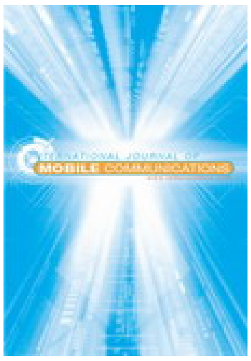


International Journal of Mobile Communications

ISSN online: 1741-5217 - ISSN print: 1470-949X
https://www.inderscience.com/ijmc

HyDroid: android malware detection using network flow combined with permissions and intent filter

Akram Zine Eddine Boukhamla, Abhishek Verma

DOI: 10.1504/IJMC.2023.10040480

Article History:

Received:	09 April 2021
Accepted:	11 July 2021
Published online:	04 July 2023

HyDroid: android malware detection using network flow combined with permissions and intent filter

Akram Zine Eddine Boukhamla*

LINATI Laboratory,
Department of Computer Science and Information Technologies,
Kasdi Merbah University,
Ouargla BP.511,30000, Algeria
Email: boukhamla.akram@univ-ouargla.dz
*Corresponding author

Abhishek Verma

Department of Computer Science and Engineering,
School of Engineering,
BBD University, Lucknow,
Uttar Pradesh, India
Email: abhishekverma@ieee.org

Abstract: Android has become one of the most widely used operating systems for mobile platforms in the recent years. With its widespread adoption, it has also became the target of malicious applications' developers and cyber threats. This in turn has stimulated research on android malware analysis and detection. Several android malware detection techniques have been proposed in the literature. In this paper, we propose a novel hybrid android malware detection method which is named as HydDroid. A hybrid dataset based on the existing CICInvesAndMal2019 dataset by selecting most relevant static features is created. HydDroid is represented by the form of a combination of binary vectors and numerical vectors. The proposed approach is evaluated using three well-known machine learning classification algorithms. The experiment results indicate that HydDroid achieves the accuracy of up to 96.3%. To show the effectiveness of our proposed approach, the performance results are compared with existing solutions.

Keywords: Android malware detection; static analysis; network flow; hybrid analysis; machine learning.

Reference to this paper should be made as follows: Boukhamla, A.Z.E. and Verma, A. (2023) 'HyDroid: android malware detection using network flow combined with permissions and intent filter', *Int. J. Mobile Communications*, Vol. 22, No. 1, pp.70–91.

Biographical notes: Akram Zine Eddine Boukhamla is an Associate Professor at the Department of Computer Science, University of Kasdi Merbah Ouargla/Algeria. He received PhD in Image Processing and Artificial Vision from Badji Mokhtar University, Annaba, Algeria in (2016). He worked as a reviewer for many academic journals Springer, Elsevier, and Wiley. His research focuses on cyber security, security analysis, internet traffic analysis and the detection of malware and attacks.

Abhishek Verma is an Assistant Professor in the Department of Computer Science & Engineering at BBD University, Lucknow, India. He received PhD (2020) in The Internet of Things Security from the National Institute of Technology Kurukshetra, Haryana, India. He completed his BTech in Computer Science and Engineering from Uttar Pradesh Technical University, India in (2014), and MTech in Computer Engineering from the National Institute of Technology Kurukshetra, India in (2016). He has more than five years of experience in research and teaching. He has published more than 15 research articles in international journals and conferences of high repute. He is an editorial board member of Research Reports on Computer Science (RRCS) and active review board member of various reputed journals, including IEEE, Springer, Wiley, and Elsevier. His current areas of interest include information security, intrusion detection, and the internet of things.

1 Introduction

In our daily lives, mobile applications become ubiquitous (Eveleth and Stone, 2020). According to International Data Corporation (IDC) (https://www.idc.com/promo/smartphone-market-share), android operating system remained the number 1 mobile operating system since 2017, occupying 85% of market share in second quarter of 2020 with smartphone shipments hitting approximately 1.37 billion units in 2019 (https://www.statista.com/statistics/263441/global-smartphone-shipments-forecast/). In mobile security field, it is believed that the commonly used operating system or application is, the more likely it is to be exposed to attacks (https://www.kaspersky.com/resource-center/threats/malware-popularity). This is main reason why android remains the favourite target of cyber-attacks and malware in present scenario. Unlike other platforms, android allows installing apps from various sources, such as Google Play Store and other third-party marketplaces. This in turn, has led to an increase in its potential as a target for malicious activity. As mobile devices have gained popularity, computing platforms and data storage units, mobile computing privacy, and security concerns also increase. In recent years, malware developers have used sophisticated techniques to evade traditional and modern malware protection mechanisms. As a result, malware analysis and detection has been an active area of research lately, and a multitude of techniques have been proposed in this area using concepts from a wide range of scientific disciplines such as graph theory, machine learning (Mateless et al., 2020) and information visualisation to name a few.

The existing approaches for android malware detection can be categorised into two main classes that are static and dynamic analysis. Static analysis is a malware detection approach that examines the malware without running it (Nath and Mehtre, 2014), which means that only the source code and the binaries are inspected. In contrast, the dynamic analysis uses behaviour and actions when running in a controlled environment like a sandbox to determine whether the application is a malware or not (Shijo and Salim, 2015). While the static analysis is straightforward and quick, it is ineffective against unknown malware with code obfuscation and encryption (Gascon et al., 2013). On the other hand, dynamic malware detection typically provides better accuracy than static methods (Wong and Lie, 2016). However, the major disadvantages are that it can detect malicious behaviour only if it is performed during analysis. Starting from exploiting both advantages of static and dynamic analysis, and that detection approaches based on permission and API usage are susceptible to instruction-level obfuscation techniques (Gascon et al., 2013), we employ a hybrid approach to detect android malwares. Our approach combines static detection and dynamic detection (Amin et al., 2020). The hybrid approach analyses the files of static applications and monitors the behaviours of the applications through execution (Mahmood et al., 2014).

This work has been motivated by the obtained results of (Lashkari et al., 2018) who created a dataset named CICInvesAndMal2017 and built an android malware detection model based on it. They have started that the extracted network-flow features were good for malware binary classifications, however they did not give their best results in the malware category and family classifications. Hence, it comes the idea of combining these network-flow features (which represent the dynamic features) with the extracted static features (Intend Filters and permissions) from android applications. Our solution increased considerably the detection accuracy of binary classification and, furthermore, it gave a highest accuracy in the malware category detection.

