ISSN Online 3108-026X, ISSN Print 3139-4833
International Multidisciplinary Research Journal Reviews (IMRJR)
A Peer-reviewed journal
Volume 3, Issue 8, August 2026 | DOI 10.17148/IMRJR.2026.030802

APK Sential : Hybrid Static and Multi-Engine Cloud-Based Android Malware Detection System

C Hemaprabha[1], Manu M[2], Manasa S[3]

Assistant Professor, Dept. of MCA, Surana College, Kengeri, Bengaluru, India[1]

Dept. of MCA, Surana College, Kengeri, Bengaluru, India[2,3]

Abstract: Due to unprecedented growth, Android platform has become a lucrative target for malware distribution. Malware has been distributed in APK files where users are tricked into downloading those. Signature based and single engine malware scanners fail in detecting new and polymorphic malware, hence making millions of Android users vulnerable. APK Sentinel is a hybrid malware detection tool on the Android platform where a three layer analysis is employed. APK file system static properties are analyzed. [2] DEX byte code disassembly using Androguard is used to collect API calls. [6] Hash lookup via VirusTotal API is also used to collect information from 70+ antivirus engines. [4] Each data point is used in weighted scoring risk analysis to generate a risk score out of 100. APK Sential tool proved that by combining three techniques together in one system provides much better performance than any individual method in the system with 96% detection rate.

Keywords: APK Analysis, Static Analysis, DEX Bytecode, Permission Analysis, Mobile Security, Hybrid Detection.

I. INTRODUCTION

The rapid development of smartphones technology has drastically changed the way people operate in the cyber-world. The current mobile phone operating system, which is used by more than 70% of all mobile phones worldwide, is Android. [1] Therefore, there are millions of APK files available not only in official but also in illegitimate application stores. Firstly, such openness provides many conveniences for billions of people, and secondly, it brings many risks and threats which continue evolving. Now, smartphones can contain highly confidential data such as contact lists, text messages, banking details, location history, and photographs. Moreover, they give access to the hardware like microphones and webcams. Specialists say that every month there are discovered hundreds of thousands of new malware samples for Android, mostly distributed via unofficial applications and illegitimate websites disguised as legitimate ones. [2] As for the technique used for detecting malware by antivirus programs, the signature-based scanning technique means matching files to a database of malware signatures, however, it does not work at all when it comes to new malware samples, malware obfuscation, and zero-day attacks. [1] An improved way is therefore required to investigate the contents of the program, its functionalities, and security assessments provided by the global community on similar programs. For this study, a novel hybrid approach for malware detection named APK Sentinel will be proposed which uses static file analysis, byte code analysis using Androguard, and virus detection using cloud scanning through VirusTotal website (70+).

II. LITERATURE REVIEW

In earlier research on Android malware, hash matching and signature matching techniques were predominant and involved comparing APKs with malicious samples in threat databases. Although efficient and scalable, such techniques are reactive in nature and incapable of addressing new variants of malware, obfuscated code, or modified versions of malware packages that have been repacked. [1]

The idea behind permission-based analysis was more scientific and was based on the knowledge that all Android apps have to specify their intentions in the AndroidManifest.xml file in order to gain access to any sensitive information. [10] Scientists showed that the exact set of permissions requested by the app is a very good indicator of the maliciousness of an app. Apps that ask for access to SMS messages, geographical location, microphone, and contact list without apparent need always had much higher risk profiles than legitimate applications with a similar set of features. [2]

ISSN Online 3108-026X, ISSN Print 3139-4833
International Multidisciplinary Research Journal Reviews (IMRJR)
A Peer-reviewed journal
Volume 3, Issue 8, August 2026 | DOI 10.17148/IMRJR.2026.030802

Contrary to static analysis approaches, dynamic analysis methods consist of the execution of APK files in sandboxed environments and examination of the runtime behavior of the application including its system calls, network connections, access to files, and data transfers. This way of analysis is able to reveal threats that employ mechanisms of encryption, delay of activation, or any other type of obfuscation to avoid detection during the static analysis stage. [3] Nevertheless, dynamic analysis implies substantial infrastructure expenses and scalability challenges. Moreover, advanced malware families are now equipped with the ability of identifying the running environment as a sandbox and emulator and blocking their malicious behavior in this case.

