WWW.IJRPR.COM

International Journal of Research Publication and Reviews

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

SecureDroid: A Static-Analysis Android Malware Screening Prototype and a Genetic-Algorithm Feature-Selection Study on the CICMalDroid 2020 Dataset

Kanya V P[a], Manya A[a], Monisha N[a], Varsha H A[a], Shilpa S[a]

[a]Department of Computer Science & Engineering, Kalpataru Institute of Technology, Tiptur 572201, Karnataka, India
Affiliated to Visvesvaraya Technological University, Belagavi, Karnataka, India

ABSTRACT :

Android's open application ecosystem makes it an appealing target for many kinds of malware. Plenty of detection techniques have been proposed, but their results are often hard to reproduce and harder to interpret. In this work we apply a Genetic Algorithm (GA) to feature selection on the public CICMalDroid 2020 dataset, and we describe SecureDroid, a web-based prototype that statically analyses Android Package Kit (APK) files. The experiments use the dataset's system-call frequency features. We first removed 77 duplicate records, which left 11,521 samples with 139 features across five classes, and divided them with a stratified 80:20 training–testing split. A Random Forest classifier evaluated the features. The GA's fitness measure was the three-fold cross-validated macro-F1 score, computed on the training set only, so the test data never influenced which features were chosen. The baseline model trained on all 139 features reached 94.27% accuracy and a macro-F1 of 93.21% on the test set. The GA-selected model kept 74 features, a 46.8% reduction, and reached 93.54% accuracy with a macro-F1 of 92.21%. Feature selection therefore shrank the feature set but did not improve classification performance. Because the evaluation rests on a single train–test split and a fairly small genetic search, repeated cross-validation and further datasets are needed before the results can be called reliable or generalizable.

Keywords: Android malware detection, static analysis, Genetic Algorithm, feature selection, Random Forest, CICMalDroid 2020.

1. Introduction

Android's open design allows applications to be installed from official stores, third-party marketplaces or files downloaded directly. That flexibility is good for users and developers, but it also opens several routes onto a device for malicious apps. Common threats include adware, banking malware, SMS-based malware and riskware. Signature-based detection can miss new variants of known malware or repackaged applications, and dynamic analysis normally requires a controlled execution environment and more processing time. On top of this, many machine-learning detection studies are difficult to reproduce or interpret. Experiments therefore need to be documented properly and evaluated under clearly defined procedures.

This paper reports a feature-selection experiment on the public CICMalDroid 2020 dataset and presents SecureDroid, a web application we built alongside the study. In the experiment, a Genetic Algorithm (GA) picks a subset of the system-call frequency features, and a Random Forest trained on that subset is compared against a baseline Random Forest that uses every feature. SecureDroid itself is a separate, application-level component. A user uploads an Android Package Kit (APK) file, the application validates it and extracts information for static inspection, and it then returns a rule-based risk assessment with explanations.

Our central research question (RQ1) is whether a Genetic Algorithm can reduce the number of system-call features in CICMalDroid 2020 without a substantial loss in classification performance. We do not propose a new classification algorithm. Instead, we look at the practical trade-off between fewer features and predictive performance under a clearly specified experimental protocol.

This work makes the following contributions:

- a GA feature-selection experiment on real data, with fitness computed on the training split only, so the test split never influences which features are selected;
- an honest result: the GA removed 46.8 % of the features but did not improve accuracy;
- complete per-class results and confusion matrices for both models;
- a documented web application with deterministic, explainable analysis modules, and a clear statement of what has not yet been evaluated.

The rest of the paper is organised as follows. Section 2 reviews related work, Section 3 presents the application, Section 4 describes the experimental method, Section 5 reports the results, Section 6 discusses them along with the limitations, and Section 7 concludes and outlines future work.

2. Related work

Surveys of machine-learning-based Android malware detection cover sample collection, preprocessing, feature selection, learning models and how detection effectiveness is assessed (Liu et al., 2020). Static approaches inspect an application without running it. Drebin gathered a broad set of static

features from the manifest and the code, embedded them in a vector space and classified them with a learning-based model; its authors also emphasised that each detection can be explained to the user (Arp et al., 2014). Permission analysis has a long history too. Felt et al. (2011) analysed compiled Android applications and found that developers often request permissions their code never uses, which is one reason permission lists are informative but noisy features. Zhou and Jiang (2012) characterised Android malware families and traced how they evolved. TaintDroid, in contrast, takes the dynamic route by tracking the flow of private data on a running device (Enck et al., 2010).

