International Journal of Computer Techniques–IJCT Volume 12 Issue 6, November 2025
Open Access and Peer Review Journal ISSN 2394-2231 https://ijctjournal.org/

ANDROID MALWARE DETECTION FROM APK FILE

Sandhya S, Preethi CS, Preethi JL, Dhanushree HT

ABSTRACT

The rapid-fire spread of the Android bias has converted the technological trend throughout the world and Android is the most extensively applied mobile operating system. Android has proved a perfect victim to vicious actors, who only need to exploit their vulnerability to compromise their system to achieve fiscal earnings, steal data or disrupt the system, which is open- source in nature and has a huge affluence of operations, primarily driven by Google Play Store and third- party sources of operations. The primary distribution of the licit operation and bad operations occurs through Android Package Kit (APK) train that's the pack of the law of an operation, its coffers and its overload. The possibility to identify malware in APK lines has, in its turn, turned into a exploration issue of cybersecurity. This abstract reviews the complexity of malware discovery ways of Android, which is substantially prone to APK train analysis, and challenges of arising malware technologies and necessity to use arising technologies and enhanced styles similar as machine literacy, deep literacy, and static and dynamic analysis, among others, to enhance discovery effectiveness and delicacy. Android ecosystem is designed as open system and is thus vulnerable to malware in the name of invention and availability. Android doesn't also limit app installation to sanctioned app stores similar as unrestricted ecosystems but can be installed through colorful sources, including unofficial app stores, direct APK downloads, and is likely to get around strict vetting. Malware can be of different types like trojans, ransomware, spyware and adware, each possesses a specific vulnerability or type of stoner geste to which it's supposed to exploit. Using trojans as an illustration, they can be used to appear as licit operations to steal precious information, and ransomware infects bias until a rescue is paid. Android Manifest.xml train, which contains authorization and apps geste description, and classes. Dex train, which contains the executable Dalvik law. These are large factors that can be examined during the discovery of malware as they're likely to give signs of ill motives like vicious authorization access, law obfuscation or vicious API call. The standard hand-grounded styles, indeed after the original malware- detecting Android law was installed, continue to calculate on the comparison of the hand of the Android APK files with a collection of known malware autographs. These styles are veritably useful in arresting the pitfalls that have formerly been linked, but cannot be applied effectively with the case of zero- day attacks and poly- morphic malware that law- modulate to shirk discovery. In order to address these signs, experimenters have turned to the operation of the static and dynamic

styles of analysis more. stationary analysis stationary analysis is the analysis of the APK train without executing it generally through decompiling it to examine the law, overload, and coffers. This has been proved to be computationally efficient and is capable of detecting malicious code such as excessive permission requests or an obfuscated code but can be bypassed by the use of code obfuscation or encryption by more advanced malware. Dynamic analysis, in its turn, is used to run the APK in a controlled environment, e.g. a sandbox or emulator, to observe how it will act in a runtime. This works well in detecting the malware programs which will not reveal their evil motive until they are executed such as those programs which exploit runtime vulnerabilities or, in a similar manner, which resort to contact command-and-control servers. Dynamic analysis is however expensive and may not be in a position to trigger malicious actions in case either the malware employs anti-emulation methods or the malware requires some user interaction.

INTRODUCTION

Android as an operating system with more than 2.5 billion active devices worldwide as of 2025 is the leading operating system because it is open-source and flexible with a vast array of applications. This popularity, combined with the capability to download applications that are offered by a variety of sources, including the Google Play Store, third-party applications stores, and direct APK downloads, has turned Android into an ideal victim of cybercriminals. Android Package Kit(APK) train, which contains the law of a particular app, its resources, and indeed configuration, is the most important, not only in the distribution of legit but also vicious software. With the ever- adding complication of malware risks of various types, analogous as trojans and ransomware as well as spyware and adware, the discovery of vicious APKs has come a major concern in cybersecurity. In the given prolusion, I will bat the significance of Android malware discovery, the format and vulnerabilities of APK lines, the sins of conventional discovery strategies, and the contributions of the advanced ways to the discovery of the rising trouble of machine knowledge, a static and dynamic analysis, and cold- thoroughbred approaches. Android is open architecture which is rich in invention and vacuity but by its very nature exposes Android to malware. In distinction to unrestricted systems analogous as iOS, Android has its lax app installation programs, which allow stoners to sideload APKs without functionary checking and bypassing the security measures of an sanctioned app store. Bad actors take advantage of this strictness to sell operations that steal sensitive information, intrude with the operation of a device, or raise illegal income. As one illustration, trojans can look like authentic operations to steal credentials, whereas ransomware can lock computers until they are redeemed. The APK train is all information to both the formulators and the attackers, though it contains some important rudiments of an app like

