Computers and Electrical Engineering 100 (2022) 107903

Contents lists available at ScienceDirect

Computers and Electrical Engineering

journal homepage: www.elsevier.com/locate/compeleceng

Mitigating adversarial evasion attacks of ransomware using ensemble learning☆

Usman Ahmed [a], Jerry Chun-Wei Lin [a,*], Gautam Srivastava [b,c]

[a] Department of Computer Science, Electrical Engineering and Mathematical Sciences, Western Norway University of Applied Sciences, 5063, Bergen, Norway
[b] Department of Mathematics & Computer Science, Brandon University, Canada
[c] Research Centre for Interneural Computing, China Medical University, Taichung, Taiwan

ARTICLE INFO

Keywords:
Android ransomware
Adversarial evasion attacks
Machine learning-based ensemble analysis
Attack mitigation
Ransomware detection

ABSTRACT

Ransomware continues to pose a significant threat to cybersecurity by extorting money from users by locking their devices and personal data. The attackers force the payment of a ransom in order to restore access to personal files. Because of the structural similarity, detection of ransomware and benign applications becomes vulnerable to evasion attacks. Ensemble learning can provide countermeasures, while attackers can use the same technique to improve the effectiveness of their respective attacks. This motivates us to investigate whether the distinct ensemble method can achieve better performance when combined with the voting-based method. This research proposes a hybrid approach that examines permissions, text, and network-based features both statically and dynamically by monitoring memory usage, system call logs, and CPU usage. Ensemble machine learning analyzers on static and dynamic features extracted from Android malware applications (ransomware and non-ransomware) are then trained in the designed model. Our experimental results show that the proposed ensemble classification and detection technique can classify unknown static and dynamic ransomware behavior to mitigate adversarial evasion attacks.

1. Introduction

Ransomware (RW) attacks have become one of the biggest security threats facing individuals and businesses worldwide. Typical malware targets users by deleting or damaging files, changing system configurations, disclosing user information to third parties, etc. Ransomware, on the other hand, gains access to user data and computer resources undetected, notifies the victim and demands a ransom to release access to the captured resources (e.g., encrypted files, etc.). Crypto-ransomware finds and encrypts the files on the device with a solid cipher to deny the user access. Locker ransomware locks the device itself, mainly by locking the user interface or using pop-up overlays, so that the user cannot enter the device in the first place [1]. Ransomware is not just a Windows operating system phenomenon. It also attacks other platforms, such as Android devices. At the end of 2018, Android has over 86.8% of the total cell phone market share.[1] Android has emerged as the most widely used operating system for mobile devices [2]. In September 2018, McAfee Lab stated that the total number of ransomware had reached 17 million. Android ransomware will be one of the most important security threats in the future.[2]

☆ This paper is for special section VSI-mlsec. Reviews were processed by Guest Editor Dr. Suyel Namasudra and recommended for publication.

* Corresponding author.
E-mail addresses: usman.ahmed@hvl.no (U. Ahmed), jerrylin@ieee.org (J.C.-W. Lin), SRIVASTAVAG@brandonu.ca (G. Srivastava).

[1] https://www.statista.com/statistics/236027/global-smartphone-os-market-share-of-Android/.

[2] https://www.mcafee.com/enterprise/en-us/assets/reports/rp-quarterly-threats-sep-2018.pdf.

https://doi.org/10.1016/j.compeleceng.2022.107903
Received 31 May 2021; Received in revised form 5 March 2022; Accepted 9 March 2022
Available online 8 April 2022

Ransomware can spread as a legitimate application or is downloaded (unintentionally) by users who intend to download software updates, apps, etc. from third-party app stores [3] or by clicking on the spam link sent in SMS messages. However, modern Android ransomware is usually spread via compromised apps that are freely available to users via third-party app stores. Ransomware attackers select a popular app to mimic a realistic app that infects a large user base. Depending on the complexity of the attack, the attackers retain the original functionality of the application and may add malicious code to it, or the application may only display the icon and name of the original application. This is done in order to install ransomware on the target device unnoticed and make the user [4]. Once installed, the ransomware collects information about the victim's device, searches for the target resources such as files, resources, etc., and communicates with the Command & Control (C&C) server to obtain the encryption key if it was not already included in the payload. After that, the ransomware hijacks (locks/encrypts) the target resource and displays a message to the victim asking him to pay the ransom, along with the payment instructions.

Currently, ransomware developed mainly for Android devices is on the rise. Due to the alarming increase of Android ransomware applications, Android ransomware analysis and detection has become an important research area. Some techniques for Android ransomware detection and classification have already been proposed. We can divide ransomware detection techniques into two categories: Static analysis and Dynamic analysis. Static analysis uses the syntax or structural properties of the application to determine its maliciousness. Static analysis relies on feature extraction (without execution) from resource files, Android manifest files, Java bytecode, etc. The Android manifest file contains all the required permissions, which are the central design point of the Android security model [5]. By default, no application has the permission to access sensitive data (such as contacts or SMS) and certain system functions (such as camera, Internet). Ransomware developers use the permissions mainly for privilege escalation and access sensitive data stored on the device.

Dynamic analysis aims to detect malicious behavior during program execution. Dynamic analysis can look at features such as dynamic code loading, the sequence of system calls collected during application execution, network activity, CPU usage, and memory usage [1]. Similarities in ransomware application behavior can help identify new (zero-day) ransomware. Most state-of-the-art techniques [2] do not take into account the structural features specific to the appearance of ransomware, such as the text in the source code. Ransomware may contain specific threats within its code, e.g., to lock, encrypt, porn, etc. Most Android ransomware requires individual permissions (such as BIND_DEVICE_ADMIN, KILL_BACKGROUND_PROCESS, and RECEIVE_BOOT_COMPLETED, etc.), which can be helpful for ransomware detection. Ransomware regularly establishes network connections to retrieve commands or send data collected from devices [4]. Therefore, network addresses (email address, IP address, URLs) may be present in the code of different ransomware samples, which can help in ransomware detection. These network-based features have never been statically analyzed for Android ransomware detection before.

1.1. Motivation

Behavioral analysis based on hardware features such as CPU usage, memory usage, and system call logs could be useful for Android ransomware classification, as they are more resistant to change compared to static features that ransomware can evade through code obfuscation and encryption [1,6,7]. To our knowledge, detection and classification (using machine learning) of Android ransomware has not yet been performed using the combination of the above features. Due to advances in machine learning techniques, a significant amount of research has been conducted on Android malware detection using machine learning techniques [8]. Although machine learning techniques have proven their effectiveness in malware detection, machine learning classifiers are not very resilient to adversarial attacks. This aspect is highlighted in the following text: ''in the learning phase, the dataset used for training remains representative of the problem domain, assuming no intentional malicious modification of the data'' [9]. Therefore, malicious users often employ adversarial attacks to fool the machine learning models. We can divide adversarial attacks into two types (1) evasion and (2) poisoning attacks [9,10]. In evasion attacks, attackers intentionally fabricate malicious inputs so that the classification model incorrectly classifies an application as benign or clean. In poisoning attacks, on the other hand, attackers poison the training data to compromise the entire learning process. The focus of this work is on countering circumvention attacks by adversaries. We identify a subset of features from several features of different behavior types. Moreover, we construct the set of multiple distinct features instead of using a single feature vector. The main motivation for using a single feature vector based on several distinct subsets was generalization. This helps the instances to contribute more to the trained classifiers. In this way, it is difficult for attackers to bypass the detection model. We use the benchmark Android ransomware to extract multiple discriminative subsets based on their behavioral analysis. We used an ensemble of multiple classifiers and combined them with the voting method. Finally, we compared the proposed method with the traditional ML-based model in adversarial environments.

