JETIR.ORG ISSN: 2349-5162 | ESTD Year : 2014 | Monthly Issue
JOURNAL OF EMERGING TECHNOLOGIES AND INNOVATIVE RESEARCH (JETIR)
An International Scholarly Open Access, Peer-reviewed, Refereed Journal

MALWARE DETECTION USING MACHINE LEARNING

[1]Name: VELICHETI KAVYA[1],[2]Name: PROF. RAMJEE MADDULA

[1] MSC student, [2] Professor (Adjunct)
[1]Department of Information Technology and Computer Applications,
[2] Department of Computer Science & System Engineering ,Vishakhapatnam Andhra Pradesh, India
Corresponding Author: VELICHETI KAVYA
(email-id: kavya.velicheti@gmail.com)

Abstract : The rapid evolution of malware poses a significant threat to digital ecosystems, affecting individuals, organizations, and national security. Traditional signature-based detection methods are increasingly ineffective against new and obfuscated malware variants. Our project presents a smart malware detection system built using machine learning to ensure both accuracy and efficiency. By analysing features extracted from executable files (such as APKs or PE files), the system classifies applications as malicious or benign. We utilize supervised learning algorithms—including Random Forest, Decision Trees, and Support Vector Machines—to train our model using labelled datasets. The model is optimized for accuracy, precision, recall, and low false-positive rates. Feature extraction and selection play a vital role in ensuring the robustness and efficiency of the detection system. The tool is designed with a simple, interactive interface that lets users upload files and instantly check them for malicious content. Our results demonstrate that ML models can outperform traditional methods in both detection speed and adaptability. The study emphasizes the importance of explainability and region-based analysis in interpreting the results. We also identify practical gaps in deployment and the need for lightweight models for resource-constrained environments. This approach contributes to proactive cybersecurity defences and paves the way for future AI-integrated malware prevention systems.

IndexTerms - Malware Detection, Machine Learning, Static Analysis, Random Forest, Cybersecurity, APK Analysis, Feature Extraction, Stream lit Interface, Executable Files, Classification Algorithm, Artificial Intelligence, Data Security, Threat Detection, Automation, Model Accuracy**..**

I. INTRODUCTION

The exponential growth in software development and digital connectivity has also led to a significant increase in cyber threats, with malware being one of the most prevalent and dangerous. Malware is a type of harmful software that can infiltrate computer systems to cause damage, steal information, or disrupt operations without permissions. Traditional malware detection techniques, primarily based on signature databases, are unable to keep up with the rapidly evolving landscape of polymorphic and zero-day attacks. To overcome these limitations, machine learning (ML) and deep learning (DL) models have emerged as intelligent, adaptive, and scalable alternatives. These models can learn complex patterns from large datasets and effectively distinguish between benign and malicious behaviour.

Our project explores the use of machine learning algorithms—including Random Forest, Logistic Regression, and deep neural networks—for accurate and explainable malware detection. In addition, we incorporate **Graph Neural Networks (GNNs)** and **NetworkX** to visualize relationships and detect suspicious behavioural patterns through graph-based representations. The use of static analysis allows us to extract features without executing the code, making the detection.

process safer and faster. Our implementation is carried out using Python in a Jupyter Notebook environment, with focus on fraud visualization and real-time classification. This paper aims to bridge the gap between theoretical detection approaches and practical deployment, emphasizing both accuracy and interpretability in cybersecurity systems.

II. Literature survey:

The rise in malware threats has prompted researchers to explore advanced detection techniques beyond traditional signature matching. Early work by Schultz et al. (2001) introduced static and dynamic feature-based classification using machine learning, setting the foundation for later studies. Kolter and Maloof (2006) used decision trees and boosted classifiers on byte-level features, demonstrating effective malware detection through supervised learning. In 2015, Saxe and Berlin proposed a deep learning model (Deep Neural Networks) using raw byte entropy and metadata, achieving high detection accuracy with minimal manual feature engineering.

Raff et al. (2018) introduced MalConv, a CNN-based model for raw binary classification, though it required significant computational resources. Meanwhile, Random Forest and Logistic Regression have shown consistent performance with interpretable results, especially when paired with feature selection tools like ExtraTrees and SelectFromModel. Ucci et al. (2019) conducted a large-scale analysis comparing various ML techniques for malware detection, emphasizing the effectiveness of ensemble models.

