Android Malware Detection Using Optimal Ensemble Learning Approach For Cyber Security

P. Pallavi[1], P. Harshitha Syamala[2], P. Sanjay Kumar[3], M. Siva Manikanta[4], Mr. K. Sudhakar [5]
[1,2,3,4] *Department of Information Technology, Vishnu Institute of Technology*
[5]*Assistant professor, Department of Information Technology, Vishnu Institute of Technology*

Abstract: **Malware attacks have increased due to the growing use of Android devices, endangering user data and system integrity. Identification of malware using machine learning can analyse large datasets and identify patterns that point to malicious behaviour, it has become increasingly popular. To increase detection accuracy and reduce false positives, this study suggests an ideal ensemble learning strategy that makes use of the advantages of several classifiers. The model uses methods for feature extraction, selection, and classification that are tailored for the detection of Android malware.**
The effectiveness of the suggested system in recognising different malware types is demonstrated by experimental results, guaranteeing strong cyber security.

Index terms**: Android Malware Detection, Cyber Security, Optimal Ensemble Learning, Machine Learning .**

I.INTRODUCTION

Android devices have become a prime target for cyberattacks due to their explosive growth. Malware presents serious risks, such as system compromise, financial loss, and data theft. Conventional malware detection techniques based on signatures find it difficult to stay up with changing threats. A promising substitute is offered by machine learning techniques, which make it possible to identify novel and unidentified malware. However, issues like high false positives, computational overhead, and malware's dynamic nature call for creative fixes. In order to improve detection performance and adaptability to emerging threats, this paper presents an optimal ensemble learning approach that combines the advantages of several models.

Key Features and Capabilities:

1. Malware Analysis: The system utilizes static, dynamic, and hybrid analysis methods to perform comprehensive feature extraction, ensuring a detailed understanding of Android malware behaviour.

2. Optimal Ensemble Learning: Diverse classifiers are combined using advanced ensemble techniques to enhance accuracy and significantly reduce false positives.

3. Scalability: The system is appropriate for real-time malware detection applications since it is built to manage big datasets effectively.

4. Feature Optimization: To improve model performance and lower computational overhead, feature selection techniques are used to determine which attributes are most pertinent for classification.

5. Robust Detection: The system effectively identifies zero-day and polymorphic malware, ensuring robust security against evolving threats.

6. Real-Time Processing: Optimized models and efficient algorithms enable real-time detection of malicious behavior in Android applications.

7. Multi-Domain Usability: Applicable across various cyber security domains, the system can be integrated into both enterprise-level and personal Android security solutions.

8. Enhanced Accessibility: The lightweight design ensures accessibility and deployment feasibility on resource-constrained Android devices.

9. Error Minimization: The ensemble learning approach minimizes errors associated with individual classifiers, delivering consistent

and reliable detection results.

10. Scalability and Adaptability: The architecture is scalable to accommodate increasing malware datasets and adaptable for future enhancements, such as integration with threat intelligence platforms and support for emerging malware types.

II. LITERATURE REVIEW

Android devices have become a popular target for malware attacks due to their rapid proliferation, which poses serious risks to user privacy, data integrity, and device functionality. Machine learning has become a viable substitute in recent years, thanks to its capacity to recognise intricate patterns and identify threats that were previously unidentified. Nevertheless, stand-alone machine learning models like Neural Networks, Decision Trees, and Support Vector Machines (SVM) frequently have low adaptability and high false-positive rates. The ability of ensemble learning techniques, like Random Forest, Gradient Boosting, Hist Gradient Boosting, and XGBoost, to combine multiple classifiers and increase detection accuracy and robustness has been investigated as a solution to these drawbacks. By putting forth an ideal ensemble learning framework that combines static, dynamic, and hybrid learning, this study aims to get past these obstacles offering a comprehensive, scalable, and effective solution for Android malware detection.