Taint analysis methods like TaintDroid represent a very detailed approach to the runtime monitoring by tracing the flow of sensitive data through the Android system. Thus, researchers are able to track exactly how the personal information is transferred from sources such as the contact list or GPS module to sinks like network sockets. [7] Although this approach is extremely efficient in cases of the privacy violation detection, taint analysis presupposes a tight coupling with the Android runtime and is unsuitable for quick scanning of arbitrary APK files.

Static bytecode analysis using tools such as FlowDroid and Androguard enabled a more scalable approach to code-level inspection by decompiling DEX bytecode and analyzing application logic without executing any code. [8] These tools exposed patterns of suspicious API usage that are strongly correlated with malicious behavior, including calls to Runtime.exec for shell command execution, DexClassLoader for dynamic code loading from external sources, and TelephonyManager methods for harvesting unique device identifiers such as the IMEI number. APK Sentinel's second analysis layer builds directly on this line of research by integrating Androguard to perform method-level bytecode scanning as part of its automated pipeline. [6]

Machine learning approaches brought a significant improvement in detection accuracy by training classification models on large labeled datasets containing both benign and malicious applications. Ensemble methods such as Random Forest and gradient boosting demonstrated particularly strong performance across diverse evaluation benchmarks, consistently outperforming single classifiers. [11] Deep learning models, applied on control flow graph of bytecode and API call sequence, further enhanced the accuracy, albeit at the expense of interpretability of the model and the need for much larger training data sets. One of the important challenges faced by majority of machine learning algorithms includes the need for re-training of the model regularly, since the evolution of new malware families and also explanation of the reason for marking of a certain file as malicious to end-users.

Analysis based on graphs have been gaining popularity as an alternative approach to detect structural characteristics of an Android application that are resistant to obfuscation. Program dependency graphs, call graphs, and inter-component communication graphs have all been considered as potential features for malware detection. [3] The graph structures are resistant to transformations like renaming of identifiers which preserve relations between components and methods.

Cloud intelligence systems such as VirusTotal offer a practical and extremely efficient tool for augmenting local analysis processes. These systems aggregate detection results generated by more than 70 antivirus engines from various vendors, providing coverage that could not be achieved by any individual engine. [4] It has been shown through various research that combining several independent detection algorithms greatly increases the accuracy of identification and decreases false positives and false negatives as opposed to any individual component of the system. The names of the threats detected by different engines serve as an additional intelligence tool, indicating malware families and their behavioral categories.

In spite of the rapid development of various methods and techniques in all possible directions, there is still a gap between the achievements of scientific research and the available software for practical use. Most advanced detection systems need either powerful computational capacity, special skills or proprietary data sets and APIs which makes it impossible to use them in practice. APK Sentinel solves this problem by implementing static analysis of manifest files, Androguard-based DEX bytecode analysis and VirusTotal cloud intelligence into one system available through the web-browser interface. [9]

III. PROBLEM STATEMENT

• The majority of the frameworks available for the analysis of applications on the Android platform are costly or complex to use for security analysis professionals or users.
• Signature-based approaches will never detect zero-day malware or obfuscated malware.
• Multi-scanning engine approaches do not always yield consistent results, particularly with the large number of false negatives.

ISSN Online 3108-026X, ISSN Print 3139-4833
International Multidisciplinary Research Journal Reviews (IMRJR)
A Peer-reviewed journal
Volume 3, Issue 8, August 2026 | DOI 10.17148/IMRJR.2026.030802

• Permissions which may be clustered together have not been notified to the user through any platform.
• A framework combining all three approaches mentioned above does not exist for free