1.2. Contributions

This study focuses on mitigating evasion attacks in Android ransomware detection, such as code obfuscation and its use to evade malware/ransomware detection. Most of the existing ransomware techniques fail to modify the input feature vector for analysis (when one aspect is used for obfuscation, it changes the entire feature vector, which causes the trained classifier to misclassify the ransomware. Therefore, this research proposes an ensemble-based analysis mechanism to detect Android ransomware and mitigate evasion attacks. The Android characteristics used for analysis here are not easy to modify and use for evasion attempts. Based on this motivation, we propose a technique that combines the effectiveness of both static (i.e., permissions, text, network-based features, etc.) and dynamic features (such as system call logs, CPU, and memory usage) to detect Android ransomware using an ensemble machine learning model. In summary, the main contributions of this research are:

1. Extraction and analysis of static network-based features such as IP addresses, email addresses, and URLs.
2. Development of two different machine learning ensemble models that include multiple machine learning algorithms for static and dynamic feature sets for Android ransomware detection and mitigation of adversarial evasion attacks.
3. Evaluating the effectiveness of the proposed model for mitigating adversarial evasion attacks using a large dataset of fabricated feature vectors from Android ransomware samples.

We have structured the remainder of the paper as follows. In the next Section 2, we present a literature review of related techniques. Section 3 describes the proposed methodology, followed by Section 4, where the experimental setup is presented. Section 5 provides a detailed evaluation and discussion of the obtained results and Section 6 concludes the paper.

2. Related work

There are several approaches to ransomware detection, and some of them relate to the Android platform. Song et al. [2] proposed a technique that carefully monitors and specifies processes and certain file directories using statistics on processor usage, memory usage, and I/O rate so that processes with unusual behavior can be detected. This technique has been implemented with three modules (i.e. configuration, monitoring and processing). The configuration module creates monitoring list tables, while the monitoring module monitors the processor, memory, and I/O usage of each process. The processing module takes care of processes that are considered suspicious by the monitoring module and makes an exception or isolates them. The proposed approach can be implemented in the Android source code. Without obtaining information about the ransomware, it can reduce the damage caused by unknown ransomware. However, this technique does not perform static analysis, which could have been more efficient.

Yang et al. illustrated the design of an automated hybrid analysis technique [3]. The proposed system uses static analysis based on matching features such as permissions, sequence of API invocations, resources, and APK structure. Dynamic analysis describes the nature of the attack in terms of data leaks such as web browser cookies and others without gaining access to the exact protected data sources in a mobile device.

Alberto et al. proposed a hybrid approach to Android ransomware detection that first examines (using static analysis based on opcode frequency) the application to be used on a device before installing it [8]. Then, dynamic analysis identifies whether the system is under attack by monitoring CPU usage, memory usage, network usage, and system call statistics. The dataset used for the experiments was small, and such analysis cannot examine different Android ransomware families.

Ensemble-based learning helps improve countermeasures in adversarial environments [9]. A generative-based attack generation without executing the malicious functionality was investigated. New instantiations based on adversarial examples are used as an instance for training. This improves the ensemble method and can be used as a more robust classification model. Ensemble-based methods are used for unique classification and fake feedback detection [11].

Gharib et al. [12] proposed a DNA droid technique, a hybrid real-time detection framework that can rapidly evaluate a sample using static analysis. If the application is only considered suspicious, it is continuously monitored and runtime behavior is profiled. Once the profile resembles a collection of malicious profiles, DNA-Droid terminates the application. The overall architecture of their proposed framework includes three main components: a static analysis, a dynamic analysis and a detection module. The static module includes three subcomponents (Text Classification Module (TCM), Image Classification Module (ICM), and API calls and permissions Module (APM)) to assess different aspects of an APK file. The dynamic module profiles malware families based on the sequences of API calls and produces DNA for each family. In the detection phase, the runtime behavior of a suspicious sample is continuously compared to the families of the DNA.

Alzahrani et al. [13] has introduced Randroid. This automated approach measures the structural similarity between the collected information of the examined application and the threat-related information collected by known ransomware variants to classify the application as ransomware or goodware. The Randroid approach extracts the application's information such as images and text from XML layout files, resources and class.dex files in the static analysis phase. The dynamic analysis captures extortion activity and examines the presence of threat letters or lock screens. Image Similarity Measurement (ISM) and String Similarity Measurement (SSM) are used to determine the similarity between the extracted information and previously collected information about known ransomware. Based on the similarity scores, the examined application is classified as suspicious, good software or ransomware.

Another proposed solution for crypto-ransomware detection was proposed by Chen et al. [14], namely RansomProber. RansomProber is a real-time detection technique that analyzes User Interface (UI) widgets of related activities, coordinates the user's finger movements, and detects whether the ransomware starts the file encryption process. Ransom prober includes three steps: encryption analysis, foreground analysis and layout analysis. The encryption analysis module is used to detect if some files are encrypted. The foreground analysis module decides whether the encryption operation belongs to the user's application, and the layout analysis module analyzes UI widgets of the corresponding activities and the user's coordinates. Ransom prober can detect repackaged ransomware that targets an application without encryption. This technique is designed to detect crypto-ransomware only. The proposed approach does not detect repackaged applications that use encryption or compression methods.

Mercaldo et al. described a model checking technique for identifying malicious payloads in Android ransomware [15]. This technique is divided into three subprocesses (construction of formal model, construction of temporal logical properties, and detection of ransomware family). In the construction of the formal model, the bytecode of the application is parsed and appropriate formal models of the system are created. The construction of Temporal Logic Properties defines the characteristic behavior of ransomware in terms of a set of properties. In ransomware family detection, the formal verification environment including a model checker is used to detect ransomware families. The special features of this approach are formal methods and detection of ransomware from Java byte code.

Table 1
Summary of the related work.

Approaches	Static features	Dynamic features	NLP	Image processing	Heuristic	Machine learning	Deep learning	Behavior analysis
Gharib et al. (2017) [12]	✓	✓	✗	✓	✓	✗	✓	Sequence analysis
Song et al. (2016) [2]	✓	✗	✗	✗	✓	✗	✗	✗
Yang et al. (2016) [3]	✓	✓	✗	✓	✓	✗	✗	✓
Alzahrani et al. (2019) [13]	✓	✓	✗	✓	✓	✗	✗	✗
Chen et al. (2017) [14]	✓	✓	✗	✗	✓	✗	✗	✗
Mercaldo et al. (2016) [15]	✓	✗	✗	✗	✓	✗	✗	✓
Alberto et al. (2018) [8]	✓	✓	✗	✗	✗	✓	✗	✓

Ensemble-based methods have been adopted by several approaches [16]. The method combines ensemble learning and transfer learning on time series data to implement incremental updates. As a result, the hybrid method independently makes predictions and improves accuracy. In another work, NATICUSdroid [17] was proposed, which selects the specific permissions as features and then classifies them as benign or malware. The selection is based on the trend of the permissions. The proposed model is evaluated using eight different machine learning algorithms. The random forest classification model achieved an accuracy of 97%, a false positive rate of 3.32%, and an F-measure of 0.96.

Another method *PerbDroid* [18] was proposed using six different approaches for feature ranking (i.e., gain ratio, OneR feature evaluation, chi-square test, information gain feature evaluation, principal component analysis (PCA), and logistic regression analysis) to create the classification algorithm based on API, permission, and intents. The model was tested with a real-world application [18]. The model used the hyper-tuning selection process and classification algorithm to select and classify malware detection. However, malware detection and countermeasures based on adversarial models still need to be explored.

We summarize the strengths and weaknesses of current approaches in Table 1. The literature review shows that most of the techniques proposed so far, whether they perform only static or dynamic detection [2]. Although static analysis is fast, secure, and accurate in identifying known ransomware samples [1], static-only detection could be vulnerable to ransomware attacks that obfuscate code to alter structure [15] and are unable to deal with samples that encrypt or compress their payloads. Dynamic analysis is resistant to evasion [8]. It can detect unknown ransomware based on the general behavioral signatures [2]. In addition, dynamic analysis has some vulnerabilities, such as some actions only trigger certain conditions that may not be available in a test environment, such as an emulator [1]. Therefore, for ransomware detection, it may be worth combining behavior-based detection capabilities that are resistant to evasion [2] with the effective capabilities of static analysis [12]. Few other techniques that use a hybrid approach are type-specific, addressing only one type such as crypto-ransomware [14] or addressing only a specific ransomware family [3]. Family-specific detection cannot generalize the solution and apply it to every type of ransomware. Previously proposed approaches for Android ransomware detection [8] using machine learning techniques are vulnerable to adversarial evasion attacks. Attackers can compromise the entire detection model by simply obfuscating code or using other evasion techniques, since changing one aspect changes the entire feature vector; thus, the ransomware remains undetected by the machine learning classifier.

