Indian Institute of Technology, Kanpur

CS658A: Malware Analysis and Intrusion Detection
Project Report

Title: Anomaly-based Malware Detection in Android Applications

---

Supervised By: Prof. Sandeep Shukla

29 April 2022

Group Name : Toddlers

Aditya Jain	20111004
Akash Halayyanavar	20111006
Aman Pratap Singh	20111010
Mani Kant Kumar	20111030
Mohit Kumar	20111034
Rohit Raj	20111051

Contents

1 Abstract

Most users have shifted from desktop-centric applications to mobile applications in the current era of stable and cheaper internet access. Ninety percent of the global users use smartphones to access the internet, and Android applications capture 70 percent of the market. The users share and exchange their information on the Android applications and create loopholes for their information by using free-of-cost applications that attackers create to track and steal the user's private data using malware. The leakage of personal information creates security issues, digital harassment, etc.

In this project, we proposed to create a Heuristic anomaly detection model which detects the presence of malware in an Android application and warns the user at the time of installation. The model is a classifier that detects the malware based on the permissions it has requested from the Android system and the types of the API-Calls it has made. On top of it, we have designed a feature selection mechanism that reduces the number of features by *100x* providing a massive speedup in the processing time of the classifier, making it feasible to deploy on embedded systems.

2 Broad Aims of the project

- We have designed a Heuristic Anomaly Detection model which detects the presence of malware in an Android application using static features.
- We have trained a classifier to identify whether an App is potentially malicious or not. The permissions and API calls are used as features to characterize each application by the classifier.
- We have improved the processing time of the anomaly detector through feature selection to make the tool deployable on the real device.

3 Datasets

We have used datasets from the University of New Brunswick, which is publicly available. Below are the datasets used:

1. CIC-AndMal2017 datasets:
   `https://www.unb.ca/cic/datasets/andmal2017.html`
2. CICMalDroid 2020:
   `https://www.unb.ca/cic/datasets/maldroid-2020.html`

These datasets contain the Android applications in the form of apk files. We have processed these apk files using the tool - Androguard to reverse-engineer the apk file to generate the AndroidManifest.xml and Dex files. From the AndroidManifest file, we have extracted the Permissions requested by the application. On parsing the Dex files, we have obtained the names and frequency of the API calls performed by the application.

The dataset is already classified into benign and malware. It has more than 3000 malware and benign real-world applications. The following is the break-up of the malicious applications:

1. Adware
2. Banking-Trojans
3. Ransomeware
4. Riskware
5. SMSware
6. Scareware

4 Methodology

4.1 Project Framework

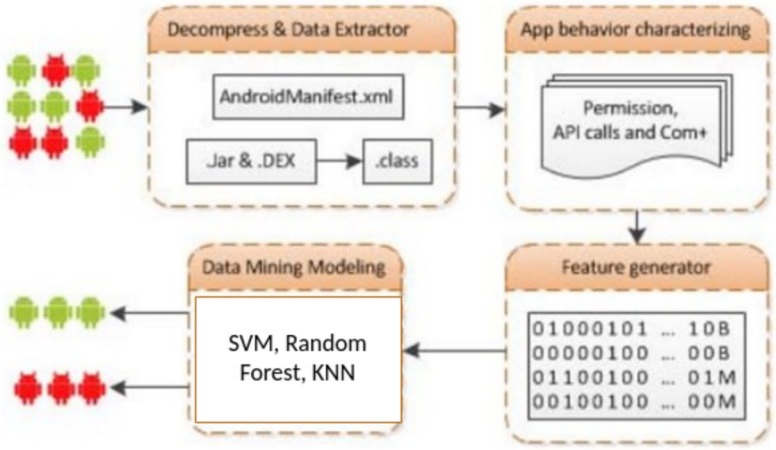

Figure 1: Framework of the project

1. The first component is an App analyzer, which decompresses an App's APK file and extracts AndroidManifest.xml and class files, which are required for app categorization.[1].

2. The second component seeks to characterize each App based on the permissions and API calls requested.

3. The feature generator, the third component, executes feature extraction, including permission features taken from the Android Manifest and API call features extracted from class files and feature selection to limit the number of features.

4. The training of categorization models from the collected data is the final step.

4.2 Feature Extraction

The datasets contain the Android applications in the form of apk files. We have processed these apk files using the tools - Androguard to reverse-engineer the apk file to generate the AndroidManifest.xml and Dex files.

- From the AndroidManifest file, we have extracted the Permissions requested by the application.

- We have created a parser-based tool on python which parses the decompiled smali code generated by the Androguard. On parsing the Dex files, we have obtained the names and frequency of the API calls performed by the application

We represented each App in the dataset as a single instance with binary permission features. The permission is the feature, and if the App has requested that permission, then the value of that feature is one; else, it is equal to zero.
We represented each App in the dataset as a single instance with discrete API call. The API call is the feature, and the number of times the App has used that API call is the value of the feature. A binary class label indicates whether the App is benign or malware.

4.3 Feature Selection

We have applied a mixture of following feature selection techniques and have performed hyperparameter tuning through cross-validation.

- Filter Methods:
  - Information Gain
  - Chi-Square
- Principal Component Analysis

We followed the methodology proposed in the academic paper - DeepDetect: A Practical On-device Android Malware Detector[2]. We experimented with reducing the number of features to 100, 250, 500, 800, and 1000 features. The following figure shows that the performance becomes constant when the number of features is around 350. Still, to be on the good side, we have picked up the top 500 features for our model.

