

Optimizing Random Forest Using Genetic Algorithm for Static Android Malware Detection

Youssef Zaz^{a,*}, Muhamad Taufiq^b and Barroon Isma'eel Ahmad^c

^aAbdelmalek Essaâdi University, Morocco

^bNusa Putra University, Indonesia

^cAl Buraimi University, Oman

ARTICLE INFO

Keywords:

random forest
genetic algorithm
drebin-215
malware detection
static analysis
android malware detection
hyperparameter tuning

ABSTRACT

The massive and increasingly complex growth of Android malware has led to a heightened threat to mobile device security. One proven effective approach is the use of machine learning, particularly the Random Forest algorithm. However, many previous studies have not fully utilized the potential of this algorithm, as they rely on default configurations without tuning. This study implements the Genetic Algorithm (GA) method to perform hyperparameter tuning on Random Forest, using the Drebin-215 dataset, which consists of 15,036 APKs and 215 static features. The study compares two configurations: Default and GA Tuning. Evaluation is conducted using 5-fold cross-validation and performance metrics including accuracy, precision, recall, and F1-score. The results show that the GA-tuned model delivers the best performance, achieving an accuracy of 98.97%. The study also tests the model on 55 real APKs to evaluate its predictive capability in real-world cases.

1. Introduction

The rapid increase in both the volume and complexity of Android malware poses a serious threat to the security of mobile devices and the services that rely on them. To address this challenge, machine learning has been widely applied in Android malware detection (Yerima and Sezer, 2019), with algorithms such as Random Forest, Support Vector Machine (SVM), and Naïve Bayes commonly used. Among these algorithms, Random Forest has demonstrated relatively high accuracy (Agrawal and Trivedi, 2021).

One of the most widely used datasets in malware detection research is DREBIN, which provides clearly labeled malware families and applies a lightweight static analysis approach to extract features from application bytecode and the manifest file (Arp et al., 2014; Morán et al., 2024). Although machine learning-based malware detection research has grown rapidly in recent years, studies specifically using the Drebin or Drebin-215 datasets remain limited and still tend to rely on conventional methods such as Random Forest, SVM, or Naïve Bayes without systematic optimization (Morán et al., 2024).

However, hyperparameter optimization is crucial for improving model accuracy and efficiency, directly affecting evaluation metrics such as accuracy, precision, recall, and

F1-score (Arp et al., 2014; Mahindru and Singh, 2017; Yang and Shami, 2022). One optimization method that can be applied is the Genetic Algorithm (GA), which has proven effective in optimizing various machine learning models (Yang and Shami, 2022; Wicaksono and Supianto, 2018).

This study aims to optimize the Random Forest algorithm using a Genetic Algorithm to improve Android malware detection performance. The study uses the Drebin-215 dataset, which consists of 215 static features extracted from APK files and includes a total of 15,036 samples. By combining Random Forest with GA-based hyperparameter tuning, this research seeks to obtain a more effective classification model for static Android malware detection.

The results of this study are expected to provide a meaningful contribution to improving the effectiveness of machine learning-based malware detection in a more optimal and practical manner (Widardo and Chowanda, 2025; Aftab et al., 2025).

2. Related Works

Most studies indicate that Random Forest is a fairly effective algorithm for Android malware detection (Agrawal and Trivedi, 2021; Rana et al., 2020); however, it is generally used with default parameters without systematic optimization (Morán et al., 2024; Yang and Shami, 2022).

*Corresponding author

 muhamad.taufiq@nusaputra.ac.id (M. Taufiq)

ORCID(s):

Optimization methods such as Genetic Algorithm (GA) and Random Search have proven to be efficient in improving model performance across various domains (Yang and Shami, 2022; Wicaksono and Supianto, 2018; Albadr et al., 2020), but they have not been widely applied for tuning Random Forest in the context of Android malware detection (Morán et al., 2024; Mahindru and Singh, 2017; Arp et al., 2014).

