

Android-manifest extraction and labeling method for malware compilation and dataset creation

Djarot Hindarto¹, Arko Djajadi²

¹Faculty of Information and Communication Technology, University of Nasional, Jakarta, Indonesia

²Department of Engineering Physics, Faculty of Engineering and Informatics, Multimedia Nusantara University, Tangerang, Indonesia

Article Info

Article history:

Received Jan 18, 2023

Revised Mar 27, 2023

Accepted Apr 7, 2023

Keywords:

Android application

Artificial neural network

Extract

Machine learning

Malware

ABSTRACT

Malware is a nuisance for smartphone users. The impact is detrimental to smartphone users if the smartphone is infected by malware. Malware identification is not an easy process for ordinary users due to its deeply concealed dangers in application package kit (APK) files available in the Android Play Store. In this paper, the challenges of creating malware datasets are discussed. Long before a malware classification process and model can be built, the need for datasets with representative features for most types of malwares has to be addressed systematically. Only after a quality data set is available can a quality classification model be obtained using machine learning (ML) or deep learning (DL) algorithms. The entire malware classification process is a full pipeline process and sub processes. The authors purposefully focus on the process of building quality malware datasets, not on ML itself, because implementing ML requires another effort after the reliable dataset is fully built. The overall step in creating the malware dataset starts with the extraction of the Android Manifest from the APK file set and ends with the labeling method for all the extracted APK files. The key contribution of this paper is on how to generate datasets systematically from any APK file.

This is an open access article under the [CC BY-SA](https://creativecommons.org/licenses/by-sa/4.0/) license.



Corresponding Author:

Arko Djajadi

Department of Engineering Physics, Faculty of Engineering and Informatics, Multimedia Nusantara University

Scientia Boulevard Road, Gading Serpong, Tangerang, Indonesia

Email: arko@umn.ac.id

1. INTRODUCTION

The growth of the smartphone market over the last two decades has made Android one of the most pervasive operating systems for smartphone devices, as it accounts for more than 80% of the global market. With its popularity, the Android operating system comes at a cost, as it is becoming one of primary targets of attack by cyber crime. Cyber attacks are so prevalent on most internet-connected systems, and various attack models are used. Online applications such as web applications that are not properly protected are prone to such attacks as SQL injection attacks, distribution denial of service, defacing or many other potential dangers. The last 15 years smartphone users have had the advantage of ever faster mobile connections and now nearly all smartphones are always connected to the internet. This is true in the case of ever dominating Android smartphones. It means that Android smartphones in the networks are compromised even more seriously and suffer from even wider cyber attacks or hijacked by fake Android application package kit (APK) files. Even worse is the fact that the majority of Android smartphone users tend to be less aware or not literate with obvious catastrophic danger once their devices are infected. Combined with social engineering attacks, users' mobile bank accounts, emails, phone books and social media apps fall quickly into the hand of cyber predators ready for exploiting the victims. Recent national news of looming attacks being handled by the national cyber security

forces confirm this critical attacks [1]–[3], indicating the eternal need for well-planned efforts to perform penetration testing [4] and malware compilation. Cryptography can be applied to further protect data [5] in case of cyber attacks, and scam or fraudulent links for phishing can possibly be anticipated by federated learning [6].

August 2010 was the first time that the existence of malware had been detected in the android operating system. It did not take long thereafter, that the number of malicious android applications reached more than thousand APKs in the following years. In the third quarter of 2018, according to Google data, the total number of Android malware touched 3.2 million and jumped by 40% year on year [7], [8]. Data from several sources indicate that there are more than 1 billion Android devices at risk due to malware. In addition, two out of five active mobile phones have security risks [9], which we believe that most Android users are not aware of their potential attacks, as the attack is stealth. This risk is exaggerated when users do not update the operating system and APKs to the latest version with the most current security update.

Users today often download applications from anywhere, which often causes problems. When installing an application and allowing whatever the application asks for, without making a conscious selection and knowing the purpose of the application, this is what causes security problems on the smartphone, such as infiltration of certain scripts. That particular script when triggered can perform an action that violates security, for example the initiation of stealthy data transfer of private data to leak out from the smartphones to the attackers. Finally, the control of the smartphone will fall into the hands of attackers.

Application development technology is currently very fast, due to the use of a framework that makes it easy to create Android-based applications. There are several web-based applications that provide solutions to quickly create Android APK files, such as MIT App Inventor [10], [11], Flutter, Appery.io, and many more. There are also those who take advantage of ready-made applications and then carry out the Reverse Engineering process and add several functions to create new applications. The manufacturing process can be done in a short time. The speed of making Android applications is used by irresponsible parties for negative purposes. By adding certain scripts such as allowing to activate storage so that the party entering the script can explore the user's smartphone storage. So that applications that have been infiltrated by malicious scripts or programs are referred to as Android malicious software.

Some survey data regarding Android malware reveal that the ease or speed of making Android applications coupled with reverse engineering APKs are very interesting for security research. Research in malware based on the latest data for February 2021, released by the AV-TEST Institute, says there are over 350,000 malwares every day. This means that it has increased sharply from the last 5 years. Attacks and threats on the internet become a major topic that is often discussed in campus forums or other forums. One of the attacks and threats that are quite trending today is malware on Android-based smartphones. Research on malware is carried out using a classification algorithm to detect whether the file is normal or malware. Many anti-virus and anti-malware sometimes cannot detect new malware variants for several reasons. There are two analyzes namely static analysis and dynamic analysis. The analysis uses a classification of Android malware. With the presence of artificial intelligence technology, malware research can take advantage of this technology, one of which is malware classification using machine learning (ML). It is an ongoing effort that the application of ML technology for malware classification will be accurate.