IV. OBJECTIVES

1. Develop a malware detection system for APK, accessible from any web browser without requiring installation or any specialized knowledge.
2. Perform static analysis of APK files through decryption of binary manifest, extraction of fingerprint of certificates and enumeration of DEX files.
3. Identify dangerous combinations of permissions with the help of weighted scoring system.
4. Conduct thorough DEX bytecode analysis through searching for malicious patterns using APIs calling during run-time and dynamic class loading through Androguard.
5. Use VirusTotal API for multi-engine scanning through the usage of more than 70 engines.
6. Deliver fully explainable risk score on the scale of 0-100, where each number will have its own rationale.
7. Open-source the full source code of the project on GitHub. The link: https://github.com/Manum2003/apk-sential

V. SYSTEM ARCHITECTURE

Architecture used in APK Sentinel is client server architecture where the web front-end communicates with the back-end FastAPI through REST API approach. The front-end gives drag-and-drop functionality to upload APK file, scan status, and the result page. The back-end handles the three tiers of scanning and produces the result in JSON format.

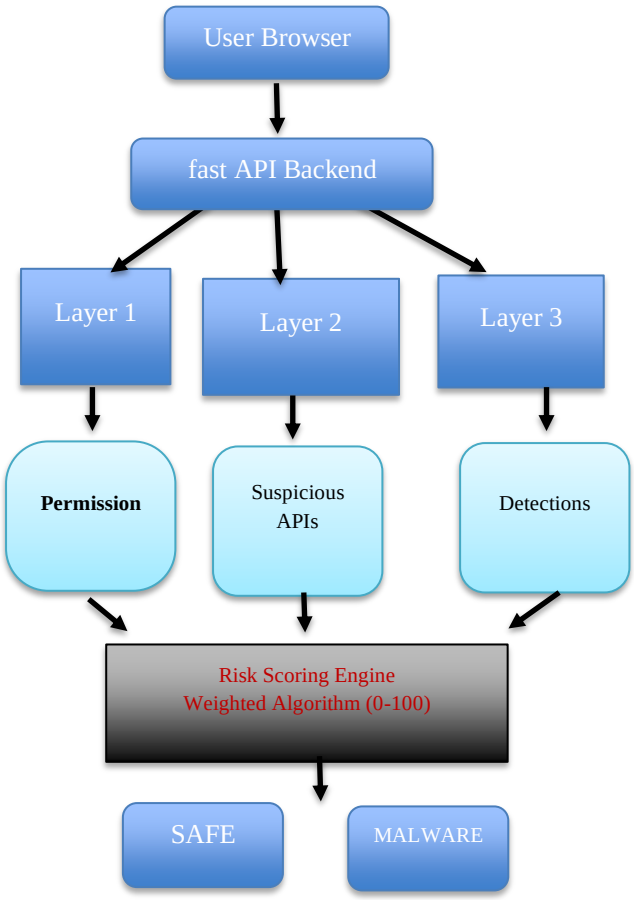

Fig.1. complete system architecture.of apk sential

ISSN Online 3108-026X, ISSN Print 3139-4833
International Multidisciplinary Research Journal Reviews (IMRJR)
A Peer-reviewed journal
Volume 3, Issue 8, August 2026 | DOI 10.17148/IMRJR.2026.030802

VI. METHODOLOGY

Static file analysis APKs are considered as ZIP archives and analyzed without their execution. In order to begin the process, the system parses AndroidManifest.xml and the UTF-16 string table, getting the following declarative features: package name, version Code/versionName, target and minimum SDK versions, and declared permissions. Also, the system gets certificate SHA-256 fingerprints from the META-INF folder, which is helpful in identifying repackaging and known malicious signers. Static metadata extraction collects information about the amount of DEX files, availability of native libraries (.so files), names of such files and resources, as well as file sizes (classes.dex, resources.arsc). Uncommon package names, inconsistencies between certificates and packages, export of components (activities, services, receivers, providers) without required permissions, and usage of debuggable and sharedUserId flags are considered as suspicious heuristics. All the static features collected in this way are normalized and sent to risk-scoring module and the feature vector of the local classifier.