Therefore, there is a promising research gap in combining Random Forest with optimization techniques like GA to enhance the accuracy and efficiency of malware detection (Siregar et al., 2023; Togatorop et al., 2022), particularly using benchmark datasets such as Drebin-215.

Table 1: Summary of Previous Studies

No	Study and Main Finding	Research Gap
1	Agrawal and Trivedi (2021) applied several machine learning algorithms for Android malware detection and found that Random Forest achieved strong performance.	Hyperparameter tuning was not systematically explored.
2	Vanusha et al. (2024) used Drebin-215 with several classifiers and reported strong performance, especially from Neural Network models.	The study focused on neural models rather than GA-based optimization of Random Forest.
3	Mahindru (2020) applied Random Forest on the Drebin dataset and obtained high accuracy for malware detection.	The model mainly relied on default configuration without systematic optimization.
4	Yang et al. (2022) reviewed hyperparameter optimization methods and showed that GA can outperform Random Search in efficiency and effectiveness.	The study was general and not specifically applied to Android malware detection.
5	Jamshidnezhad (2011) showed that GA could improve classification performance by optimizing SVM.	The work was conducted in facial expression recognition, not malware detection.
6	Hadiprakoso (2022) evaluated supervised learning methods for Android malware detection and reported strong results from SVM.	Random Forest was not the focus, and evolutionary optimization was not used.
7	Rana et al. (2020) compared several tree-based models for Android malware detection and reported good performance.	Systematic tuning methods such as GA and Random Search were not emphasized.
8	Yerima and Sezer (2019) proposed a feature-fusion based model for Android malware detection with promising results.	The focus was on ensemble fusion rather than optimizing a single Random Forest model.
9	Arp et al. (2014) introduced the DREBIN benchmark and demonstrated the effectiveness of static analysis for Android malware detection.	The study did not investigate Random Forest optimization.
10	Albadr et al. (2020) explained the effectiveness of Genetic Algorithm in solving optimization problems across domains.	The work did not specifically address Random Forest tuning for Android malware detection.
11	Wicaksono and Supianto (2018) showed that GA can optimize machine learning hyperparameters efficiently and achieve competitive results.	The study was not applied to Android malware detection.

3. Terminology

3.1. Random Forest

Random Forest is an ensemble learning algorithm that constructs multiple decision trees during training and outputs the mode of the classes (classification) or mean prediction (regression) of the individual trees (Dutt et al., 2018). It

operates by randomly sampling subsets of data and features, offering robustness against overfitting and high accuracy, particularly in high-dimensional datasets. Its flexibility allows it to be used for both classification and regression tasks. (Dutt et al., 2018)

3.2. Hyperparameter Optimization Methods

Hyperparameter optimization refers to the process of selecting the best combination of model parameters that are not learned during training (e.g., number of trees, tree depth). Common optimization methods include manual tuning, Grid Search, Random Search, and metaheuristic approaches such as Genetic Algorithms. These methods aim to improve model performance metrics such as accuracy, precision, recall, and F1-score (Yang and Shami, 2022).

3.3. Genetic Algorithm

Genetic Algorithm (GA) is a bio-inspired metaheuristic optimization technique based on the principles of natural selection and genetics. It evolves a population of candidate solutions through iterative processes of selection, crossover, and mutation. In the context of hyperparameter tuning, GA represents each set of hyperparameters as a chromosome, evolving over generations to find the configuration that yields the highest model performance (Albadr et al., 2020).

3.4. Hyperparameter Tuning in Random Forest

In Random Forest, key hyperparameters include `n_estimators`, `max_depth`, `criterion`, `min_samples_split`, `min_samples_leaf`, and `max_features`. Hyperparameter tuning involves finding the most effective combination of these parameters to maximize classification performance (Siregar et al., 2023; Togatorop et al., 2022). GA can encode these parameters into chromosomes and optimize them using evolutionary strategies to avoid overfitting and improve generalization. In Genetic Algorithm (GA)-based optimization, there are several key terms, including:

- a chromosome represents a potential solution—typically a vector of encoded hyperparameter values (Siregar et al., 2023).
- An individual refers to a single candidate solution that contains a chromosome (Togatorop et al., 2022).
- A population is a set of individuals in a given generation (Siregar et al., 2023).
- a generation is one cycle of evolution comprising evaluation, selection, crossover, and mutation (Togatorop et al., 2022).