1. Traditional Approaches: Signature-based methods, such as those employed by conventional antivirus software, rely on known malware signatures for detection. While effective for previously identified threats, these methods fail to adapt to rapidly evolving malware, rendering them ineffective against zero-day and polymorphic attacks.

2. Machine Learning in Malware Detection: Methods such as Neural Networks, Support Vector Machines (SVM), and Decision Trees have been extensively employed in malware detection. These methods have proven to be effective in spotting intricate patterns in malware behaviour. They frequently have high false positive rates, though, which reduces their dependability in practical applications.

3. Ensemble Learning: To increase overall detection accuracy, ensemble learning techniques like Random Forest, Gradient Boosting, and XGBoost combine several classifiers. These models minimise the drawbacks of individual classifiers while utilising their advantages. Ensemble approaches have demonstrated potential for improving malware detection systems' resilience and lowering false positives.

4. Challenges in Existing Systems: Despite advancements, existing malware detection systems face several challenges:
    a. Lack of Adaptability: Limited ability to detect new and unknown malware types.
    b. High Computational Costs: Resource-intensive models are unsuitable for real-time malware detection.
    c. Limited Accuracy: Difficulty in identifying complex, obfuscated, or polymorphic malware.

5. Proposed Solutions: Addressing these challenges requires the integration of ensemble learning techniques with optimized feature selection and hybrid analysis approaches. This study aims to develop a scalable, adaptable, and efficient system that overcomes the limitations of traditional and current machine learning-based solutions.

III. EXISTING SYSTEM

The majority of the current Android malware detection systems rely on conventional techniques like standalone machine learning models or signature-based detection, which present a number of difficulties in terms of computational efficiency, accuracy, and adaptability. The main characteristics and drawbacks of the current methods for malware detection on Android platforms are highlighted in this section.

1. Static Analysis
    - Features: Relies on extracting features such as permissions, API calls, and manifest file attributes from APK files. This method is lightweight and does not require executing the app, making it faster and resource-efficient.
    - Limitations: Highly vulnerable to

obfuscation techniques employed by malware developers, which can mask critical features and bypass detection.

2. Dynamic Analysis
   - Features: Observes app behavior in a controlled sandbox environment, monitoring runtime actions such as network activity, file modifications, and API usage. This approach provides insights into the actual behavior of the application.
   - Limitations: Time-consuming and resource-intensive, as it requires setting up and maintaining a sandbox. It also struggles to scale for large datasets or real-time detection needs.

3. Hybrid Analysis
   - Features: Combines static and dynamic methods to leverage the strengths of both approaches, improving accuracy and robustness.
   - Limitations: High detection computational overhead due to the integration of both methods, making it less suitable for resource-constrained environments or real-time applications.

Identified Challenges in Existing Systems

- Difficulty in adapting to obfuscated and polymorphic malware.
- High computational costs for dynamic and hybrid analysis methods.
- Limited scalability for real-time malware detection in resource-constrained settings.

IV. PROPOSED SYSTEM

The proposed solution introduces a comprehensive and efficient Android malware detection framework, addressing the limitations of existing systems through the integration of optimal ensemble learning and advanced analysis techniques.

Key Components:

1. Static, Dynamic, and Hybrid Analysis: Static features (e.g., permissions, API calls) and dynamic behaviors (e.g., network activity, file access) are extracted and combined for robust detection.
2. Optimal Ensemble Learning: Utilizes advanced ensemble methods like Random Forest and XGBoost to combine diverse classifiers, enhancing accuracy and reducing false positives.
3. Scalable Architecture: Designed to handle large datasets and real-time applications efficiently, enabling deployment in both enterprise and resource-constrained environments.
4. Feature Optimization: Reduces computational overhead while preserving high accuracy by using feature selection techniques to find the most pertinent attributes.
5. Accessibility and Usability: Lightweight design ensures usability on Android devices with limited resources, making the system practical for widespread adoption.