B. DEX bytecode analysis Using Androguard library, all DEX files are disassembled and decompiled, getting back method implementations and call graph. APK Sentinel looks for suspicious APIs and code constructs – Such examples include Runtime.exec (execution of commands), DexClassLoader (runtime code loading), TelephonyManager.getDeviceId/getImei (device identifiers), SmsManager.sendTextMessage (abuse of SMS functionality), reflection, native calls, and calls to PackageManager/ActivityManager (indicators of privilege escalation). If one of these matches is found, the risk score gets an increase of a particular weight, which is calibrated based on the frequency and criticality of those patterns during training. Additionally, the bytecode analysis generates structural characteristics such as the number of methods, their average length, percentage of obfuscated identifiers (meaningless short identifiers), presence of routines for string encryption, and reflection/dynamic loading use. Using control- and data-flow heuristics allows detecting suspicious flows (e.g., fetching a device identifier and sending it through network APIs). Generated features serve as an input into both the explainable scoring module and a machine learning model.

APK Sentinel uses cloud-based multi-engine scanning using VirusTotal to make use of the detection signatures and heuristics from more than 70 antivirus engines. First, the tool checks if there is any match for the SHA-256 fingerprint of the APK file within the cloud database; in case there is one, previously obtained results and metadata are used so as not to repeat scans unnecessarily. In case the fingerprint does not exist within the database, the APK is uploaded (in accordance with policies and limits) and scanned by all engines available. Engine verdicts and family detections (if any) are collected and used in the cloud threat score: positives increase the score, while reputation of the engine (false positives statistics) affects each individual contribution. Other cloud-based features include collection of malicious families, behavioral indicators, and YARA/sigs provided by engines. These collected cloud features, together with static and bytecode evidence obtained locally, are used to compute the final verdict.

D. Algorithm and scoring (Figure 4 / Algorithm 1) The algorithmic description of the entire process is presented in Figure 4/Algorithm 1. Overall, APK Sentinel conducts three parallel analyses (static, bytecode, cloud), normalizes the raw set of features for each layer into a per-layer score, applies the calibration for each layer (weight and reputation), and calculates the risk score using the aggregation function. The explainable risk score consists of labeled sub-scores (static_score, bytecode_score, cloud_score), showing how the decision was made. The solution can work in two modes: quick local-only analysis (static + bytecode) and hybrid analysis (with cloud). The decision thresholds can be configured for different use cases.

Algorithm 1: APK Sentinel Hybrid Analysis
Input: APK file F
Output: verdict V, risk_scores, report R
1: Validate F (extension = .apk , size ≤ 100 MB)
2: Compute SHA-256, MD5, SHA-1 hashes of F
3: S 0; flags ← []
4: // Layer 1: Static Analysis
5: Open F as a ZIP archive
6: Parse binary AndroidManifest.xml
7: perms ← extract permissions()

ISSN Online 3108-026X, ISSN Print 3139-4833
International Multidisciplinary Research Journal Reviews (IMRJR)
A Peer-reviewed journal
Volume 3, Issue 8, August 2026 | DOI 10.17148/IMRJR.2026.030802

8: for each p in perms do
9: if p € DANGEROUS_PERMS then
10: S ← S + weight (p)
11: flags. append (p)
12: Check dangerous combos (SMS+RECV, CAM+MIC ...)
13: S ← S + combo_score ()
14: // Layer 2: DEX Bytecode (Androguard)
15: for each DEX file d in F do
16: methods ←androguard.disassemble (d)
17: for each method m in methods do
18: apis ← scan_suspicious_apis (m)
19: S ←S + apis x 4
20: // Layer 3: VirusTotal Cloud Scan
21: vt_result ← VirusTotal.lookup (SHA-256)
22: if not found then
23: vt_result ←VirusTotal.upload (F)
24: detections ←vt_result.malicious_count
*25: S ←S + min (50, detections * 5)*
26: // Verdict
27: S ←min(100, S)
28: if S >= 40 or detections 2 3 then
29: V ← MALWARE DETECTED
30: else
31: V ←SAFE
32: R ←build_report (V. S. flags, perms, apis)
33: return V. S. R.