Table 1 List of abbreviations

Abbreviation	*Stands for*
IDC	International Data Corporation
APK	Android Package Kit
RF	Random Forest
k-NN	K Nearest Neighbour
API	Application Programming Interface
SVM	Support Vector Machine
GS	Genetic Search
FPR	False Positive Rate
LSI	Latent Semantic Indexing
IoT	Internet of Things
IP	Internet Protocol
AUC	Area Under Curve
DNN	Deep Neural Network
MKL	Multiple Kernel Learning
HADM	Hybrid Analysis for Detection of Malware
TAM	Tree Augmented naïve Bayes
SMS	Short Message Service
CSV	Comma Separated Values
DT	Decision Tree
TPR	True Positive Rate

In our proposed approach, both Intent Filters and Permissions characteristics (static features) from APK (android package kit) files are extracted. The resulted features are then combined with network flow (dynamic features) as a unique dataset. Then the most relevant features are selected by applying the CfsSubsetEval (Correlation-based Feature Subset Evaluation) method to reduce the dimension and the complexity of the dataset. To evaluate our proposed approach, various machine learning algorithms are used. To the best of our knowledge, this framework is the first one that combines apps network flow features with static ones. Furthermore, it proposes a multiclass classification to detect malware category. Table 1 shows the list of abbreviations used throughout the paper.

The contributions of this paper are as follows:

1 A new hybrid dataset that combines both static (permissions, Intent filters) and dynamic features (Network flow) is developed.
2 A feature selection selected method is used to reduce the dimensionality of the developed dataset.
3 A novel hybrid android malware detection approach is proposed.
4 The study proposes multiclass classification method to detect the malware category.

The remainder of this paper is organised as follows. In Section 2, we discuss relevant background information, including related work. Section 3 contains a background of android application file and its components. Section 4 discusses the proposed android malware detection approach. In Section 5, the developed dataset is presented with a detailed discussion on experimental results. The results and discussions are presented in Section 6. The paper is concluded with some future directions in Section 7.

2 Related works

Android malware analysis can be classified into three classes: static analysis, dynamic analysis, and hybrid analysis. In this section, we focus on the works that are based on usage of machine learning for android malware analysis.

2.1 Static analysis

The droidDet framework (Zhu et al., 2018) was proposed to detect android malware. The study employed Rotation Forest (RF) to build the model evaluated on multiple types of features (permissions, sensitive APIs, monitoring system events, and permission-rate). In the same context, a static analysis technique is also used by extracting permissions from android manifest files (Varna and Visalakshi, 2020). The main contribution lies in the k-NN based Relief algorithm to select relevant features from permissions. Then an optimised SVM algorithm was used to evaluate the model's performances. Feature's selection techniques were likewise addressed (Firdaus et al., 2018) by proposing a genetic search (GS) algorithm to select the best features that give a higher score to some features in permissions and directory path. However, the results suffer from high false positive rate. Latent semantic indexing (LSI) was proposed (Singh et al., 2020) as a dimensionality reduction technique to build a lightweight detection system. Opcode features were integrated with permissions and intent in a single vector. End-to-end deep learning architectures using Bidirectional long short-term memory (BiLSTMs) neural networks that detect and attribute android malware using opcodes obtained from bytecode was proposed in Amin et al. (2020). They decomposed their system into four layers (input, pre-processing, decision and output). The proposed system gave an accuracy up to 99.9% tested on large dataset of more than 1,8 million apps. Although the promising results, the use of a large dataset could bring to an intense time consuming. In

Song et al. (2016), a malware detection framework based on permissions was proposed that combines four layers of filtering mechanisms that is, the message digest (MD5) values, the combination of malicious permissions, the dangerous permissions, and the dangerous intention applied on hash, permissions and action as detection objects. A threshold of threat-degree was defined specially to detect dangerous permissions. The drawback of their suggestion lies in the size of the dataset that contains only 1,000 malicious apps and 100 non-malicious apps used to build the framework.

2.2 Dynamic analysis

Despite the most state-of-the-art contributions which satisfy by binary classification (benign, malware), EnDroid (Feng et al., 2018) proposes a semi-supervised malware family classification by predicting the family of the malware. It employs a two-fold dynamic analysis approach to detect android malware and applies Ensemble learning classifiers to verify the effectiveness of EnDroid. This approach's main limitation is that it detects only the executed malicious behaviour and considers only the IP address and the port number as a feature for network flow. Burguera et al. (2011) proposed a method named CrowDroid to detect android malware. The proposed method is based on the dynamic analysis of app behaviour for anomalies. CrowDroid uses k-means clustering for classifying attack and normal instances. The authors showed that the proposed method could isolate the malware and alert the users at the same time. Another dynamic approach named AntiMalDroid is proposed by Zhao et al. (2011). AntiMalDroid performs monitoring of apps behaviors to find malware. It is capable of detecting unknown malware. The results shown by the authors indicated that AntiMalDroid achieved acceptable detection rates. Enck et al. (2010) proposed TaintDroid, a real-time analysis tool to detect android malware. TaintDroid performs real-time analysis of sensitive data sources and tries to find data leakage points in the android system. The authors carried out the performance evaluation in an android emulation environment and showed that the proposed method achieves high efficiency with no false positives. However, it fails to track the information that goes and comes back to the network. An emulation-based technique for malware detection in android is proposed by Yan et al. (2012). The proposed approach is known as the DroidScope that monitors the activities of the android operating system. It is capable of detecting privilege-based attacks. DroidScope achieves high accuracy of malware detection if input features are significant. However, the major limitation of this technique is that it has limited code coverage. Taheri et al. (2019) generated a second part of their android malware dataset CICAndMal2017 which includes permissions and intents as static features, and API calls as dynamic features. By introducing these features with their two-layer android malware analyser they assumed that their precision achieved 95.3% in static-based malware binary classification at the first layer, 83.3% precision in dynamic-based malware category classification and 59.7% precision in dynamic-based malware family classification at the second layer.

2.3 Hybrid analysis

Roy et al. (2020) presented a feature engineering method for the detection of android malware. The authors emphasised on employing a hybrid of static technique and machine learning for improving the detection rate. The static analysis is used to map each API call to certain features, which are further aggregated. Then, machine learning classifiers, including Logistic Regression, K-Nearest Neighbor, Support Vector Machine, Random Forest, are utilised to classify instances into malware and benign classes. The proposed hybrid method achieves an AUC score of 98.87%. Deep learning techniques are widely used in the hybrid approaches for android malware analysis due to its high accuracy for predicting android applications' nature. One such deep learning-based malware detection technique is Droid-Sec (Yuan et al., 2014). It uses a hybrid analysis by combining both static features (permissions, sensitive API) and dynamic feature (Dynamic behaviour) extracted by running APK files in the DroidBox, which permits to obtain 18 dynamic behaviours. However, the number of training samples should be increased despite their highest accuracy of 96%, since authors used only 250 apps for both normal and malware apps. Xu et al. proposed a deep learning-based android malware detection model named as HADM (Xu et al., 2018). HADM is based on the idea that a combination of advanced features that are extracted using Deep Neural Network (DNN) with the original features can improve detection performance. In this regard, both original dynamic features and static features are fed as input to deep learning classifiers to get new features that are further combined with original features to create DNN vector sets. Also, the dynamic information is transformed into graph-based representations. A hybrid classifier is then built by combining the learning results from vectors and graph features with hierarchical Multiple Kernel Learning (MKL). HADM achieved a classification accuracy of 94.7%. Surendran et al. (2020) proposed a tree augmented naïve bayes (TAM) based hybrid method for detecting android malware. The authors argued that the interdependency of static and dynamic features must be considered in machine learning-based detection models to avoid multicollinearity problems. Therefore, the proposed method employs conditional dependencies among useful static and dynamic features like API calls, permissions, and system calls. TAM achieved the detection accuracy of up to 97%. BRIDEMAID framework (Martinelli et al., 2017) combines static and dynamic analysis to detect android malware. For android malware detection, it employs three consecutive steps: static analysis, meta-data analysis and dynamic analysis. In the static analysis (which occurs just after the download phase), n-grams classification is applied on the decompiled file. The meta-data analysis (which occurs during the installing phase) includes features such as permissions, developer’s reputation, rating, etc. as metrics to detect the suspicious apps. Finally, during the runtime phase, the dynamic analysis exploits both classifiers and security policies to control suspicious activities related to text messages, system call invocations and administrator privilege abuses. Authors evaluated their framework’s performances using energy consumption and performance overhead. The proposed framework achieves an accuracy in android malware detection equal to 99.7%. Despite the high accuracy detection, the proposed framework is expected to consume more time in the detection phases, which is not discussed in this work.