Overall, this work has inspired us to propose a hybrid analysis-based Android ransomware classification technique that employs both effective static and dynamic features and ensemble learning-based machine learning to mitigate the adversarial evasion attack.

3. Hybrid distinct ensemble analyze

We have discussed the proposed framework for ransomware classification in Fig. 1. The proposed methodology consists of two parts, i.e. *offline training* and *online prediction*, as shown in Fig. 1(a) and (b). In the offline training part, feature extraction, selection, training, and testing are performed. In the online prediction method, the trained classifier is used to predict ransomware and non-ransomware applications. Feature extraction based on static and dynamic analysis is discussed in 3.1 and 3.2. In offline training, static analysis is performed to extract permission, network, and text as shown in Figure (a). Then separate classifiers are trained for each type of feature. In this way, a static meta-classifier is trained based on permission, network features, and text. In dynamic analysis, the applications are run in a controlled environment, then the features are extracted and analyzed. We logged the system calls, usage of CPU, and memory consumption of the application and used them as features. Separate classification models are then trained depending on the nature of the features. In this way, a dynamic feature-based meta-classifier is trained. The proposed hybrid ensemble analysis involves two separate ensemble machine learning models trained on static and dynamic feature vectors. Each ensemble model consists of an odd number of machine learning classification algorithms, e.g., three or five. The stacking ensemble method is used to train and test the classification algorithms, where each algorithm is trained on the entire feature vector. In the offline prediction method, the static and dynamic meta-classifiers are then used in the voting method to predict the output, as mentioned in Fig. 1(b). Due to the separate ensemble learning models used in the methodology, we call it a hybrid distinct ensemble analyzer.

We started with a dataset of malware APK files that contained both Android ransomware and non-ransomware as input. Each Android application is packaged in an .apk file, a compressed file that contains several other files and folders, such as classes-dex files, assets, resources, META-INF and AndroidManifest.xml files, etc. In the first phase, the features to be used for static and dynamic analysis are extracted from an APK file. The extracted features and their compilation into feature vectors are then used to perform the static and dynamic analysis simultaneously. To extract static features, the APK file is decomposed into Java and XML files. For this

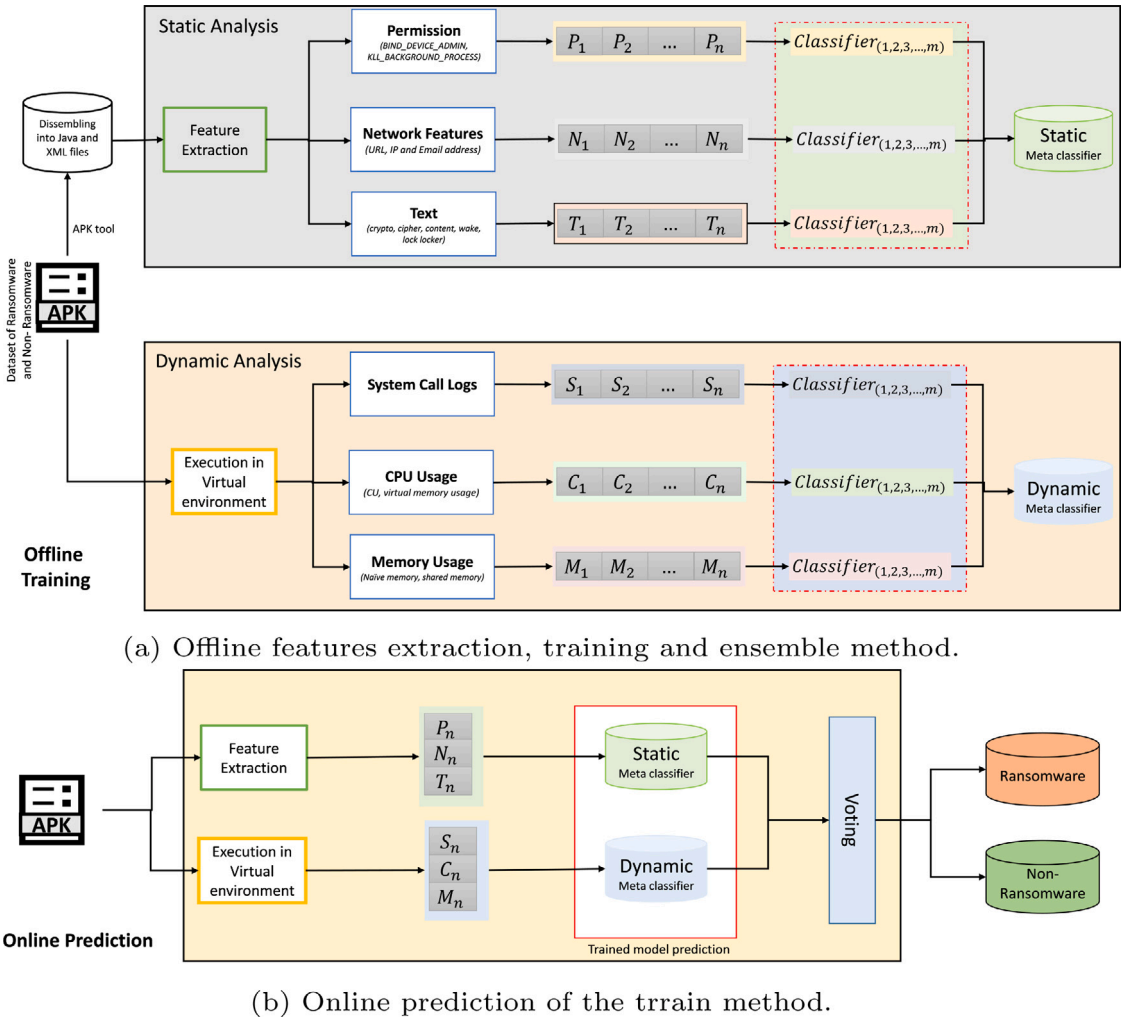


Fig. 1. The overall methodology of distinct ensemble analysis approach.

purpose, we used the Apk tool[3]. The Apk tool is a free open-source utility that unpacks an APK file into its individual resources [19]. The obtained Java and XML files are further scanned to extract features. Android has a special permission strategy. Permissions are granted by the user when the application is installed [12]. These permissions are extracted from the AndroidManifest.xml files. While text and network based features like email addresses, IP addresses and URLs are extracted from the Java files. These network features describe who the application communicates with after installation. Since Android ransomware activities are mostly network-based [4], these network features help in detecting Android ransomware. These features are converted into a combined feature vector and fed to the proposed static ensemble machine learning model in the second phase. The static ensemble machine learning model is trained using these feature vectors. Once this static ensemble is trained, it can classify the application and assign a label RW/NRW based on the identical static features.

Similarly, each APK file is run in an emulation environment to record the dynamic features for dynamic analysis. The extracted dynamic features of an APK file are converted into a feature vector. These feature vectors are further used to train the dynamic ensemble model. Then, the dynamic ensemble classifies the application and assigns the label RW/NRW. The final decision (on the classification of the application) takes into account feedback from both the static and dynamic ensemble models. Suppose one of the machine learning models (part of the ensemble framework) classifies the application as ransomware (by assigning it a RW label). In this case, the application is classified as ransomware. The application is classified as non-ransomware only if both the static and dynamic ensemble models assign it a similar label (i.e., non-ransomware NRW).