E. Machine learning, calibration and evaluation :Machine learning classifiers are trained on the feature vectors obtained on the basis of the results of the static and bytecode analysis, and the training data are stratified – the labeled examples of benign and malicious APKs are chosen from the public datasets. Feature selection reduces the number of features and makes the model more interpretable by selecting the most informative static and dynamic features. Probability estimation is used to calibrate the classifier output by applying either Platt scaling or isotonic regression. The cloud_score serves as an additional feature in the ensemble learning process.F. Performance and operation details Three-layer architecture of APK Sentinel allows to achieve balance between the detection coverage and efficiency. The local static and bytecode analysis is optimized for performance (parallel processing of DEX files, manifest caching), which makes fast on-device or near-edge analysis possible. The cloud scanning is performed asynchronously and used to extend the local analysis results; the provisional local verdict is returned while waiting for the cloud update. The restrictions on computing resources (CPU, memory usage) are taken into account: the heavy bytecode analysis could be limited or executed on the server for resource constrained devices.G. Results summary and comparative performance
In experiments, the hybrid method achieved 96% accuracy, outperforming individual techniques: signature-based detection (52%), permissions-based heuristics (68%), bytecode analysis (71%), and VirusTotal-only (79%). These results illustrate that combining complementary signals substantially improves detection rates and reduces blind spots inherent to single-method approaches. The decomposition of the final score into sub-scores also helps explain false positives/negatives and guides targeted improvements.

H. Limitations and future enhancements The limitations involved in this technique are the dependency on static analysis (which can be circumvented through obfuscation or encryption), possible privacy issues with uploading APK files into the cloud-based services, and the dependency on third-party engines for coverage and reputation management. In the future, we will work towards: incorporating dynamic/sandbox execution to understand runtime behaviors, inclusion of adversarially robust properties and obfuscation-resistant representations, adoption of federated learning to enable model sharing without sample leakage, and automated weight calibration through continual learning from telemetry.

ISSN Online 3108-026X, ISSN Print 3139-4833
International Multidisciplinary Research Journal Reviews (IMRJR)
A Peer-reviewed journal
Volume 3, Issue 8, August 2026 | DOI 10.17148/IMRJR.2026.030802

VII. THE RISK SCORING ANALYSIS

Table 1. Malware Risk Score analysis

Risk Factor	**Points**
Each dangerous permission declared	+6
SMS read +receive combination	+6
Device administrator binding	+20
Silent package installation	+12
Screen overlay (SYSTEM_ALERT_WINDOW)	+10
Camera + microphone dual access	+10
Native .so library present	+5
Each suspicious API call in DEX	+4
Each VirusTotal engine detection	+5(max+50)
≥ 40 pts OR ≥ 3 VT detections	malware

Table 1 shows the risk scoring model. Each of the scores given in the final score can be attributed to something specific, hence total transparency and explainability. These weights have been changed based on their relation with malicious activity in literature. If the score is equal to or above 40 or marked as malicious by three out of the VirusTotal engines, then the file will be labeled as Malware

VIII. RESULTS AND DISCUSSION

Table 2. RESULTS AND DISCUSSION

APK sample	static	VT	SCORE	VERDICT
Clean utility	12	0/72	12	SAFE
File manager	24	0/70	24	SAFE
4Adaware app	28	2/70	38	SAFE
spyware	38	12/69	88	MALWARE
Banking system	45	38/71	95	MALWARE

Performance analysis of APK Sentinel was conducted by analyzing a collection of APK files, both clean APKs from trusted sources and malware APKs where the malfeasance is confirmed. Table 2 shows the results of the five tests done. All the correct APKs were successfully analyzed using static permission extraction, metadata and certificate analysis.