Malware in smartphones today can be very annoying and disturbing for users [12]. The disturbance causes it to not run normally on the Smartphone. This problem is often solved by installing anti-malware or antivirus for smartphones. But another problem arises, namely the use of anti-malware from untrusted sources, where the anti-malware has been infiltrated with malware. In addition, anti-malware does not run normally, because several new variants keep appearing, and many anti-malwares do not detect new variants. So anti-malware remains a problem. Anti-malware producing companies also continue to innovate to detect malware. Even so, it still cannot detect the latest malware variants that keep popping up.

Many works have been produced in previous research, where the writings discuss various works discussing malware research with static, dynamic and hybrid methods or those that combine static and dynamic. One of the results is forensic analysis of mobile devices using scoring (FAMOUS) of application permissions [13], which proposes a predictive approach to forensics in detecting suspicious Android APKs. The next study is to detect Android APK malware and benign by weighting the prediction-based feature set using ML. Various experiments were carried out on the features of the Android APK properties with an accuracy rate of 99%. FAMOUS extracts Android APK files using the permissions feature only in classifying and analyzing each Android feature using the AndroidManifest.xml file.

Next work is longitudinal performance analysis (LPA) of ML based Android malware detectors [14]. The aim of the study was to examine the performance degradation over time for various classifiers with ML, which were trained with static features extracted from a collection of applications and date-labeled malware. It is a static analysis with quantitative methods, namely by collecting malware dataset, Application features extraction and noting ML classification for performance evaluation. The investigation is repeated by training with time periods and samples from the latest datasets. The review of this work is that the method chosen is

the static method with datasets for 2013, 2014 and 2015-2016. Also, the ML algorithms used are support vector machine (SVM), J48 decision tree (DT), naive Bayes (NB), simple logistic (SL), and random forest (RF).

Decompiled APK based malicious code classification [15]. The purpose of this study is to adapt the decompile source code APK technique based on natural language processing for the classification of source code malware. Using static analysis with quantitative methods it proceeds as follows using Rocky framework: Decompiling APK files into source code, preprocessing of source code, generalizing N-tokens, feature representations, and classification. Algorithm baselines are permissions, API calls and neural network (NN) based. Android malware dataset (AMD) from Argus Lab contains 24,553 sample APKs, grouped into 135 types, 71 malware families from sampling year in 2010 – 2017. Evaluation of metrics are confusion matrix, true positive ratio (TPR), F1, accuracy, receiver operating characteristic (ROC) and area under the ROC curve (AUC) [16], [17]. Classification is done with NB, RF, logistic regression, with 10 validation tests. Test results from decompiled APK based malicious code classification research reached 97% accuracy.

Next is APK auditor-a permission-based Android malware detection system [18]. In APK auditor permissions feature, the system performs a malware assessment with three main components: Android client, signature database, central server communicating with both. Presents Android's permission-based malware detection system [19]–[22] using static analysis in classifying benign and malware Android applications. In conducting the experiment using data as many as 8,762 APKs, consisting of 1,853 benign APKs and 6,909 APK malware. The results of the accuracy of the model produce 88%.

As can be summarized from the reviewed studies above, there are no standardized ways yet to address the key problem of malware identifications. The research questions are focused on the current state of the art in data engineering for dataset creation and in the labeling approach to support ML algorithms for Android malware study. The potential ability of the ML methods to detect, analyze and predict new variants of malware that are currently widespread looks promising. Therefore, both dataset creation and detection algorithms are two key enablers for solving the problem. Both are interesting and logical to be appointed as research questions (RQ) as follows: i) How does feature extraction of APK files produce the best malware dataset? (RQ 1) and ii) How does feature selection from the dataset help optimize the resulting detection model? (RQ2)

The scope of this current research is as follows. The method used in analyzing malware is a static method. Data collection (public data) for the dataset will be used as material for making models that will be tested. Data preparation, analyzed datasets, which attributes are used, which attributes are most influential in the malware are considered very important in modeling and on the performance of ML algorithms. Dataset creation is a data engineering process that largely relies on the feature extraction method and feature selection method required, before data consuming ML algorithms can start.

2. THE PROPOSED METHOD (EXTRACTING APK ANDROID FILES)

Figure 1, is a big picture of the system that will extract the Android APK file into a malware dataset, then the analysis process will be carried out on the features contained in the APK file. The resulting dataset is the result of reverse engineering using the Jadx module [23], [24], which is a tool from Reverse Engineering. Reverse Engineering is the process of converting APK files into source code form. This source code will be carried out. Further analysis, whether the file is malware or benign.