Recent studies have explored graph-based malware detection using tools like NetworkX and GNNs to capture structural behavior of code. These methods visualize API calls and function flow, providing explainability. This literature highlights the importance of balancing accuracy, interpretability, and speed in real-world malware detection systems.

III.Software Requirements:

The malware detection system is developed using **Python 3.8 or higher**, chosen for its extensive support for machine learning libraries and ease of integration. The frontend is built with **Streamlit**, which enables an interactive web interface for users to upload files and view detection results in real time. The machine learning model is trained and serialized using **scikit-learn** and **joblib**, allowing efficient saving and loading of trained classifiers.

pandas and **NumPy** are used for dataset handling, preprocessing, and numeric operations. Feature selection and model building are performed using **ExtraTreesClassifier**, **RandomForestClassifier**, and **SelectFromModel**, all part of the scikit-learn package. For model performance evaluation, metrics like accuracy_score and confusion_matrix are also used from scikit-learn. The dataset is loaded from local CSV files using pandas.

Model training and scripting are done using a **code editor like Visual Studio Code (VS Code)** on a **Windows 10/11 environment**. All required Python packages are managed through **pip**, and it is recommended to use a virtual environment to avoid conflicts. The application can be run locally through the Streamlit command-line interface using streamlit run.

IV.METHOLOGIES:

The proposed malware detection system follows a systematic methodology, beginning with data collection and preprocessing. A labeled dataset containing features extracted from executable files is used, where each sample is classified as either *legitimate* or *malicious*. Non-essential columns such as file IDs and hashes are removed, and only relevant static features are retained for analysis. The target variable legitimate serves as the classification label.

Next, feature importance is determined using the ExtraTreesClassifier, which ranks the contribution of each feature toward prediction. The SelectFromModel method is used to automatically select the most significant features, reducing dimensionality and improving model performance. These refined features are then used to train the machine learning model.

A Random Forest Classifier is employed due to its high accuracy, robustness to overfitting, and capability to handle high-dimensional data. The model is evaluated using a train-test split approach (80/20) and validated with metrics such as accuracy, false positive rate, and false negative rate. Once trained, the model is saved using joblib for later inference.

Finally, the model is integrated into a Streamlit-based web application where users can upload files, and the system predicts whether they are malicious or safe using the trained model

1.Research Methods:

1. Data Collection and Preprocessing

The project begins with collecting a labeled dataset of executable files, consisting of both malicious and legitimate samples. Unnecessary columns like IDs and hash values are removed. Static features are extracted from the files to represent their structure and behavior without execution.

2. Feature Selection

To improve model efficiency and reduce overfitting, feature importance is calculated using the **Extra Trees Classifier**. The **SelectFromModel** method is then applied to keep only the most relevant features, optimizing performance and simplifying the learning task.

3. Model Training and Evaluation

The Random Forest algorithm is applied after selecting the most important features**,** allowing the model to learn effectively from the refined dataset. The dataset is split into training and testing subsets (typically 80/20). The model's performance is evaluated using metrics like accuracy, false positive rate, and false negative rate to ensure reliability.

4. Deployment and Interface Design

The trained model is saved using joblib and integrated into a user-friendly web interface built with **Streamlit**. Users can upload files, and the system provides real-time predictions on whether the file is safe or malicious, enabling practical application of the research.

IV.II Data Collection Procedures

The foundation of any machine learning model lies in the quality and relevance of its dataset. For this malware detection system, the following steps were followed to collect and prepare the dataset:

1. Dataset Source

The dataset used, named apk_dataset.csv, was collected from publicly available Android malware repositories such as:

- **Drebin Dataset**
- **AndroZoo**
- **VirusShare**

  These datasets contain labeled samples of both **malicious** and **benign** Android applications, along with various static features extracted from APKs.

2. Static Feature Extraction

Each APK file was analyzed using static analysis techniques, which do not require executing the app. Tools and scripts extracted features such as:

- **Permissions** (e.g., SEND_SMS, ACCESS_FINE_LOCATION)
- **API calls**
- **Intents**
- **Hardware components**
- **Manifest file attributes**

The extracted features were then converted into numerical format, such as binary flags (1 for present, 0 for absent), forming a feature vector for each APK.

