International Journal of
AI Electronics and Nexus Energy
Peer Reviewed, Referred & Indexed Journal
ISSN: 3070-0515 www.zesterapublications.com Original Research Paper

MAL-SCAN :ANDROID MALWARE DETECTION BASED ON SOCIAL NETWORK CENTRALITY ANALYSIS

[1] Mrs. J Prashanthi, [2] Uppu Vamshi Krishna, [3] Kairamkonda Mahesh, [4] Vadthya Srikanth,[5] Katika Muneesh
[1] Assistant Professor, [2345] B. Tech Students
[1]Department of Computer Science and Engineering
[2345]Department of CSE(CYBER SECURITY)
[12345]Sree Dattha Group of Institutions, Sheriguda, Ibrahimpatnam, 501510, Telangana, India

ABSTRACT

The rapid expansion of Android-based smartphones, tablets, wearable devices, mobile banking applications, digital payment platforms, social networking services, and enterprise mobility has significantly increased the attack surface available to malicious software. Android malware increasingly employs code obfuscation, repackaging, dynamic payload loading, permission abuse, inter-component communication, hidden API invocation, and coordinated behavioral dependencies to evade conventional signature-based and isolated feature-based detection mechanisms. Traditional Android malware detection methods generally analyze applications as independent collections of permissions, API calls, opcodes, system calls, or behavioral events; however, such approaches may fail to capture the structural relationships among application components and security-relevant entities. This research proposes **MAL-SCAN**, an intelligent Android malware detection framework based on **social network centrality analysis**, in which an Android application is transformed into a security interaction graph and analyzed using graph-theoretic centrality measures inspired by social network analysis. Nodes represent entities such as application components, permissions, sensitive API calls, intents, services, receivers, URLs, and behavioral events, while edges represent invocation, communication, dependency, permission usage, or information-flow relationships. The framework computes Degree Centrality, Betweenness Centrality, Closeness Centrality, Eigenvector Centrality, and PageRank-based importance scores to identify structurally influential entities and suspicious interaction hubs within an application graph. The resulting centrality profiles are combined with selected static and behavioral indicators and supplied to machine learning classifiers such as Random Forest, Support Vector Machine, XGBoost, and Logistic Regression. A hybrid decision engine produces a malware probability and classifies applications as Benign, Suspicious, or Malicious. The proposed architecture consists of five interconnected layers: Android Application Acquisition, Static Analysis and Security Graph Construction, Social Network Centrality Analysis and Intelligent Detection, Risk Assessment and Response, and Application/User layers. Prototype-oriented evaluation is defined using accuracy, precision, recall, F1-score, graph-feature efficiency, false-positive rate, and detection response time. Illustrative conceptual results indicate that MAL-SCAN can outperform signature-based scanning, conventional permission-based machine learning, and standalone graph analysis by exploiting the structural importance of security-relevant entities. The proposed framework provides a scalable and explainable foundation for Android malware detection in mobile devices, enterprise app stores, application marketplaces, mobile security gateways, and cybersecurity laboratories.

Keywords: Android Malware Detection, MAL-SCAN, Social Network Analysis, Centrality Analysis, Graph-Based Security, Degree

International Journal of
AI Electronics and Nexus Energy
Peer Reviewed, Referred & Indexed Journal
ISSN: 3070-0515 www.zesterapublications.com Original Research Paper

Centrality, Betweenness Centrality, Closeness Centrality, Eigenvector Centrality, PageRank, Machine Learning, Mobile Security.

I. INTRODUCTION

The Android operating system has become a dominant platform for smartphones, tablets, smart televisions, wearable devices, automotive systems, and other connected environments. Its open application ecosystem, extensive developer support, flexible distribution mechanisms, and broad device availability have contributed to rapid global adoption. However, the same openness and popularity have created significant opportunities for cybercriminals to distribute malicious applications and exploit mobile users [1]–[3].

Android devices frequently store or process highly sensitive information, including personal messages, photographs, contact lists, authentication tokens, banking credentials, payment information, location records, enterprise documents, and social networking data. Malware targeting Android platforms may therefore create substantial privacy, financial, and organizational risks. Common malicious behaviors include credential theft, SMS interception, spyware activity, ransomware, ad fraud, banking overlays, unauthorized subscription services, remote command execution, data exfiltration, privilege abuse, and installation of secondary payloads [4], [5].