In the context of Random Forest hyperparameter tuning, each chromosome might encode values such as `n_estimators`, `max_depth`, and `criterion`, while the population evolves over successive generations toward better-performing configurations (Togatorop et al., 2022).

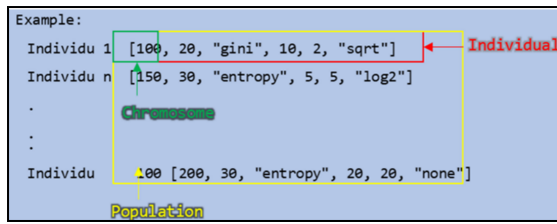


Figure 1: Chromosome, Individual and Population Representation

3.4.1. Threshold Function / Fitness Function

The evaluation of each individual within the population is carried out using a fitness function, which serves as the core of the selection process in a Genetic Algorithm (GA). The fitness value is calculated based on the average accuracy of the model on the training data, evaluated using a k-fold cross-validation scheme (Siregar et al., 2023). The purpose of the fitness function is to select hyperparameter configurations that yield the highest accuracy while avoiding overfitting. This is critical, as models that are overly fitted to the training data may fail to generalize effectively to unseen test data (Togatorop et al., 2022).

Description:

- P_g : refers to the population at generation- g ,
- $x_j^{(g)}$: is the j -th individual in the population,
- $M(\cdot)$: denotes the Random Forest model with the hyperparameter configuration from that individual,
- D_i : represents the i -th fold subset of the data,
- $k = 5$: is the number of folds used in cross-validation.

3.4.2. Evolution Stages: Selection, Crossover, and Mutation

The evolution stages in a Genetic Algorithm consist of three key operations:

Selection Selection identifies the most promising individuals based on their fitness scores. Techniques like tournament selection are often used to probabilistically favor better-performing candidates (Togatorop et al., 2022).

Crossover (or recombination) Crossover (or recombination) creates new offspring by exchanging gene segments between pairs of parent chromosomes. This allows the algorithm to explore new regions of the search space (Siregar et al., 2023).

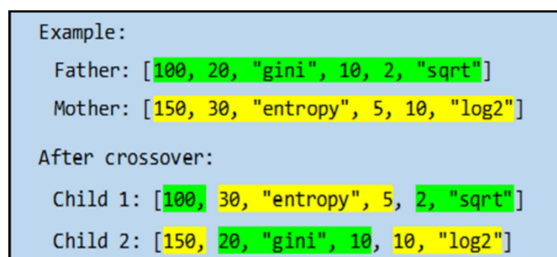


Figure 2: Crossover Visualization

Mutation Mutation introduces random changes to certain genes in an individual, helping to maintain diversity in the population and avoid premature convergence to sub-optimal solutions. These stages are repeated over multiple generations to improve the overall quality of the population (Wicaksono and Supianto, 2018).

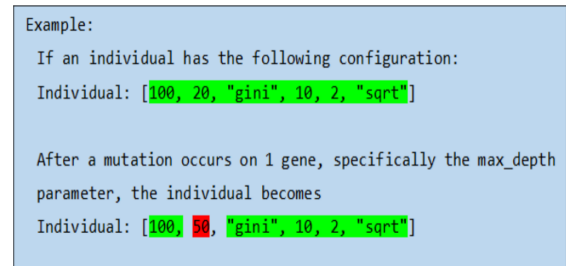


Figure 3: Mutation Visualization

3.5. Drebin Dataset

The Drebin dataset is a widely used Android malware benchmark dataset based on static analysis. Introduced by Arp et al. (2014) (Arp et al., 2014), it contains over 15,000 labeled APK samples (malicious and benign) and extracts features from the manifest file and DEX bytecode without executing the application. A refined version, Drebin-215, uses a selected subset of 215 features to facilitate lightweight and interpretable malware classification.