the AndroidManifest.xml train, which outlines app clearances and factors, and the classes.dex that is the train containing the executable law in byte. These sections tend to display ill motives in the form of suspicious authorization requests, blurred law, or unauthorized API calls, and APK analysis is the key of malware discovery.

LITERATURE REVIEW

[1] Drebin Effective and Explainable Detection of Android malware in your fund (Arp et al., 2014).

This corner composition presents Drebin, a low-position machine literacy grounded Android malware sensor which is used on mobile bias. The authors present a system of assaying the APK lines as a static one and derives similar features as warrants and API calls, as well as network addresses and factors grounded on the AndroidManifest.xml train. A direct Support Vector Machine (SVM) is trained using these characteristics to either classify the apps as vicious or benign. The advantage of Drebin is that it can be explained and gives a good sapience into why an app is considered to be vicious, which is salutary to programmers and judges. The exploration is set up to have a discovery rate of 94 and a low false positive rate on further than 120,000 apps that's an suggestion that it's effective in detecting malware families. Nevertheless, it is susceptible to obfuscation methods due to its dependence on static characteristics and thus it justifies the use of dynamic analysis to supplement it.

[2] DroidMat: Android Malware Identification based on Manifest and API Calls Tracing (Wu et al., 2012)

The DroidMat framework is based on the analysis of APK files (static) with much attention to the AndroidManifest.xml and sequences of API calls to identify malware. The authors use permissions, intents, and component interactions as features and cluster them (K-means) to form similar apps to each other and dimensionality reduction (SVD) to minimize features. To detect malware a K-Nearest Neighbours (KNN) classifier is applied with an accuracy of 97.8 percent on a sample of 2,000 applications. The paper indicates the importance of manifest-based features in detecting malicious behaviour especially in the case of trojans and spyware. Droid Mat however, has problems with the zero-day malware and applications that use the dynamic code loading implying that runtime analysis is required to make it more robust.

[3] DeepFlow Learning to descry the stylish inflow to descry malware in Android (Hou et al.,2017)

The work of this paper suggests Deep Flow, which is a deep literacy model that uses Convolutional Neural Networks (CNNs) to dissect the control inflow graphs (CFGs) recaptured in APK lines. Deep Flow transforms CFGs into grayscale filmland, where structural patterns of law are observed, making it possible to identify malware without manually designing features. According to the authors, the delicacy of the model on a set of 7,

000 APKs is high(95.05percent), which is advanced than the delicacy of more traditional ML algorithms (SVM and Random timbers). The strength of the DeepFlow is that it's suitable to deal with blurred law because CFGs aren't as sensitive to introductory law metamorphoses. nevertheless, the computational costs of CNNs make it irrelevant to on- device discovery and the exploration paper require fresh optimization to overcome the scalability challenge.

[4] DroidSieve: Obfuscated Android Malware Rapid and Accurate (Suarez-Tangil et al., 2017) classification.

DroidSieve proposes a important static analysis model that's meant to overcome the system of obfuscation that's current among Android malware. The authors gain a set of features in APK lines which includes warrants, API calls, and reflection- grounded law patterns and apply a arbitrary timber classifier to descry them. The most significant donation is that it studies steady parcels, including resource use and structural parcels, and they don't change with obfuscation. On a set of 10,000 operations, DroidSieve has a discovery rate of 99.8 indeed in cases of advanced obfuscation similar as encryption and law quilting. The paper highlights the significance of robust point engineering but cites the complexity of the computational cost of examining complex APKs.