3 Background

In this section, we briefly introduce the APK file components and give an overview of two APK components used in our experiments, i.e., permissions and intent filters. Besides, a brief description of android malwares used in our approach.

3.1 Android application structure

Android package kit (APK) is a zip-compressed file containing all components of the android application. It includes four directories (META-INF, res, libs, and assets) and three files (AndroidManifest.xml, classes.dex, resource.asc). Table 2 lists the APK components.

Table 2 The APK file components

File	*Description*
META-INF	APK signatures and certificates directory
Res/	Resources directory
Libs/	Libraries directory
Assets/	Application assets directory
AndroidManifest.xml	APK configuration file
Classes.dex	The classes compiled in the dex file format understandable by the Dalvik virtual machine
Resource.asc	Precompiled resources file

3.1.1 Permissions

The main goal behind the permissions is to ensure the android user's privacy. For that purpose, each application must demand permission to access the user's data (contacts, SMS, etc.) or device components like camera, WIFI, etc. Permissions might be grant either by the system automatically or by request of the user (Dong et al., 2018). It can be retrieved from the AndroidManifest.xml file under the <uses-permissions> tag. Depending on the nature of each request of the application, we can reveal if that application is malicious or not.

3.1.2 Intent filter

The intent filter is an expression in the manifest file of the application that defines the form of intent that the component would like to obtain. For example, by creating an intent filter for activity (https://developer.android.com/guide/components/intents-filters), you make it possible for other applications to start your activity with some kind of intent explicitly. Furthermore, if there is no announcement of any intent filters for an activity, it can only be initiated with explicit intent.

3.2 Android malwares

This part contains a brief description of android malwares categories used in our approach.

Adware is unwanted software designed to display pop-up ads on the user's screen, most commonly in a web browser and which transmits information to its publisher, allowing these advertisements to be adapted to the user's profile. Although not harmful to the device, adware is considered malicious software (malware) because of its aggressive and disruptive operation.

Ransomware is malicious software that infects a computer, typically when the user clicks on a link or file received as an email attachment. The criminals can then remotely block the device and encrypt the files. Users lose control of all information stored on the device, and the malware displays a screen asking for a ransom, often in virtual currency (e.g., bitcoins).

Scareware is a particularly insidious technique that uses the user's fear to attack him. It simulates the warning messages sent by the Windows Security Centre to trick you into scanning your hard drive and downloading protection software.

SMS In this kind of malware, the app introduces itself as a normal application for SMS messaging and uses its permissions to send or receive SMS. Since several mobile service providers offer services that allow users to transfer credits/units via SMS, this service is exploited by the application to illegally transfer credits from users (Chehab et al., 2012).

4 Proposed HyDroid system for android malware detection

This paper presents an android malware detection approach named HyDroid. By using this approach, users can identify if the android application is normal or malware. We have extended out approach by providing the category of the detected malware, which helps users or professionals to plan the countermeasures. The workflow of our proposed approach (Figure 1) is as follows:

First, we collect APK files from CICInvesAndMal2019 (https://www.unb.ca/cic/datasets/invesandmal2019.html) public dataset, and then apply the reverse engineering technique to extract the static features in the APK file. For this reason, Apktool is used in order to unzip the compressed APK file and obtain its components which includes four directories (META-INF, res, libs, and assets) and three files (AndroidManifest.xml, classes.dex, resource.asc). In experiments, permissions and intent filters are extracted from the AndroidManifest.xml file because they represent as critical features that can be manipulated by malicious persons, using feature engineering techniques. We created a python script that: interprets the AndroidManifest.xml files to extract all existing permissions in benign and malware applications of the dataset, places it in a table, then performs another analysis by comparing the features in the table with each sample (Application) and generating for each of them a vector filled with zeros and ones, the same for the Intent filter attributes.

Using a feature selection technique, the permissions with a high impact on the class, and the intent filters are selected. This considerably reduces the number of training features and makes the detection model less complex.

We applied the CfsSubsetEval method with the Best First search method located in the Weka tool. It was applied to the static dataset to select the static features (Permission, Intent Filter) to get the most relevant features that will be used to train our model. It is an attribute evaluator in Weka for the CfsSubsetEval method. Its principle of operating is to evaluate the value of a subset's attributes, considering the individual predictive ability of each one. By applying this method, the number of features remained 59 after 2,054 features, among which 36 features were intent filters, and 23 features were Permissions. Table 3 shows the static features obtained after the application of CfsSubsetEval.

Table 3 List of selected features (intend filters and permissions)

N°.	*Intent Filters*	*N°.*	*Permissions*
1	1006TV	37	ACCESS_CHECKIN_PROPERTIES
2	ACTION_068A22E3	38	ACCESS_MTK_MMHW
3	ACTION_08250282	39	BILLING
4	ACTION_0C1EF840	40	BROADCAST_PACKAGE_REMOVED
5	ACTION_0C2961E6	41	BROADCAST_SMS
6	ACTION_COLLECT_POPUP_ADS	42	DELETE_PACKAGES
7	ACTION_DEVICE_ADMIN_DISABLED	43	FULL_SCREEN
8	ACTION_DEVICE_ADMIN_DISABLE_REQUESTED	44	INSTALL_PACKAGES
9	ACTION_EXTERNAL_APPLICATIONS_AVAILABLE	45	MOUNT_UNMOUNT_FILESYSTEMS
10	ACTION_LAUNCH_HOME	46	PROVIDER_ACCESS_MODIFY_CONFIGURATION
11	ACTION_PUSHAD	47	READ_GSERVICES
12	ACTION_ROUSE	48	READ_HISTORY_BOOKMARKS
13	APPWIDGET_UPDATE	49	READ_PHONE_STATE
14	AUDIO_BECOMING_NOISY	50	REAL_GET_TASKS
15	BS	51	RECEIVE
16	CBOOT_COMPLETED	52	SEND_SMS
17	DETECT	53	SET_ANIMATION_SCALE
18	DEVICE_ADMIN_ENABLED	54	SET_PROCESS_FOREGROUND
19	EULA_RECEIVER	55	STATUS_BAR_SERVICE
20	EULA_RESPONSE_RECEIVER	56	SYSTEM_ALERT_WINDOW
21	INNER_BROADCAST	57	WRITE_PUSHINFOPROVIDER
22	LAUNCHER_SERVICE	58	WRITE_SECURE_SETTINGS
23	MAIN	59	WRITE_SMS