3.6. Static Features Indicative of Android Malware

Static features are non-runtime attributes extracted from APKs, such as permissions, intents, API calls, and embedded resources. Specific features indicative of malware include dangerous permissions (e.g., SEND_SMS, READ_PHONE_STATE), suspicious API calls (e.g., getDeviceId(), DexClassLoader), and system broadcast intents (e.g., BOOT_COMPLETED) (Yerima and Sezer, 2019; Arp et al., 2014). These features are crucial for distinguishing malware from benign applications using static analysis techniques. The following is a list of features explicitly categorized as dangerous by both studies, grouped into several categories (Yerima and Sezer, 2019; Arp et al., 2014):

3.6.1. Dangerous Permissions

This feature represents high-level privileges that can be exploited by malware:

- SEND_SMS, RECEIVE_SMS, READ_SMS, WRITE_SMS
- READ_PHONE_STATE, CALL_PHONE, READ_CONTACTS, WRITE_CONTACTS
- READ_CALL_LOG, WRITE_CALL_LOG, RECORD_AUDIO, CAMERA
- INTERNET, ACCESS_NETWORK_STATE, ACCESS_FINE_LOCATION, ACCESS_COARSE_LOCATION

- READ_EXTERNAL_STORAGE,
WRITE_EXTERNAL_STORAGE,
INSTALL_PACKAGES, DELETE_PACKAGES
- SYSTEM_ALERT_WINDOW, WAKE_LOCK,
BIND_DEVICE_ADMIN,
BIND_ACCESSIBILITY_SERVICE

3.6.2. Broadcast Intents dan System Events

This feature represents high-level privileges that can be exploited by malware. Malware leverages system events to trigger automatic execution:

- android.intent.action.BOOT_COMPLETED
- android.intent.action.SCREEN_ON
- android.intent.action.SCREEN_OFF
- android.intent.action.PACKAGE_ADDED
- android.intent.action.PACKAGE_REMOVED
- android.intent.action.ACTION_POWER_CONNECTED
- android.intent.action.ACTION_POWER_DISCONNECTED
- android.intent.action.BATTERY_LOW
- android.intent.action.SEND
- android.intent.action.SENDTO
- android.provider.Telephony.SMS_RECEIVED
- android.intent.action.NEW_OUTGOING_CALL
- android.intent.action.RECEIVE_BOOT_COMPLETED

3.6.3. Dangerous API Calls (in .dex / smali)

Several API calls frequently identified as malicious patterns include:

- `getDeviceId()` (retrieves the IMEI)
- `getSimSerialNumber()` (accesses the SIM serial number)
- `getLine1Number()` (retrieves the phone number)
- `sendTextMessage()` (sends SMS)
- `DexClassLoader` (performs dynamic code loading)
- `exec()` or `Runtime.getRuntime().exec()` (executes shell commands)
- `URLConnection` or `URLConnection.connect()` (external connection)
- `TelephonyManager` (accesses operator information)

3.6.4. Malicious Embedded Files

Malware often hides additional code in the form of files that are loaded at runtime:

- Additional .dex files in the assets/ or res/raw/ folders
- .jar files for dynamic class loading
- .so libraries (native code)
- .exe files embedded within the APK

3.6.5. String Pattern & Native Commands

In DroidFusion, the extraction of string literals also reveals uncommon shell commands and URLs:

- Linux commands such as: `chmod`, `rm -rf`, `su`, `sh`, `mount`
- Static URLs or IP addresses: `http://`, `https://`, or numeric IP addresses within strings
- Suspicious filenames: `payload.dex`, `update.jar`, `exploit.so`

3.6.6. APK Structure Anomalies Commands

Certain file attributes or structures are considered abnormal:

- Extremely small or large .dex file size
- Presence of dummy resources (e.g., empty icons, unsynchronized manifest resources)
- Massive use of Base64 string encryption within the code.