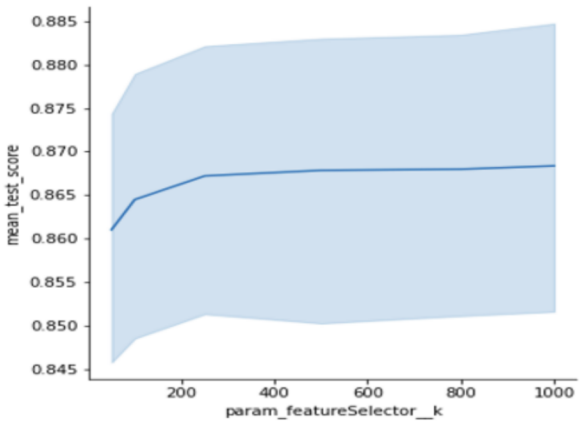

Figure 2: Number of features vs Accuracy

4.4 Classifier Models

We have experimented with the following classifier models:

- Logistic Regression
- Decision Tree
- Random Forest
- Support Vector Classification
- K-Nearest Neighbours

5 Results

5.1 Without Hyperparameter Tuning

5.1.1 Permissions as features

Results on Permission Features

Models Used	Accuracy	Recall	Precision	F1 score
Logistic Regression	92.99%	75.90%	87.50%	81.28%
Decision Tree	92.99%	87.95%	79.34%	83.42%
Random Forest	95.16%	79.51%	95.65%	86.83%
SVM	93.96%	73.49%	95.31%	82.98%
KNN	93.96%	85.54%	84.52%	85.02%

Figure 3: Results for the permissions as features

5.1.2 API Call as features

Results on API-Call Features

Models Used	Accuracy	Recall	Precision	F1 score
Logistic Regression	92.89%	94.38%	90.32%	92.30%
Decision Tree	89.34%	93.25%	84.69%	88.76%
Random Forest	93.40%	94.38%	91.30%	92.81%
SVM	91.37%	94.38%	87.50%	90.80%
KNN	92.38%	88.76%	94.04%	91.32%

Figure 4: Results for the API Call as features

5.1.3 API Call + Permissions as features

Results on API+ Permission Call Features

Models Used	Accuracy	Recall	Precision	F1 score
Logistic Regression	93.40%	97.75%	88.77%	93.04%
Decision Tree	89.84%	92.13%	86.31%	89.12%
Random Forest	94.92%	95.50%	93.40%	94.43%
SVM	87.81%	91.01%	83.50%	87.09%
KNN	73.60%	94.38%	64.12%	76.36%

Figure 5: Results for the API Call + Permissions as features

5.2 With Hyperparameter Tuning

5.2.1 Permissions as features

Results on Permission Dataset
(Feature Selection + Hyperparameter Tuning(PCA))

Models Used	Accuracy	Recall	Precision	F1 score
Logistic Regression	92.75%	73.49%	88.40%	80.25%
Decision Tree	89.13%	68.67%	75.00%	71.69%
Random Forest	93.71%	73.49%	93.84%	82.42%
SVM	95.16%	84.33%	90.90%	87.49%
KNN	94.44%	84.33%	87.50%	85.88%

Figure 6: Results for the permissions as features

Results on API Dataset
(Feature Selection + Hyperparameter Tuning(PCA))

Models Used	Accuracy	Recall	Precision	F1 score
Logistic Regression	93.90%	96.62%	90.52%	93.47%
Decision Tree	86.80%	86.51%	84.61%	85.55%
Random Forest	92.38%	87.64%	95.12%	91.22%
SVM	91.37%	84.26%	96.15%	89.82%
KNN	90.86%	95.50%	85.85%	90.42%

Figure 7: Results for the API Call as features

6 Conclusions

We have successfully designed a working model with high prediction rates for all the metrics - Accuracy, Recall, Precision, and F1-score for detecting the malware from the android applications.

Results on API+Permission Dataset
(Feature Selection + Hyperparameter Tuning(PCA))

Models Used	Accuracy	Recall	Precision	F1 score
Logistic Regression	89.34%	95.50%	83.33%	89.00%
Decision Tree	91.37%	89.88%	90.90%	90.38%
Random Forest	94.41%	91.01%	96.42%	93.63%
SVM	91.87%	89.88%	91.95%	90.90%
KNN	90.35%	88.76%	89.77%	89.26%

Figure 8: Results for the API Call + Permissions as features

Our designed feature selection technique can reduce a large number of features (500K) into only 400 features without affecting the model's performance. This reduction in the number of features provides the model with significant speedup that, after some certain modifications, it can deploy to a real-time device.

7 Limitations

- Any kind of permissions outside the manifest file is not accessible.
- All the API calls made by the application is present in the .dex file.
- Our proposed method, i.e., static analysis, cannot handle the part of the code downloaded during execution.
- Attacker makes the API call obscure, which cannot be detected by Androguard, which can impact the model's accuracy.

References

[1] S. Kumar, D. Mishra, B. Panda, and S. K. Shukla, "Deepdetect: A practical on-device android malware detector," in *2021 IEEE 21st International Conference on Software Quality, Reliability and Security (QRS)*, pp. 40–51, IEEE, 2021.

[2] K. Arun, S. Kumar, D. Mishra, and B. Panda, "Snip: An efficient stack tracing framework for multi-threaded programs," 2022.