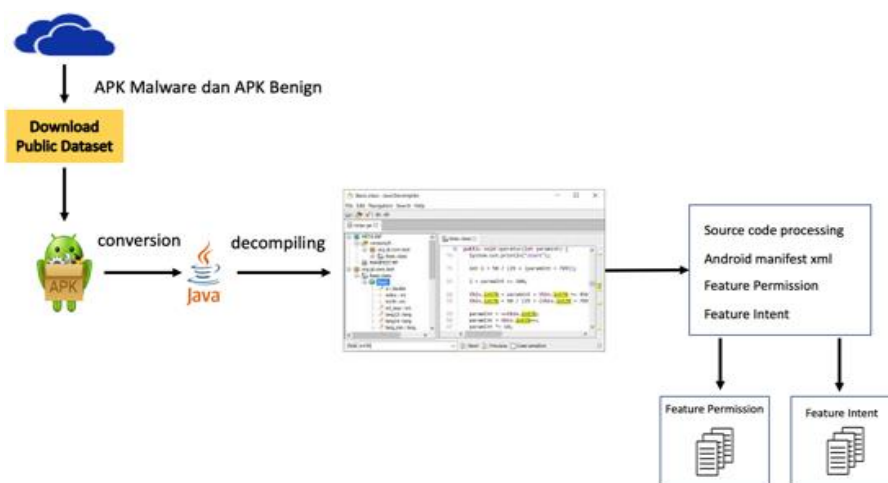


Figure 1. Steps in extracting APK Android files

3. METHOD

3.1. Download APK File

The proposed research framework is as follows: dataset collection in the form of an application package kit (APK) containing malware APK and benign APK. Source dataset from University of New Brunswick [25], Virusshare [26], VirusTotal [27]. After collecting the dataset in the form of APK malware files and APK benign files, the data extraction process is carried out. APK files are converted and decompiled to get feature permission and feature intent. Feature permission and feature intent will be processed into a dataset. This process is called feature extraction. Feature selection is done to reduce the features or dimensions of the malware dataset. Feature permission and feature intent are modeled by dividing training data and test data.

The APK files in the training come from the Google Play store and the Canadian Institute (UNB), consisting of five classes. The files are Benign APK, Ransomware APK, Riskware APK, Banking APK, and short message service (SMS) APK. The downloaded data is stored in their respective folders according to their class. To make sure the files are malware or not malware, check them. Checking through virustotal.com Website. The website is able to detect the types of malwares. Because the virustotal.com [27] website is supported by security companies such as Avast, Norton and others. In carrying out this experiment, download files of around 14,170 APKs measuring 60 GB from various sources above. Android APK files are placed in a folder according to the type of Android APK, such as Benign will be placed in the benign folder, while banking will be placed in banking. Figure 2 shows the overall process in a pseudocode. The collection of Android APK files will later be extracted based on the type or family of malware. The results of the extraction using a reverse engineer, which will be processed is AndroidManifest.xml. The feature set that is processed is the permission feature and the intent feature, both of which are the basis of the Android malware classification. It is hoped that the extraction process will produce the best malware dataset.

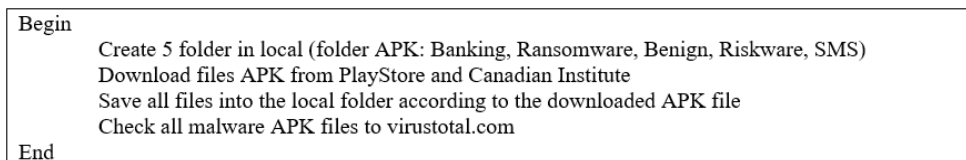


Figure 2. Pseudocode for creating folders and storing APK malware

After downloading the malware APK and Benign APK, the next step is to check before carrying out the extraction process. The purpose of checking the APK file is whether the malware file is malware, not benign APK. Likewise, with the benign Android APK, whether the file is a benign APK not a malware APK. In addition, the file can be categorized so that it is not wrong in determining the class in the classification. Later it will produce a dataset that is not mixed between the malware APK and the benign APK.

3.2. Checking APK file

Virustotal.com [27] is a reference for checking files that contain malware. Not only APK files, but any files can be checked on the website. Actually, in our society, the term harmful file is called a virus. The virus is part of malicious software or malware. There are many families of malware [28], [29], there are viruses, trojans, ransomware, banking, SMS, and others. There are those that infect the computer and there are also those that infect the smartphone. Making malware has different goals, depending on the malware maker. There is some malware that does not damage the operating system, but has an effect on data theft [30], spying on smartphone users [31].

Some smartphone users do not realize that they are being spied on by malware, by sending smartphone user data to the malware maker's servers [32]. This is very detrimental for smartphone users. Therefore, before downloading the Android APK file, the APK file should not be installed directly, but must be checked on the APK file to virustotal.com or other APK file checking websites.

3.3. Extraction APK file

This stage is reading the APK Files in the folder then doing a reverse engineer to read the AndroidManifest.xml File. Feature permission, reverse engineer process; Extract the APK file and save it to the unpacked-permissions folder. The next step that is read is the AndroidManifest.xml file, the XML file parser is carried out to read the value from "uses-permission" then saved into the UpdatePermList.txt file. feature intent, reverse engineering process; Extract the APK file and save it to the unpacked-intent folder. The process is continued by reading the AndroidManifest.xml file, followed by the XML file parser process to read the feature intent value from the contents of the AndroidManifest.xml file. The contents of the file,

“*application/activity/intent-filter/action*”, “*application/receiver/intent-filter/action*”, “*application/activity/intent-filter/category*”, if any, will be assigned a value of 1 and 0 if not found. The process is continued by saving the results of the process into the UpdateIntentList.txt file. APK file, is an executable format in the Android operating system. To find out the APK file structure, you can use the Jadx module or Jadx-graphical user interface (GUI).