4. Methodology

The research methodology in this study follows a common machine learning research model, as illustrated in Figure 4. This study is divided into five stages: Data Collection, Data Pre-processing, Model Selection, Model Training and Testing, and Final Model.

There are two main processes running simultaneously: first, converting the Drebin dataset into a machine-processable format, and second, decompiling real APK samples into a file named `sample.csv`, which is structured similarly to the feature columns in `drebin215.csv`.

The data collection process involves two primary sources: the Drebin-215 dataset, which consists of 15,036 APK files (including 5,560 malware and 9,476 benign applications), and real APK samples that have been decompiled and saved as `sample.csv`. Static analysis is conducted using tools such as `Apktool`, `AXMLPrinter2`, and `Baksmali`. These tools are used to extract features from the `AndroidManifest.xml` file (such as permissions and intents) and the .dex files (to analyze API call patterns). The extracted features are then converted into a binary CSV format, where a value of 1 indicates the presence of a feature and 0 indicates its absence.

gen	nevals	avg	min	max
0	100	0.980293	0.967268	0.988956
1	76	0.983559	0.972989	0.98889
2	69	0.986296	0.977646	0.989289
3	83	0.987359	0.971592	0.989422
4	71	0.987506	0.966536	0.989422
5	66	0.988378	0.97891	0.989422
6	75	0.988365	0.973455	0.989422
7	68	0.988497	0.978112	0.989422
8	68	0.988699	0.980973	0.989422
9	81	0.988612	0.967933	0.989422

Figure 4: Research Methodology

The data preprocessing phase ensures that the dataset is clean and suitable for machine learning. This includes removing irrelevant data, normalizing numerical values, and applying label encoding so that the features can be properly interpreted by the learning algorithm.

For model selection and optimization, the Random Forest algorithm is used as the baseline. To improve its performance, hyperparameters are optimized using a Genetic Algorithm (GA). The optimization process begins with an initial population of 100 individuals, each representing a different random combination of hyperparameters. This population evolves over 20 generations. Each individual is evaluated using a fitness function based on model accuracy through 5-fold cross-validation. The process includes stages

of selection, crossover, and mutation to improve the overall quality of the population.

The model is trained and tested in a computing environment running Windows 11, with an Intel i7 processor, 16 GB of RAM, and Python 3.11. The model is evaluated using 5-fold cross-validation, splitting the data into 80% for training and 20% for testing. Evaluation metrics include accuracy, precision, recall, and F1-score.

The final model is a Random Forest classifier with the best hyperparameter configuration found by the Genetic Algorithm. This optimized model is then integrated into a Python-based static analyzer system, which is capable of predicting whether an APK file is malware or not (Yerima and Sezer, 2019; Arp et al., 2014).

5. Performance of Random Forest with Hyperparameter Optimization Using GA

After the initialization of the population, definition of the fitness function, and determination of generation count and population size, the Genetic Algorithm proceeds with evolutionary operations consisting of selection, crossover, and mutation. Each individual in the population—representing a unique combination of Random Forest hyperparameters—is evaluated using a fitness function based on the average accuracy from 5-fold cross-validation. Individuals with higher fitness scores are more likely to be selected as parents through tournament selection (with $k = 3$), followed by uniform crossover applied to 70% of the population, enabling genetic information exchange between selected pairs. Mutation is then introduced to 20% of individuals to maintain population diversity and prevent premature convergence by randomly altering hyperparameter values. This cycle of evaluation and evolution is repeated over 20 generations, with the population being updated iteratively until the optimal hyperparameter configuration is found.

The best individual found was: Best Hyperparameters Found: {'n_estimators': 131, 'max_depth': 45, 'criterion': 'gini', 'min_samples_split': 2, 'min_samples_leaf': 1, 'max_features': 'log2'}, with the Best Fitness (Accuracy): 0.9897, corresponding to an accuracy of 98.97%.