Advantages Over Existing Systems:

1. Enhanced Detection Accuracy: Combines static, dynamic, and hybrid analysis methods with optimal ensemble learning, significantly reducing false positives and improving the overall accuracy of malware detection.
2. Resilience to Obfuscation: By leveraging dynamic analysis and hybrid approaches, the system can detect obfuscated and polymorphic malware that traditional signature-based and static analysis methods often fail to identify.
3. Real-Time Capability: Optimized architecture and ensemble learning techniques enable real time malware detection, overcoming the computational delays commonly associated with dynamic and hybrid analysis methods.
4. Scalability: Since the system can effectively manage big datasets, it can be used for enterprise-level applications and widespread deployment without sacrificing performance.
5. Feature Optimization: Feature optimisation lowers computational overhead while

preserving strong detection capabilities by utilising sophisticated feature selection techniques to determine the most pertinent attributes for classification.

6. Robustness Against Zero-Day Threats: The integration of static, dynamic, and hybrid analysis ensures comprehensive coverage, enabling the system to detect zero-day and previously unseen malware effectively.

7. Reduced Computational Overhead: By optimizing the combination of static and dynamic methods and applying feature optimization, the system minimizes resource consumption compared to traditional hybrid analysis techniques.

8. Improved Accessibility: Lightweight design ensures compatibility with resource-constrained environments, including low-end Android devices, facilitating widespread adoption.

9. Customizable and Adaptive: The system is kept current and effective against changing malware trends thanks to its modular design, which makes it simple to integrate new features and adjust to new threats.

10. Lower False Positive Rate: Ensemble learning methods mitigate the weaknesses of individual classifiers, providing a more balanced and reliable detection framework compared to standalone machine learning models.

11. Comprehensive Analysis: The system provides a deeper understanding of malware behaviour by combining static signatures with runtime behaviour analysis, offering a holistic detection approach not commonly found in existing systems.

Proposed Workflow:

1. Data Collection: Collect features manually from Android APK files using static, dynamic, and hybrid analysis methods.
2. Feature Extraction: Extract relevant features such as API calls, permissions, network activity, and file operations.
3. Ensemble Learning: Combine diverse classifiers (e.g., Random Forest, GradientBoost, XGBoost) to improve detection accuracy.
4. Detection and Classification: Classify APKs as benign or malicious based on the optimized features and classifier outputs.
5. Real-Time Analysis: Process and analyse malware behaviour in real-time for immediate threat detection.

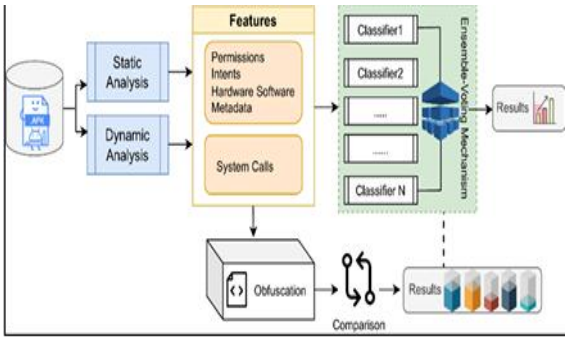

Fig.1. Project Workflow

Applications:

1. Android Malware Detection: Detect and classify Android malware in apps to protect users from potential threats.
2. Enterprise Security: Enhance mobile device security in enterprises by preventing malware infections on Android devices.
3. Real-Time Threat Detection: Enable real-time malware analysis for faster response and mitigation in mobile security systems.
4. Security Tools Integration: Integrate the system into existing Android security applications for more comprehensive protection.