Fitur permission [33]; The APK file [34] is reverse engineered to produce the AndroidManifest.xml file. Then the XML parser is done by checking the ‘uses-permission’ feature, if there is a uses-permission, then it is given a value of 1 and if it is not there is given a value of 0. Fitur Intent; The APK file is reverse engineered to produce the AndroidManifest.xml file. Followed by parser by checking “*activity/intent-filter/action*,” “*receiver/intent-filter/action*,” “*activity/intent-filter/category*”. If there is a checking condition such as a checking condition, it will be given a value of 1 and if it is not there, it will be given a value of 0. The intent feature is the most basic feature in Android, which is used to process data from other components [35]. Reverse engineering for Android APK files [36], using the Jadx module. This Jadx (APK-tools) module is a module that can extract Android APK files by creating folders and extracting them into source code, resources and assets. The resources folder contains the AndroidManifest.xml file. Figure 3 shows the pseudocode for performing the APK feature extraction.

```

Begin
  do
    Read file APK(folder)
    Reverse file APK with Jadx module
    Save file AndroidManifest.xml in folder UnpackedPermission
    Read file AndroidManifest.xml in folder UnpackedPermission
    ParseXML in permission root.findall("user-permission")
    if feature_permission=1
      set value_permission=1
    else
      set value_permission=0
    while until eof ()
      save update_permission.txt
      save datapermission.csv
  End
  Procedure Intent_Extraction(folder)
  Begin
    do
      Read file APK(folder)
      Reverse file APK with Jadx module
      Save file AndroidManifest.xml in folder UnpackedIntent
      Read file AndroidManifest.xml in folder UnpackedIntent
      ParseXML in permission root.findall("intent-filter/action")
      if intent-filter-action=1
        set value-filter-action=1
      else
        set value-filter-action=0
      endif
      if intent-filter-receiver=1
        set value-filter-receiver=1
      else
        set value-filter-receiver=0
      endif
      if intent-filter-category=1
        set value-filter-category=1
      else
        set value- filter-category=0
      endif
      while until eof ()
        save update_intent.txt
        save dataintent.csv
    End
  Procedure merge_dataset
  Begin
    merged_dataset=[]
    for row in dataset_permission:
      merged dataset append (row)
    for row in dataset_intent:
      merged_daataset append (row)
    return merged_dataset
  End

```

Figure 3. Pseudocode for performing the APK feature extraction

4. RESULTS AND DISCUSSION

In conducting an experiment for Android APK extraction, using a Macintosh 2020 Notebook, with 8 GB RAM, 256 GB hard drive. Python programming language, NumPy and pandas library, xml. NumPy and pandas are the main library packages for computational mathematics and data science. The time it takes to extract Android APK files is 7 days non-stop. Process 14,170 Android APK, 1,179 feature, malware and Benign. This process generates two datasets namely dataparmission.cvs and dataintent.csv. The two dataset files are merged, resulting in datamalware.csv.

4.1. Extract APK

Table 1 is the result of the extraction of the Android APK file. Where the process has been described above, explaining about reverse engineering using the JADX module [37] and parsing each Android APK file. Table 1, the score of permission features that often appear are *SEND_SMS* (Developer), *ACCESS_COARSE_LOCATION* (Developer), *SYSTEM_ALERT_WINDOW*, *READ_PHONE_STATE*, *RECEIVE_SMS*, *RECEIVE_BOOT_COMPLETED*, *GET_TASKS*, *READ_SMS*, *ACCESS_WIFI_STATE*, *WRITE_EXTERNAL_STORAGE*. The following is a description of the features.

4.2. Dataset

The result of this research is a dataset that can contribute to the detection of malware and non-malware. So that other researchers can directly use the dataset from the extraction process on various original malware on the internet. The extraction process was carried out according to the algorithm described in Figure 3. Table 1 shown the sample result of a malware dataset.

Table 1. The result of the dataset creation process

| Name File | Android permission access | | Android intent access | |
|--------------------------------------|---------------------------|-----------------|-----------------------|------------------|
| | Downloads | Bluetooth_Share | Package_Removed | Package_Replaced |
| ffa01b3ce624d6efc8028b3c2dfa17a4.apk | 0 | 0 | 0 | 1 |
| fe73930d1a24fb7d81693471c3677f8f.apk | 1 | 0 | 0 | 1 |
| f9e6378ebfbd69e77c451e32cf2af90c.apk | 0 | 0 | 0 | 1 |
| fa8ac1e84089e249e2e4a52cc588b810.apk | 0 | 1 | 0 | 1 |
| f83a2cc8303ea8af2c8f55c059564485.apk | 0 | 1 | 0 | 0 |
| f7296fa9243869375577e7770ca148f9.apk | 1 | 1 | 0 | 0 |
| f7013204327c182fb0bc2b9a45adc734.apk | 1 | 1 | 0 | 0 |
| f6c4919d0f465cc4e0d346c285ccd297.apk | 1 | 1 | 0 | 1 |
| f6515bfa39b1754867b6fd66c9cdc864.apk | 0 | 0 | 1 | 1 |
| fd6d6f5467370d6d3921192d8e7f6466.apk | 0 | 0 | 1 | 0 |
| fd4a86dfa65eb92eb8780e9cebdb822e.apk | 0 | 0 | 1 | 0 |
| f9c00e18740daf9a5dc3bd7dff25f14e.apk | 0 | 0 | 1 | 0 |
| f5e3158eba53be270164494bb4c830a2.apk | 0 | 0 | 1 | 1 |
| fe16e96706eb4a2b9132589a4f9fe582.apk | 0 | 1 | 0 | 0 |

Explanation of the final result of the extraction process from the android file (.apk) using the reverse engineering process, extract the android manifest file, selecting the permission keywords and intent keywords in the android manifest file. The result of this process becomes a dataset file for Android malware classification or detection. Table 1 explains the features in the resulting dataset. The explanation for the left column is *NAME*, *NAME* is the name of the Android APK file. The *android_permission_ACCESS_All* column is a request to access all functions on a smartphone device. If an Android APK application executes certain instructions and requests access to all functions on a smartphone device, then an irregularity appears. A column or feature with a value of 1 indicates the function of the permissions feature requesting access rights to run on smartphone device functions. Column or feature if it has a value of 0 indicates the function of the permissions feature does not ask for access rights to run on smartphone device functions.