Table 2: Hyperparameter Recapitulation

No	Hyperparameter	Default	GA Tuned
1	n_estimator	100	131
2	max_depth	none	45
3	criterion	gini	gini
4	min_samples_split	2	2
5	min_samples_leaf	1	1
6	max_features	sqrt	log2

After hyperparameter optimization, the model's performance improved across all metrics. The accuracy increased from 98.87% to 98.97%, indicating a higher prediction accuracy. Precision rose from 99.29% to 99.43%, reflecting more accurate positive predictions. Recall improved from 97.64% to 97.77%, showing the model's enhanced ability to detect more positive instances than before. The F1-Score

also increased from 98.46% to 98.59%, representing a better balance between precision and recall.

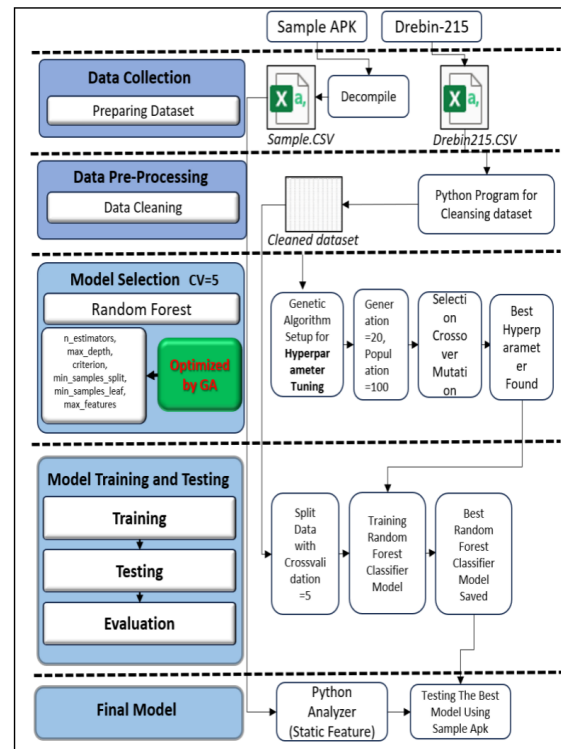


Figure 5: GA Tuning Results on Random Forest

This comparison shows that the optimization process successfully improved the model's performance. These results confirm that hyperparameter tuning can positively impact the model's ability to perform classification tasks with greater precision.

Table 3: Model Evaluation

No	Evaluation Criteria	Default	GA Tuned
1	Accuracy	98,87	98,97
2	Precision	99,29	99,43
3	Recall	97,64	97,77
4	F1-score	98,46	98,59
5	Runtime	13.7 sec	35 min and 27 sec

6. Malware Detection on Sample APKs

The best hyperparameters obtained from the tuning process were then applied to the Random Forest configuration. A total of 55 APK samples—comprising 25 malware samples from Malware Bazaar, 5 real malware samples, and 25 common APKs—were decompiled and analyzed using a static analysis approach, as applied by Yerima et al. (2019). The purpose of this evaluation is to assess the performance of the Random Forest model optimized using Genetic Algorithm (GA) and trained on the Drebin-215 benchmark dataset.

6.1. APK Analysis Procedure

The analysis of APK files begins with the decompilation and extraction of the application structure. Tools such as

Apktool or JADX are employed to decompile the APK and access its internal components. Key files analyzed during this process include the `AndroidManifest.xml`, which reveals requested permissions and declared application components; the `classes.dex` file, which contains bytecode and can be used to identify invoked API functions; and various directories such as `res/`, `assets/`, and `smali/`, which may contain embedded resources or obfuscated code fragments. These components provide valuable insights into the static behavior of the application.