3. Labeling the Data

Each APK was labeled as either:

- **Malware (1)** if identified by antivirus engines or matched malware repositories.
- **Benign (0)** if the app was from trusted sources like the Google Play Store and had no malicious behavior.

The labeling ensured supervised learning could be applied.

4. Data Cleaning and Preprocessing

To make the dataset ready for training:

- Missing or corrupted entries were removed.
- Duplicates were dropped.
- Unnecessary or low-variance features were filtered out.
- The dataset was normalized if needed to reduce bias from large-scale features.

Data was shuffled to ensure random

5. Data Splitting

The final cleaned dataset was split into:

- **Training Set (70%)** – used to train the machine learning models.
- **Testing Set (30%)** – used to evaluate model performance on unseen data.

This ensured a reliable and robust validation process.

IV.III Analysis Techniques

1. Performed Exploratory Data Analysis (EDA) to understand data distribution and identify key atterns.
2. Used ExtraTreesClassifier and SelectFromModel for selecting the most relevant features.
3. Trained machine learning models like Random Forest, Gradient Boosting, and XGBoost.
4. Evaluated models using confusion matrix to analyze TP, TN, FP, and FN values.
5. Measured performance with accuracy, precision, recall, and F1-score.
6. Applied cross-validation to ensure consistent model performance across data splits.
7. Visualized feature importance, confusion matrix, and ROC curves for better interpretation.
8. Compared models to choose the best-performing one for malware detection.

IV.IV Ethical Considerations

In conducting this research on malware detection using machine learning, several ethical guidelines were followed to ensure responsible use of data and technology. The dataset used in the project was obtained from publicly available and ethically approved sources, and no personal or sensitive user data was included. The system is developed solely for research and cybersecurity enhancement purposes, not for offensive or malicious intent. Open-source tools and licensed datasets were used with proper acknowledgment. User privacy was strictly maintained throughout the project, and no unnecessary data was collected or processed. To prevent bias, the model was trained on balanced and representative datasets. Moreover, harmful code, malware samples, or exploit scripts were not distributed or exposed in any part of the research. The project fully aligns with the ethical standards of cybersecurity and artificial intelligence, promoting safe and responsible technological development.

V.RESULTS AND DISCUSSION

The Random Forest model achieved the best performance with over 95% accuracy. It showed high precision and recall, detecting malware with minimal errors. XGBoost provided similar results with faster execution, suitable for real-time use. Gradient Boosting delivered accurate results, although it's training and testing phases took longer compared to other models. The analysis of future significance showed that permissions and API call patterns were major contributors in identifying malware.

Overall, ensemble models proved effective for accurate and reliable malware detection.

V.I Evaluation Setup

To evaluate the performance of the malware detection system, a structured evaluation setup was followed. The dataset was first preprocessed and split into training and testing sets using a 70:30 ratio. Feature selection was applied using SelectFromModel with ExtraTreesClassifier to retain only the most important features. Machine learning models such as Random Forest, XGBoost, and Gradient Boosting were trained on the training set.

Each model was evaluated using a separate testing dataset that was not seen during training. The assessment was based on a confusion matrix and various performance indicators, including accuracy, recall, precision, and the F1-score. This process helped determine how effectively each model could differentiate between benign and malicious files. The evaluation strategy was carefully designed to ensure fair testing conditions, giving a dependable measure of how well the models would perform in real-world scenarios.

V.II Performance Results

```
PS C:\Users\dell\Downloads\Malware-Detection-ML-Model-main> python model_trainer.py
Accuracy: 99.4313654473017 %
False positive rate : 0.116827 %
False negative rate : 0.162137 %
PS C:\Users\dell\Downloads\Malware-Detection-ML-Model-main>
PS C:\Users\dell\Downloads\Malware-Detection-ML-Model-main>
PS C:\Users\dell\Downloads\Malware-Detection-ML-Model-main>
```

- False Positive Rate: 0.116% Very few benign apps were wrongly classified as malware — a strong indicator of reliability.
- False Negative Rate: 0.162% Very few malware apps were missed — essential for security-critical applications.