4.3. Discussion

APK files are the package files used to share and install apps on Android devices. To extract the source code from an APK file, you can use a tool called Jadx. Jadx is a reverse engineering tool that can decompile an APK file into Java source code. It can also convert DEX bytecode to Java source code and provides an option to view the code in a graphical format. To use Jadx, you can download the tool from the official website and then open the APK file you want to extract using the Jadx GUI. Once the file is open, you can navigate through the package hierarchy and view the source code for each package, class, and method. Additionally, you can also export the source code as a zip or a jar file. Reverse engineering the Jadx module to create a dataset involves using the Jadx tool to decompile multiple APK files and then organizing the resulting source code into a structured dataset. This dataset can then be used for various purposes such as code analysis, malware

detection, and more. One way to create a dataset using Jadx is to first gather a set of APK files that you want to decompile. These files can be obtained from various sources such as the Google Play Store or from other sources like GitHub. Once you have a set of APK files, you can use Jadx to decompile each file and extract the source code. Next, organize the extracted source code into a structured dataset. This can be done by creating a new directory for each APK file and placing the decompiled source code into the corresponding directory. Alternatively, you can also organize the source code into a spreadsheet (format file csv), with each row representing an APK file and its corresponding source code.

The methodological or framework approach in this research is reverse engineering and extraction of the AndroidManifest file. Using the JADX module reverse engineering method, to perform reverse engineering. The process is that the android file (example file-android.apk) is reversed to become the source code files collected in the folder. A collection of source-code files contains the AndroidManifest file. This file is extracted and selects the permission keywords and intent keywords. If the permission keyword is enabled, it will be written into the dataset with a value of 1, if the permission keyword is disabled, then the dataset permission feature is 0. The algorithm has been explained in the APK file extraction section, complete with pseudocode *Feature_Extraction*.

The design of the system used is as follows: So far in detecting malware APK, the difficulty encountered is getting the malware dataset if it detects malware APK with a static method. The first difficulty is getting the dataset in the form of an android virus (Apk file). To get the original virus file, download it at University of New Brunswick [25], Virusshare [26], VirusTotal [27]. After getting the Android virus, the second step is to do reverse engineering using the JADX module tools. The third step is to read the android manifest file in the reverse engineering results (android manifest file) and select the permission and intent keywords. The fourth step saves the results of the third step (selection of permission keywords and intent keywords in the android manifest file) into the dataset.csv file. By carrying out four stages, a dataset is obtained. The design explanation is in Figure 1 steps in extracting APK Android files.

Evaluate the performance of the dataset if the simulation is carried out using the grid search cross validation, 5 folds cross and the multi-layer perceptron (MLP) Classifier. For the code from GridSearchCV as given in Figure 4 whose results are as shown in the next Figure 5. Figure 5 shows the simulation results of the malware.csv dataset using artificial neural network (ANN) MLP. The dataset resulting from the reverse engineering process and the extraction of the AndroidManifest file produces a better performance accuracy model, reaching 100%. The number of datasets processed is 14,170 malwares. Simulations using ML algorithms are also carried out using DT, SVM and KNN algorithms. So that the simulation using the ML algorithm will be compared. The simulation results of ML algorithms are compared with NNs such as ANNs.

```
GridSearchCV (mlpc, mlpc_params, cv=5, n_jobs=-1, verbose=2)
  mlpc_params={'alpha': [0.1, 0.01, 0.0001], "hidden_layer_sizes": [(10,10,10),
    (100,100,100), (100,100)], "solver": ["lbfgs", "adam", "sgd"], "activation": ["relu", "logistic"]}
  mlpc_cv_model.fit (X_train, y_train)
  mlpc_tuned=mlpc_cv_model.best_estimator_
  mlpc_tuned.fit (X_train, y_train)
For the value of K-fold F1_weighted as follows:
from sklearn. model_selection import Kfold
from sklearn. model_selection import cross_val_score
kf=Kfold (shuffle=True, n_splits=5)
cv_results_kfold=cross_val_score (mlpc_tuned, X_test, np.argmax (y_test, axis=1), cv=kf, scoring= 'f1_weighted')
print ("K-fold Cross Validation f1_weighted Results: ", cv_results_kfold)
print ("K-fold Cross Validation f1_weighted Results Mean: ", cv_results_kfold.mean ())
The result is as follows:
K-fold Cross Validation f1_weighted Results: [0.99823636 1. 1. 1. 1.]
K-fold Cross Validation f1_weighted Results Mean: 0.9996472717814008
K-fold accuracy
from sklearn. model_selection import Kfold
from sklearn. model_selection import cross_val_score
kf=Kfold (shuffle=True, n_splits=5) # To make a 5-fold CV
cv_results_kfold=cross_val_score (mlpc_tuned, X_test, np.argmax (y_test, axis=1), cv=kf, scoring= 'accuracy')
print ("K-fold Cross Validation accuracy Results: ", cv_results_kfold)
print ("K-fold Cross Validation accuracy Results Mean: ", cv_results_kfold.mean())
The result is as follows:
K-fold Cross Validation accuracy Results: [1. 1. 0.99823633 1. 0.99646643]
K-fold Cross Validation accuracy Results Mean: 0.9989405525330142
```