Following this, static feature extraction is performed without executing the application. The features extracted include suspicious permissions like `SEND_SMS`, `READ_PHONE_STATE`, and `INTERNET`; hidden components such as uncommon services or broadcast receivers; and high-risk API calls including `getDeviceId()`, `getSimSerialNumber()`, and `DexClassLoader()`. Additionally, the analysis looks for embedded executable files such as `.dex`, `.jar`, `.so`, or `.exe`, as well as hardcoded IP addresses, URLs, and Linux shell commands found in string literals. These extracted features are then converted into a binary vector representation, where each feature is assigned a value of 1 if present or 0 if absent, in accordance with the Drebin methodology.

The next step is model training and execution. Training is performed using the command `python drebin.py -train`, which initiates the learning process using the previously labeled Drebin-215 dataset. Once trained, features of new APKs are extracted with the command `python drebin.py -extract`. Finally, predictions are made using `python drebin.py -predict`, which processes the extracted binary vector and classifies the APK as either benign or malicious.

6.2. Prediction on Case Study APK: IKO.apk

As a case study, the APK file named `IKO.apk` was analyzed to evaluate the performance of the optimized Random Forest model. The file details are as follows:

- SHA256 hash value: `6d55d90d021b0980528f56d040e78fa7b85a96f5c244e23f330f24c8e80c1cb2`
- File size: 1,916 KB
- Discovered on: May 29, 2025
- Categorized as malware by Malware Bazaar

```
PS C:\Users\anonymous\OneDrive\Desktop\THESIS\JUDUL_BARU\data
[3/3] Mulai prediksi sampel...

Hasil Prediksi Numerik: [1]
Hasil Prediksi Label: ['malware']
```

Figure 6: Feature Extraction Results on IKO.apk

Upon inspection, `IKO.apk` was found to contain 29 out of the 215 predefined static features defined in the Drebin framework. These included dangerous permissions such as:

- `SEND_SMS`, `RECEIVE_SMS`, `WRITE_SMS`

- `READ_PHONE_STATE`, `CAMERA`, `INTERNET`
- `DELETE_PACKAGES`, `CALL_PHONE`
- `WRITE_CONTACTS`, `READ_CONTACTS`

It also registered multiple broadcast intents such as:

- `BOOT_COMPLETED`
- `PACKAGE_REMOVED`
- `PACKAGE_ADDED`
- `SCREEN_ON`, `SCREEN_OFF`
- `BATTERY_LOW`
- `ACTION_POWER_CONNECTED`
- `ACTION_POWER_DISCONNECTED`

Additional suspicious indicators the presence of:

- `BIND_DEVICE_ADMIN`
- `BIND_ACCESSIBILITY_SERVICE`
- `BROADCAST_SMS`
- `BROADCAST_WAP_PUSH`
- `RECEIVE_WAP_PUSH`

```
2025-06-16 19:25:27.434 | INFO | androguard.core.analy
in 00s
APK: 6d55d90d021b0980528f56d040e78fa7b85a96f5c244e23f330f2
Matched features:
- SEND_SMS
- READ_PHONE_STATE
- RECEIVE_SMS
- READ_SMS
- android.intent.action.BOOT_COMPLETED
- android.intent.action.SEND
- WRITE_SMS
- CAMERA
- INTERNET
- android.intent.action.PACKAGE_REMOVED
- WAKE_LOCK
- RECEIVE_BOOT_COMPLETED
- android.intent.action.ACTION_POWER_DISCONNECTED
- android.intent.action.PACKAGE_ADDED
- ACCESS_NETWORK_STATE
- DELETE_PACKAGES
- RECEIVE_WAP_PUSH
- android.intent.action.SENDTO
- BIND_DEVICE_ADMIN
- android.intent.action.SCREEN_ON
- BROADCAST_WAP_PUSH
- BIND_ACCESSIBILITY_SERVICE
- CALL_PHONE
- android.intent.action.SCREEN_OFF
- BROADCAST_SMS
- android.intent.action.BATTERY_LOW
- WRITE_CONTACTS
- android.intent.action.ACTION_POWER_CONNECTED
- READ_CONTACTS
Processed: apk_folder\6d55d90d021b0980528f56d040e78fa7b85a
```

Figure 7: Feature Extraction Results on IKO.apk

After these features were transformed into a binary vector and processed through the trained model, the prediction result for `IKO.apk` was `[1]`, corresponding to the output label `['malware']`. This confirms that the model successfully identified the APK as malicious based on the presence of multiple high-risk static features.