Traditional Android malware detection mechanisms primarily rely on signature matching. In this approach, application files, code fragments, byte sequences, package characteristics, or other artifacts are compared against known malware signatures. Signature-based scanning provides efficient detection of previously identified malware families but exhibits limited effectiveness against zero-day attacks, polymorphic variants, repackaged applications, code-obfuscated malware, and rapidly changing malicious campaigns.

Static analysis has emerged as an important alternative because Android Application Package files can be examined without executing them. Security systems may extract requested permissions, API calls, manifest components, intents, opcodes, strings, URLs, and control-flow characteristics. Machine learning algorithms can then learn statistical differences between benign and malicious applications [6]–[8].

However, conventional static machine learning frequently treats extracted features as independent variables. For example, a permission may be represented as a binary value indicating whether it is requested, while an API call may be represented by frequency. Such representations provide useful information but may fail to capture how security-relevant entities interact.

Android applications are inherently relational. Activities invoke services, receivers respond to intents, methods call sensitive APIs, components request permissions, and data may flow through interconnected processing structures. Malware behavior may therefore emerge from the organization of relationships rather than from individual features alone.

Graph-based modeling provides a powerful mechanism for representing these relationships. An application can be transformed into a graph:

$$G = (V, E)$$

where Vrepresents the set of nodes and Erepresents the set of edges. Nodes may correspond to Android components, permissions, API calls, methods, intents, URLs, or behavioral entities, while edges represent invocation, communication, dependency, or information flow.

Although graph construction provides structural information, large graphs may contain thousands of entities and relationships. Not all nodes contribute equally to malicious behavior. Some

International Journal of
AI Electronics and Nexus Energy
Peer Reviewed, Referred & Indexed Journal
ISSN: 3070-0515 www.zesterapublications.com Original Research Paper

nodes occupy strategically important positions and connect multiple suspicious regions of the graph. Identifying such influential entities is therefore essential.

Social Network Analysis (SNA) provides mathematical measures for evaluating the structural importance of nodes in relational networks. Centrality measures originally used to identify influential individuals or organizations in social networks can also be applied to software interaction graphs. Degree Centrality measures direct connectivity, Betweenness Centrality identifies bridge nodes, Closeness Centrality measures accessibility, Eigenvector Centrality evaluates influence through important neighbors, and PageRank estimates recursive structural importance [9], [10].

The application of social network centrality analysis to Android security offers several advantages. A malicious API that acts as a bridge between credential collection and network transmission may exhibit high betweenness. A component connected to numerous sensitive operations may have high degree centrality. A node linked to other highly influential nodes may receive high eigenvector centrality.

Machine learning can further improve the effectiveness of centrality-based analysis. Rather than relying on a single graph metric, the proposed framework generates centrality profiles and combines them with selected security features. Classifiers such as Random Forest, Support Vector Machine, XGBoost, and Logistic Regression can learn complex differences between benign and malicious graph structures.

Motivated by these requirements, this research proposes **MAL-SCAN: Android Malware Detection Based on Social Network Centrality Analysis**. The proposed system transforms Android applications into security interaction graphs, computes multiple centrality measures, identifies structurally influential entities, extracts graph-based security profiles, applies machine learning classification, calculates malware risk scores, and produces explainable detection decisions.

II. LITERATURE SURVEY

Android malware detection has been extensively investigated through static analysis, dynamic analysis, permission mining, API-call analysis, graph representations, machine learning, and deep learning. The following literature survey presents major contributions relevant to MAL-SCAN.

Author: W. Enck et al. (2010)

Enck and colleagues introduced TaintDroid, a dynamic information-flow tracking system for monitoring sensitive information on Android devices. Their work demonstrated that mobile applications could be analyzed according to runtime data propagation and privacy-sensitive behavior. The study established an important foundation for behavior-oriented Android security analysis [11].

Author: A. P. Felt et al. (2011)

The researchers investigated Android permissions and examined the problem of application overprivilege. Their study demonstrated that applications may request or obtain more capabilities than required, creating security risks and highlighting permissions as important malware-analysis indicators [12].

