Enhancing Mobile Security through Machine Learning: A Study on Android Malware Detection

Authors:
Moti Dahari308212570
amit koobani 204804488

Machine Learning-based Approach for Android Malware Detection: An Evaluation of Performance and Limitations

Introduction

Our presentation covers a machine learning-based approach for detecting malware on Android devices. We used a dataset of 4591 apps and trained various models on it. We will discuss the methodology, results, evaluation metrics, and insights, as well as limitations and future directions for this research.

Description of the dataset

This project used a dataset of 4591 apps, during the data extraction 1817 apps were selected, 165 labeled as malicious and 1652 as benign. Various features were extracted to train machine learning models like SVM with a linear kernel, Logistic Regression, KNeighborsClassifier, DecisionTreeClassifier, and GradientBoostingClassifier.

The dataset also included categories and Android API calls. The extracted information was stored in a JSON file, any missing values were filled with 0, and helper functions were written to extract and organize the information.

Pre-processing

We pre-processed the data to prepare it for training the model. The steps included data cleaning, transformation, dimensionality reduction, and balancing the dataset. This was necessary to ensure the data was of high quality and to overcome challenges faced in the project.

Details of the features

This project extracted various features from the dataset of Android apps:

```
{
    sha256,
    label,
    app_permissions,
    api_permissions,
    api_calls,
    activities,
    s_and_r,
    interesting_calls,
    urls,
    providers
}
```


```
"sha256": {"malicious": 2755,"benign": 1836},
"interesting_calls::getSystemService": {"malicious": 2563,"benign": 1334},
"app_permissions::android_permission_INTERNET": {"malicious": 2510,"benign": 1423},
"api_permissions::android_permission_INTERNET": {"malicious": 2460,"benign": 1306},
"app_permissions::android_permission_READ_PHONE_STATE": {"malicious": 2353,"benign": 286},
"api_permissions::android_permission_READ_PHONE_STATE": {"malicious": 2079,"benign": 330},
"api_calls::java/net/URL;->openConnection": {"malicious": 1864,"benign": 951},
"interesting_calls::getDeviceId": {"malicious": 1857,"benign": 446},
"api_calls::android/telephony/TelephonyManager;->getDeviceId": {"malicious": 1856,"benign": 238},
"interesting_calls::printStackTrace": {"malicious": 1840,"benign": 654},
"api_calls::java/net/HttpURLConnection": {"malicious": 1804,"benign": 932},
"app_permissions::android_permission_ACCESS_NETWORK_STATE": {"malicious": 1778,"benign": 9
"app_permissions::android_permission_WRITE_EXTERNAL_STORAGE": {"malicious": 1776,"benign": 509},
"api_calls::android/webkit/WebView": {"malicious": 1720,"benign": 1049},
"api_calls::android/content/Context;->startService": {"malicious": 1710,"benign": 682},
"api_calls::org/apache/http/impl/client/DefaultHttpClient": {"malicious": 1668,"benign": 623},
"api_permissions::android_permission_VIBRATE": {"malicious": 1612,"benign": 722},
"api_calls::android/content/Context;->startActivity": {"malicious": 1576,"benign": 1001},
"api_calls::android/app/NotificationManager;->notify": {"malicious": 1535,"benign": 637},
"api_permissions::android_permission_ACCESS_NETWORK_STATE": {"malicious": 1508,"benign": 1086},
"api_calls::java/net/HttpURLConnection;->connect": {"malicious": 1474,"benign": 679},
"interesting_calls::HttpPost": {"malicious": 1437,"benign": 492},
"app_permissions::android_permission_SEND_SMS": {"malicious": 1417,"benign": 35},
"api_calls::android/net/ConnectivityManager;->getActiveNetworkInfo": {"malicious": 1411,"benign": 1008},
"interesting_calls::Read/Write External Storage": {"malicious": 1376,"benign": 782},
"interesting_calls::getPackageInfo": {"malicious": 1333,"benign": 1317},
```

```
[
  {
    "sha256": "B8CC23EC7D68F320558A9572F1C14935AFC820A0A1C053ECA94888F341D208DF",
    "label": 1,
    "app_permissions::android_permission_READ_PHONE_STATE": 1,
    "app_permissions::android_permission_SEND_SMS": 1,
    "app_permissions::android_permission_RECEIVE_SMS": 1,
    "app_permissions::android_permission_INTERNET": 1,
    "api_permissions::android_permission_INTERNET": 1,
    "api_permissions::android_permission_READ_PHONE_STATE": 1,
    "api_permissions::android_permission_SEND_SMS": 1,
    "api_calls::android/app/Activity;->startActivity": 1,
    "api_calls::android/app/Activity;->startActivityForResult": 1,
    "api_calls::org/apache/http/impl/client/DefaultHttpClient": 1,
    "api_calls::org/apache/http/impl/client/DefaultHttpClient;->execute": 1,
    "api_calls::android/telephony/TelephonyManager;->getLine1Number": 1,
    "api_calls::android/telephony/SmsManager;->sendTextMessage": 1,
    "activities::b'_FirstActivity'": 1,
    "activities::b'_RulesActivity'": 1,
    "activities::b'_FinishActivity'": 1,
    "activities::b'QuestionActivity'": 1,
    "activities::b'MemberActivity'": 1,
    "s_and_r::b'_services_SMSSenderService'": 1,
    "s_and_r::b'_sms_BinarySMSReceiver'": 1,
    "interesting_calls::printStackTrace": 1,
    "interesting_calls::getSystemService": 1,
    "interesting_calls::getSimCountryIso": 1,
    "interesting_calls::sendSMS": 1
  },
```