Figure 4. Source code GridSearchCV and K-fold accuracy

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0 | 1.00 | 1.00 | 1.00 | 576 |
| 1 | 1.00 | 1.00 | 1.00 | 543 |
| 2 | 1.00 | 1.00 | 1.00 | 529 |
| 3 | 1.00 | 1.00 | 1.00 | 573 |
| 4 | 1.00 | 1.00 | 1.00 | 613 |
| accuracy | | | 1.00 | 2834 |
| macro avg | 1.00 | 1.00 | 1.00 | 2834 |
| weighted avg | 1.00 | 1.00 | 1.00 | 2834 |

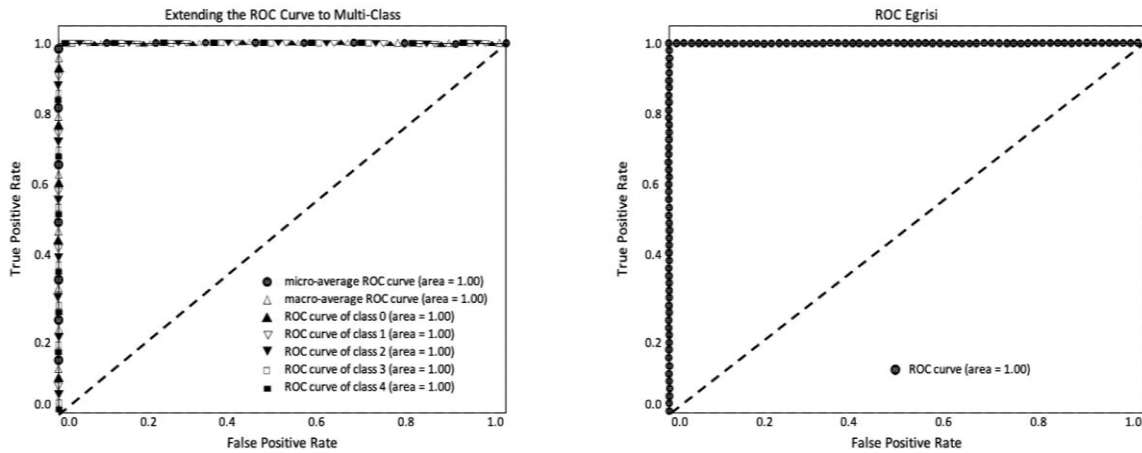


Figure 5. Performance simulation using ANN MLP

Table 2, shows a comparison of simulations using several algorithms such as DT, SVM, KNN+PCA, ANN+GridSearch+MLP. The result is that the DT algorithm produces a very good Precision, Recall, F1-Score of 100%, when using a relatively small dataset of 600 malwares. If the 7,000 dataset and 14,170 malware datasets experience a decrease in precision, Recall and F1-score. The same thing happened to SVM and KNN+PCA. DT, SVM and KNN are included in ML. In this simulation, the ML algorithm will decrease if the number of datasets increases. The difference occurs in the ANN, if the dataset is larger in number, the Precision, Recall and F1-score will increase. The average yield reaches 100%. The contribution of this study provides an alternative dataset that will be used for further research. The next work is how to use the dataset from the extracted APK file into ML and deep learning (DL) algorithms. The reason for using an ANN with GridSearchCV and MLP is because the dataset is large. For the use of ML methods such as SVM, DT does not have the maximum precision, recall and F1-score. The dataset is simulated using an ANN with GridSearchCV and MLP, resulting in 100% performance.

Table 2. Comparison of DT, SVM, KKN+PCA, ANN+GridSearchCV+MLP

| Method | Dataset size | Precision | Recall | F1-score | Support |
|----------------------|--------------|-----------|--------|----------|---------|
| Decision tree | 600 * | 1.00 | 1.00 | 1.00 | 58 |
| | 7000 | 0.89 | 0.89 | 0.89 | 1401 |
| | 14170 | 0.92 | 0.91 | 0.91 | 2834 |
| SVM | 600 | 0.99 | 0.99 | 0.99 | 58 |
| | 7000 | 0.90 | 0.89 | 0.89 | 1401 |
| | 14170 | 0.91 | 0.91 | 0.90 | 2834 |
| KNN+PCA | 600 | 0.85 | 0.84 | 0.84 | 1401 |
| | 7000 | 0.85 | 0.85 | 0.85 | 2834 |
| | 14170 | 0.88 | 0.88 | 0.88 | 2834 |
| ANN+GridSearchCV+MLP | 600 | 0.99 | 0.99 | 0.99 | 116 |
| | 7000 * | 1.00 | 1.00 | 1.00 | 1401 |
| | 14170 * | 1.00 | 1.00 | 1.00 | 2834 |

5. CONCLUSION

The authors purposefully focus on the process of building quality malware datasets as it is seen as the most demanding approach and implementation, and not on machine learning itself, because implementing machine learning requires another effort only doable after the reliable dataset is fully built. The overall steps in creating the malware dataset have been extensively described systematically, starting with the collection,

reverse engineering, followed by extraction of the Android Manifest from the APK file set, and ending with the labeling method for all the extracted APK files. The core contribution of this paper is on how to generate datasets systematically from any APK file. The conclusion of this study is very insightful and useful for researchers working in the various fields of ML. The constructed dataset can be directly used for various purposes, especially for supervised classification and malware identification.