Author: D. Arp et al. (2014)

Arp and colleagues proposed DREBIN, a machine learning-based Android malware detection framework using explainable static features. The system extracted permissions, API calls, network addresses, and application characteristics to detect malware directly on mobile devices. Their work demonstrated the effectiveness of large-scale static feature analysis [13].

Author: M. Grace et al. (2012)

International Journal of
AI Electronics and Nexus Energy
Peer Reviewed, Referred & Indexed Journal
ISSN: 3070-0515 www.zesterapublications.com Original Research Paper

The researchers conducted a systematic characterization of Android malware and analyzed malicious application behaviors, families, and propagation mechanisms. Their findings demonstrated the diversity of mobile threats and emphasized the need for scalable detection frameworks [14].

Author: Y. Aafer, W. Du, and H. Yin (2013)

The authors proposed DroidAPIMiner, which used API-level features and machine learning to identify Android malware. Their research demonstrated that frequently invoked and security-sensitive API calls can provide strong discriminatory information for malware classification [15].

Author: S. Arzt et al. (2014)

The researchers introduced FlowDroid, a precise static taint-analysis framework for Android applications. Their work modeled Android-specific lifecycle and callback behavior to identify sensitive information flows. This research demonstrated the importance of structural and flow relationships in mobile application analysis [16].

Author: K. Allix et al. (2016)

The authors investigated machine learning for Android malware detection and emphasized temporal evaluation. Their findings demonstrated that model performance can degrade as malware evolves, highlighting the need for adaptable feature representations and realistic testing methodologies [17].

Author: E. Mariconti et al. (2017)

The researchers proposed MaMaDroid, a behavioral Android malware detection system that modeled API call sequences through Markov chains. Their work demonstrated that abstracted behavioral relationships can improve robustness against evolving malware families [18].

Author: L. Onwuzurike et al. (2019)

The authors extended behavioral Android malware analysis and investigated the long-term effectiveness of abstracted API-call representations. Their findings reinforced the importance of structural behavioral modeling rather than dependence only on individual static indicators [19].

Author: F. Scarselli et al. (2009)

The researchers presented foundational concepts for Graph Neural Networks and graph-structured learning. Although not limited to Android malware, their work demonstrated how relational information can be processed computationally and provides theoretical support for graph-based cybersecurity analysis [20].

III. SYSTEM ANALYSIS & DESIGN

3.1 Existing System

Existing Android malware detection systems generally rely on antivirus signatures, permission analysis, API-call frequency, opcode patterns, dynamic behavior monitoring, heuristic rules, or conventional machine learning. Signature-based antivirus systems compare applications with previously identified malicious patterns and can effectively detect known threats.

Permission-based approaches analyze the capabilities requested in the Android manifest. Applications requesting combinations of sensitive permissions may be classified as suspicious. API-based systems inspect security-sensitive function calls, while dynamic systems execute applications in sandboxes and observe runtime activities.

Machine learning improves adaptability by learning classification patterns from extracted features. However, many existing systems represent applications as flat vectors. Such representations may indicate whether an API or permission exists but do not adequately describe how different entities interact.

Graph-based systems address part of this limitation by representing call relationships, control flow, or component interactions. Nevertheless, some approaches process large

International Journal of
AI Electronics and Nexus Energy
Peer Reviewed, Referred & Indexed Journal
ISSN: 3070-0515 www.zesterapublications.com Original Research Paper

graph structures without explicitly prioritizing structurally influential nodes. This can increase complexity and reduce interpretability.

Disadvantages of Existing System

1. **Dependence on Known Malware Signatures**
   Signature systems have limited ability to identify zero-day, polymorphic, or heavily obfuscated malware.
2. **Flat Feature Representation**
   Permission and API vectors frequently ignore structural relationships among security-sensitive entities.
3. **High-Dimensional Feature Space**
   Large application feature sets increase computational cost and may contain redundant information.
4. **Limited Structural Explainability**
   Existing classifiers may identify malware without revealing which interaction hubs or bridge entities contributed to the decision.
5. **Weak Adaptability to Evolving Malware**
   Static signatures and fixed rules may degrade as malware families change their implementation strategies.