[3] https://ibotpeaches.github.io/Apktool/documentation/

3.1. Static feature extractor

Our experimental dataset includes .apk files (i.e., 50% ransomware and 50% non-ransomware). The Android Package Kit (APK) is a file format that Android uses to distribute and install applications. It contains all the elements such as classes (.dex files), resources, and manifest files that an application needs to be installed correctly on a device. The manifest file contains permissions and other configuration details of the application. Our feature extraction process starts with capturing APK files using a feature extraction script. We wrote a Python script to extract permissions from the manifest.xml file, text and network-based features (i.e., IP addresses, email addresses and URLs) from .dex files. The script decompiles the APK files, extracts these features, and then saves them to text files. We use these .txt files from both ransomware and non-ransomware applications to create feature vectors. The feature vector script reads the .txt files from both ransomware and non-ransomware applications. It outputs the feature vectors of each application after capturing a dataset of all features to identify characteristic features of each application and save the dataset in the output file.

We create a binary sequence for each application at its position in the dataset (representing the feature vector). All detected individual permissions are then ordered as a sequence of 0s and 1s. A particular authorization is denoted by a one and the absence of an authorization is denoted by a 0 in the list. The last bit of the vector represents the category of the application (i.e., ransomware or non-ransomware). All redundant permissions are removed from the dataset, as redundancy could have a negative impact on classification. After removing redundant permissions, we get 166 unique permissions.

Both text- and network-based features contain strings; therefore, we create their feature vectors using the TF-IDF vectorizer [20]. The TF-IDF vectorizer converts textual features into feature vectors that can be used as input to the classification algorithm. TF-IDF is an extravagant and increasingly effective representation for classification mechanisms of textual data [21]. Next, we use all the created static feature vectors to train static ensemble analyzers based on machine learning. Algorithm 1 describes the feature extraction process for both static and dynamic analysis of APK files, the conversion of extracted features into feature vectors, and the classification mechanism used to detect Android ransomware.

Algorithm 1 Feature extraction and classification detection

INPUT: APK_{File}.
OUTPUT: Malware or Ransomware.

for $all\ f \in F$ **do** ▷ F is APK folder
  $APK_{File} \leftarrow Open(file)$;
  $manifest_{File}, java_{File} \leftarrow APK\_Tool(APK_{File})$;
  if $manifest_{File} == androidmanifest.xml$ **then**
    permission$\leftarrow Get_Permission(androidmanifest.xml)$;
    for $all\ permission_(i) \in permission$ **do**
      if $Permission_{(list)}[i] == permission_{(i)}$ **then**
        $Vector_{(Permission)}[\ ] \leftarrow 1$;
      end if
      $Vector_{(Permission)}[\ ] \leftarrow 0$;
    end for
  end if
  $network_{vector} \leftarrow TF\_IDF(manifest_{File}, java_{File}, network_{File})$;
  $Text_{vector} \leftarrow TF\_IDF(manifest_{File}, java_{File}, Text_{File})$;
  $Dynamic_{vector} \leftarrow Virtual\_Environment(APK_{File})$;
  $Output_1 \leftarrow Classify(network_{vector} + Text_{vector} + Vector_{Permission})$;
  $Output_2 \leftarrow Classify(Dynamic_{vector})$;
  $Output \leftarrow XOR(Output_1, Output_2)$;
end for
Return $Output$.

3.2. Dynamic feature extraction

Dynamic analysis involves running an Android application in the virtual environment to examine its runtime behavior. Dynamic analysis can be used to detect the malicious behavior of applications that remain undetected in static analysis. In previous research related to dynamic analysis, much attention has been paid to the investigation of data leakage and the sequence of API calls sequence [3,12]. A promising approach to efficient dynamic detection of Android ransomware is to identify a helpful set of features that allow distinguishing between ransomware and non-ransomware behaviors. Our proposed framework explores such types of dynamic features (i.e., CPU usage, system call statistics, memory usage, etc.) that are less costly and more informative for Android ransomware detection [8]. Therefore, the execution traces containing this data must be collected by running the application in a

controlled environment. These traces were recorded manually by running applications one at a time for 10 minutes in an Android emulator. However, some traces are shorter because the emulator has minor weaknesses. However, a longer execution period gives us more meaningful results. The Android emulator Genymotion version 3.0.2 was chosen for dynamic analysis of Android malware applications. The Genymotion tool is used because it is open source software and supports Android Studio. The reason for using the Android emulator software instead of the original device is that the emulation environment provides more capacity to run a large number of malware programs within a reasonable time.

Dynamic analysis on the device is immune to emulator bypass techniques. However, in the case of Android ransomware, the physical device cannot be reset to a clean state. In contrast, the emulator can be reinitialized after analyzing each application. However, the anti-emulation might have some impact on the dynamic feature extraction performed on Genymotion. Therefore, in our proposed model, when using dynamic features (memory usage and CPU), we assume that even if applications are repackaged, obfuscated, or equipped with techniques to avoid detection errors, they will still exhibit similar behavior traces during their execution. An ensemble analyzer trained on these characteristics could adequately distinguish Android ransomware applications from other malware applications.

The virtual device is reinitialized each time before a new malicious application is executed to avoid interference from the previously executed applications, such as changed settings, execution of background processes, changes related to the operating system configuration, etc. Android Debug Bridge (ADB) is used to monitor the memory and CPU usage of the applications. The ADB is a command line tool that allows PC to communicate with an emulator instance or an Android device. Strace (a system call tracking tool) is used to collect system calls from applications. For dynamic feature extraction, the following steps are performed for each application as mentioned in Algorithm 2.

Algorithm 2 Controlled environment feature extraction process.

INPUT: APK_{File}.
OUTPUT: Dynamic features.
$Start_{Device} \leftarrow AVM(genymotion)$;
for $all\ f \in F$ **do** ▷ F is APK folder
  $Package \leftarrow APK(f)$;
  $Events - Execution \leftarrow Apply(wipes, presses, touchscreens)$;
  $Memory \leftarrow ADB(meminfo)$;
  $CPU \leftarrow ADB(cpuinfo)$;
  $Processes \leftarrow PID()$;
  $System - call \leftarrow Command(strace --p\ pid)$;
  $Terminate\ (f,\ 10minutes)$;
  $Exit(f)$;
end for
$Feature\_set \leftarrow Package,\ Memory, CPU, Processes, System - call$;
Return $Feature\_set$.

We have considered all the features related to system calls, memory, and CPU usage that can be accessed in Android. In total, there are 73 features for each running application. Five features relate to CPU: three to the use of CPU and two to virtual memory exceptions (major and minor errors). 63 features relate to the various aspects of memory usage, and 5 represent statistics of system calls. These features are further converted into feature vectors (i.e., numerical values) for classification. These feature vectors and application category (i.e., ransomware/non-ransomware) are used to train dynamic ensemble analyzers based on machine learning.

3.3. Ensemble learning

Two separate machine learning ensemble models are used to classify applications based on their static and dynamic features. Each ensemble is trained to deliver feature vectors along the category (i.e., ransomware with a value of 1 and non-ransomware with a value of 0) to the ensemble models. Each classifier (e.g., Naïve Bayes, Decision Tree, Random forest, etc.) in the ensemble model is trained with all the feature vectors. Once all these ensemble models are trained, they can classify the applications and assign class labels such as RW/NRW. All the membership classifiers are provided to a meta-classifier, which combines these results using a combination rule (i.e., majority voting) to assign the final label. Since we are dealing with two-class problems, we use the majority decision scheme to determine the final label. Based on the results of both ensemble models, the final label is assigned to the applications. To assign the final label, a "OR" operation is applied to the outputs of both ensembles. Thus, if either ensemble assigns the RW label to the application, it is classified as ransomware. The application is classified as non-ransomware only if both models of the static and dynamic ensembles assign it a similar label (i.e., NRW). Since supervised learning is used, the training set consists of Android application samples that are assigned to one of the two classes: Ransomware or Non-Ransomware. For ensemble models, we use Naïve Bayes, Decision Tree (j48/ c4.5), Random Tree, Random Forest, Support Vector Classifier, Logistic Regression, Adaptive Boosting (Ada boosting), Gradient Boosting, Support Vector Machine with Sequential Minimal Optimization, JRip, etc.