Table 3 List of selected features (intend filters and permissions) (continued)

N°.	*Intent Filters*	*N°.*	*Permissions*
24	MESSAGE_ARRIVED		
25	NS		
26	PUSH_AD_CLICK		
27	P_START		
28	RSSI_CHANGED		
29	SERVICE_RESCHEDULE		
30	SMS_RECEIVED2		
31	SPRINGBOARD		
32	TIME_SET		
33	UMS_DISCONNECTED		
34	USER_PRESENT		
35	VIEW		
36	ZS		

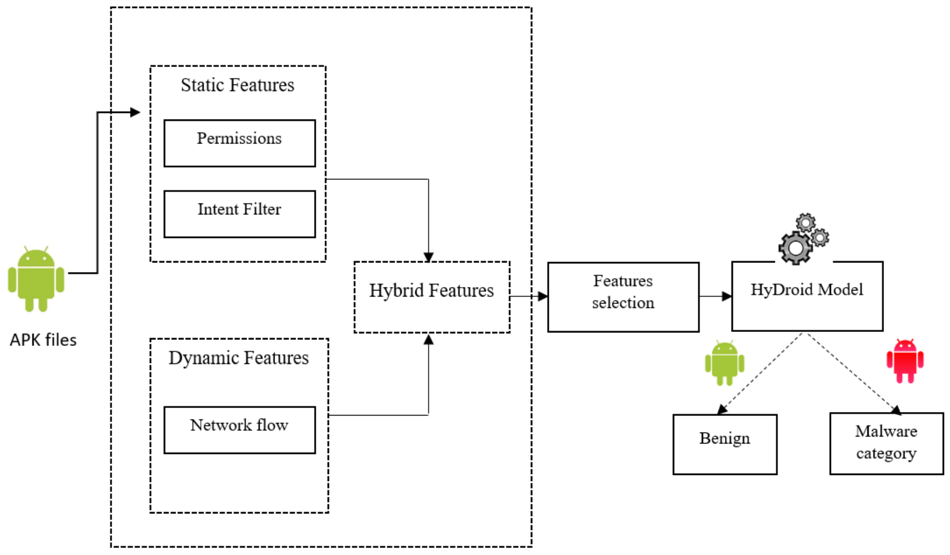
On the other hand, the network flow of each android application located in a separate folder in the CICInvesAndMal2019 dataset as dynamic features are employed. The selected static features from permissions and intent filters are then combined with network flow as dynamic features for creating a hybrid dataset.

The hybrid features are normalised then fed to machine learning models to evaluate the effectiveness of HyDroid. This step serves to normalise the quantitative data by putting them all in the same scale, this considerably facilitates the learning of machine learning models which are based on gradient descent, distance calculation or the variance calculation. For this, there are several normalisation methods and the one we used to be MINMAX, this method consists of transforming each variable so that these values will be all between 0 and 1, for that we subtract each value of a variable at the minimum of this variable then we divide by the difference between the max of the variable and the min of that variable, this is what the mathematical formula of minimax looks like:

$$X_{scaler} = \frac{X - X_{\min}}{X_{\max} - X_{\_\min}}$$

Hence, we use three machine learning classifiers (k-NN, SVM and DT). The model can predict the nature of the android application (benign, malware). Furthermore, it can even predict the category of malware (Adware, Ransomware, Scareware, and SMS) with high accuracy.

Figure 1 Architecture of the proposed approach (see online version for colours)



5 Experiment

5.1 APK dataset

Our work is based on the CICInvesAndMal2019 (https://www.unb.ca/cic/datasets/invesandmal2019.html) dataset, which contains 1594 APK samples in total, in which 426 malware applications and 1168 benign. Table 4 shows the distribution of captured characteristics between static and dynamic features. We notice the existence of permissions and intent characteristics as static features and API calls, network flow as dynamic features in three steps (During installation, before restarting, and after restarting the phone).

Table 4 List of captured characteristics in CICInvesAndMal2019 dataset

Captured static samples						*Captured dynamic samples*				
Stats	Permissions	Intent	Component	Certificate	Source code	API calls	Network flows	System call	Information flow	Logs
Yes	Yes	Yes	No	No	No	Yes	Yes	No	No	No

The dataset is completely labelled and includes network flows, logs, API/SYS calls, phone statistics, and memory. Next, authors in their dataset extract over 80 network flow features for all benign and malware applications using Cicflow meter software, which is publicly available on the Canadian Institute for Cyber Security website (https://www.unb.ca/cic/datasets/invesandmal2019.html). The samples come from 42 families of malware applications and 1,168 benign applications. The category and the numbers of the samples captured (Table 5).

Table 5 Distribution of android applications based on category

Malware category	*Captured samples*
Adware	104
Ransomware	101
Scareware	112
SMS Malware	109
Benign	500
	600
	68
Total	*1,594*

5.2 Preprocessing phase

The data pre-processing phase is essential (Azzaoui et al., 2021), and it has a significant impact on the quality of learning. In this step, the data is prepared and processed to be in an acceptable format to generate models that describe applications' behaviour better. First, in the static analysis, the application's source code is analysed without being executed in an emulator or a real device. For this, we used the Apktool (https://ibotpeaches.github.io/Apktool) to obtain the AndoidManifest.xml files by unzipping the APK archives. Second, in the dynamic analysis, the results of network stream captures were retrieved as csv files from the Canadian Institute website.

5.3 Features extraction phase

After the pre-processing phase, the dataset becomes easy to handle at this stage. The appropriate features are extracted to build models that are used to classify the apps and detect any malicious behaviour. Our proposal created two datasets; the first consists of static features only, and the second consists of hybrid features (static and dynamic).

5.4 Permissions and intent filters inspection

In general, this type of features depends on the analysis of applications (APK files) after decompression. There are several classes in which static features can be extracted from, for example, manifest files, source code, semantic characteristics, application metadata, etc. As far as our work is concerned, we have chosen only the features that identify the applications, which are the filter intent and the device permissions in the manifest file. We extract these features to convert the APK apps to the csv file that characterises these features' existence or absence. We created a python script that: interprets the AndroidManifest.xml files to extract all existing permissions in benign and malware applications of the dataset, places it in a table, then performs another analysis by comparing the features in the table with each sample (application) and generating for each of them a vector filled with zeros and ones, the same for the Intent filter attributes.