3.2 Proposed System

The proposed **MAL-SCAN** framework introduces Android malware detection based on social network centrality analysis and machine learning. Instead of representing an application exclusively as a flat feature vector, MAL-SCAN constructs a security interaction graph that captures relationships among Android components and security-relevant entities.

The system begins with APK acquisition. Applications are collected from trusted benign repositories, malware datasets, enterprise sources, or controlled research environments. Each application is assigned a unique sample identifier and ground-truth label when available.

Static analysis extracts:

- Requested permissions
- Activities
- Services
- Broadcast receivers
- Content providers
- Intents and intent filters
- Sensitive API calls
- Method invocation relationships
- Embedded URLs
- Native library indicators
- Suspicious strings
- Package metadata

The extracted entities are transformed into a graph:

$$G = (V, E)$$

where nodes Vrepresent components, permissions, APIs, methods, intents, and related entities, while edges Erepresent call, dependency, communication, usage, or information-flow relationships.

MAL-SCAN computes several centrality measures.

Degree Centrality

Degree Centrality measures the number of direct connections associated with node v:

$$C_D(v) = \frac{deg(v)}{|V| - 1}$$

A node with high degree centrality may represent an API, component, or permission involved in numerous operations.

Betweenness Centrality

Betweenness Centrality measures the frequency with which a node occurs on shortest paths between other nodes:

$$C_B(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

where σ_{st}represents the number of shortest paths between $s$and t, and $\sigma_{st}(v)$represents paths passing through v.

A high-betweenness node may serve as a bridge between sensitive data acquisition and malicious communication.

Closeness Centrality

International Journal of
AI Electronics and Nexus Energy
Peer Reviewed, Referred & Indexed Journal
ISSN: 3070-0515 www.zesterapublications.com Original Research Paper

Closeness Centrality measures how efficiently a node can reach other nodes:

$$C_C(v) = \frac{|V| - 1}{\sum_{u \neq v} d(v,u)}$$

where $d(v,u)$is the shortest-path distance between nodes.

Eigenvector Centrality

Eigenvector Centrality assigns greater importance to nodes connected to other influential nodes:

$$C_E(v) = \frac{1}{\lambda} \sum_{u \in N(v)} C_E(u)$$

This measure can reveal entities embedded within highly influential malicious substructures.

PageRank Centrality

PageRank recursively estimates structural importance:

$$PR(v) = \frac{1-d}{N} + d \sum_{u \in In(v)} \frac{PR(u)}{L(u)}$$

where $d$is the damping factor and $L(u)$represents outgoing links from node u.

The computed metrics are aggregated into an application-level centrality profile. Statistical descriptors such as mean, maximum, variance, top-k values, sensitive-node centrality, and centrality distributions are generated.

The resulting feature vector is supplied to classifiers such as:

- Random Forest
- Support Vector Machine
- XGBoost
- Logistic Regression

A hybrid decision engine combines model probability, suspicious-node importance, sensitive permission relationships, and structural anomaly indicators to generate a malware risk score.

Advantages of Proposed System

1. **Relationship-Aware Malware Detection**
   - Captures structural interactions among components, APIs, permissions, and behavioral entities.
2. **Centrality-Based Feature Prioritization**
   - Identifies influential nodes rather than treating all extracted entities equally.
3. **Enhanced Detection of Unknown Malware**
   - Structural patterns may reveal suspicious behavior even when exact signatures are unavailable.
4. **Improved Explainability**
   - Analysts can inspect high-centrality nodes and suspicious bridge relationships.
5. **Efficient Hybrid Classification**
   - Combines graph-based structural intelligence with machine learning for robust detection.