CICMalDroid 2020, released by the Canadian Institute for Cybersecurity, contains more than 17,000 samples in five categories: adware, banking malware, SMS malware, riskware and benign applications (Canadian Institute for Cybersecurity, n.d.). It has previously been used for semi-supervised classification with pseudo-label deep networks on dynamic behaviour profiles (Mahdavifar et al., 2020) and for a pseudo-label stacked auto-encoder on hybrid static and dynamic features (Mahdavifar et al., 2022). Those studies differ from ours in features, models and protocols, so we do not compare numbers with them.

Feature selection reduces dimensionality and can improve efficiency and, sometimes, accuracy (Guyon & Elisseeff, 2003). Genetic algorithms (Holland, 1975) have been used as wrapper methods, searching the space of feature subsets and scoring each candidate with a classifier (Yang & Honavar, 1998). We apply this wrapper idea in its simplest form to see whether it pays off on this dataset.

3. The SecureDroid application

Architecture

SecureDroid is a web application that examines Android Package Kit (APK) files through static analysis and gives users an understandable picture of possible security risks. It uses a three-tier architecture: a frontend interface, a backend processing service and a database layer.

The frontend is a React single-page interface where users upload APK files and use the analysis features. It talks to the backend by sending requests as JSON.

The backend is built with FastAPI. It validates uploaded files, runs the analysis operations and coordinates the application's modules. With Androguard, it pulls information out of each uploaded APK, such as the requested permissions, references to potentially suspicious APIs and counts of application components. That information feeds the later analysis and the risk assessment.

Results of each analysis are stored in an SQLite database. Besides the core scan, SecureDroid has several supporting modules: a rule-based risk assessment engine, a plain-language explanation generator, security recommendations, a dashboard, scan comparison and a lexical URL scanner. The URL scanner judges a submitted address from its text alone and never contacts the target website.

All analysis modules are deterministic, meaning the same extracted information and rules always produce the same result. This keeps the system transparent and makes it possible to trace how each extracted characteristic contributes to the final assessment.

1. In short, the architecture joins a user-friendly frontend, a backend analysis service and persistent storage into one structured workflow for examining Android applications. The prototype's rule-based analysis should not be confused with the machine-learning experiment on CICMalDroid 2020, though. The quantitative results in this study come from the dataset experiment and say nothing about the detection accuracy of SecureDroid's live APK analysis.

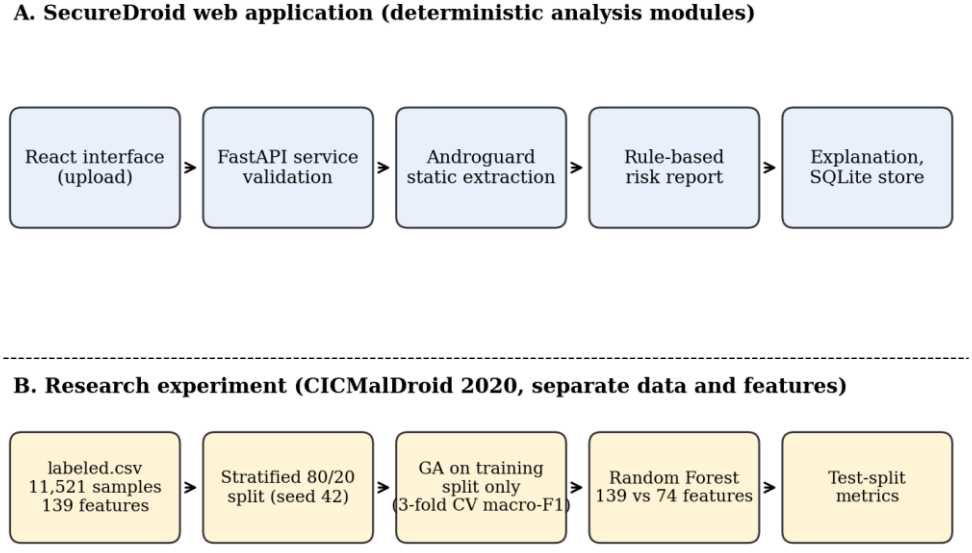


Fig. 1 - Overview of (A) the SecureDroid web application and (B) the CICMalDroid experiment evaluated in this paper.