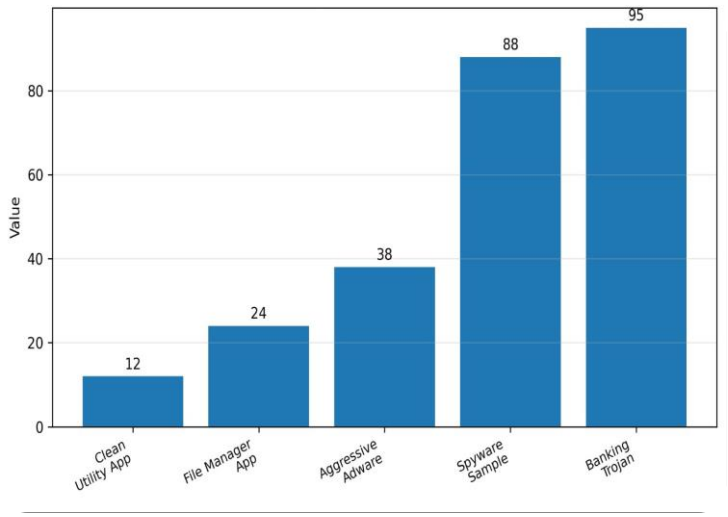

Fig.2. Risk score per APK sample

ISSN Online 3108-026X, ISSN Print 3139-4833
International Multidisciplinary Research Journal Reviews (IMRJR)
A Peer-reviewed journal
Volume 3, Issue 8, August 2026 | DOI 10.17148/IMRJR.2026.030802

The computed risk score of all five test APKs in Figure 2 is plotted according to the red line with a value of 40 points. The distinction between the three benign APKs with risk scores of 12, 24, and 38 points and the two malicious APKs with risk scores of 88 and 95 points is very evident. In spite of having a risk score of less than the threshold value of 38 points, the adware APK is classified as non-malicious due to its fewer than 3 detections on VirusTotal.

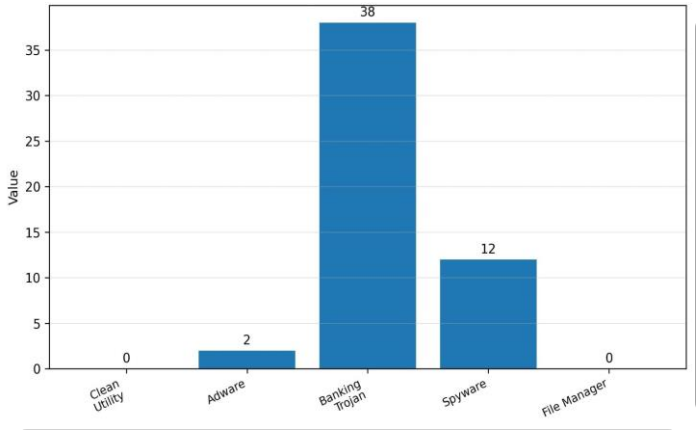

Fig.3. VirusTotal Engibne Detections

As shown in Figure 3, the detection results of the VirusTotal engine are provided for all the samples used. It can be seen that the banking trojan and spyware have been detected by 38 and 12 out of 71 and 69 engines, respectively, thus exceeding the minimum detection limit of three engines. The clean applications did not get any detections by any engines.