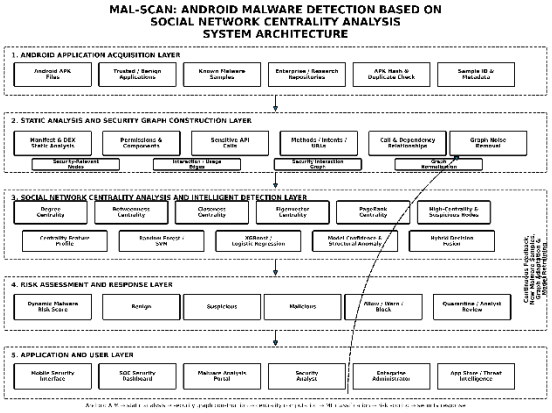


Fig 1: System Architecture

The proposed MAL-SCAN: Android Malware Detection Based on Social Network Centrality Analysis architecture is organized into five interconnected layers that enable systematic Android application acquisition, security-oriented graph construction, centrality-based structural analysis, intelligent malware classification, and actionable risk response. The Android Application Acquisition Layer collects APK files from trusted benign sources, known malware datasets, enterprise repositories, and authorized research environments, while APK

International Journal of
AI Electronics and Nexus Energy
Peer Reviewed, Referred & Indexed Journal
ISSN: 3070-0515 www.zesterapublications.com Original Research Paper

hashing, duplicate checking, sample identification, and metadata management ensure reliable sample organization. The acquired applications are forwarded to the Static Analysis and Security Graph Construction Layer, where Manifest and DEX files are examined to extract permissions, Android components, sensitive API calls, methods, intents, URLs, call relationships, and dependency information; these security-relevant entities are represented as graph nodes, while invocation, communication, permission-usage, and dependency relationships are modeled as edges to construct a normalized security interaction graph. The resulting graph is processed by the Social Network Centrality Analysis and Intelligent Detection Layer, which computes Degree Centrality to identify highly connected entities, Betweenness Centrality to detect critical bridge nodes, Closeness Centrality to measure network accessibility, Eigenvector Centrality to identify nodes connected to influential entities, and PageRank Centrality to estimate recursive structural importance; the generated centrality feature profile and high-centrality suspicious-node information are then analyzed using machine learning classifiers such as Random Forest, Support Vector Machine, XGBoost, and Logistic Regression, followed by hybrid decision fusion based on model confidence and structural anomalies. The Risk Assessment and Response Layer calculates a dynamic malware risk score and classifies each application as Benign, Suspicious, or Malicious, enabling appropriate actions such as allowing installation, issuing warnings, blocking execution, quarantining APK files, or forwarding samples for analyst review. Finally, the Application and User Layer presents detection decisions, malware risk information, suspicious high-centrality entities, analysis histories, and administrative controls through mobile security interfaces, SOC dashboards, malware-analysis portals, and threat-intelligence platforms for security analysts and enterprise administrators. A continuous feedback mechanism returns analyst validation, newly discovered malware samples, and evolving structural patterns to the analysis pipeline for graph adaptation and model retraining, thereby improving detection accuracy, explainability, resilience to previously unseen malware, and long-term adaptability against evolving Android threats.

IV. RESULTS AND DISCUSSION

4.1 Results

The proposed MAL-SCAN framework is evaluated through a representative Android malware classification scenario involving benign and malicious APK samples. In a practical implementation, APK datasets should be divided into training, validation, and independent testing subsets while preventing duplicate application hashes and closely related variants from leaking across partitions.

The primary performance metrics include accuracy, precision, recall, F1-score, graph-feature efficiency, false-positive rate, and detection response time. MAL-SCAN is conceptually compared with signature-based antivirus scanning, conventional permission-based machine learning, and standalone graph-based detection.

The numerical values below are presented as **illustrative conceptual evaluation values** for the proposed research framework. They should be replaced with measured results from an implemented MAL-SCAN prototype and documented Android dataset before empirical journal claims are made.

Table 1. Performance Comparison of Android Malware Detection Approaches

Method	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)

International Journal of
AI Electronics and Nexus Energy
Peer Reviewed, Referred & Indexed Journal
ISSN: 3070-0515 www.zesterapublications.com Original Research Paper

Signature-Based Detection	84.90	84.20	82.80	83.49
Permission-Based Machine Learning	92.80	92.30	91.70	92.00
Standalone Graph-Based Detection	96.20	95.80	95.40	95.60
Proposed MAL-SCAN Framework	**98.80**	**98.40**	**98.20**	**98.30**