We opted to represent the vector of each of the applications as follows: 1,725 attributes of an intent filter and 329 attributes of permission, the whole equal to 2,054 numerical attributes with one and/or zero values, respectively representing the presence or absence of each of the permissions and intent filter in the AndroidManifest.xml file, in a specific order, ended by two attributes. The first class contains two labels, malware or benign, to indicate whether the software is malicious or not, the second to show the category of malware.

Let R be a vector that contains 329 permissions (1N725 Intent filter). For each i^{th} application, we generate a binary sequence:

$$R_i = \{r_1, r_2, r_3, ..., r_j\} \; with \; rj = \begin{cases} 1, \textit{ if permission (Intent Filter) exists} \\ 0, \textit{ else} \end{cases}$$

The identified permissions and Intent Filters are stored as a binary sequence of 0 or 1 in a comma separated form. This sequence usually contains permission /intent filters bits separated by commas which indicates 1 if the corresponding permission/ Intent Filters is present or 0 if it is absent, here is an example:

0;0;0;0;0;0;0;0;0;0;0;1;0;0;0;0;0;0;0;0;0;0;0;0;1;0;0;0;0;0;0;0;0;1;0;0;0;0;0;0;0;0;0;0;
0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;1;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;
0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;1;0;1;0;1;1;1;1;1;1;1;0;
0;0;0;0;…………0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;
0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;

0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;;0;0;0;0;0
;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;Malware; RANSOMWARE

Table 6 List of the extracted network flow features

Features' name
Source Port, Destination Port, Protocol, Flow Duration, Total Fwd Packets, Total Backward Packets,Total Length of Fwd Packets, Total Length of Bwd Packets, Fwd Packet Length Max, Fwd Packet Length Min, Fwd Packet Length Mean, Fwd Packet Length Std,Bwd Packet Length Max, Bwd Packet Length Min, Bwd Packet Length Mean, Bwd Packet Length Std,Flow Bytes/s, Flow Packets/s, Flow IAT Mean, Flow IAT Std, Flow IAT Max, Flow IAT Min,Fwd IAT Total, Fwd IAT Mean, Fwd IAT Std, Fwd IAT Max, Fwd IAT Min,Bwd IAT Total, Bwd IAT Mean, Bwd IAT Std, Bwd IAT Max, Bwd IAT Min,Fwd PSH Flags, Bwd PSH Flags, Fwd URG Flags, Bwd URG Flags, Fwd Header Length1, Bwd Header Length,Fwd Packets/s, Bwd Packets/s, Min Packet Length, Max Packet Length, Packet Length Mean, Packet Length Std, Packet Length Variance,FIN Flag Count, SYN Flag Count, RST Flag Count, PSH Flag Count, ACK Flag Count, URG Flag Count, CWE Flag Count, ECE Flag Count, Down/Up Ratio, Average Packet Size, Avg Fwd Segment Size, Avg Bwd Segment Size, Fwd Header Length,Fwd Avg Bytes/Bulk, Fwd Avg Packets/Bulk, Fwd Avg Bulk Rate, Bwd Avg Bytes/Bulk, Bwd Avg Packets/Bulk,Bwd Avg Bulk Rate,Subflow Fwd Packets, Subflow Fwd Bytes, Subflow Bwd Packets, Subflow Bwd Bytes,Init_Win_bytes_forward, Init_Win_bytes_backward, act_data_pkt_fwd, min_seg_size_forward,Active Mean, Active Std, Active Max, Active Min,Idle Mean, Idle Std, Idle Max, Idle Min.

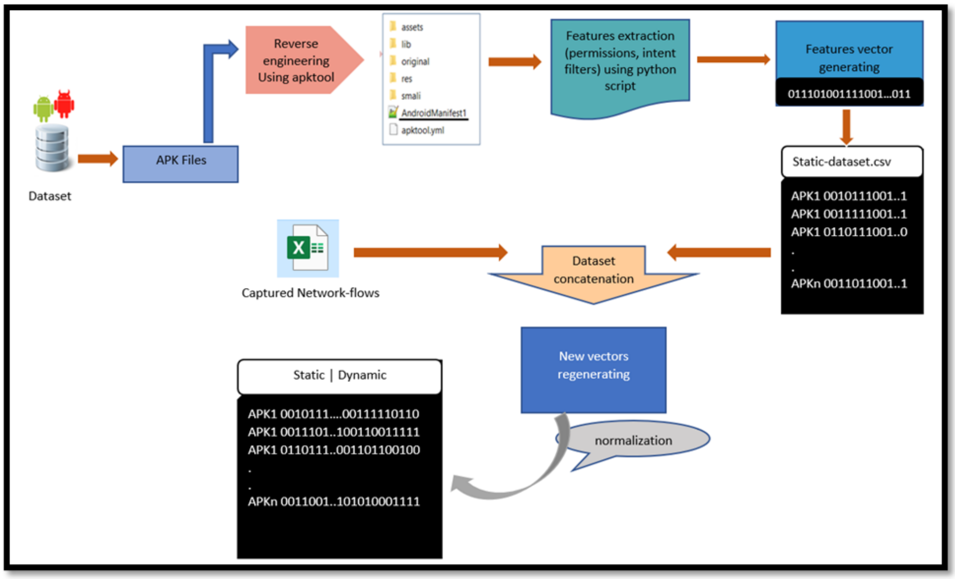
5.5 *Hybrid features*

First, the network flow files collected from each application belonging to dataset were processed, grouped and organised by the name of the application and then listed in folders identified by the type of application (e.g. Benign 2015, Benign 2016, Benign 2017), in the case of malware applications, the files are named by category then by family (in our case we consider only category), e.g., the Scareware category, the fake Job Offer family, then categorised which make it easily to be combined with their corresponding static dataset vectors, so that we take each network flow.csv file from the application and associate it with the equivalent static dataset vector in a redundant manner. Therefore, we have established a dataset that represents a combination of the static features (permissions and Intent Filter) and dynamic features (network flow). Due to the large volume of the obtained hybrid dataset, we randomly sampled 30 % of each application.

5.6 *Features selection phase*

This phase consists of selecting the most relevant features among the existing features in the dataset in order to build an efficient model. Thus, it represents a critical task since it affects classifier performance by limiting the number of irrelevant features. First, we applied the CfsSubsetEval method with the Best First search method located in the Weka tool. It was applied to the static dataset to select the static features (permission, intent filter) to get the most relevant features that will be used to train our model. It is an attribute evaluator in Weka for the CfsSubsetEval method. Its principle of operating is to evaluate the value of a subset's attributes, considering the individual predictive ability of each one. By applying this method, the number of features remained 59 after 2,054 features, among which 36 features were intent filters, and 23 features were permissions. Table 6 shows the static features obtained after the application of CfsSubsetEval.