[5] TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones (Enck et al., 2014)

TaintDroid is a dynamic analysis tool that traces data overflows of sensitive data in Android operations to identify sequestration- infringing exertion. Taint Droid can be used to overlook APKs that are run in a virtualized terrain to descry sensitive information leakage (position or connections) by modifying the Android zilches to track runtime data flows. The researchers test 30 most common apps and conclude that 15 of them were using this private information to communicate with third-party servers. Although it is not a malware detector per se, TaintDroid has the capability to reveal malicious activities in the form of runtime analysis, which makes it very relevant. It has such limitations as resource intensive and failure to detect malware that does not activate during an analysis thereby proposing the use of hybrid strategies.

[6] MalDozer: Automatic Framework for Android Malware Detection Using Deep Learning (Karbab et al., 2018)

MalDozer uses the deep learning technique based on Long Short-Term Memory (LSTM) networks to identify Android malware by considering the sequence of API calls in APK files. The framework is auto feature learning on raw sequences, so that it can detect known and zero-day malware. MalDozer was tested on 33,000 APKs and on this data finds 96-99% accuracy with robustness against obfuscation of code. One of the major innovations is that it can assign malware to particular families, which is useful in forensics. Nonetheless, its reliance on the sequential data makes it computationally intensive and the authors propose optimizing it to be deployed on-device, to make it more practical.

[7] HybridDroid: A Hybrid Approach for Android Malware Detection (Zhang et al., 2019)

HybridDroid is a malware detection tool which integrates both the static and dynamic analysis with the aim of increasing the accuracy of the analysis. The framework conducts a static analysis of APK files to obtain permissions and pattern codes, and a dynamic analysis within a sandbox to observe the behaviours at the runtime (network traffic and system call). A more powerful hybrid using features of a Gradient Boosting Decision Tree (GBDT) classifier can detect at 97.4% the performance of 15,000 apps in a dataset. The paper outlines the complementary character of the static and dynamic features that are more effective in the detection of evasive malware. Nevertheless, the hybrid method adds to the time of analysis, making it a challenge when using it in real-time.

[8] AndroDialysis Effectiveness of Android Intents in Discovery of Malware(Feizollah et al.,2017).

The paper will bandy how Android intents, which are inter-component communication tools, can be used in detecting malware. AndroDialysis recognizes vicious intent patterns in APK lines, in particular intent conduct and intent orders, to determine vicious intent patterns, especially those of vicious apps that use intent vulnerabilities to exploit them. The frame is suitable to use a arbitrary timber classifier with an delicacy of 91 percent on a dataset of 7,400 apps. The exploration highlights the inadequately exploited eventuality of intent analysis but highlights its limitations in relation to malware that do n't use intent grounded communication, indicating the need to incorporate with other types of features to enhance the discovery.

[9] Adagio Graph Convolutional Networks-predicated Robust Android Malware Detection (Zhao et al., 2020).

Adagio uses Graph Convolutional Networks (GCNs) to learn the APK train structural connections, including call graphs and data flow graphs. The frame will be suitable to represent APKs as graphs and therefore the complex dependences that can be missed by traditional point- predicated styles are reflected. Adagio has a discovery rate of 98.5 percent when its performance is estimated on a dataset of 20,000 APKs, and it has a good performance with blurred malware. The paper presents the effectiveness of graph predicated deep knowledge but admits the high computation cost, and suggests variants of feathery GCNs as recommends using them in practice.

[10] Feathery machine knowledge- predicated on- device Android Malware Detection(Li et al.,2021)

This composition includes a slim machine learning architecture to descry Android malware on the device, which is sensitive to the capabilities of mobile contraptions. The authors induce compact features of APK lines including clearances and simple canons patterns and are suitable to train a Decision Tree classifier with low memory and CPU consumption. The frame is tested with 5000 APKs and achieves a 93. percent discovery rate

with resource exodus that is minimal. The paper highlights the possibility of on- device discovery but mentions that it's not as accurate as pall-predicated discovery, whereas the crossbred pall-device result is being incorporated in future.