Naïve Bayes (NB) is a probabilistic classifier. It applies probability theory and Bayes theorem to make the assumption that features are independent [22]. **Decision tree** generates rules for predicting the target variables [22]. A tree classification algorithm makes it easy to understand the desired distribution of the data. **J48** (i.e., an open-source Java implementation of C4.5 in WEKA data mining tool) performs missing value calculation, decision tree pruning, continuous attribute value ranges, rule derivation, etc, [23]. **Random tree** is a tree created randomly from a set of potential trees that have a number of k random features at each node. In this case, ‘‘random’’ means that each individual tree has an equivalent stroke in the set of trees from which a sample is drawn. Then each tree can be said to have a ‘‘uniform’’ spread. Random trees can be made well, and integrating huge collections of random trees requires exact models [22]. **A random forest (RF)** consists of several random decision trees. There are two types of randomness built into the trees. First, each tree is built on a random sample from the original data. Second, at each node, a subset of features is randomly selected to generate the best split [22]. The goal of **Support vector machine (SVM)** is to fit the provided data and determine the best fitting hyperplane that categorizes the provided data. Once the hyperplane is determined, the features can be input to the classifier to determine the predicted class. In the case of multiple classes, SVM uses a ‘‘one versus one’’ strategy.

Logistic regression (LR) is a linear classifier that calculates the restrictive probabilities of the outcomes and selects the one with the highest probability. Boosting is a common ensemble method that generates a strong classifier from the various weak classifiers. This is done by constructing a model from the training data and then creating a model to correct the errors of the main model. Models are added until the training set is perfectly predicted with no errors or extreme values. **AdaBoost (AD)** works by weighting the observations. In this process, problematic samples or samples that are difficult to classify are weighted more heavily and those that are treated effectively are weighted more weakly. We used AdaBoost M1 with SVM basis for ensemble evaluation because it performs better than AdaBoost with a different type of weak learner [22]. Gradient Boosting identifies the weakness by using gradients in the loss function.

The output of different weak learners is combined so that their loss function can be optimized [22]. The loss function is a measure of how well the predictive model classifies the underlying data. **Gradient boosting (GB)** allows the loss function to be optimized by adding weak learners in the gradient descent process. **Support Vector Machine (SVM)** examines, identifies and matches patterns of data for classification. It uses a hyperplane to partition the data into regions of n-dimensional space. The hyperplane keeps the values of a margin between regions at the maximum. SVM uses a kernel function that results in a nonlinear classification surface instead of a linear hyperplane. **Sequential Minimal Optimization (SMO)** is an iterative algorithm for solving optimization problems that arise in the training phase of the Support Vector Machine (SVM). SMO performs a fragmentation of the problem into a sequence of smallest possible subproblems, which are then solved analytically [24]. **JRip** is a rule-based classification algorithm. It develops a proportional rule learner called ‘‘Repeated Incremental Pruning to Produce Error Reduction (RIPPER)’’ to extract the rule directly from the data and use successive coverage algorithms to produce requested rule lists. The algorithm goes through four stages. (1) growing a rule (2) pruning (3) optimization and (4) selection [24].

4. Experimental results

This study conducted an empirical evaluation to assess the effectiveness of our proposed Hybrid Distinct Ensemble Analysis approach. The evaluation method uses a 10-fold cross-validation technique to evaluate the proposed model. In k-fold cross-validation, the original dataset is randomly divided into k subsets of equal size. A single subset of the k subsets is used for validation to test the algorithm. The remaining $k - 1$ subsets are used as the training set. The cross-validation process is repeated precisely k times (the number of folds). Therefore, each of the k subsets are used exactly once as a validation set [22]. The advantage of this strategy is that all observations are used for both training and testing, and each observation is used exactly once for validation. After the k experiments, the weighted average of the classification accuracy is calculated. This research used the performance metrics given in Table 2, where **True Positive (TP)** is an accurate positive result that detects ransomware when ransomware is present.**True Negative (TN)** is an actual negative result that does not detect ransomware when ransomware is not present.**False Positive (FP)** is a false positive result, meaning ransomware is detected when ransomware is not present.**False Negative (FN)** is a false negative result, meaning ransomware is not detected when ransomware is present. The computing power used in the experiments is listed in Table 3.

4.1. Dataset

The entire dataset used in this study includes two subsets: Android malware (Drebin[4]) and Android ransomware (RansomProber[5]) applications as .apk files. The Drebin malware dataset is used for the malware dataset, a benchmark repository with more than five thousand malware applications (from 179 different malware families). The ransomware dataset used for analysis comes from the ransomware repository used for experiments in a research effort called RansomProber. The RansomProber dataset includes more than two thousand samples taken from related security alerts, threat reports from antivirus companies, and security blogs cite6chen2017uncovering. The RansomProber dataset shows good coverage of existing Android ransomware families. The dataset contains 5500 ransomware and 2280 non-ransomware. The experimentation used 275 samples of each class for evasion attacks. The classifiers used were trained using the behavioural characteristics of ransomware and non-ransomware applications

[4] https://www.sec.cs.tu-bs.de/~danarp/drebin/.

[5] http://csp.whu.edu.cn/RansomProber/download.html/.

Table 2
Summary of the performance metrics used for evaluation.

Metrics	Calculation Method
Precision (P)	$\frac{TruePositive}{TruePositive + FalsePositive}$
Recall (R)	$\frac{TruePositive}{TruePositive + FalseNegative}$
F-Measure	$2 * \frac{(Precision * Recall)}{(Precision + Recall)}$

Table 3
Experimental setup.

CPU	Intel® Core™ i5-5200 CPU @ 2.20 GHz
Memory	8 GB
OS	Window 8
APK Decompilation Tool	Apk Tool
Machine learning library	sklearn
Android emulator configuration	
Platform	Genymotion 2.12.2
Device	Custom Tablet
Android version	6.0.0
API Level	21
CPU	1.5 GHz
Memory size	2048 MB
Data disk capacity	16 384

Table 4
Summary of the ensemble and classification model configuration.

Ensemble	Classification models
1	C45 Decision tree, Random tree, Random forest
2	Logistic regression, C45, SVM with SMO
3	Random tree, Random forest, SVM with SMO
4	SVM with SMO, Logistic regression, Random forest
5	SVM with SMO, Logistic regression, AdaBoost with SVM base
6	Logistic regression, Jrip, Random forest, C45, SVM with SMO
7	SVM with SMO, logistic regression, Simple regression, AdaBoost with SVM base

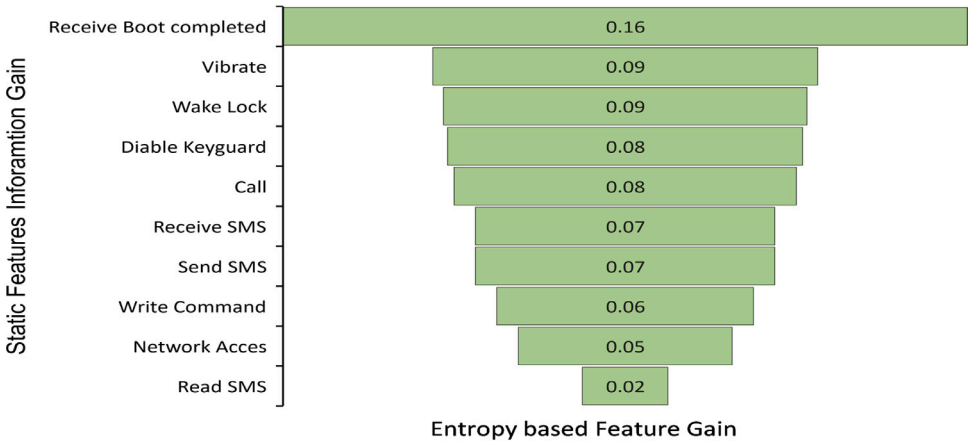
and an explicit label (i.e., ransomware/non-ransomware). The development method uses disjoint datasets for training and testing purposes. This study used many applications to ensure that our dataset is unbiased. The dataset used in this study is not limited to applications with specific attributes that can help in generating results. However, if the new features come from any functionality for the extended application, the model can incorporate the updated features. Table 2 describes the methods used to calculate all values. These measurements were made by evaluating each classification algorithm and various combinations of ensemble learning.