7. Conclusion

The results of this study indicate that optimizing Random Forest hyperparameters using a Genetic Algorithm (GA) significantly improves model performance compared to the default configuration. The GA-tuned model achieved a higher accuracy of 98.97% and an F1-score of 98.59%, surpassing the default model's accuracy of 98.87%. This improvement demonstrates that the evolutionary process in GA effectively identifies more suitable hyperparameter combinations—such as the optimal number of trees and tree depth—which enhances the model's ability to classify Android malware. Additionally, the integration of this optimized model into a Python-based static analyzer system, built upon Drebin-215 features, has proven effective in detecting real APK samples. The combination of static analysis and GA-based optimization not only increases detection accuracy but also ensures practical applicability for identifying malware threats in real-world Android environments.

References

- Aftab, M. et al. (2025). Hyper-parameter tuning through innovative designing to avoid over-fitting in machine learning modelling. *Journal of Statistical Computation and Simulation*.
- Agrawal, P. and Trivedi, B. (2021). Machine learning classifiers for android malware detection. In *Machine Learning Approaches for Convergence of IoT and Blockchain*, pages 343–352. Springer.
- Albadr, M., Tiun, S., Ayob, M., and Al-Dhief, F. T. (2020). Genetic algorithm based on natural selection theory for optimization problems. *Symmetry*, 12(11):1758.
- Arp, D., Spreitzenbarth, M., Hübner, M., Gascon, H., and 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)*.
- Dutt, S., Chandramouli, S., and Das, A. K. (2018). *Machine Learning*. Pearson Education, 1 edition.
- Mahindru, A. and Singh, P. (2017). A study of machine learning algorithms for detection of android malware. *International Journal of Engineering and Computer Science*, 6(6):21894–21897.
- Morán, P., Robles-Gómez, A., Duque, A., Tobarra, L., and Pastor-Vargas, R. (2024). Machine learning models and dimensionality reduction for improving android malware detection. *PeerJ Computer Science*, 10:e2616.
- Rana, M. S., Ahmad, A., Nazir, B., and Khan, A. A. (2020). Evaluation of tree-based machine learning classifiers for android malware detection. In *Advances in Intelligent Systems and Computing*, volume 701, pages 395–404. Springer.
- Siregar, M. U., Effendy, S., and Nurhidayati, N. (2023). Optimized random forest classifier based on genetic algorithm for heart failure prediction. In *Proceedings of Seminar Informatika UIN Sunan Kalijaga*.
- Togatorop, P. R., Purba, R. N., and Junaidi, A. (2022). Optimizing random forest using genetic algorithm for heart disease classification. *LIKMI Journal of Information Technology and Informatics*, 13(1):39–47.
- Wicaksono, A. S. and Supianto, A. A. (2018). Hyperparameter optimization using genetic algorithm on machine learning methods for online news popularity prediction. *International Journal of Advanced Computer Science and Applications (IJACSA)*, 9(12):263–270.
- Widardo, F. and Chowanda, A. (2025). Hyperparameter tuning for deep learning model used in multimodal emotion recognition data. *International Journal of Electrical and Engineering Informatics (IJEI)*, 14(1):115–123.
- Yang, L. and Shami, A. (2022). On hyperparameter optimization of machine learning algorithms: Theory and practice. *arXiv preprint arXiv:2007.15745*.
- Yerima, S. Y. and Sezer, S. (2019). Droidfusion: A novel multilevel classifier fusion approach for android malware detection. *IEEE Transactions on Cybernetics*, 49(2):453–466.