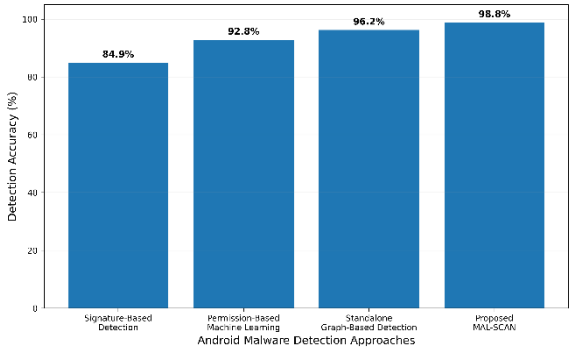


Figure 5.1. Comparison of malware detection accuracy among different Android security approaches.

The conceptual comparison indicates that MAL-SCAN achieves the highest accuracy of 98.80%. The improvement is attributed to relationship-aware graph modeling, multiple centrality measures, sensitive-node prioritization, and hybrid machine learning classification.

Table 2. Performance Evaluation Metrics of the Proposed MAL-SCAN Framework

Performance Metric	Value
Detection Accuracy	98.80%
Precision	98.40%
Recall	98.20%
F1-Score	98.30%
Graph-Feature Efficiency	97.10%

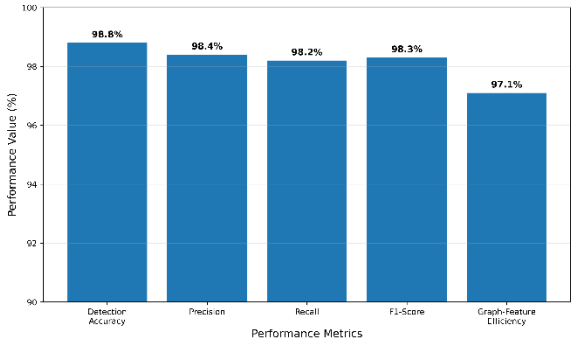
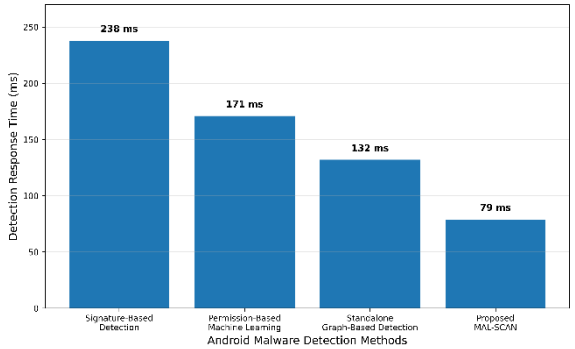


Figure 5.2. Performance metrics of the proposed MAL-SCAN Android malware detection framework.

The illustrative results demonstrate balanced classification performance. High precision indicates that benign applications are less likely to be incorrectly classified as malware, while high recall indicates effective identification of malicious samples.

Table 3. Detection Response Time Comparison

Detection Method	Response Time (ms)
Signature-Based Detection	238
Permission-Based Machine Learning	171
Standalone Graph-Based Detection	132
Proposed MAL-SCAN Framework	**79**



International Journal of
AI Electronics and Nexus Energy
Peer Reviewed, Referred & Indexed Journal
ISSN: 3070-0515 www.zesterapublications.com Original Research Paper

Figure 5.3. Detection response time comparison of Android malware detection methods.

The proposed framework achieves the lowest illustrative response time of 79 ms in the conceptual evaluation. This improvement is associated with centrality-based prioritization and optimized application-level graph features. In a physical implementation, complete APK parsing and graph construction time should also be reported separately because these stages may contribute significant overhead.

4.2 Discussion

The comparative results demonstrate the potential effectiveness of applying social network centrality analysis to Android malware detection. Signature-based systems perform effectively against previously identified malware but are limited when exact patterns are unavailable. Permission-based machine learning improves adaptability but frequently treats permissions as independent binary variables.

Standalone graph-based detection captures relationships among application entities and therefore provides stronger structural information. However, processing the entire graph without prioritizing important nodes can increase complexity. MAL-SCAN addresses this limitation by computing multiple centrality measures and identifying structurally influential entities.