4.2. Static and dynamic feature selection using InfoGain

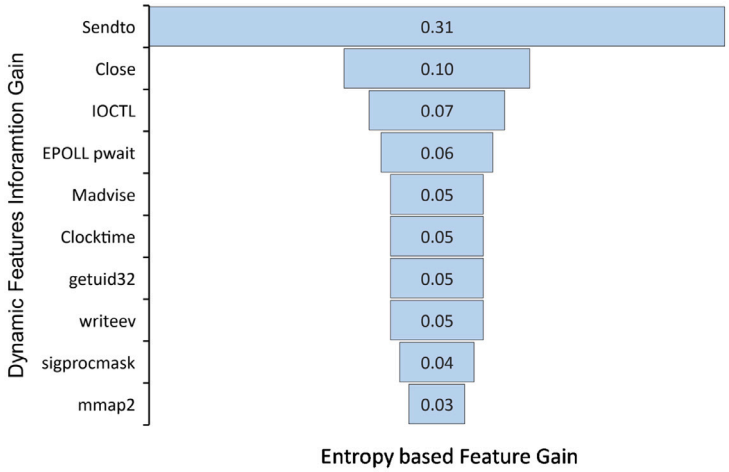
During the development of the proposed model, this study performed feature selection to select important attributes of the dataset that are most appropriate and useful for identifying application classes (i.e., RW/NRW). To this end, the study performs feature selection based on the information gain criterion to find the most appropriate features by assigning weights to the information to highlight the effectiveness of the features [22]. The method selected 72 static features from 2911 features after applying the feature selection process to the static features. The method selected 45 dynamic features from 130 dynamic features. In this work, the study combines the classification model and evaluates different combinations as given in Table 4.

Fig. 2 describes the top ten features obtained from the entire dataset of static and dynamic features using the InfoGain method. To check the performance of the proposed model, this study used the performance measures Precision, Recall, and F-measure. Table 2 describes the methods used to calculate all values. These measurements were performed by evaluating each classification algorithm and different combinations of ensemble learning. Figs. 3 and 4 show the effects of dataset shuffling on the F-measure during model development and the classification results with a single classifier and classification with different ensembles (for the selected static and dynamic features by the InfoGain method). The values for precision and recall are in the same range due to the dataset used.

Fig. 3 shows the performance metrics of our proposed model for ranked static and dynamic data using single machine learning algorithms. In contrast, Fig. 4 shows the performance metrics for ranked static and dynamic data using ensemble algorithms. From Figs. 3 and 4, it can be seen that the ensemble algorithms perform well compared to the single algorithms on the ranked hybrid data. Among the individual algorithms, Ada Boost (AB) [22] are the best with values of 0.853, 0.886 and 0.849 for precision, recall and F-measure, respectively. However, performing the ensemble of Random Tree + Random Forest + SVM with SMO as member classifiers is significantly below all others with 0.863 precision, 0.892 recall and 0.86 F-measure.



(a) Ranked static features.



(b) Ranked dynamic features.

Fig. 2. Comparison of static and dynamic information gain features.

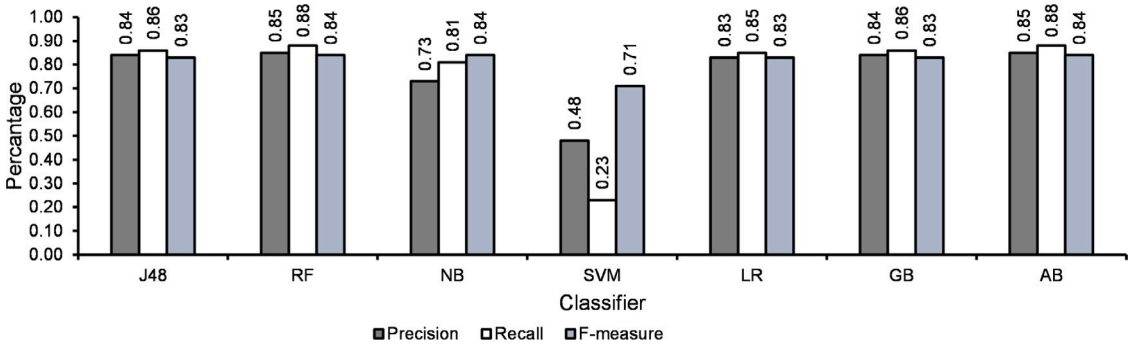


Fig. 3. Evaluation results of ranked data using a single machine learning.

4.3. Feature selection experiments using PCA

It is desirable to select a subset of unique features in certain applications rather than finding an assignment that uses all features. Using a subset of features can reduce computational costs and eliminate noisy features while preserving information using clean features. The other feature selection algorithms are either very computationally intensive or select a subset of redundant features.

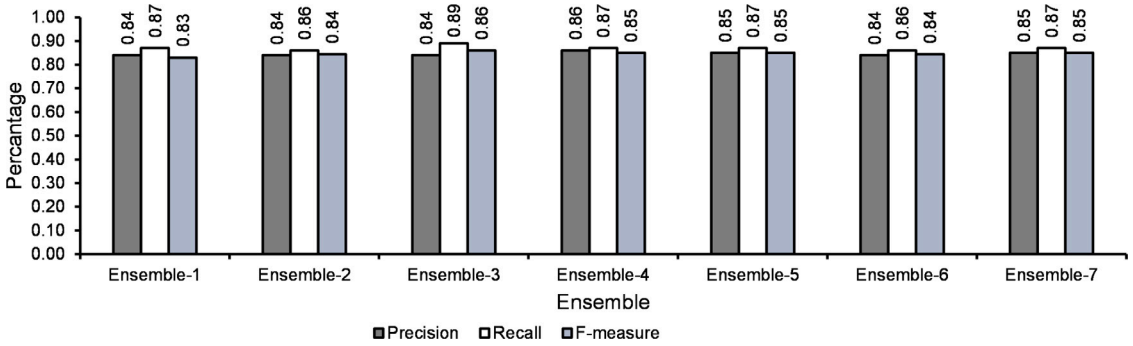


Fig. 4. Evaluation results of ranked data using ensemble learning.

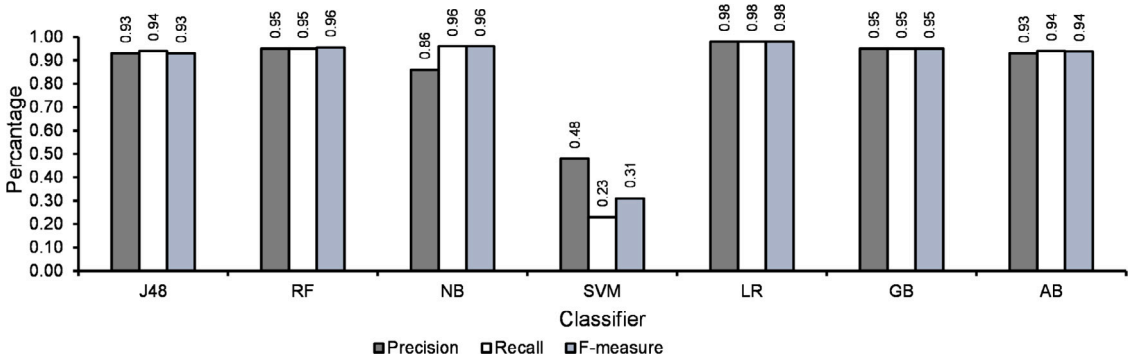


Fig. 5. Evaluation results of ranked data (PCA) Using a single machine learning algorithm.

The study performed feature selection using Principal Component Analysis (PCA). PCA is a dimensionality reduction algorithm that allows us to identify correlations and patterns in the dataset [25]. Thus, the dataset can be converted to a low-dimensional dataset by removing these correlations without losing important information. PCA is a mathematical procedure that converts multiple correlated variables into a few uncorrelated variables known as principal components. This small subset of uncorrelated variables is much easier to work with to identify and use in analysis than the large set of correlated features.

Figs. 5 and 6 describe the performance results of single classifiers and ensembles on selected static and dynamic data of PCA. Fig. 5 illustrates that the performance of LR is highest with 0.993 precision, recall, and F-measure.