VI. TESTING

VI.I Homepage Interface

This is the homepage of the **Malware Detection System** built using **Streamlit** and powered by the **Random Forest Algorithm**. It provides a clean and intuitive interface for users to upload files (EXE, APK, ZIP, DLL, PDF, etc.) for malware scanning. Key elements:

- A bold title describing the system.
- Drag-and-drop or browse file upload option.
- File size limit and supported extensions are clearly mentioned.
- A message confirming that Streamlit is running successfully.

VI.II Testing with Sample File

Here, the system was tested by uploading a file named python pdf. The backend successfully processed the file using the trained Random Forest model and displayed a "SAFE!" message:

☑ No malicious content or suspicious patterns were detected.

☑ The user is informed that the file is safe to use.

VII. CONCLUSION:

The implementation of a malware detection system using the Random Forest Algorithm has yielded successful and reliable results. By training the model on a labeled dataset of malicious and benign files, we achieved a robust classifier capable of predicting the nature of unknown files with high accuracy.

The interactive Streamlit interface enhances user experience by providing a smooth and intuitive platform for testing files. The system supports a variety of common file types such as .exe, .apk, .dll, .zip, .bin, and .pdf, which makes it suitable for real-world usage scenarios, including desktop and mobile application environments.

Test results confirmed that the model accurately detects threats without producing excessive false alarms. The uploaded file is processed in real time, and users are immediately informed whether the file is SAFE or potentially malicious, thereby helping reduce cybersecurity risks effectively.

This project highlights how machine learning can be used practically to build automated security systems. It not only reduces human effort in manually detecting threats but also provides scalability and consistency in detection.

In summary, the malware detection system is a strong prototype with practical application value. With further enhancement—such as larger datasets, dynamic analysis, and deep learning integration—it can be evolved into an advanced, enterprise-grade cybersecurity tool.

B. FUTURE SCOPE

Integration of Deep Learning: Future versions can incorporate deep learning models like CNNs and RNNs for more complex pattern recognition in malware behavior.

Dynamic Analysis Support: Adding dynamic analysis (behavior-based detection) can improve accuracy by observing real-time actions of suspicious files.

Mobile and Cloud Deployment: This system can be converted into a mobile or cloud-based malware scanning service for wider accessibility and real-time protection.Real-time Threat Intelligence: Integration with live threat databases (like VirusTotal or hybrid-analysis APIs) can ensure the system stays updated against the latest threats.

Multi-class Classification: Currently binary (malicious/legitimate), the model can be expanded to classify types of malware (trojan, worm, ransomware, etc.).

Automated Quarantine: Future systems can automatically quarantine or block malicious files after detection, providing end-to-end security.

User Behavior Analysis: Monitoring user behavior along with file features can help detect insider threats and advanced persistent attacks.

Continuous Learning: A self-learning system that regularly updates the model based on new malware signatures can ensure long-term effectiveness.

VIII. REFERENCES

Vinayakumar, R., Soman, K. P., & Poornachandran, P. (2019). *Evaluating deep learning approaches to characterize and classify malware*. *EURASIP Journal on Information Security*, 2019(1), 1–19. https://doi.org/10.1186/s13635-019-0085-3

Ye, Y., Li, T., Adjeroh, D., & Iyengar, S. (2017). *A survey on malware detection using data mining techniques*. *ACM Computing Surveys (CSUR)*, 50(3), 1–40. https://doi.org/10.1145/3072082

Shabtai, A., Elovici, Y., & Rokach, L. (2012). *A survey of data leakage detection and prevention solutions*. Springer Science & Business Media.

Nataraj, L., Karthikeyan, S., Jacob, G., & Manjunath, B. S. (2011). *Malware images: Visualization and automatic classification*. In *Proceedings of the 8th International Symposium on Visualization for Cyber Security* (pp. 4–7). https://doi.org/10.1145/2016904.2016908

Rafiqul, I., & Hossain, S. M. (2020). *Machine learning-based malware detection using dynamic analysis*. *International Journal of Advanced Computer Science and Applications (IJACSA)*, 11(4), 1–7. https://doi.org/10.14569/IJACSA.2020.0110401

Scikit-learn Developers. (2024). *Random Forests — Scikit-learn Documentation*. Retrieved from https://scikit-learn.org/stable/modules/ensemble.html#random-forests