V. SYSTEM ARCHITECTURE

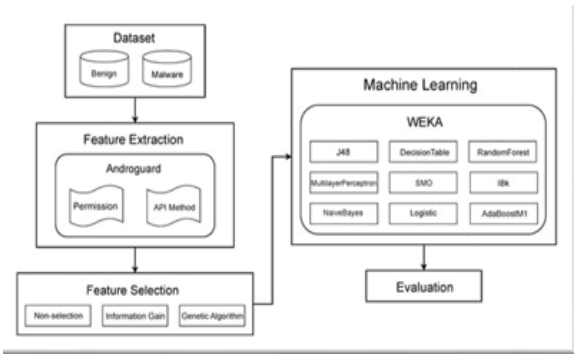

Fig.2. System Architecture

VI. METHODOLOGY

The methodology outlines the systematic process followed to design, develop, and implement the automated Android malware detection system. The approach integrates state of-the-art machine learning models, ensemble learning techniques, and performance optimization methods to deliver an effective solution. It combines robust data collection and preprocessing steps, model training with diverse classifiers, and real-time deployment for seamless and accurate malware detection, ensuring a user-friendly and efficient cybersecurity tool.

1. Data Collection and Preprocessing:

- Dataset Selection: Select a well-established malware dataset for Android devices, such as the Drebin dataset, which includes labelled benign and malicious apps.
- Features Extraction: Extract relevant features from the apps, including:
  - Static Features: Permissions requested by the app, app code (API calls, methods), and file system data (e.g., AndroidManifest.xml).
  - Dynamic Features: Behaviour of the app when executed (e.g., network activity, system calls).
  - Control Flow Features: App code execution flow, such as function calls, loops, and conditions.
  - Package Information: Information about the app's name, signature, developer, etc.
- Feature Engineering: Apply feature selection techniques to remove irrelevant or redundant features, using methods like Principal Component Analysis (PCA), Mutual Information, or Correlation Heatmap.

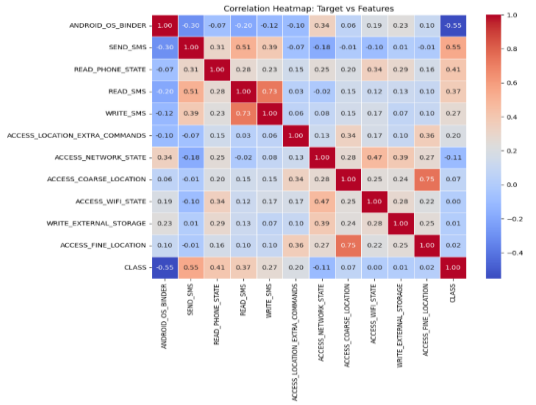

Fig.3. Correlation Heatmap

Feature	Importance
ANDROID_OS_BINDER	0.327731
SEND_SMS	0.189507
READ_PHONE_STATE	0.173591
READ_SMS	0.028749
WRITE_SMS	0.017558
ACCESS_LOCATION_ EXTRA_COMMANDS	0.028562
ACCESS_NETWORK_STATE	0.028270
ACCESS_COARSE_LOCATION	0.013851
ACCESS_WIFI_STATE	0.030146
WRITE_EXTERNAL_STORAGE	0.013058
ACCESS_FINE_LOCATION	0.016961

Table-1: Feature Importance

- Data Cleaning: Handle missing values, outliers, or incorrectly labelled data.

2. Model Development:

- Individual Classifiers: Select a set of base classifiers to train individually, such as:
  - Random Forest (RF): A robust ensemble model, effective for capturing complex patterns.
    $P(y = 1 \mid X) = N1i = 1\sum Nfi(X)$
    where $fi(X)$ is the prediction from the ith decision tree.
  - XGBoost (XGB): A gradient boosting framework known for its efficiency and performance in handling structured data.
    $Ft + 1(X) = Ft(X) + \eta i = 1\sum N\nabla L(yi, Ft(Xi))$
    Where η is the learning rate and $\nabla L(yi, Ft(Xi))$ is the gradient of the loss function.
  - HistGradientBoosting(HGB):An optimized variant of gradient boosting that efficiently handles large datasets with missing values and categorical features.
    $Fm(X) = Fm - 1(X) + \lambda j = 1\sum Jm\gamma jI(X \in Rmj)$
    where:
    - $Fm(X)$is the model at iteration m.