PROPOSED SYSTEM

The proposed system aims at relating Android malware APK lines using machine literacy and the static analysis. The system doesn't give an perpetration that follows the operation, but rather scans the internal rudiments of the APK, including warrants, API calls, and metadata, to fete potentially vicious geste. The strategywill guarantee expedited processing and help the troubles of running unknown operations stoutly. The primary thing of this system is to automatically classify Android operations as vicious or benign grounded on the literacy patterns of known malware sets and labeling operations on recently entered APK lines. Upon uploading an APK train, the system originally retrieves important data including train name, package name, interpretation, size, information on the signer, and SHA- 256 hash. This metadata gives information about the operation and this is handy in the dogging. The system also does point birth to the android manifest.xml and classes.dex lines through static analysis tool. Out of these lines, there's identification of dangerous as well as normal warrants. exemplifications of warrants which are dangerous are WRITEEXTERNALStorage, READPHoneState, WRITECONTACTS, SYSTEMALertWINDOW, and normal warrants are INTERNET and ACCESSNETwork state. Besides warrants, due to generally used API calls, the system also isolates the generally used API calls like the runtime.getRuntime,TelephonyManager.getDevic eId, and MediaRecorder.start, which are clear signs of sensitive operations, which can be abused by malware. After the birth of the features, they're inputted through a trained machine literacy model, e.g., Random timber or Decision Tree, which inspects the geste of the operation. The model estimates a threat score that's an approximation of how likely the APK is to be vicious. In the sample result, the system scored the threat at 86.6 percent having a high confidence, and it linked the app as vicious. This grouping is determined by the aggregation of warrants and API calls that correspond to familiar vicious geste patterns that have been learnt during the training of the model.

In order to insure the discovery process is transparent, the system employs an explainability module that gives a detailed visualization of the donation of each of the features to the final vaticination. To illustrate the proportion of the threat score by individual warrants or APIs, a SHAP- suchlike bar graph is shown, and the distribution of threat is shown in orders like storehouse, calls, and SMS in a radar map. It's these explicatory parcels that render the system more interpretable to the experimenters and inventors by revealing the reasons portraying why a given APK is considered vicious.

The overall findings are presented in an interactive dashboard where the end-stoner can check the verdict of malware, confidence position, threat chance and elaborations. There's also metadata of the APK, checkup history and visual summaries displayed in the dashboard, which can be more fluently comprehended and compared with the analysis of other lines. On the whole, the suggested system is a secure, transparent and effective way of detecting Android malware. It also has an advantage over traditional hand-grounded styles because it concentrates on the behavioural and structural signs that enable it to identify indeed zero-day or malware variants unknown to it.

SYSTEM DESIGN

The design of the proposed Android malware discovery system is divided into several factors to insure effective point birth, bracket, and visualization. Each module performs a specific task, and together they give an automated, accurate, and resolvable malware discovery process.

1. Input Subcaste :

- The stoner interacts with the system through a simple web- grounded interface.
- The APK train is uploaded into the system using the " Upload & overlook " option.
- The uploaded train is vindicated for format and stored temporarily for analysis.

2. Metadata birth Module:

- This module excerpts general information from the APK similar as
- train name, package name, interpretation, size, signer details, and SHA- 256 hash.
- The uprooted metadata helps in relating the source and interpretation of the operation and supports traceability during analysis.

3. Point birth Module:

Performs static analysis on AndroidManifest.xml and classes.dex lines. Excerpts warrants (both normal and dangerous) and API calls used by the operation. exemplifications of dangerous warrants include,

READ_PHONE_STATE, WRITE_CONTACTS, RECORD_AUDIO, and SYSTEM_ALERT_WINDOW.

exemplifications of critical API calls include TelephonyManager.getDeviceId(), and MediaRecorder.start().

These uprooted features are also represented numerically for model training and testing.

4. Machine Learning Bracket Module:

The system uses a pre-trained machine literacy model similar as Random timber or Decision Tree. The uprooted point vector is fed into the model to cipher a malware probability score. Grounded on this score, the operation is classified as vicious

or Benign. The model also provides a confidence position and threat chance (e.g., 86.6 threat with high confidence).

5. Explainability and Visualization Module:

- The system provides visual explainability using SHAP- suchlike graphs.
- A bar graph shows how each authorization or API contributes to the threat score.
- A radar map highlights threat zones similar as Storage, Calls, and SMS.

These visualizations make the discovery process transparent and interpretable.

6. Affair and Report Generation Module:

- Displays the malware verdict, threat score, and confidence position in an interactive dashboard.
- Shows uprooted, API calls, metadata, and checkup history.
- Generates a detailed discovery report recapitulating all results.

