traffic Analysis

5.1 Abstract

Network business analysis is an effective fashion for detecting Android malware which communicates with remote waiters or performs vicious online conditioning. By assaying packet patterns, DNS queries, IP addresses, encryption behaviour, and data transfer frequentness, abnormal communication flows can be linked. Malware similar as spyware, botnets, and ransomware generally calculate on command- and-control waiters, making network examination a precious discovery approach. Machine literacy models help classify business as benign or vicious grounded on statistical and behavioural features. still, translated business, VPN operation, and stealthy communication styles reduce discovery delicacy.

5.2 Approach

1. Packet Capture and Logging : Few Tools like Wireshark, tcpdump, and DroidSheep capture outgoing and incoming packets while the app is running, that generates traffic logs.

2. Flow-Level Feature Extraction : At the communication flow -perspective, attributes such as packet size, session duration, protocol type, port numbers, and inter-arrival times are collected to uncover unusual or suspicious communication patterns.

3. Domain and IP Reputation Analysis : Take every IP and domain it talks to and look them up on blocklists, VirusTotal, AbuseIPDB, or your own C2 blacklist. One hit on a known botnet server and you're basically done — case closed.

4. Behaviour Pattern Modeling : Malware loves routines: pinging home every 5 minutes, blasting out weird DNS TXT queries, or suddenly dumping 50 MB of encrypted data at 3 a.m. You write little scripts or sequence models that go "yep, that's not normal."

5. Machine Learning Classification : Throw all those features into something simple like Random Forest, XGBoost, or even a one-class anomaly detector. After training on a bunch of good and bad apps, it can look at new traffic and say "benign" or "get this thing off my phone" in milliseconds.

5.3 Contributions

1. Detects Remote- Controlled Malware : Botnets, spyware, ransomware — pretty much anything that needs to chat with its evil overlord gets caught the second it tries to reach out.

2 Behaviour based detection : Pack it, encrypt it, obfuscate it all you want — the second it hits the network, the behaviour is out in the open.

3. Real- Time Network Monitoring : Works great for enterprise setups or even personal phones: just keep an eye on traffic and get alerts the moment something sketchy happens.

4. Minimal Device Resource Usage : Compared to firing up a full sandbox and running the app for minutes, just sniffing packets is basically free.

5. Useful for Zero- Day Discovery : Abnormal traffic behaviour reveals new or unknown malware families.

5.4 Limitations

1. Encrypted Traffic Evasion : The Malware which use HTTPS, VPN, or TLS tunneling may hide communication patterns.

2. High False Positives : The benign apps such as gaming, social media generates anomalus traffic similar to malware.

3 Demands Ongoing Oversight : Long-term data collection increases overhead and storage requirements.

4. Relies on Network Availability : Offline or dormant malware cannot be detected.

5. Inadiquate isolation : Must be combined with static or dynamic methods for higher accuracy.

6. Android Malware Detection Using Machine Learning (General ML Approaches)

6.1 Abstract

Machine Literacy- grounded malware discovery focuses on erecting classifiers that distinguish vicious APKs from benign ones using uprooted features similar as warrants, API calls, opcodes, system calls, and network patterns. ML models automate the identification of complex malware patterns that traditional rule- based systems can not detect. By training models on large datasets, malware families can be recognized with high delicacy. Although ML significantly improves scalability and rigidity, it depends heavily on point quality, data balance, and nonstop retraining to detect new pitfalls.

6.2 Approach

1. Feature Collection : APK files are anatomized statically or stoutly to prize warrants, API calls, intent pollutants, opcodes, system calls, and network behaviour.

2. Point Engineering and Selection : Redundant and inapplicable features are removed using ways like PCA, Chi-square, and collective Information to ameliorate model performance.

3. Dataset Preparation : Malware and benign apps are collected from depositories like Drebin, AndroZoo, Virus Share, and Google Play to make balanced datasets.

4. Model Training : ML algorithms similar as SVM, Random Forest, Naïve Bayes, Logistic Regression, and K- means are trained using uprooted point vectors.

5. Model Evaluation : delicacy, perfection, recall, F1- score, and ROC- AUC criteria are used to measure the discovery capability of trained classifiers.

6.3 Contributions

1. High Classification Accuracy: ML models effectively detect both known and unknown malware based on statistical patterns.

2. Automation of Malware Detection: It reduces manual analysis time through automated feature-based classification.

3. Scalable for Large Datasets: Suitable for app-store level malware screening.

4. Flexible Integration: ML models can work with static, dynamic, or hybrid features.

5. Family-Level Classification: Supports grouping malware into families for detailed threat analysis.

6.4 Limitations

1. Point reliance: Model performance heavily depends on the quality of uprooted features.

2. High threat of Overfitting : inadequately balanced datasets may beget prejudiced models.

3. Frequent Model Updates demanded : New malware variants bear nonstop retraining.

4. Limited Interpretability : Some ML models don't easily explain why an app is flagged vicious.

5. Vulnerability to Adversial Attacks : Malware can designedly manipulate features to evade discovery.

CONCLUSION

TITLE AND YEAR	AUTHORS	PROS	CONS
Android Malware Detection using Static Analysis (2021)	Li Wang, M. Kumar, Banerjee	Fast analysis; no execution required; low resource usage; suitable for large-scale screening.	Ineffective against obfuscated or packed malware; cannot detect runtime behaviour; limited zero-day detection.
Dynamic Behaviour-Based Android Malware Detection (2022)	R. Sharma, D. Patel, Thomas	Captures real execution traces; resistant to obfuscation; reveals hidden payloads; strong for spyware and ransomware.	High computational overhead; slow execution; vulnerable to sandbox-evasion; requires realistic input simulation.
Hybrid Static–Dynamic Android Malware Detection (2023)	H. Zhao, S. Mehta, Vivek Rao	High detection accuracy; reduced false positives; strong zero-day detection; resilience against code hiding.	Complex architecture; high hardware requirement; difficult on-device deployment; requires large datasets.

Deep Learning–Driven Android Malware Classification (2024)	A.Gupta, F. Hernandez, Y. Cho	Learns deep patterns automatically; very high accuracy; effective for malware families and zero-day attacks.	Requires GPUs/TPUs; large dataset dependency; limited model interpretability; prone to overfitting.
Network Traffic Analysis for Android Malware (2021)	J. Singh, R. Alami, O. Musa	Effective for botnets and C2 detection; low device resource usage; independent of code structure.	Encrypted traffic evades detection; high false positives; requires continuous monitoring; cannot detect offline threats.
Machine Learning-Based APK Classification (2020)	Priya Nair, Sohail Khan, Ankit Verma	Automated classification; scalable; integrates static and dynamic features; supports family-level grouping.	Feature dependency; frequent retraining required; vulnerable to adversarial manipulation; dataset imbalance issues.