Figure 2 The process of feature vector generation (see online version for colours)



To evaluate the performances of our approach, we use three machine learning algorithms K-nearest neighbours (k-NN), support vector machine (SVM), and decision tree (DT). The previous step's selected features are fed into the commonly used machine learning classifiers like k-NN, SVM, and DT by 10-fold cross-validation technique to measure the performance of HyDroid. Machine learning algorithms are performed in python scripts through the sklearn library (https://scikit-learn.org/stable/).

5.7 *Evaluation metrics*

In our experiments, we train our models on a classification problem with five classes. A confusion matrix is used in our method to evaluate the effectiveness of different models. We can calculate the TPR, FPR, and Accuracy of each model based on the resulted confusion matrix.

True positive rate (TPR) is defined as TP divided by the total count of malicious applications.

$$TPR = \frac{TP}{TP + FN}$$

False positive rate (FPR) is defined as FP divided by the total count of benign applications.

$$FPR = \frac{FP}{FP + TN}$$

Accuracy is defined as the sum of TN and TP divided by all applications' total count.

$$Accurancy = \frac{TP + TN}{TP + FP + TN + FN}$$

6 Results and discussion

Tables 7–9 summarise the performance metrics of the k-NN, SVM, and DT. Table 7 shows the evaluation metrics of HyDroid with a highest TPR (93.8%) and accuracy (96.3%) for k-NN classifier.

Table 7 Evaluation results of HyDroid

Algorithm	*TPR*	*FPR*	*Accuracy*
k-NN	93.8 %	6.1%	96.3 %
SVM	91.1 %	3.01 %	95.7 %
DT	93.4 %	5.3 %	95.6 %

Table 8 Android detection accuracy in static analysis

Apps category	*Accuracy*		
	k-NN	*SVM*	*DT*
Benign	89.3%	89.8%	89.3%
Adware	96%	96%	96%
Ransomware	97.6%	96.6%	96%
Scareware	95%	93.2%	94.2%
SMS	96.3%	96.6%	95.3%
Overall accuracy	92.7%	92.7%	92.3%

Tables 8–9 present the results of malware category classification for static and hybrid analysis respectively by considering four well-known malware categories namely Adware, Ransomware, Scareware and SMS malware described in Section 3.2.

From Table 8, we notice that the accuracy of benign class in static analysis is roughly equal in all classifiers, with a slight difference in favour of SVM classifier with an accuracy of 89.8 %. The obtained results can be justified because the models cannot learn only from the static patterns of benign applications due to its high variance. In contrast, we see that the three models perform well on malware category classification with more than 94% for all malware categories except the SVM model that gives an accuracy of 93.2 % for the scareware category. The overall accuracy for the three models is almost similar in all classifiers, with 92%. Table 9, however, shows a high accuracy as compared with Table 7 results. We notice a significant improvement in the class being's accuracy in the three classification models, especially the SVM classifier that gives the highest accuracy achieving 96.4%. We interpret the notable increase of the benign class accuracy by adding more informative data related to the network flow as dynamic features. The static features described by permissions and intent filters help the model predict well the benign class. As for the category classification with HyDroid, the model shows a slight improvement in the four malware categories with the highest accuracy of 95.9%, 97.3%, 94.2%, and 97.1%, respectively, for adware, ransomware, scareware, and SMS.

Table 9 Android detection accuracy with HyDroid

Apps category	*Accuracy*		
	k-NN	*SVM*	*DT*
Benign	91.8%	96.4%	89.8%
Adware	95.3%	95.9%	95.4%
Ransomware	97.1%	97.2%	97.3%
Scareware	94%	94.2%	93.2%
SMS	97.1%	96.8%	96.8%
Overall accuracy	95%	96.1%	94.5%

This experiment approves our initial assumption that, by combining network flow as dynamic features with permissions and intent filter as static features, we can considerably improve benign prediction accuracy, thus improving the overall android application model accuracy.

Figure 3 Accuracy of multi-class static analysis classification (see online version for colours)

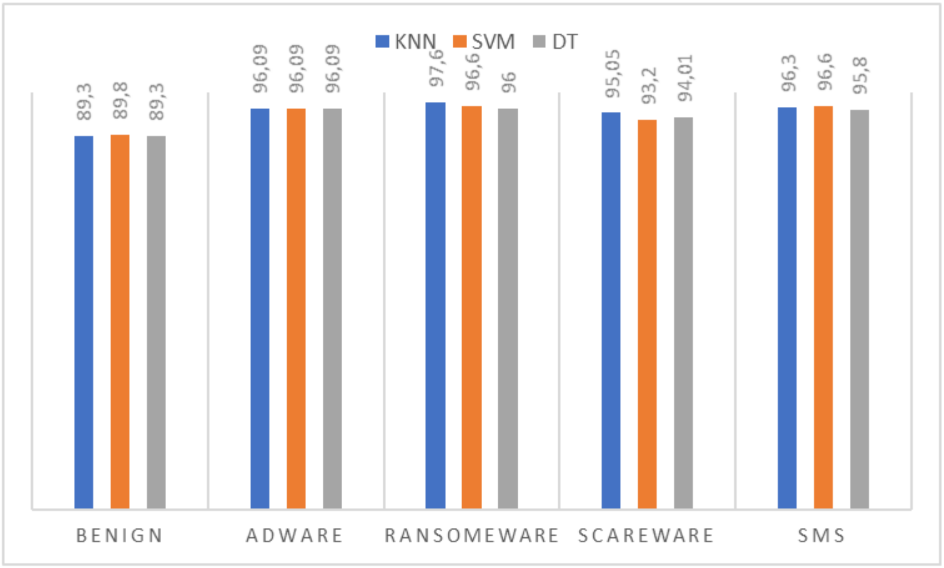
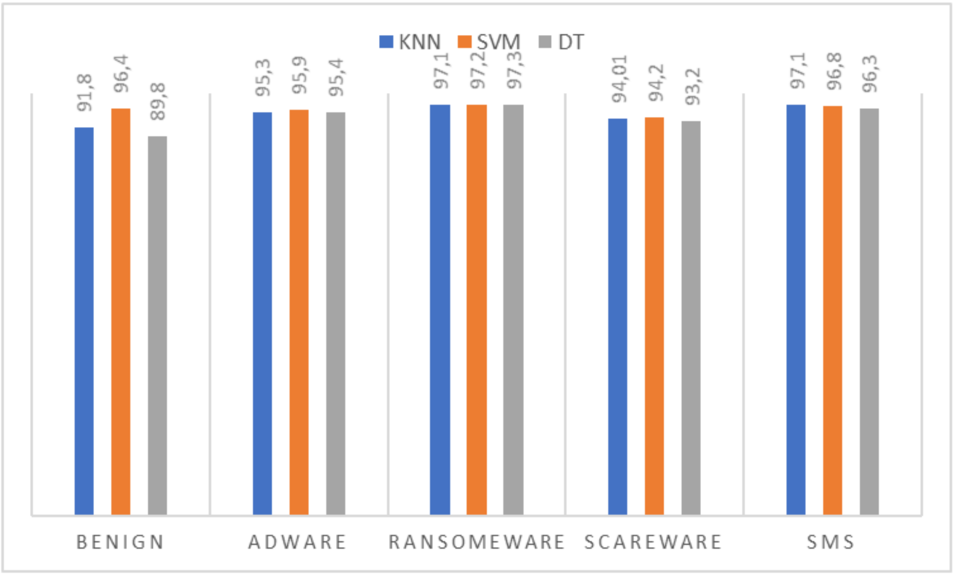