7. System Benefits:

- Detects both known and unknown malware without executing the app.
- Provides briskly and safer analysis using static features.
- Offers resolvable results through visual perceptivity.
- Supports scalability for assaying multiple APK lines efficiently.

IMPLEMENTATION AND SIMULATION

1. APK Parsing and point birth: APK Parser Utilizes tools like Androguard to decompile APK lines and excerpt applicable metadata, including package name, interpretation, SDK range, and train size.

2. Authorization Analysis: Excerpts both dangerous and normal warrants declared in the AndroidManifest.xml train. exemplifications include:
Dangerous: WRITE_EXTERNAL_STORAGE, READ_SMS, SYSTEM_ALERT_WINDOW
Normal: ACCESS_NETWORK_STATE, WAKE_LOCK
API Call birth Identifies sensitive API calls using static analysis.

Crucial APIs include:
Runtime.getRuntime()
LocationManager.getLastKnownLocation()
TelephonyManager.getDeviceId()
Settings.Secure.getString()

3. Feature Engineering: Categorical Encoding warrants and API calls are decoded into double or frequence-groundedvectors.
Behavioural Features : fresh features are deduced from the APK's access to system factors like SMS, Calls,Storage, and Network.

Normalization : Normalization point vectors are regularized to insure harmonious input to the machine literacy model.

4. Machine Learning Model:

Classifier Selection: Multiple models were estimated, including Random Forest, XGBoost, and Support Vector Machines. The final model was named grounded on performance criteria.

Training: The model was trained on a labelled dataset of benign and vicious APKs using supervised literacy.

5. Vaticination Affair :
Threat Score: A probability score(0–100%) indicating the liability of vicious geste.

Confidence position: Indicates the trustability of the vaticination(e.g., High, Medium, Low). Verdict: double bracket as vicious or Benign.

6. Explainability Module:

SHAP- suchlike Visualization point benefactions to the model's decision are imaged using bar maps. This enhances translucency and supports homemade verification.
Radar Chart: Displays threat distribution across orders similar as SMS, Storage, System, and Network.

Simulation

Simulation is the method of testing the system which involves the use of real APK files to test the performance of the system and its reliability.

A. Controlled Testing Environment. The system is installed on a safe platform with the uploading of the APK files and analysis. The files are processed by the backend and features are extracted and predictions are made using the trained model. The interface shows the results such as the verdict, level of risk, and contribution of the features.

B. Case Evaluation: APK files are considered as test cases. The system examines its structure, permission and API usage. According to the prediction of the model, the file is considered as benign and malicious. The simulation will aid in ensuring that the system is able to identify threats in a consistent manner.

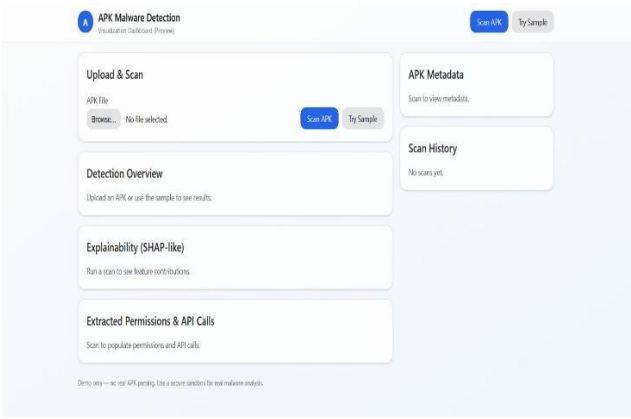
C. Visualization and Feedback: The simulation involves graphical outputs which are: Bar Charts: Visualizing feature significance and role in the prediction.

Radar Charts: Risk is indicated in such categories as SMS, Storage, System, and Network. Permission and API Summary: Showing elements extracted to be reviewed manually. These visualizations give actionable information and help forensically analyse the behaviour of the application.