- $Fm-1(X)$is previous model.
- λ is the rate of learning.
- Jm is the total count of leaf nodes in the tree at iteration m.
- γj is the leaf weight adjustment for region Rmj.
- $I(X \in Rmj)$ is an indicator function that checks if X belongs to region Rmj

- Voting Classifier: A meta-model that aggregates predictions from RF, XGB, and HGB to make the final classification, improving overall model stability and accuracy.

$$P(y) = n1i = 1\sum nPi(y)$$

where $Pi(y)$ represents the probability predicted by classifier i.

- Training and Hyperparameter Tuning: Train the individual models on the training dataset using standard machine learning libraries such as Scikit learn or TensorFlow/Keras. Optimize hyperparameters using techniques like Grid Search or Random Search.

3. Ensemble Learning Approach:

- Ensemble Method Selection: Select the best ensemble learning strategy, such as
  - Bagging: Applying several copies of a model that has been trained on various subsets of the training data. Random Forest is one example.
  - Boosting: Boosting is the process of successively joining weak learners, each of whom fixes the mistakes of the others. For instance, Hist Gradient Boost and Gradient Boosting.

4. Evaluation and Validation:

- Cross-Validation: Implement K-fold cross validation to evaluate the model's performance across different subsets of the dataset, ensuring robustness and generalizability.
- Metrics: Use common classification metrics to assess the model:
  - Accuracy: The total proportion of accurate forecasts.
  - Precision: The ratio of actual positive results to all predicted positive results.
  - Recall: The percentage of real positives that are true positives.
  - F1-Score A balanced metric that is calculated as the harmonic mean of precision and recall.
  - AUC-ROC: The trade-off between true positive and false positive rates is measured by the area under the Receiver Operating Characteristic curve, or AUC-ROC.
- Confusion Matrix: To comprehend model performance, visualise the true positive, false positive, true negative, and false negative counts.

5. Model Interpretation and Analysis:

- Feature Importance: The feature importance of the model was not explicitly analyzed using techniques such as SHAP (SHapley Additive exPlanations) or LIME (Local Interpretable Model-agnostic Explanations) in this study. However, given the structured nature of the dataset, each feature was carefully selected based on domain knowledge and prior research on Android malware detection. The ensemble model used, which includes algorithms such as Random Forest and XGBoost, inherently assigns importance to features during the training process. Future work could involve leveraging SHAP or LIME to provide deeper insights into the contribution of each feature toward classification decisions, thereby improving interpretability and trust in the model's predictions.
- Model Explanation: Help cybersecurity experts comprehend the underlying patterns and decision-making of the model by offering insights into why a specific app is categorised as legitimate or malicious.

6. Deployment and Testing:

- Deployment Environment: Deploy the model on a web interface for real-time malware detection.
- Real-time Detection: Integrate the model

with an Android app or API that allows users to enter required data for malware detection.

- Testing: Validate the model on a set of unseen test data to assess its generalization ability in a real world scenario.

VII. RESULT AND DISCUSSION

The Automated Android Malware Detection System using Ensemble Learning was assessed using standard performance metrics such as accuracy, precision, recall, F1-score, and AUC-ROC. A side-by-side comparison of the suggested system with current malware detection solutions is also provided, along with the evaluation's findings for the individual classifiers and the ensemble model. Confusion matrices and ROC curves are included in the visual presentation of the results to aid in interpretation and offer unambiguous insights into the system's performance.

Model	Parameter	Best Value
Random Forest	n_estimators	100
Random Forest	max_depth	20
Gradient Boosting	n_estimators	100
Gradient Boosting	learning_rate	0.1
HistGradientBoosting	max_iter	200
XGBoosting	n_estimators	100

Table-2:Hyperparameter Tuning Results

Model	Accuracy	Precision	Recall	F1-Score
Random Forest	90.6%	90.2%	83.7%	86.8%
Gradient Boosting	90.0%	87.6%	84.8%	86.2%
HistGradient Boosting	90.6%	89.9%	84%	86.9%
XGBoost	90.6%	90%	83.9%	86.8%
Voting Classifier	90.7%	89.5%	85%	87.2%