Table 10 compares the proposed method with the state-of-art android malware analysis in terms of accuracy. It can be seen that HyDroid approach proposed by us achieve a detection accuracy of 96.3% which outperforms most methods in terms of performance. We notice from the summary table that most of the proposed works are based on classifying the android apps as benign or malware which is not the case of HyDroid that predicts also the category of the detected malware with a highest accuracy with k-NN classifier as mentioned in Table 9.

Table 10 Summary of related work

Ref	*Analysis technique*	*Features*	*Classifiers*	*Classes*	*Dataset*	*Accuracy*
Zhu et al. (2018)	Static	Permissions, sensitive APIs, monitoring system events, permission-rate	Rotation Forest	binary	Customised	88.26%
V.P.D and V.P (2020)	Static	Manifest	Optimised SVM	binary	AAGM google play store	–
Firdaus et al. (2018)	Static	Permissions, code-based, directory path,	Naïve bayes, functional trees, J48, RF, MLP	binary	Drebin google play store	95%
Xu et al. (2016)	Hybrid		DNN	binary	Google play store virus share	94.7%
Singh et al. (2020)	Static	Permissions, Intents, opcodes	Random Forest	binary	CICInvesAndMal2019	93.92%
Roy et al. (2020)	Dynamic	API calls	SVM	binary	Drebin CICInvesAndMal2019	88.72%
Shyong et al. (2020)	Hybrid	Permissions, tcp, http, dns	Random forest	multi-class	Drebin, google play store	96%
Yuan et al. (2014)	Hybrid	Required permission, Sensitive API, dynamic behavior	Deep learning	binary	Contagio google play store	96%
Our study	Hybrid	Permissions, intent filters, network flows	k-NN, SVM, DT	multi-class	CICInvesAndMal2019	96.3%

Figure 4 Accuracy of multi-class hybrid analysis classification (see online version for colours)



7 Conclusions

Over the last years, android has occupied a high market share, making it the preferred target of malicious applications. This paper proposes a hybrid android security approach that detects android malware and provides the malware category. The approach aims at combining both static (permissions, Intent filters) and dynamic features (network flow) to perform an android hybrid analysis. The experimental results showed that the proposed approach improves android category classification accuracy compared with only static analysis. Thus, it can be concluded that by combining network flow as dynamic features with permissions and intent filter as static features, we can considerably improve benign prediction accuracy. This consequently improves the overall detection model's accuracy. In practice, this methodology is used to improve the mobile apps security by incorporating the network aspect into the static method when scanning an apps. Some android malware attacks (Adware, Scareware, Ransomware, and so on) execute their commands over the network, allows attacker to send requests and receive personal information, as well as alter the victim's system. The idea of exploiting network flow features and reinforcing them with static analysis of the apps increases the detection accuracy of android malware, because this method does not satisfy with static analysis results but also allows to involve network parameters, which can be decisive in the phase of the scan. Moreover, the concept of this framework permits not only the detection of potential malware in the installed apps that are not running but also the infected running apps which tries to run a malicious action. In future, we will try to add other patterns from both static and dynamic behaviours of android applications to have an in-depth vision of the malware behaviour. Also, we will consider improving the effectiveness of the framework by combining several machine learning algorithms to give an additional layer that represents the android malware family class.

Acknowledgements

This work is a part of PRFU project N° C00L07UN300120210001 supported by Ouargla University https://www.univ-ouargla.dz/.

References

Amin, M., Tanveer, T.A., Tehseen, M., Khan, M., Khan, F.A. and Anwar, S. (2020) 'Static malware detection and attribution in android byte-code through an end-to-end deep system', *Future Generation Computer Systems*, January, Vol. 102, pp.112–126, https://DOI: 10.1016/j.future.2019.07.070.

Azzaoui, H., Boukhamla, A.Z.E., Arroyo, D. and Bensayah, A. (2021) 'Developing new deep-learning model to enhance network intrusion classification', *Evolving Systems*, January, https://DOI: 10.1007/s12530-020-09364-z.

Burguera, I., Zurutuza, U. and Nadjm-Tehrani, S. (2011) 'Crowdroid: behavior-based malware detection system for android', in *Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, New York, NY, USA, October, pp.15–26, https://DOI: 10.1145/2046614.2046619.

Chehab, A., Elhajj, I.H., Hamandi, K., Chehab, A., Elhajj, I.H. and Kayssi, A. (2013) *Android SMS malware: Vulnerability and Mitigation*.

Dong, S. et al. (2018) 'Understanding android obfuscation techniques: a large-scale investigation in the wild', in *Security and Privacy in Communication Networks, Cham*, pp.172–192, https://DOI: 10.1007/978-3-030-01701-9_10.

Enck, W. et al. (2010) 'TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones', in *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, USA, October, pp.393–407.

Eveleth, L.B. and Stone, R.W. (2020) 'User's perceptions of perceived usefulness, satisfaction, and intentions of mobile application', *IJMC*, Vol. 18, No. 1, p.1, https://DOI: 10.1504/IJMC.2020.104431.

Feng, P., Ma, J., Sun, C., Xu, X. and Ma, Y. (2018) 'A novel dynamic android malware detection system with ensemble learning', *IEEE Access*, Vol. 6, pp.30996–31011, https://DOI: 10.1109/ACCESS.2018.2844349.

Firdaus, A., Anuar, N.B. Karim, A. and Razak, M.F.A. (2018) 'Discovering optimal features using static analysis and a genetic search-based method for android malware detection', *Frontiers Inf Technol Electronic Eng.*, June, Vol. 19, No. 6, pp.712–736, https://DOI: 10.1631/FITEE.1601491.

Gascon, H., Yamaguchi, F., Arp, D. and Rieck, K. (2013) 'Structural detection of android malware using embedded call graphs', in *Proceedings of the ACM Workshop on Artificial Intelligence and Security*, New York, NY, USA, November, pp.45–54, https://DOI: 10.1145/2517312.2517315.

Hamandi, K., Chehab, A., Elhajj, I.Hand Kayssi, A. (2013) 'Android SMS Malware: vulnerability and mitigation', *2013 27th International Conference on Advanced Information Networking and Applications Workshops*, March, pp.1004–1009, doi: 10.1109/WAINA.2013.134.

