International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

AUTOMATED ANDROID MALWARE DETECTION USING ENSEMBLE LEARNING APPROACH FOR CS

Syed Anas [1], Mohammed Nizam Uddin Saif [2], Syed Khaja Osmane Haroon [3]

Department of IT, Nawab Shah Alam Khan College of Engineering and Technology, Hyderabad, India
Email: stoa4451@gmail.com.

A B S T R A C T:

This research paper presents the development of a web-based Android malware detection system that leverages static analysis and machine learning for accurate classification of malicious applications. Built using Python and the Flask framework, the system enables users to upload feature-extracted CSV files derived from Android APKs and obtain real-time malware predictions. It incorporates multiple machine learning models—including Random Forest, Extra Trees, Artificial Neural Networks, and Convolutional Neural Networks—to evaluate detection performance through key metrics such as accuracy, precision, recall, and F1-score. A feature selection mechanism enhances model performance by isolating the most impactful attributes from the input data. The application also includes role-based access (admin and user), performance visualization, and a streamlined prediction module powered by a trained CNN model. This work demonstrates the practical potential of combining static analysis and supervised learning techniques to build efficient and user-friendly malware detection platforms.

Keywords: Android Malware Detection, Static Analysis, Machine Learning, Convolutional Neural Network, Flask Web Application, Feature Selection, APK Analysis, Cybersecurity, Malware Classification, Supervised Learning

1. Introduction:

The rapid growth of the Android ecosystem has transformed smartphones into essential tools for communication, banking, entertainment, and productivity. However, this widespread adoption has also made Android a prime target for cybercriminals, leading to a significant rise in malicious applications that threaten user privacy, data integrity, and device security. Traditional antivirus solutions often rely on signature-based detection, which fails to recognize newly emerging malware variants. As a result, there is a growing demand for intelligent, automated malware detection systems that can identify threats based on behavioral patterns and code-level features.

2. Literature Review:

The rapid proliferation of Android malware has led to significant research efforts aimed at enhancing mobile security through machine learning and deep learning techniques. Numerous studies have investigated static and dynamic analysis, feature engineering, and model optimization to effectively detect and classify malicious applications. The following are the few literature provides a foundation for the current work by reviewing key contributions in this field.

- Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, Konrad Rieck (2014)
  Proposed Drebin, a lightweight static-analysis tool using SVM that extracts features like permissions and API calls from APKs. It achieved 94% detection accuracy with minimal overhead and offered explainable outputs.

- Feizollah, N. B. Anuar, R. Salleh, A. W. A. Wahab (2015)
  Reviewed 100 studies on feature selection for mobile malware detection, categorizing features into static, dynamic, hybrid, and metadata. Emphasized the importance of feature selection for improving model accuracy and performance.

- K. Zhao, D. Zhang, X. Su, W. Li (2015)
  Developed FEST, a tool for extracting and selecting features from Android apps using a method called FrequenSel. It prioritizes features frequent in malware but rare in benign apps to boost classification accuracy.

- Saracino, D. Sgandurra, G. Dini, F. Martinelli (2016)
  Introduced MADAM, a multi-layer behavior-based detector analyzing kernel, app, user, and package features. It achieved 96% malware detection on real-world datasets with minimal performance impact.

3. Methodology:

The Android malware detection system was developed using Python, leveraging various libraries and tools for feature extraction, data processing, and machine learning model training. The methodology involved the following key steps:

3.1 Setting Up the Environment

The first step was to prepare the development environment by installing all necessary Python packages and tools. This included libraries for static APK analysis, data handling, and machine learning, such as apktool, pandas, scikit-learn, and TensorFlow. Additionally, tools for feature extraction were configured to analyze APK files and extract relevant permission and API call data.

3.2 Dataset Collection and Preparation

A dataset of Android application packages (APKs) was gathered, consisting of both benign and malicious samples. The APK files were organized and labeled accordingly to ensure the model could learn to distinguish between safe and harmful apps.

3.3 Feature Extraction

Static analysis was performed on the APK files to extract meaningful features related to app behavior. Custom Python scripts parsed APK manifests and code to identify permissions requested, API calls made, and other indicators such as intent filters. These extracted features were structured into a tabular format (CSV) for further processing.