REFERENCES

- [1] "Avoid fraud with wedding invitation mode APK," *Republika.co.id*, 2023. <https://kampus.republika.co.id/posts/200272/tips-menghindari-penipuan-dengan-apk-modus-undangan-pernikahan> (accessed Feb. 03, 2023).
- [2] "Alert! The latest online fraud mode through APK files," *Departemen Komunikasi Bank Indonesia*, 2023. <https://www.bi.go.id/id/publikasi/ruang-media/cerita-bi/Pages/Waspada!-Modus-Penipuan-Online-Terbaru-lewat-File-APK.aspx> (accessed Feb. 01, 2023).
- [3] "Many fraud using APK files, understand how it works and tips to avoid it," *Kompas.com*, 2023. <https://money.kompas.com/read/2023/02/05/053000226/ramai-penipuan-bermodus-file-apk-pahami-cara-kerja-dan-tips-menghindarinya?page=all> (accessed Feb. 05, 2023).
- [4] A. Djajadi and N. Sutisna, "Penetration testing: Dumping data from web application using SQL injection attack (case study: eArsip)," *InterNetworking Indonesia Journal*, vol. 13, no. 1, pp. 3–9, 2021.
- [5] M. Chen and D. Kusuma Halim, "Federated learning for scam classification in small Indonesian language dataset: an initial study," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 30, no. 1, pp. 325–331, Apr. 2023, doi: 10.11591/ijeecs.v30.i1.pp325-331.
- [6] A. A. Permana and L. A. Pratiwi, "Implementation of the advanced encryption standard (AES) algorithm for digital image security," *Jurnal Teknik Informatika*, vol. 15, no. 1, pp. 44–51, Jun. 2022, doi: 10.15408/jti.v15i1.25735.
- [7] F. Di Cerbo, A. Girardello, F. Michahelles, and S. Voronkova, "Detection of malicious applications on Android OS," in *IWCF 2010: Computational Forensics*, 2011, pp. 138–149.
- [8] M. A. Omer *et al.*, "Efficiency of Malware detection in Android system: A survey," *Asian Journal of Research in Computer Science*, pp. 59–69, Apr. 2021, doi: 10.9734/ajrcos/2021/v7i430189.
- [9] X. Ge, Y. Pan, Y. Fan, and C. Fang, "AMDroid: Android Malware detection using function call graphs," in *2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, Jul. 2019, pp. 71–77, doi: 10.1109/QRS-C.2019.00027.
- [10] Z. R. Mohsin, A. M. Dayish, and B. A. Hamdan, "Android projects on Android attack application," *Journal of Xidian University*, vol. 14, no. 5, pp. 16–21, May 2020, doi: 10.37896/jxu14.5/233.
- [11] W. Stallings, *Cryptography and network security: Principles and practice*, Seventh Ed. Harlow: Pearson Education Limited, 2017.
- [12] S. Peng, S. Yu, and A. Yang, "Smartphone Malware and its propagation modeling: A survey," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 2, pp. 925–941, 2014, doi: 10.1109/SURV.2013.070813.00214.
- [13] A. Kumar, K. S. Kuppusamy, and G. Aghila, "FAMOUS: Forensic analysis of mobile devices using scoring of application permissions," *Future Generation Computer Systems*, vol. 83, pp. 158–172, Jun. 2018, doi: 10.1016/j.future.2018.02.001.
- [14] S. Y. Yerima and S. Khan, "Longitudinal performance analysis of machine learning based Android malware detectors," in *2019 International Conference on Cyber Security and Protection of Digital Services (Cyber Security)*, Jun. 2019, pp. 1–8, doi: 10.1109/CyberSecPODS.2019.8885384.
- [15] R. Mateless, D. Rejabek, O. Margalit, and R. Moskovitch, "Decompiled APK based malicious code classification," *Future Generation Computer Systems*, vol. 110, pp. 135–147, Sep. 2020, doi: 10.1016/j.future.2020.03.052.
- [16] K. Gajowniczek and T. Ząbkowski, "ImbTreeAUC: An R package for building classification trees using the area under the ROC curve (AUC) on imbalanced datasets," *SoftwareX*, vol. 15, Jul. 2021, doi: 10.1016/j.softx.2021.100755.
- [17] A. R. Rachakonda and A. Bhatnagar, "A: Extending area under the ROC curve for probabilistic labels," *Pattern Recognition Letters*, vol. 150, pp. 265–271, Oct. 2021, doi: 10.1016/j.patrec.2021.06.023.
- [18] K. A. Talha, D. I. Alper, and C. Aydin, "APK Auditor: Permission-based Android malware detection system," *Digital Investigation*, vol. 13, pp. 1–14, Jun. 2015, doi: 10.1016/j.diin.2015.01.001.
- [19] S. K. Smmarwar, G. P. Gupta, S. Kumar, and P. Kumar, "An optimized and efficient android malware detection framework for future sustainable computing," *Sustainable Energy Technologies and Assessments*, vol. 54, Dec. 2022, doi: 10.1016/j.seta.2022.102852.
- [20] A. Mathur, L. M. Podila, K. Kulkarni, Q. Niyaz, and A. Y. Javaid, "NATICUSdroid: A Malware detection framework for Android using native and custom permissions," *Journal of Information Security and Applications*, vol. 58, May 2021, doi: 10.1016/j.jisa.2020.102696.
- [21] A. S. Shatnawi, Q. Yassen, and A. Yateem, "An Android Malware detection approach based on static feature analysis using machine learning algorithms," *Procedia Computer Science*, vol. 201, pp. 653–658, 2022, doi: 10.1016/j.procs.2022.03.086.
- [22] P. Bhat and K. Dutta, "A multi-tiered feature selection model for android malware detection based on feature discrimination and information gain," *Journal of King Saud University - Computer and Information Sciences*, vol. 34, no. 10, pp. 9464–9477, Nov. 2022, doi: 10.1016/j.jksuci.2021.11.004.
- [23] H. Ali *et al.*, "Security hardened and privacy preserved Android Malware detection using fuzzy hash of reverse engineered source code," *Security and Communication Networks*, vol. 2022, pp. 1–11, Sep. 2022, doi: 10.1155/2022/7972230.
- [24] P. Agrawal and B. Trivedi, "Unstructured data collection from APK files for Malware detection," *International Journal of Computer Applications*, vol. 176, no. 28, pp. 42–45, Jun. 2020, doi: 10.5120/ijca2020920308.
- [25] UNB, "Android adware and general Malware Dataset (CIC-AAGM2017)," *University of New Brunswick (UNB)*, 2017. <https://www.unb.ca/cic/datasets/andmal2017.html> (accessed: Jan. 12, 2023).
- [26] J.-M. Roberts and Melissa, "Virusshare: Report for a sample recently added to the system," *VirusShare.com*, Accessed: Jan. 20, 2023. [Online]. Available: <https://virusshare.com/>
- [27] VirusTotal, "VirusTotal: Dataset malware," VirusTotal. Accessed Jan. 05, 2023. [Online]. Available: <https://www.virustotal.com/gui/home/upload>
- [28] S. Turker and A. B. Can, "AndMFC: Android Malware family classification framework," in *2019 IEEE 30th International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC Workshops)*, Sep. 2019, pp. 1–6, doi: 10.1109/PIMRCW.2019.8880840.