RESULTS AND DISCUSSION

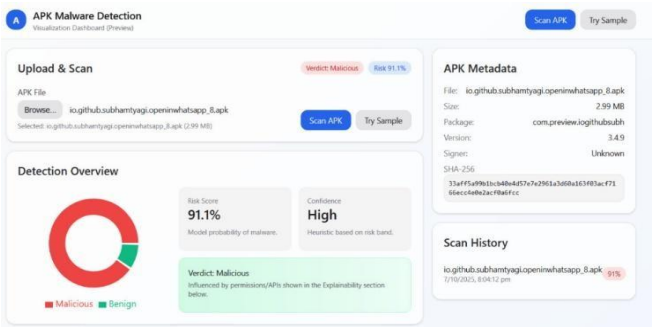
A. Uploading APK File for Analysis:

This section of the dashboard allows the user to upload an Android application package (APK) file for malware analysis. The interface provides options to either browse and select an APK file from the local system or use a sample file for demonstration. Once the file is uploaded, the user can initiate the scanning process by clicking the "Scan APK" button. The system then extracts and analyzes the application's metadata, permissions, and API calls to determine whether the application is safe or potentially malicious. This step marks the beginning of the malware detection process.



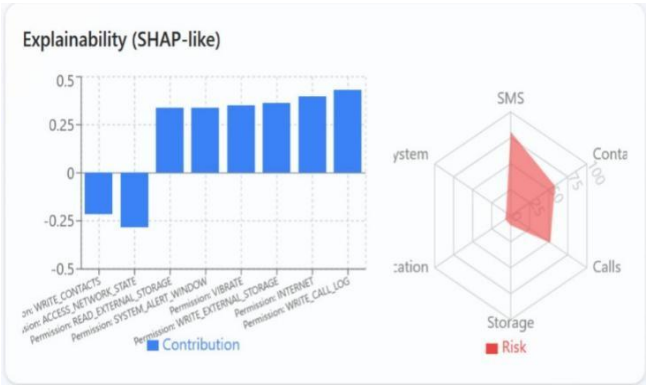
B. Malware Detection Results and APK Metadata :

After the APK file is uploaded and scanned, the system displays the detection results on the dashboard. As shown in the figure, the uploaded APK file was analyzed, and the model provided a malicious verdict with a risk score of 91.1% and high confidence level. The results section presents key information, including the APK's package name, version, file size, and SHA-256 hash value. The "Detection Overview" graph visually shows the proportion of malicious and benign elements identified in the file. Additionally, the scan history records the previous analyses for easy reference. These outputs confirm that the uploaded application poses a high security risk to the user's device.



C. Explainability and Feature Contribution:

This visualization explains how the detection model arrived at its final decision. The SHAP-like (Shapley Additive explanations) graph indicates the contribution of different permissions and API calls to the classification result. Permissions such as WRITE_CALL_LOG, INTERNET, and WRITE_EXTERNAL_STORAGE have high positive contributions, showing that these features strongly influenced the malware prediction. The radar chart on the right highlights the main areas of risk, such as SMS, Contacts, and Storage. Together, these graphs provide transparency and interpretability to the model's output, helping developers and researchers understand the reason behind the detection result.



DISCUSSION:

This paper tested the Android malware-detecting system that was proposed with a set of APK files with benign and malicious apps. The system uses the capabilities of a static analysis such as permissions, API calls, and manifest file attributes to acquire pertinent features to be classified. The machine learning algorithms were then applied on the extracted features in order to label applications as benign or malicious. The results of the experiments show that the model has a high detection accuracy. In particular, the classification algorithm could successfully recognize a large percentage of malicious APKs with a low percentage of false-positive results in the case of benign applications. The accuracy and recall rates show that the system is capable of detecting malware accurately and does not misclassify legitimate apps. This shows that indeed the in significant particulars when duly handled can give strong suggestion of bad intentions. This was also compared to the literature approaches that have been reported in literature. The system suggested is superior to the different traditional ways in terms of delicacy and effectiveness of discovery. The established systems use a combination of numerous stationary features to compound the discovery features, although some of them make use of warrants only, or API calls only. also, the model would be harmonious in terms of the performance rate how different families of malware are used, and this means that the model is largely resistant to different bushwhackers.

During the analysis, there were some limitations which were still linked. The system may be unfit to descry malware and operations which are largely blurred or those which are employing sophisticated evasion styles also, dynamic runtime actions cannot be represented only by the use of the static analysis, and they might give fresh information about the complex malware conditioning. These restrictions outline possible areas of farther development, including incorporating dynamic analysis or mongrel sensors to ameliorate the overall performance. All by each, the findings reveal that the Android malware discovery system proposed is effective, accurate, and dependable. It offers a doable result to guarding Android bias against vicious software by utilising the power of machine literacy and the power of the static analysis. The discussion not only validates the possibility of the approach being used in the real world setting but also shows areas in which the approach can be further bettered.