Testing and scope of this paper

Twenty-six automated tests covering the server and the interface passed in the final review. Tests like these confirm that the software meets its specification; they tell us nothing about how well malware is detected. The quantitative evaluation in this paper is the experiment in Sections 4 and 5, which uses the real CICMalDroid 2020 data. The application's analysis modules are not evaluated on real applications here, and connecting a model trained on real data to the application is left for future work.

4. Experimental method

Dataset and split

The experiment uses labeled.csv from CICMalDroid 2020, which holds 139 system-call frequency features and a class label for every sample. These are dynamic features, unlike the static features used by the live scanner. The file has 11,598 rows, of which 77 were exact duplicates; we deleted them, leaving 11,521 samples with no missing values. The file does not store class names, so we assigned them by matching the number of samples per label to the counts published for the dataset (Canadian Institute for Cybersecurity, n.d.). The match is exact for all five labels (Table 1). A single stratified 80/20 split with random seed 42 gave 9,216 training and 2,305 test samples. The classes are unbalanced (SMS is roughly three times the size of Adware), so we report macro-averaged metrics alongside accuracy and apply balanced class weights.

Table 1 - Class distribution of the research dataset.

Class	Raw	After duplicate removal	Test split	Share of test (%)
1 (Adware)	1,253	1,253	251	10.9
2 (Banking)	2,100	2,044	409	17.7
3 (SMS)	3,904	3,903	781	33.9
4 (Riskware)	2,546	2,529	506	22.0
5 (Benign)	1,795	1,792	358	15.5
Total	**11,598**	**11,521**	**2,305**	**100.0**

Genetic algorithm and classifier

$$\text{fitness}(c) = (1/3)\ \Sigma_{k=1..3}\ \text{macro-F1}_k(\text{RF trained on the folds, using columns where } c = 1) \quad (1)$$

Table 2 - Settings of the research experiment.

Item	Value
Classifier	Random Forest (Breiman, 2001), 200 trees, balanced class weights, seed 42; no feature scaling
GA	Population 12, 5 generations, tournament selection (k = 3), one-point crossover (probability 0.8), bit-flip mutation (0.03 per gene), elitism, repair of empty masks
Compared models	Baseline (all 139 features) and GA + Random Forest (selected features)
Metrics	Accuracy, macro precision, macro recall, macro F1 and confusion matrix, on the held-out test split only
Software	Python 3.12, scikit-learn 1.8.0 (Pedregosa et al., 2011)

Feature reduction is calculated with Eq. (2); here it comes to (139 − 74)/139 × 100 = 46.8 %.

$$\text{Feature reduction } (\%) = (N_{orig} - N_{sel}) / N_{orig} \times 100 \quad (2)$$

Evaluation metrics

For each class c, TP, FP and FN denote the true positives, false positives and false negatives on the held-out test split. Eq. (3) defines the precision, recall and F1-score of a class. The macro-averaged F1-score in Eq. (4) weights each of the K = 5 classes equally, which suits classes of unequal size. Accuracy is the share of test samples predicted correctly. Every metric is computed on the test split only, never on training data.

$$P_c = TP_c / (TP_c + FP_c),\ \ R_c = TP_c / (TP_c + FN_c),\ \ F1_c = 2P_cR_c / (P_c + R_c) \quad (3)$$

$$\text{Macro-F1} = (1/K)\ \Sigma_{c=1..K}\ F1_c \quad (4)$$

5. Experimental results

During the search, the best cross-validated macro-F1 on the training split rose from 0.9072 in the first generation to 0.9103 in the fifth, and the best chromosome grew from 61 to 74 selected features. These values come from the training split and are not test results. The final mask keeps 74 of the 139 features.

Table 3 compares the two models on the held-out test split. The baseline comes out ahead on accuracy and on all three macro metrics, and the GA-selected model gets 17 fewer test samples right (2,156 compared with 2,173 out of 2,305).