- [29] A. Walenstein and M. Venable, "Exploiting similarity between variants to defeat malware," in *Proc. BlackHat Briefings DC*, 2007, pp. 1–12.
- [30] A. Das and H. U. Khan, "Security behaviors of smartphone users," *Information & Computer Security*, vol. 24, no. 1, pp. 116–134, Mar. 2016, doi: 10.1108/ICS-04-2015-0018.
- [31] D. Perakovic, S. Husnjak, and V. Remenar, "Research of security threats in the use of modern terminal devices," in *Conference: Annals of DAAAM for 2012 & Proceedings of the 23rd International DAAAM Symposium*, 2012, pp. 545–548.
- [32] A.-D. Schmidt, F. Peters, F. Lamour, C. Scheel, S. A. Çamtepe, and Ş. Albayrak, "Monitoring Smartphones for anomaly detection," *Mobile Networks and Applications*, vol. 14, no. 1, pp. 92–106, Feb. 2009, doi: 10.1007/s11036-008-0113-x.
- [33] F. Tchakounte and P. Dayang, "System calls analysis of Malwares on Android," *Maejo International Journal of Science and Technology*, vol. 2, no. 9, pp. 669–674, 2013.
- [34] B. Gruver, "Smali/Baksmali," *GitHub.com*. Accessed: Jan. 12, 2023. [Online]. Available: <https://github.com/JesusFreke/smali>
- [35] S. Wu, P. Wang, X. Li, and Y. Zhang, "Effective detection of android malware based on the usage of data flow APIs and machine learning," *Information and Software Technology*, vol. 75, pp. 17–25, Jul. 2016, doi: 10.1016/j.infsof.2016.03.004.
- [36] T. Liu, "Software vulnerability mining techniques based on data fusion and reverse engineering," *Wireless Communications and Mobile Computing*, vol. 2022, pp. 1–6, Apr. 2022, doi: 10.1155/2022/4329034.
- [37] M. T. Kyaw, Y. N. Soe, and N. S. M. Kham, "Security analysis of Android application by using reverse engineering," in *Proceedings of 2019 the 9th International Workshop on Computer Science and Engineering*, 2019, pp. 171–177, doi: 10.18178/wcse.2019.03.029.

BIOGRAPHIES OF AUTHORS



Djarot Hindarto    received the B.Eng. degree in computer engineering from Sepuluh Nopember Institute of Technology (ITS), Indonesia, in 1994 and the Master of Information Technology Pradita University, in 2022, respectively. Currently, he is a lecture at the Faculty of Communication and Information Technology (FKTI), Universitas Nasional (UNAS) Jakarta, Indonesia. His research interests include security, artificial intelligence, deep learning, machine learning, internet of things and blockchain. He can be contacted at email: djarot.hindarto@civitas.unas.ac.id.



Arko Djajadi    received his Bachelor and Master degrees from the Delft University of Technology – Netherlands in 1992 and his Ph.D. from the University of Manchester – UK in 1999, all of them are in Electrical and Electronics Engineering. His research interests include smart embedded systems, mechatronics, instrumentation, Big Data and IoT, EV and renewable energy both in academic and industrial settings. Currently, he is with the Faculty of Engineering and Informatics at the Multimedia Nusantara University (UMN). He can be contacted at email: arko@umn.ac.id.