3.4 Data Preprocessing

The raw extracted data was cleaned and preprocessed to prepare it for machine learning. This involved handling missing values, encoding categorical variables, and normalizing feature values. The data was then split into training and testing sets to enable proper model evaluation.

3.5 Model Training and Evaluation

Several machine learning models, including Random Forest, Artificial Neural Networks (ANN), and Convolutional Neural Networks (CNN), were trained on the processed dataset. The models learned to classify APKs as malicious or benign based on the extracted feature vectors. Performance metrics such as accuracy, precision, recall, and F1-score were calculated to evaluate the effectiveness of each model.

3.6 Malware Prediction

After training, the best-performing model was integrated into the prediction pipeline. New APK files could be input into the system, where their features would be automatically extracted and fed into the model to generate a prediction about the app’s maliciousness.

4. Illustrations:

```python
@app.route("/preprocessing")
def preprocessing():
    return render_template("data_preprocessing.html")

@app.route("/data_preprocessing",methods =["GET", "POST"] )
def data_preprocessing():
    fname = request.form.get("file")
    preprocess(fname)

    return render_template("data_preprocessing.html",msg="Data Preprocessing Completed..!")
```

Fig. 1 – Data Preprocessing

```python
@app.route("/features_selection" )
def features_selection():
    return render_template("features_selection.html")

@app.route("/selected_features",methods =["GET", "POST"] )
def selected_features():
    fname = request.form.get("file")
    df = pd.read_csv(fname)
    print("date=",df.head())
    features_list,targetcol=getFeatures(df)
    X=df[features_list]
    print("date2=",df.head())
    y=targetcol
    dict['X'] = X
    dict['y'] = y
    df2=df[features_list]
    df2["class"]=y
    df2.to_csv("features.csv",index=False)

    return render_template("features_selection.html",features_list=features_list)
```

Fig. 2 – Feature Selection

Fig. 3 – Metrics Evaluation

```python
@app.route("/evaluations")
def evaluations():
    rf_list=[]
    etc_list = []
    ann_list = []
    cnn_list = []
    metrics=[]
    X=dict['X']
    y=dict['y']

    # Split train test: 70 % - 30 %
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=15)


    accuracy_rf, precision_rf, recall_rf, fscore_rf = rfc_evaluation(X_train, X_test, y_train, y_test)
    rf_list.append("RFC")
    rf_list.append(accuracy_rf)
    rf_list.append(precision_rf)
    rf_list.append(recall_rf)
    rf_list.append(fscore_rf)

    accuracy_etc, precision_etc, recall_etc, fscore_etc = etc_evaluation(X_train, X_test,y_train, y_test)
    etc_list.append("ETC")
    etc_list.append(accuracy_etc)
    etc_list.append(precision_etc)
    etc_list.append(recall_etc)
    etc_list.append(fscore_etc)

    accuracy_ann, precision_ann, recall_ann, fscore_ann = ann_evaluation(X_train, X_test, y_train, y_test)
    ann_list.append("ANN")
    ann_list.append(accuracy_ann)
    ann_list.append(precision_ann)
    ann_list.append(recall_ann)
    ann_list.append(fscore_ann)

    accuracy_cnn, precision_cnn, recall_cnn, fscore_cnn = cnn_evaluation()
    cnn_list.append("CNN")
    cnn_list.append(accuracy_cnn)
    cnn_list.append(precision_cnn)
    cnn_list.append(recall_cnn)
    cnn_list.append(fscore_cnn)

    metrics.clear()
    metrics.append(rf_list)
    metrics.append(etc_list)
    metrics.append(ann_list)
    metrics.append(cnn_list)


    return render_template("evaluations.html", evaluations=metrics)
```

5. Result:

The application successfully detects whether an Android APK file is malicious or benign based on the extracted features. The integration of static analysis tools and machine learning models enables accurate and automated malware detection without requiring the app to be executed. Users can input any APK file, and the system analyzes its behavior patterns—such as requested permissions and API usage—to produce a prediction. The trained models demonstrated high accuracy, with the Random Forest classifier providing the most consistent performance across test datasets. This showcases the application's effectiveness, practicality, and potential for real-world cybersecurity use cases.