Stages to work on this project:

The project had several stages:

1. data collection
2. 2. feature extraction
3. 3. data preprocessing
4. model training
5. evaluation.

The features were used to train machine learning models such as

1. SVM with a linear kernel
2. Logistic Regression
3. KNeighborsClassifier
4. DecisionTreeClassifier
5. GradientBoostingClassifier.

The performance of the models was evaluated using metrics such as accuracy, precision, and recall. The best performance was achieved by the DecisionTreeClassifier model with an accuracy of 0.984, a precision of 1.000, and a recall of 0.982. The proposed approach was found to outperform traditional methods in terms of accuracy, precision, and recall, but there is still room for improvement. We also wrote various scripts and helper functions throughout the project to ensure smooth execution.

Information fragmentation

In summary, information fragmentation was used in this project to break down a large dataset of Android applications into smaller, more manageable chunks. This helped the team to easily extract relevant features, perform in-depth analysis, and store and retrieve data efficiently.
It played a crucial role in making the data usable for training the machine learning model.

Information fragmentation was crucial in this project as it helped the team break down the large dataset of 1817 Android applications into smaller chunks for better management, analysis, and storage. Scripts and helper functions were written to aid in data extraction and preprocessing, allowing for targeted analysis of specific aspects such as categories and API calls. This helped identify patterns and trends relevant to malware detection, and improved the performance of the machine-learning models. Overall, fragmentation played a key role in effectively working with the data and achieving accurate results.

Algorithms

In this project, various machine learning models were trained to detect Android malware. Each model had its own strengths and weaknesses and the team had to evaluate their performance using metrics like accuracy, precision and recall. SVM had an accuracy of 0.780 and Logistic Regression had an accuracy of 0.835. However, the solver failed to converge in SVM and a convergence warning was received. The best performance was achieved by DecisionTreeClassifier with an accuracy of 0.984.

Algorithms

In this project, various machine learning algorithms were used to train models for Android malware detection, including: SVM, Logistic Regression, KNeighborsClassifier, DecisionTreeClassifier, and GradientBoostingClassifier.
Each algorithm had its strengths and weaknesses. We evaluated the results and found that DecisionTreeClassifier had the best performance with an accuracy of 0.984, precision of 1.000, and recall of 0.982. However, further research is needed to improve the accuracy and performance of the proposed method.

Challenges

We faced several challenges in the project, including dealing with a large dataset, complex data, and data quality. They also had to use dimensionality reduction techniques to handle high dimensionality, and carefully evaluate different algorithms to determine which one was most suitable for the task.

Assumptions

We made several assumptions to complete the task of Android malware detection, such as assuming the dataset was representative, labeled correctly, features were relevant and informative, and the ML algorithms used were suitable. However, these assumptions may not be true and could lead to inaccurate models.

links

github :https://github.com/motidahari/Mobile-Security-ML-Android-Malware-Detection

dataset :https://github.com/motidahari/Mobile-Security-ML-Android-Malware-Detection/tree/main/data/apks/result