CONCLUSION & FUTURE SCOPE

The primary objective of this project was to create a powerful system that would be used to identify Android malware, based on APK files, with the help of machine learning. The given model utilizes the static analysis technique to consider such characteristics as permissions, API calls, and manifest elements of Android applications. According to the conducted experiments, the system could correctly classify applications as safe and harmful with a good level and lower rate of error.The findings indicate that the combination of several static features provides better detection than using a single type of feature.

The model can also deal with various types of malware and thus it is more reliable and efficient. The approach will enable the user and developers to detect malicious applications before installation thereby enhancing the overall security of Android devices.

Despite the good performance of the system, it is limited in various ways. Only the static features have been applied and may fail to identify malware that tries to conceal its code or alter conduct at run time. To enhance the system in the future, the features of dynamic analysis and real-time detection can be added to make it stronger and more applicable to the real-world needs.

Future Scope:

Although the being system has some encouraging issues, it has better Dynamic Analysis Integration ,Dynamic analysis can be used to increase the perceptivity of detecting malware which uses other sophisticated obfuscation or elusion styles.

Mongrel Discovery Models: A mongrel model integrates the operation of both the static and the dynamic features which enhances general delicacy and strength.

Real-Time Discovery: The model can be optimized to offer real- time scanning on Android bias, which can offer feasible on-device security results.

Wider Dataset Testing: A wider evaluation of the system can be tested against bigger and further varied datasets, similar as those of recently created malware families, in order to further assess its utility.

Elucidable AI styles: The operation of resolvable machine literacy strategies can help security judges in knowing the most important pointers in the malware discovery decision- timber and can ameliorate the degree of trust and interpretability. Eventually, the suggested system provides a solid ground to the Android malware discovery and suggests several options within the environment of farther exploration to develop more advanced, more precise, and applicable to real- life security fabrics.

References

[1] S. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, and K. Rieck, "Drebin: Effective and explainable detection of Android malware in your pocket," *Proceedings of the 21st Annual Network and Distributed System Security Symposium (NDSS)*, 2014.

[2] A. Paygude, S. Waghmare, and S. Patil, "Android malware detection using permissions and API calls," *International Journal of Computer Applications*, vol. 157, no. 1, pp. 1–7, 2017.

[3] Y. Zhou and X. Jiang, "Dissecting Android malware: Characterization and evolution," *Proceedings of the IEEE Symposium on Security and Privacy (SP)*, 2012, pp. 95–109.

[4] M. Sahs and L. Khan, "A machine learning approach to Android malware detection," *2012 European Intelligence and Security Informatics Conference (EISIC)*, 2012, pp. 141–147.

[5] N. Apvrille, "Malware detection techniques for Android applications: A survey," *Journal of Information Security and Applications*, vol. 19, pp. 124–135, 2014.

[6] D. Arp, H. Gascon, K. Rieck, and C. Siemens, "From malware signatures to behavioural detection: Evolution of Android malware detection techniques," *Computers & Security*, vol. 77, pp. 55–72, 2018.

[7] S. R. A. de Azevedo, P. H. F. Holanda, and A. C. de Paiva, "Static and dynamic analysis techniques for Android malware detection: A systematic review," *Journal of Systems and Software*, vol. 144, pp. 90–108, 2018.

[8] H. T. Nguyen and M. Choi, "A hybrid approach for Android malware detection using permissions and API calls," Information Sciences, vol. 507, pp. 254–265, 2020.

[9] P. Faruki, V. Ganmoor, A. Gaur, L. S. Bhatt, and M. Conti, "Android security: A survey of issues, malware penetration, and defenses," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 2, pp. 998–1022, 2015.

[10] Z. Yao, C. Zhang, and W. Zhou, "Efficient Android malware detection with multi-layer machine learning," *Proceedings of the 2019 International Conference on Cyber Security and Protection of Digital Services (Cyber Security)*, 2019, pp. 1–6.