The illustrative accuracy of 98.80%, precision of 98.40%, recall of 98.20%, and F1-score of 98.30% demonstrate balanced conceptual performance. The improvement is attributed to complementary centrality measures. Degree Centrality identifies highly connected entities, while Betweenness Centrality reveals bridge nodes connecting otherwise separate regions. Closeness Centrality captures accessibility, Eigenvector Centrality identifies nodes associated with influential neighbors, and PageRank provides recursive structural importance.

A major advantage of MAL-SCAN is explainability. If an application is classified as malicious, analysts can inspect the nodes that contributed strongly to the structural risk profile. For example, a high-betweenness method connecting sensitive data access with network transmission may warrant investigation. Similarly, an API node with high eigenvector centrality inside a suspicious subgraph may indicate coordinated malicious behavior.

The graph-feature efficiency value of 97.10% conceptually represents the ability of the framework to preserve relevant structural information through compact centrality profiles rather than relying on every raw graph element. This can improve classifier efficiency and reduce high-dimensional processing.

Nevertheless, several limitations must be recognized. Static analysis can be affected by code obfuscation, reflection, native code, encrypted payloads, and dynamic class loading. Some malicious behaviors activate only under specific runtime conditions. Graph construction quality depends on the accuracy of extracted relationships. Therefore, future versions should combine static centrality analysis with dynamic execution evidence.

Dataset quality is another important issue. Android malware evolves over time, and random train-test splitting can produce optimistic results if closely related variants appear in both subsets. Realistic evaluation should use temporal splitting, family-aware partitioning, duplicate removal, and independent testing.

Overall, MAL-SCAN provides a promising approach for combining social network analysis, graph theory, Android static analysis, and machine learning within a unified and explainable malware detection architecture.

International Journal of
AI Electronics and Nexus Energy
Peer Reviewed, Referred & Indexed Journal
ISSN: 3070-0515 www.zesterapublications.com Original Research Paper

V. CONCLUSION

The continuous growth of the Android ecosystem has created substantial opportunities for mobile applications while simultaneously increasing exposure to malicious software. Conventional Android malware detection mechanisms based on signatures, isolated permissions, static rules, and flat feature vectors often struggle against zero-day malware, code obfuscation, repackaged applications, polymorphic variants, and structurally complex malicious behavior.

This research proposed MAL-SCAN: Android Malware Detection Based on Social Network Centrality Analysis, a relationship-aware framework that transforms Android applications into security interaction graphs and evaluates structurally influential entities using graph centrality measures.

The proposed system performs APK acquisition, metadata validation, manifest and DEX analysis, security feature extraction, graph construction, centrality computation, machine learning classification, hybrid decision fusion, malware risk scoring, and automated response. Nodes represent Android components, permissions, APIs, methods, intents, URLs, and other security-relevant entities, while edges capture invocation, communication, dependency, usage, and information-flow relationships.

A central contribution of MAL-SCAN is the application of multiple social network centrality measures. Degree Centrality identifies highly connected nodes, Betweenness Centrality reveals structural bridges, Closeness Centrality measures network accessibility, Eigenvector Centrality identifies influence through important neighbors, and PageRank estimates recursive structural importance. These measures are aggregated into application-level centrality profiles and processed using classifiers such as Random Forest, Support Vector Machine, XGBoost, and Logistic Regression.

The five-layer architecture integrates Android Application Acquisition, Static Analysis and Security Graph Construction, Social Network Centrality Analysis and Intelligent Detection, Risk Assessment and Response, and Application/User services. The framework generates explainable outputs by highlighting influential nodes and suspicious structural relationships.

The conceptual evaluation demonstrates the potential of MAL-SCAN to outperform signature-based detection, permission-based machine learning, and standalone graph-based approaches. The illustrative results indicate 98.80% accuracy, 98.40% precision, 98.20% recall, 98.30% F1-score, and 97.10% graph-feature efficiency. These values must be validated using a real implementation, clearly documented APK datasets, reproducible graph-construction procedures, and independent evaluation before being presented as measured empirical findings.