Table 3 - Baseline versus GA-selected model on the test split (n = 2,305). Differences are as reported in the project report.

Metric	Baseline (139 features)	GA + RF (74 features)	Difference (points)
Accuracy (%)	94.27	93.54	−0.73
Macro precision (%)	93.34	92.21	−1.12
Macro recall (%)	93.10	92.24	−0.86
Macro F1-score (%)	93.21	92.21	−1.00
Number of features	139	74	−65
Feature reduction (%)	0	46.8	

Table 4 gives the per-class results. The drop appears in every class. It is smallest for SMS, the largest class, where F1 goes from 98.22 % to 98.09 %, and biggest for Adware, the smallest class, where F1 falls from 91.02 % to 88.45 %. Benign recall slips from 90.78 % to 89.11 %. In the confusion matrices (Fig. 2), the baseline sends 33 benign test samples to a malware class while the GA model sends 39. Flagging a benign application as malware is the kind of error that drives the false-positive rate, so this small increase is worth noting, although on a single split it may not be stable.

Table 4 - Per-class precision (P), recall (R) and F1 (%) on the test split.

Class	Support	Base P	Base R	Base F1	GA P	GA R	GA F1
1 (Adware)	251	91.20	90.84	91.02	86.92	90.04	88.45
2 (Banking)	409	94.46	91.69	93.05	94.88	90.71	92.75
3 (SMS)	781	97.36	99.10	98.22	97.35	98.85	98.09
4 (Riskware)	506	94.39	93.08	93.73	93.79	92.49	93.13
5 (Benign)	358	89.29	90.78	90.03	88.12	89.11	88.61
Macro average	**2305**	**93.34**	**93.10**	**93.21**	**92.21**	**92.24**	**92.21**

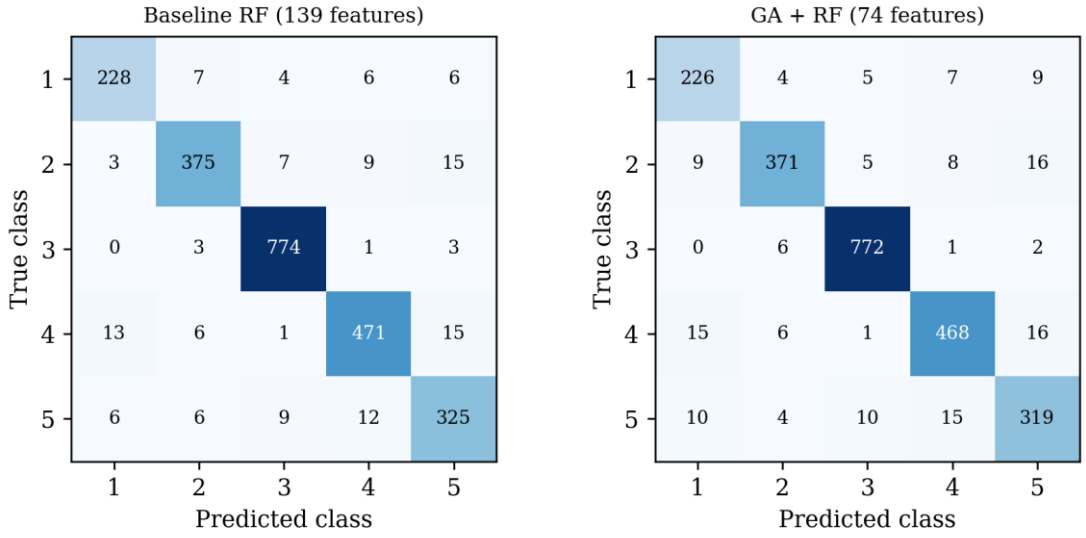


Fig. 2 - Confusion matrices on the test split (rows: true class, columns: predicted class; 1 Adware, 2 Banking, 3 SMS, 4 Riskware, 5 Benign).