SVM performance remained lowest at 0.481 precision, 0.231 recall, and 0.312 F-measure, which is not as expected. All other individual classifiers except SVM performed significantly well on the selected hybrid data. Fig. 6 shows that the ensemble with base classifiers *"SVM with SMO + logistic regression + simple logistic regression + AdaBoostM1 with SVM base + Adaboosting"* outperformed all other ensembles on selected hybrid data by PCA, achieving 0.989 precision, recall, and F-measures. The performance of the other ensembles is also significant. From Figs. 5 and 6, we can see that both the individual classifiers and the ensemble classifiers performed much better on the selected data by PCA than on the selected data by information retrieval. Removing redundant features increased classification accuracy and reduced computational costs.

4.4. Evaluation without feature selection experiments

Fig. 7 shows the performance of all individual classifiers on hybrid data that include both static and dynamic analysis features (using a single classifier instead of an ensemble). The results show that the performance of Ada Boost (AB) is significantly high among all other single classifiers. All single algorithms performed well for the hybrid data provided, except for SVM, whose performance is consistently the lowest across all experiments.

Fig. 7 shows the performance of our proposed approach, a hybrid ensemble analysis explained in Section 3. It includes two different ensemble machine learning models for static and dynamic feature datasets. The results of all different ensembles are shown in Fig. 7. They show that the ensemble with *SVM with SMO + Logistic regression + Simple Logistic regression + AdaBoostM1 with SVM base and Ada Boosting* outperforms all other ensemble and individual machine learning algorithms by achieving 0.99 precision, recall, and F values. The precision of the ensemble using SVM with SMO + Logistic regression + AdaBoostM1 with SVM base member's algorithms and the ensemble using SVM with SMO + Logistic regression + Random forest-based algorithms is the second-highest among all other ensembles with 0.997 precision, recall and F-measure.

It is clear from Fig. 7 that the highest performance is obtained when classification is performed using an ensemble with base classifiers:

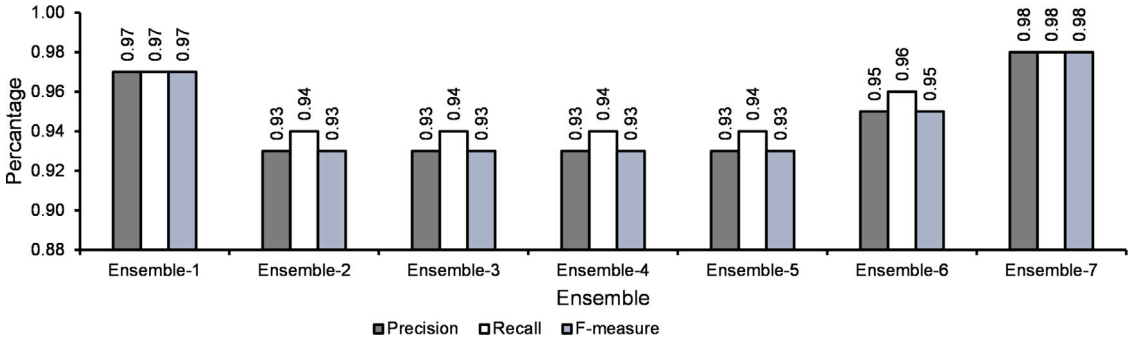


Fig. 6. Evaluation results of ranked data (PCA) using Ensemble learning.

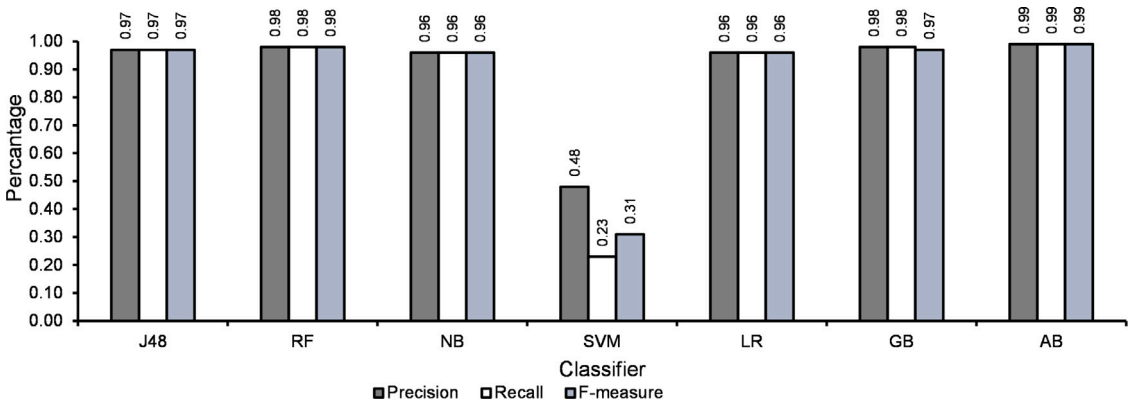


Fig. 7. Evaluation results of without feature selection using a single machine learning algorithm.

1. SVM with SMO, Logistic regression, Simple Logistic regression, AdaBoost with SVM base and Adaboosting;
2. SVM with SMO, Logistic regression, AdaBoost with SVM base;
3. SVM with SMO, Logistic regression, and Random forest.

The first ensemble achieved the highest performance in classification over a given hybrid dataset, improved by only 0.3% over the other two ensembles: *SVM with SMO + Logistic regression + AdaBoostM1 with SVM base and SVM with SMO + Logistic regression + Random forest.* This shows that these ensembles provide similar results in Android ransomware classification and detection.

4.5. Adversarial evasion attacks experimentation

To validate the resilience of the proposed hybrid distinct ensemble model against adversarial evasion attacks, we tried the ensemble model employing the fabricated inputs. These fabricated inputs (to mimic an adversarial evasion attack) are generated by minor changes in known Android ransomware feature vectors. We evaluated the performance of the proposed model in mitigating circumvention attacks by making 1-bit, 10-bit, and 20-bit changes to the input feature vectors of the known ransomware. Most of the existing machine learning-based Android ransomware detection techniques are vulnerable to adversarial evasion attacks. The bit change in the input feature vector causes the underlying classification model to be bypassed. To test the proposed model against such a change in the input feature vector, we randomly selected 100 feature vectors of known ransomware. We changed the permissions of each vector by one bit. This is because permissions are the most vulnerable feature to encryption or renaming. We tested our proposed hybrid ensemble analyzer with non-ransomware and the fabricated feature vectors using different ensemble models. The test results of the different ensemble models ML in terms of Precision, Recall and F-Measure are shown in Table 5. It can be seen that each ensemble generated 0.98% Precision, Recall, and F-Measure values for one-bit data and tended to misclassify as the number of input models increased. To further test the model's resilience, we fabricated the data for randomly selected 100 feature vectors of Android ransomware. In each feature vector, 10 bits related to permissions are changed. Again, the hybrid unique ensemble model is tested on these modified feature vectors.

5. Discussion

From Table 6, it can be seen that the ensemble with the member classifier achieved 0.97, 0.98, and 0.99 precision, recall, and F-measure values for 1, 10, and 20 bit fabricated data, respectively, and these values are the lowest among all. All other models in

Table 5
Evaluation of ensemble learning algorithm without feature selection.

Classifiers	Precision	Recall	F-measure
Ensemble-1	0.97	0.97	0.97
Ensemble-2	0.98	0.99	0.98
Ensemble-3	0.97	0.97	0.97
Ensemble-4	0.98	0.99	0.98
Ensemble-5	0.98	0.99	0.98
Ensemble-6	0.99	0.98	0.98
Ensemble-7	0.99	0.98	**0.99**

Table 6
Accuracy of model against multiple fabricated inputs.

Input fabrication	Precision	Recall	Fmeasure
1-bit fabricated	0.98	0.99	**0.98**
10-bit fabricated	0.96	0.96	0.96
20-bit fabricated	0.94	0.93	0.94
30-bit fabricated	0.92	0.90	0.92
40-bit fabricated	0.90	0.88	0.90

the ML ensemble achieved 0.99% precision, recall, and F-measures. The accuracy of the different ensembles for 10-bit fabricated data can be seen in Table 6, which shows the same trend regarding the performance of the different ensembles.