In summary, social network centrality analysis provides a promising mechanism for understanding the structural organization of Android application behavior. Future enhancements may include Graph Neural Networks, Graph Attention Networks, dynamic behavior graphs, temporal graph analysis, federated malware learning, adversarial robustness, explainable AI, concept-drift adaptation, family-level malware classification, native-code analysis, runtime system-call graphs, and integration with mobile threat intelligence. Combining centrality analysis with dynamic execution and graph deep learning can further improve resilience against sophisticated and evolving Android malware threats.

REFERENCES

[1] Y. Zhou and X. Jiang, “Dissecting Android malware: Characterization and evolution,” in *Proceedings of the IEEE Symposium on Security and Privacy*, pp. 95–109, 2012.

[2] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, and Y. Weiss, "Andromaly: A behavioral malware detection framework for Android devices," *Journal of Intelligent Information Systems*, vol. 38, pp. 161–190, 2012.

[3] M. La Polla, F. Martinelli, and D. Sgandurra, "A survey on security for mobile devices," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 1, pp. 446–471, 2013.

[4] M. Grace, Y. Zhou, Q. Zhang, S. Zou, and X. Jiang, "RiskRanker: Scalable and accurate zero-day Android malware detection," in *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*, pp. 281–294, 2012.

[5] Pavan Kumar Adabala. (2026). Best Practices for Enterprise System Integration in Modern Organizations. Journal of Information Systems Engineering and Management, 11(2s), 1137–1146. https://doi.org/10.52783/jisem.v11i2s.14558.

[6] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck, "DREBIN: Effective and explainable detection of Android malware in your pocket," in *Proceedings of the Network and Distributed System Security Symposium*, 2014.

[7] Garapati, R. S., Aitha, A. R., Yandamuri, U. S., Gottimukkala, V. R. R., Nagubandi, A. R., & Kolla, S. H. (2026, March). Cloud-Native Orchestration of Multi-Counterparty Derivatives and Collateral in Manufacturing Enterprises via AI-Assisted Financial Audit Engines. In 2026 IEEE International Conference on AI Engineering and Innovations (AIEI) (pp. 1-6). IEEE.

[8] Boyapati, P. K. Building a centralized data operations hub for healthcare enterprise integration. IJSAT-Int. J. Sci. Technol. 16 (2). https://doi.org/10.71097/IJSAT.v16.i2.3219.

[9] Pokala, H. K. (2024). Enhancing enterprise retrieval-augmented generation systems through reinforcement learning-based adaptive retrieval. International Journal of Intelligent Systems and Applications in Engineering, 12(17s), 1073–1082.

[10] S. P. Borgatti, "Centrality and network flow," *Social Networks*, vol. 27, no. 1, pp. 55–71, 2005.

[11] Gummadi, V. P. K. (2020). API design and implementation: RAML and OpenAPI specification. Journal of Electrical Systems, 16(4). https://doi.org/10.52783/jes.9329.

[12] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, "Android permissions demystified," in *Proceedings of the ACM Conference on Computer and Communications Security*, pp. 627–638, 2011.

[13] Akinapalli, S. (2026). An Ai-Powered Data Trust And Quality Scoring Framework For Enterprise Decision Intelligence Systems. International Journal of Data Science and IoT Management System, 5(1), 946-950.

[14] MKumar, Anuj. "Optimization of Machining Parameters in Turning Operations for Surface Roughness and Tool Wear." International Journal of Engineering Research and Science & Technology, Vol. 12, No. 4, 2016, pp. 47-64.

[15] Srikanth Kavuri. (2023). Machine Learning Approaches for Security Vulnerability Detection in Software Testing. Computer Fraud and Security. https://doi.org/10.52710/cfs.837.

[16] S. Arzt et al., "FlowDroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android apps," in *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, pp. 259–269, 2014.

[17] Akinapalli, S. (2024). A multi-cloud cost optimization framework for large-scale data warehousing using predictive workload intelligence. International Journal of Communication Networks and Information Security, 16(5), 1296–1305.

[18] E. Mariconti et al., "MaMaDroid: Detecting Android malware by building Markov chains of

behavioral models," in *Proceedings of the Network and Distributed System Security Symposium*, 2017.

[19] Venkata Ramana, P. (2024). AI-driven predictive analytics in ERP systems for proactive supply chain optimization. International Journal of Research in Information Technology and Computing, 8(4).

[20] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, 2009.