Lashkari, A.H., Kadir, A.F.A., Taheri, L. and Ghorbani, A.A. (2018) 'Toward developing a systematic approach to generate benchmark android malware datasets and classification', in *2018 International Carnahan Conference on Security Technology (ICCST)*, Montreal, QC, October, pp. 1–7, https://DOI: 10.1109/CCST.2018.8585560.

Mahmood, R., Mirzaei, N. and Malek, S. (2014) 'EvoDroid: segmented evolutionary testing of Android apps', in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, New York, NY, USA, November, pp.599–609, https://DOI: 10.1145/2635868.2635896.

Martinelli, F., Mercaldo, F. and Saracino, A. (2017) 'BRIDEMAID: A hybrid tool for accurate detection of android malware', in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, Abu Dhabi United Arab Emirates, April, pp.899–901, https://DOI: 10.1145/3052973.3055156.

Mateless, R., Rejabek, D., Margalit, O. and Moskovitch, R. (2020) 'Decompiled APK based malicious code classification', *Future Generation Computer Systems*, September, Vol. 110, pp.135–147, https://DOI: 10.1016/j.future.2020.03.052.

Nath, H.V. and Mehtre, B.M. (2014) 'Static Malware Analysis Using Machine Learning Methods', in *Recent Trends in Computer Networks and Distributed Systems Security*, Berlin, Heidelberg, pp. 440–450, https://DOI: 10.1007/978-3-642-54525-2_39.

Roy, A., Jas, D.S., Jaggi, G. and Sharma, K. (2020) 'Android malware detection based on vulnerable feature aggregation', *Procedia Computer Science*, Vol. 173, pp.345–353, https://DOI: 10.1016/j.procs.2020.06.040.

Shijo, P.V. and Salim, A. (2015) 'Integrated static and dynamic analysis for malware detection', *Procedia Computer Science*, January, Vol. 46, pp. 804–811, https://DOI: 10.1016/j.procs.2015.02.149.

Shyong, Y-C., Jeng, T-H. and Chen, Y-M. (2020) 'Combining static permissions and dynamic packet analysis to improve android malware detection', in *2nd International Conference on Computer Communication and the Internet (ICCCI)*, Nagoya, Japan, June, pp. 75–81, https://DOI: 10.1109/ICCCI49374.2020.9145994.

Singh, A.K., Wadhwa, G., Ahuja, M., Soni, K. and Sharma, K. (2020) 'Android malware detection using lsi-based reduced opcode feature vector', *Procedia Computer Science*, Vol. 173, pp.291–298, 2020, https://DOI: 10.1016/j.procs.2020.06.034.

Song, J., Han, C., Wang, K., Zhao, J., Ranjan, R. and Wang, L. (2016) 'An integrated static detection and analysis framework for android', *Pervasive and Mobile Computing*, October, Vol. 32, pp. 15–25, https://DOI: 10.1016/j.pmcj.2016.03.003.

Surendran, R. Thomas, T. and Emmanuel, S. (2020) 'A TAN based hybrid model for android malware detection', *Journal of Information Security and Applications*, Vol. 54, p.102483, October, https://DOI: 10.1016/j.jisa.2020.102483.

Taheri, L., Kadir, A.F.A. and Lashkari, A.H. (2019) 'Extensible android malware detection and family classification using network-flows and API-calls', in *International Carnahan Conference on Security Technology (ICCST)*, October, pp.1–8, https://DOI: 10.1109/CCST.2019.8888430.

Varna, P.D. and Visalakshi, P. (2020) 'Detecting android malware using an improved filter-based technique in embedded software', *Microprocessors and Microsystems*, July, Vol. 76, p.103115, https://DOI: 10.1016/j.micpro.2020.103115.

Wong, M.Y. and Lie, D. (2016) 'IntelliDroid: a Targeted input generator for the dynamic analysis of android malware', *Presented at the Network and Distributed System Security Symposium*, San Diego, CA, https://DOI: 10.14722/ndss.2016.23118.

Xu, L., Zhang, D., Jayasena, N. and Cavazos, J. (2018) 'HADM: hybrid analysis for detection of malware', in *Proceedings of SAI Intelligent Systems Conference (IntelliSys)*, Vol. 16, Bi, Y., Kapoor, S. and Bhatia, R. (Eds.): Springer International Publishing, Cham, pp. 702–724, https://DOI: 10.1007/978-3-319-56991-8_51.

Yan, L.K. and Yin, H. (2012) 'DroidScope: seamlessly reconstructing the OS and Dalvik semantic views for dynamic Android malware analysis', in *Proceedings of the 21st USENIX Conference on Security Symposium*, USA, August, p.29.

Yuan, Z., Lu, Y., Wang, Z. and Xue, Y. (2014) 'Droid-Sec: deep learning in android malware detection', in *Proceedings of the ACM Cconference on SIGCOMM*, New York, NY, USA, August, pp.371–372, https://DOI: 10.1145/2619239.2631434.

Zhao, M., Ge, F., Zhang, T. and Yuan, Z. (2011) 'AntiMalDroid: An Efficient SVM-Based Malware Detection Framework for Android', in *Information Computing and Applications*, Berlin, Heidelberg, pp.158–166, https://DOI: 10.1007/978-3-642-27503-6_22.

Zhu, H-J., You, Z-H., Zhu, Z-X., Shi, W-L., Chen, X. and Cheng, L. (2018) 'DroidDet: Effective and robust detection of android malware using static analysis along with rotation forest model', *Neurocomputing*, January, Vol. 272, pp. 638–646, https://DOI: 10.1016/j.neucom.2017.07.030.

Websites

Adoption Rate and Popularity [online] www.kaspersky.com (accessed 11 September 2017) [online] https://www.kaspersky.com/resource-center/threats/malware-popularity (accessed 13 September 2020).

Apktool – A Tool for Reverse Engineering 3rd Party, Closed, Binary Android Apps [online] https://ibotpeaches.github.io/Apktool/ (accessed 13 September 2020).

Global smartphone shipments 2010–2022', *Statista* [online] https://www.statista.com/statistics/263441/global-smartphone-shipments-forecast/ (accessed 8 October 2020).

IDC – smartphone market share – OS*, Idc: The Premier Global Market Intelligence Company* [online] https://www.idc.com/promo/smartphone-market-share (accessed 8 October 2020).

Intents and intent filters, *Android Developers* [online] https://developer.android.com/guide/components/intents-filters (accessed 16 September 2020).

Investigation on Android Malware 2019 | Datasets | Research | Canadian Institute for Cybersecurity | UNB [online] https://www.unb.ca/cic/datasets/invesandmal2019.html (accessed 19 September 2020).

Scikit-Learn: Machine Learning in Python — Scikit-Learn 0.23.2 Documentation [online] https://scikit-learn.org/stable/ (accessed 29 September 2020).