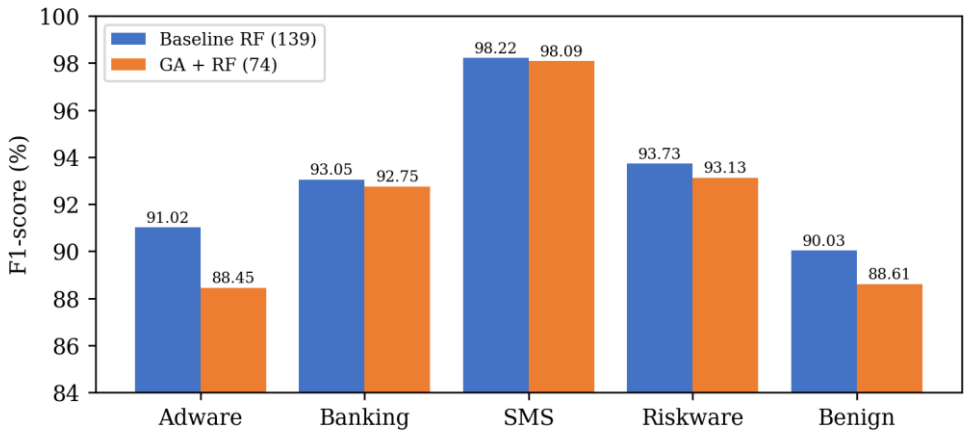


Fig. 3 - Per-class F1-score of the baseline and the GA-selected model.

Table 5 lists the largest off-diagonal entries of the two confusion matrices. In both models the most frequent errors involve the Benign and Riskware classes. That is plausible, since riskware consists of legitimate programs that can be misused (Canadian Institute for Cybersecurity, n.d.). The GA-selected model makes slightly more errors of each kind, in line with its lower recall for Benign, Banking and Riskware in Table 4.

Table 5 - Largest off-diagonal entries of the confusion matrices (true class → predicted class, number of test samples).

Rank	Baseline (139 features)	Count	GA + RF (74 features)	Count
1	Riskware → Benign	15	Banking → Benign	16
2	Banking → Benign	15	Riskware → Benign	16
3	Riskware → Adware	13	Benign → Riskware	15
4	Benign → Riskware	12	Riskware → Adware	15

On RQ1, the GA reduced the number of features by 46.8 % in this experiment, at a cost of 0.7 to 1.1 percentage points on the reported metrics, and without improving classification performance. Whether that trade-off is acceptable depends on the deployment. Fewer features mean a smaller model and faster training and inference, but we did not measure run time or memory.

6. Discussion and limitations

The main finding is modest and negative: GA-based feature selection with a small budget gave us a smaller model, not a better one. We report it anyway, because a documented null result from a leakage-free protocol is more useful to readers than an improvement nobody has examined. The limitations are as follows.

- **Single split.** All test metrics come from one stratified split with one random seed, and no standard deviation is given, so differences below one percentage point cannot be considered significant. Repeated cross-validation is needed to confirm them.
- **Small GA budget.** A population of 12 over 5 generations is small. A larger search, or a fitness function that penalises the feature count, might yield a different subset and trade-off.
- **One dataset and one feature type.** The experiment uses the dynamic system-call frequencies of one dataset, so its findings cannot be carried over to other datasets or to static permission and API features.
- **Dataset age.** The CICMalDroid samples were collected in 2017–2018 and may not reflect current malware. Published results on this dataset use different features, models and protocols, so we do not compare against them here.
- **Application not evaluated.** The application's analysis modules were not tested on real applications in this paper, so we make no detection claim for them. Rate limiting is configured but not enforced, and the system has no authentication, so in its present form it should not be deployed publicly.

Mistakes by a malware detector have real consequences. A false positive may push a user to remove a legitimate application, and a false negative may create false confidence. The output of such a system should be treated as one input to a decision and checked with independent tools. We see SecureDroid as an educational and research prototype, not a substitute for a commercial anti-malware product.

The design reflects these concerns: uploaded files are never executed, submitted URLs are never visited, every verdict is presented as an estimate, and secrets are kept out of the source code. Users should not upload files containing confidential data to any hosted instance without permission.

Others can repeat the experiment: the dataset is publicly available from the Canadian Institute for Cybersecurity, the split is stratified with a fixed random seed, and the classifier and GA settings are given in Table 2. The reported figures come from a single run; repeating the experiment with other seeds is part of the future work.