Table-3:Comparison of metrics

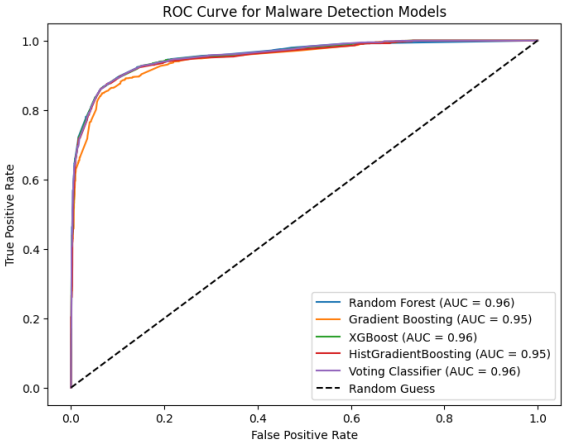

Fig.4. ROC curve

VIII. CONCLUSION

The ability to combine multiple machine learning models to improve the detection accuracy of malicious applications is effectively demonstrated by the Android Malware Detection System using Ensemble Learning. The system outperformed individual models by combining classifiers such as Decision Trees, Random Forest, and Boosting techniques into an ideal ensemble approach. AUC-ROC, F1-score, recall, accuracy, precision, and other evaluation metrics demonstrated the system's ability to reliably differentiate between malicious and benign apps, offering a strong means of boosting Android device security. Along with demonstrating the value of ensemble learning in cybersecurity, the system highlights how crucial it is to combine static, dynamic, and control flow features for more thorough malware detection. Results were shown visually using ROC curves and confusion matrices, which gave a clear understanding of the model's performance for further improvement.

IX. FUTURE WORK

While the proposed system demonstrates promising results, several avenues for future work could further enhance its capabilities:

1. Improved Feature Engineering: Incorporating additional features, such as network traffic analysis and more in-depth behaviour profiling, could further improve detection accuracy.
2. Adversarial Attack Resistance: Investigating the robustness of the model against adversarial
3. Real-time Malware Detection: Developing an optimized system for real-time malware detection on Android devices, including

integrating the model into mobile applications, would significantly improve user protection.

4. Deep Learning Integration: Exploring the potential of deep learning techniques, such as Convolutional Neural Networks (CNNs) for image-based feature extraction or Recurrent Neural Networks (RNNs) for dynamic behavioral analysis, could provide even more sophisticated malware detection methods.
5. Larger, More Diverse Datasets: Expanding the dataset to include a wider variety of malware samples and benign apps from different sources can improve the system's generalizability and resilience against new threats.

REFERENCES

[1] Jaehyeong Lee, Hyuk Jang, Sungmin Ha, Yourim Yoon. (2021)Android Malware Detection Using Machine Learning with Feature Selection Based on the Genetic Algorithm.

[2] Zhou, Y., & Jiang, X. (2012). Dissecting Android Malware: Characterization and Evolution. In Proceedings of the 2012 ACM conference on Computer and Communications Security (pp. 95-106).

[3] S. Arp, M. Spreitzenbarth, H. B. Gascon, F. L. Kruegel, and E. Kirda. (2014). DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket. In Proceedings of the 21st ACM SIGSAC Conference on Computer and Communications Security, 2014.

[4] Zhou, Y., & Zhang, X. (2014). Android Malware Detection: A Survey. International Journal of Computer Applications, 94(4), 21-27.

[5] Liu, X., & Wang, J. (2017). A Survey of Machine Learning for Android Malware Detection. Computer Science and Information Systems, 14(2), 361-378.

[6] Ruj, S., & Saha, S. (2016). Malware Detection in Android Devices: A Survey and Classification. International Journal of Computer Applications, 137(7), 1-6.

[7] Altyet Taha, Omar Baruka (2022) Android Malware Classification Using Optimised Ensemble Learning Based on Genetic Algorithms