After evaluating the model for 1-bit and 10-bit fabricated inputs and achieving high accuracy for all ensembles except one, we test the model with 20-bit fabricated input data. The fabricated input feature vectors are determined using the ransomware feature vector that has changed twenty random bits (related to the permissions). The bit-alteration in the feature has changed almost the entire permissions aspect. Based on the extracted permission data, it was found that hardly any ransomware application can require more than twenty individual permissions. We evaluated our proposed hybrid unique ensemble analysis model using these fabricated feature vectors. Table 6 shows the obtained test results for different ensembles. From the results of the 20-bit fabrication test, it is clear that this considerable fabrication of permissions does not affect the performance of the hybrid distinct ensemble model, except for the one ensemble whose precision and F-Measure are 0.9 and 0.9, respectively, for 40-bit fabrication (the lowest of all).

The results of these three experiments (i.e., 1-bit, 10-bit, 30-bit, and 40-bit altered data) demonstrate that the proposed hybrid ensemble analysis approach can mitigate adversarial evasion attacks. The proposed approach can detect Android ransomware and spoofed patterns with high accuracy. To the best of our knowledge, our proposed model is the first research work that focuses on mitigating adversarial evasion attacks through ensemble learning. It analyzes the behaviour of Android ransomware samples to detect their malicious nature, which is different from other Android malware. No recent study shares permissions, text, network-based features, system call logs, CPU usage, and memory usage and uses ensemble learning to distinguish Android ransomware from other Android malware. Moreover, our results validate our conclusion by excellently training the ensemble analyzer to classify Android ransomware and mitigate adversarial evasion attacks.

6. Conclusion and future work

In this paper, a machine learning based ensemble approach has been presented that considers various application features to detect ransomware. The static and dynamic ensemble learners consist of an odd number of classifiers such as C4.5, Random Forest, JRip, Logistic Regression, SVM, and AdaBoost. The ensemble model is trained offline and tested online to analyze the dynamic behavior. The meta-classifier based on majority voting made the final prediction. The study has shown that permissions and system call logs are the two most important features for detecting and classifying Android ransomware and non-ransomware.

We also investigate the ability of the proposed model to mitigate adversarial evasion attacks by testing it with fake inputs in the experiments. The results support our decisions regarding training ensemble analyzers for both ransomware detection and mitigation of evasion attacks, as we obtain good results. The proposed distinct ensemble analyzer mechanism shows promising results by achieving high precision, recall, and F-measure in Android ransomware detection. The proposed model proves to be a resilient model against adversarial evasion attacks by achieving good accuracy on the provided 1-bit, 10-bit, 20-bit, 30-bit and 40-bit fabricated inputs.

The future research problem should address the characteristics of malicious ransomware instances, effective attacks, cost-effective and robust feature extraction, malicious feature estimation, metrics to validate the performance of malicious defense, and designed countermeasures for ransomware defense. We also plan to analyze sequential events and their impact on attacker samples in the future.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] Al-rimy BAS, Maarof MA, Shaid SZM. Ransomware threat success factors, taxonomy, and countermeasures: A survey and research directions. Comput Secur 2018;74:144–66.
[2] Song S, Kim B, Lee S. The effective ransomware prevention technique using process monitoring on android platform. Mob Inf Syst 2016;2016.
[3] Yang T, Yang Y, Qian K, Lo DCT, Qian Y, Tao L. Automated detection and analysis for android ransomware. In: IEEE International Conference on High Performance Computing and Communications. 2015, p. 1338–43.
[4] Ameer M, Murtaza S, Aleem M. A study of android-based ransomware: Discovery, methods, and impacts. J Inform Assurance Secur 2018;13(3).
[5] Zavarsky P, Lindskog D, et al. Experimental analysis of ransomware on windows and android platforms: evolution and characterization. Procedia Comput Sci 2016;94:465–72.
[6] Nieuwenhuizen D. A behavioural-based approach to ransomware detection. Whitepaper. MWR Labs Whitepaper, 2017.
[7] Banin S, Dyrkolbotn GO. Multinomial malware classification via low-level features. Digital Invest 2018;26:S107–17.
[8] Ferrante A, Malek M, Martinelli F, Mercaldo F, Milosevic J. Extinguishing ransomware-a hybrid approach to android ransomware detection. In: International Symposium on Foundations and Practice of Security. Springer; 2017, p. 242–58.
[9] Li D, Li Q. Adversarial deep ensemble: Evasion attacks and defenses for malware detection. IEEE Trans Inf Forensics Secur 2020;15:3886–900.
[10] Biggio B, Roli F. Wild patterns: Ten years after the rise of adversarial machine learning. Pattern Recognit 2018;84:317–31.
[11] Taneja H, Kaur S. An ensemble classification model for fake feedback detection using proposed labeled CloudArmor dataset. Comput Electr Eng 2021;93:107217.
[12] Gharib A, Ghorbani A. Dna-droid: A real-time android ransomware detection framework. In: International Conference on Network and System Security. Springer; 2017, p. 184–98.
[13] Alzahrani A, Alshehri A, Alshahrani H, Alharthi R, Fu H, Liu A, Zhu Y. RanDroid: Structural similarity approach for detecting ransomware applications in android platform. In: IEEE International Conference on Electro/Information Technology. IEEE; 2019, p. 0892–7.
[14] Chen J, Wang C, Zhao Z, Chen K, Du R, Ahn GJ. Uncovering the face of android ransomware: Characterization and real-time detection. IEEE Trans Inf Forensics Secur 2017;13(5):1286–300.
[15] Mercaldo F, Nardone V, Santone A, Visaggio CA. Ransomware steals your phone. formal methods rescue it. In: International Conference on Formal Techniques for Distributed Objects, Components, and Systems. Springer; 2016, p. 212–21.
[16] Wang H, Li M, Yue X. IncLSTM: Incremental ensemble LSTM model towards time series data. Comput Electr Eng 2021;92:107156.
[17] Mathur A, Podila LM, Kulkarni K, Niyaz Q, Javaid AY. NATICUSdroid: A malware detection framework for android using native and custom permissions. J Inform Secur Appl 2021;58:102696.
[18] Mahindru A, Sangal A. PerbDroid: Effective malware detection model developed using machine. Journey Towards Bio-Inspired Techn Softw Eng 2020;185:103.
[19] Aminordin A, Ma F, Yusof R. Android malware classification base on application category using static code analysis. J Theor Appl Inform Technol 2018;96(20).
[20] Pektaş A, Acarman T. Ensemble machine learning approach for android malware classification using hybrid features. In: International Conference on Computer Recognition Systems. Springer; 2017, p. 191–200.
[21] Wang W, Li Y, Wang X, Liu J, Zhang X. Detecting android malicious apps and categorizing benign apps with ensemble of classifiers. Future Gener Comput Syst 2018;78:987–94.
[22] Ahmed U, Lin JCW, Srivastava G, Aleem M. A load balance multi-scheduling model for OpenCL kernel tasks in an integrated cluster. Soft Comput 2021;25(1):407–20.
[23] Narudin FA, Feizollah A, Anuar NB, Gani A. Evaluation of machine learning classifiers for mobile malware detection. Soft Comput 2016;20(1):343–57.
[24] Veeralakshmi V, Ramyachitra D. Ripple down rule learner (ridor) classifier for iris dataset. Issues 2015;1(1):79–85.
[25] Aurangzeb S. A machine learning based hybrid approach to classify and detect windows ransomware. 2018.

Usman Ahmed is currently a Ph.D. Research Fellow in Western Norway University of Applied Sciences, Bergen, Norway. His research interests include ML/DL, optimization and data mining.

Jerry Chun-Wei Lin is currently a Full Professor in Western Norway University of Applied Sciences, Bergen, Norway. His research interests include data analytics, ML/DL, NLP, security and privacy, optimization and soft computing.

Gautam Srivastava is currently an Associate Professor in Brandon University, Brandon, Canada. His research interests include cryptography, security and privacy, blockchain technology, data mining, and Internet of Things.