6. Requirements:

6.1. Hardware Requirements

- Processor : Any Update Processer
- Ram : Min 4 GB
- Hard Disk : Min 100 GB

6.2. Software Requirements

- Operating System : Windows family
- Technology : Python 3.6
- Front-end Technology : HTML, CSS, JS
- Back-end Technology : MySQL
- IDE : PyCharm
- Web Framework : Flask

7. Conclusion:

This paper presents a practical approach to developing an Android malware detection system using static analysis and machine learning. The application provides an automated and user-friendly way to analyze APK files and predict their malicious behavior based on extracted features. This project demonstrates the effectiveness of integrating cybersecurity techniques with machine learning to enhance mobile security. It highlights how modern technologies can be leveraged to detect threats efficiently, making malware analysis more accessible and scalable for real-world use.

REFERENCES

[1] H. Rathore, A. Nandanwar, S. K. Sahay, and M. Sewak, ''Adversarial superiority in Android malware detection: Lessons from reinforcement learning based evasion attacks and defenses,'' Forensic Sci. Int., Digit. Invest., vol. 44, Mar. 2023, Art. no. 301511.

[2] H. Wang, W. Zhang, and H. He, ''You are what the permissions told me! Android malware detection based on hybrid tactics,'' J. Inf. Secur. Appl., vol. 66, May 2022, Art. no. 103159.

[3] A. Albakri, F. Alhayan, N. Alturki, S. Ahamed, and S. Shamsudheen, ''Metaheuristics with deep learning model for cybersecurity and Android malware detection and classification,'' Appl. Sci., vol. 13, no. 4, p. 2172, Feb. 2023.

[4] M. Ibrahim, B. Issa, and M. B. Jasser, ''A method for automatic Android malware detection based on static analysis and deep learning,'' IEEE Access, vol. 10, pp. 117334–117352, 2022

[5] J. Pye, B. Issac, H. Rafiq, and N. Aslam, "Android malware classification using machine learning and bio-inspired optimization algorithms," in Proc. 2015 IEEE Int. Conf. Comput. Sci. Educ. (ICCSE), Cambridge, UK, 2015, pp. 1–6.

[6] A. Saracino, D. Sgandurra, G. Dini, and F. Martinelli, "MADAM: Effective and efficient behavior-based Android malware detection and prevention," IEEE Trans. Dependable Secure Comput., vol. 15, no. 3, pp. 441–455, May–Jun. 2018.

[7] K. Zhao, D. Zhang, X. Su, and W. Li, "Fest: A feature extraction and selection tool for Android malware detection," in Proc. 2015 IEEE Symp. Comput. Commun. (ISCC), Larnaca, Cyprus, 2015, pp. 714–720.

[8] J. Li, L. Sun, Q. Yan, Z. Li, W. Srisa-An, and H. Ye, "Significant permission identification for machine-learning-based Android malware detection," IEEE Trans. Ind. Informat., vol. 14, no. 7, pp. 3216–3225, Jul. 2018.

[9] A. Feizollah, N. B. Anuar, R. Salleh, and A. W. A. Wahab, "A review on feature selection in mobile malware detection," Digit. Invest., vol. 13, pp. 22–37, Mar. 2015.

[10] Z. Yuan, Y. Lu, and Y. Xue, "DroidDetector: Android malware characterization and detection using deep learning," Tsinghua Sci. Technol., vol. 21, no. 1, pp. 114–123, Feb. 2016.

[11] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, and Y. Weiss, "Andromaly: A behavioral malware detection framework for Android devices," J. Intell. Inf. Syst., vol. 38, no. 1, pp. 161–190,Feb. 2012.

[12] S. Arp, H. Spreitzenbarth, M. Hübner, M. Gascon, and K. Rieck, "Drebin: Effective andexplainable detection of Android malware in your pocket," in Proc. 2014Netw. Distrib. Syst. Secur.Symp. (NDSS), San Diego, CA, USA, 2014, pp. 1–15.