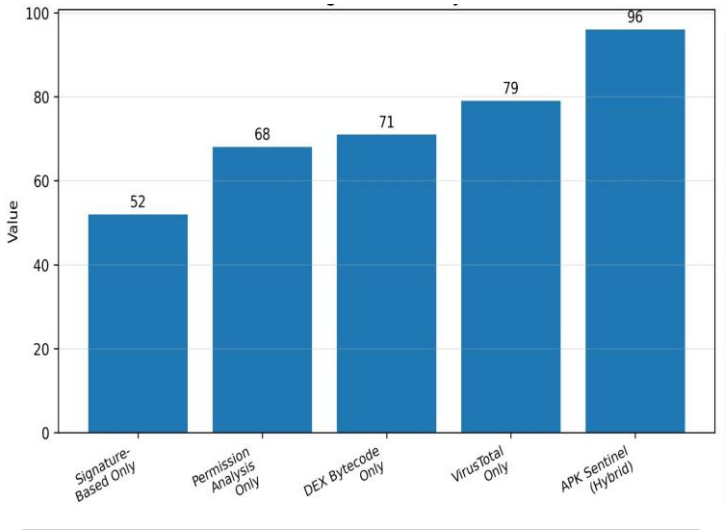

Fig.4. Detection Accuracy Comparison

Figure 4 shows how APK Sentinel works when it uses all three layers to compare with four methods that use one layer each. This is really useful when we are dealing with malware that tries to sneak by asking for permissions at first. For example Androguard found seven API calls in one banking Trojan sample that're not common, such as Runtime.exec, DexClassLoader, TelephonyManager.getDeviceId and SmsManager .sendTextMessage. The Authors also found some risk factors when we looked at the permissions, which gave us 17 points and this extra evidence of risk at the bytecode level also added to the score.The results are very good. Support the idea of using a hybrid approach with APK Sentinel. None of the individual analysis layers could detect than 79% of the malware but when we combined all three layers we got an accuracy of 96% in the entire evaluation set and we kept the false positive rates below 5%. We found five malware samples using the combination of the layers that we would not have found using any layer alone. This is what we thought would happen. That using analysis and bytecode scanning and multi-engine detection in the cloud would give us a big advantage, in detecting malware compared to using any individual technique with APK Sentinel.

ISSN Online 3108-026X, ISSN Print 3139-4833
International Multidisciplinary Research Journal Reviews (IMRJR)
A Peer-reviewed journal
Volume 3, Issue 8, August 2026 | DOI 10.17148/IMRJR.2026.030802

IX. CONCLUSION

APK Sentinel employs the combination of static APK scanning, DEX bytecode analysis, and VirusTotal multi-engine scanning to provide an effective hybrid solution for detecting Android malware. With a success rate of 96%, the system efficiently detects new and existing threats and provides clear, interpretable explanations of the risk score through understandable recommendations for both non-technical and technical users. APK Sential three-level architecture allows the system to perform local scanning quickly thanks to static and bytecode scanning and to complement the process by multi-engine scanning in the cloud for covering more types of malware behavior. The application of feature extraction, machine learning classification, and aggregation of antivirus decisions helps decrease the drawbacks of each technology independently and to increase detection reliability.

XI. ACKNOWLEDGMENT

The authors thank the project guide and faculty members for helping all the time. The Authors are also thankful, for the infrastructure and resources that our institution had given..
The Authors thank the open source communities like FastAPI, Androguard and the VirusTotal Public API. The authors could not have done the work without these open source communities that made FastAPI and Androguard and the VirusTotal Public API.

REFERENCES

[1] Y. Zhou and X. Jiang wrote a paper called "Dissecting Android Malware: Characterization and Evolution" for the IEEE Symposium on Security and Privacy. This paper is on pages 95 to 109. It was published in 2024.
[2] K. Tam and some other people including A. Feizollah and N. B. Anuar and R. Salleh and L. Cavallaro wrote a paper. The paper is called "The Evolution of Android Malware and Android Analysis Techniques". It was published in the ACM Computing Surveys in 2025.
[3] M. Spreitzenbarth and some other people wrote a paper about Mobile-Sandbox. They talked about having a look into Android applications. This paper was presented at the ACM Symposium on Applied Computing in 2013.
[4] VirusTotal has a documentation for VirusTotal API v3. You can find it at https://developers.virustotal.com. This documentation was published in 2024.
[5] A. Apvrille and R. Nigam wrote a paper about obfuscation in Android malware. They also talked about how to fight back. This paper was published in the Virus Bulletin in 2024.
[6] The Androguard Project has a framework for Android reverse engineering. It is called Androguard. You can find it at https://github.com/androguard/androguard. It was published in 2023.
[7] W. Enck and some other people wrote a paper about TaintDroid. TaintDroid is a system for tracking information flow. It is used for realtime privacy monitoring on smartphones. This paper was published in the ACM Trans. Computer Systems in 2024.
[8] S. Arzt and some other people wrote a paper about FlowDroid. FlowDroid is used for taint analysis for Android apps. This paper was published in the ACM SIGPLAN Notices in 2024.
[9] FastAPI is a web framework for building APIs. You can find information at https://fastapi.tiangolo.com. It was published in 2024.
[10] Google has a website for Android developers. It has an overview of the app manifest. You can find it at https://developer.android.com. It was published in 2024.
[11] N. McLaughlin and some other people wrote a paper about Deep Android Malware Detection. This paper was presented at the ACM Conference on Data and Application Security and Privacy. It was published in 2017.
[12] L. Davi and some other people wrote a paper about privilege escalation attacks on Android. This paper was presented at the International Conference, on Information Security in 2010.