7. Conclusion and future scope

On the real CICMalDroid 2020 dataset, a genetic algorithm shrank the system-call feature set from 139 to 74 features (46.8 % fewer). The Random Forest trained on them reached 93.54 % accuracy and 92.21 % macro-F1, against 94.27 % and 93.21 % with all features. So the GA cut the feature set at a small cost and did not improve accuracy. Future work should (i) apply repeated stratified cross-validation with standard deviations and a nested protocol; (ii) widen the GA search, add a feature-count penalty and compare other selection methods; (iii) assess further Android datasets and the static features of CICMalDroid 2020; (iv) extract the application's static features from a labelled corpus of real benign and malicious APKs, train and evaluate its classifiers on them and report false-positive and false-negative rates; (v) generate per-prediction explanations and test them with users; and (vi) add sandboxed dynamic analysis for behaviour that static analysis cannot observe.

8. Acknowledgements

The authors are grateful to their guide, Shilpa S, for her valuable guidance throughout this work, and to the Principal, Dr. Sateesh Kumar H C, the Head of the Department of Computer Science and Engineering, Dr. Maithri C, and the project coordinators, Dr. Sanjay Kumar N V and Dr. Kavitha M G, of Kalpataru Institute of Technology, Tiptur, for their support.

REFERENCES :

1. Arp, D., Spreitzenbarth, M., Hübner, M., Gascon, H., & Rieck, K. (2014). Drebin: Effective and explainable detection of Android malware in your pocket. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, USA. https://doi.org/10.14722/ndss.2014.23247

2. Breiman, L. (2001). Random forests. *Machine Learning, 45*(1), 5–32. https://doi.org/10.1023/A:1010933404324
3. Canadian Institute for Cybersecurity. (n.d.). *Android malware dataset (CICMalDroid 2020)*. University of New Brunswick. https://www.unb.ca/cic/datasets/maldroid-2020.html (accessed 27 October 2026).
4. Desnos, A., Gueguen, G., et al. (n.d.). *Androguard: Reverse engineering, malware and goodware analysis of Android applications* [Software]. https://github.com/androguard/androguard
5. Enck, W., Gilbert, P., Chun, B.-G., Cox, L. P., Jung, J., McDaniel, P., & Sheth, A. N. (2010). TaintDroid: An information-flow tracking system for realtime privacy monitoring on smartphones. In *Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*.
6. Felt, A. P., Chin, E., Hanna, S., Song, D., & Wagner, D. (2011). Android permissions demystified. In *Proceedings of the 18th ACM Conference on Computer and Communications Security (CCS)*. https://doi.org/10.1145/2046707.2046779
7. Guyon, I., & Elisseeff, A. (2003). An introduction to variable and feature selection. *Journal of Machine Learning Research, 3*, 1157–1182.
8. Holland, J. H. (1975). *Adaptation in natural and artificial systems*. Ann Arbor, MI: University of Michigan Press.
9. Liu, K., Xu, S., Xu, G., Zhang, M., Sun, D., & Liu, H. (2020). A review of Android malware detection approaches based on machine learning. *IEEE Access, 8*. https://doi.org/10.1109/ACCESS.2020.3006143
10. Mahdavifar, S., Kadir, A. F. A., Fatemi, R., Alhadidi, D., & Ghorbani, A. A. (2020). Dynamic Android malware category classification using semi-supervised deep learning. In *Proceedings of the IEEE International Conference on Dependable, Autonomic and Secure Computing (DASC)*. https://ieeexplore.ieee.org/abstract/document/9251198
11. Mahdavifar, S., Alhadidi, D., & Ghorbani, A. A. (2022). Effective and efficient hybrid Android malware classification using pseudo-label stacked auto-encoder. *Journal of Network and Systems Management, 30*, Article 22.
12. Pedregosa, F., et al. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research, 12*, 2825–2830.
13. Yang, J., & Honavar, V. (1998). Feature subset selection using a genetic algorithm. *IEEE Intelligent Systems, 13*(2), 44–49.
14. Zhou, Y., & Jiang, X. (2012). Dissecting Android malware: Characterization and evolution. In *Proceedings of the IEEE Symposium on Security and Privacy*. https://doi.org/10.1109/SP.2012.16