electronics

MDPI

Article

Ransomware Detection System for Android Applications

Samah Alsoghyer [1] and Iman Almomani [2,3,*]

[1] King Abdulaziz City for Science and Technology, Riyadh 11442, Saudi Arabia
[2] Department of Computer of Science, Prince Sultan University, Riyadh 11586, Saudi Arabia
[3] Department of Computer Science, The University of Jordan, Amman 11942, Jordan
* Correspondence: imomani@psu.edu.sa

Received: 13 June 2019; Accepted: 1 August 2019; Published: 5 August 2019

Abstract: Android ransomware is one of the most threatening attacks nowadays. Ransomware in general encrypts or locks the files on the victim's device and requests a payment in order to recover them. The available technologies are not enough as new ransomwares employ a combination of techniques to evade anti-virus detection. Moreover, the literature counts only a few studies that have proposed static and/or dynamic approaches to detect Android ransomware in particular. Additionally, there are plenty of open-source malware datasets; however, the research community is still lacking ransomware datasets. In this paper, the state-of-the-art of Android ransomware detection approaches were investigated. A deep comparative analysis was conducted which shed the key differences among the existing solutions. An application programming interface (API)-based ransomware detection system (API-RDS) was proposed to provide a static analysis paradigm for detecting Android ransomware apps. API-RDS focuses on examining API packages' calls as leading indicator of ransomware activity to discriminate ransomware with high accuracy before it harms the user's device. API packages' calls of both benign and ransomware apps were thoroughly analyzed and compared. Significant API packages with corresponding methods were identified. The experimental results show that API-RDS outperformed other recent related approaches. API-RDS achieved 97% accuracy while reducing the complexity of the classification model by 26% due to features reduction. Moreover, this research designed a proactive mechanism based on a high quality unique ransomware dataset without duplicated samples. 2959 ransomware samples were collected, tested and reduced by almost 83% due to samples duplication. This research also contributes to constructing an up-to-date, unique dataset that covers the majority of existing Android ransomware families and recent clean apps that could be used as a labeled reference for research community.

Keywords: Android; malware detection; ransomware; static analysis; dataset; classification; machine learning

1. Introduction

Computers and electronic devices are vulnerable to viruses and all kinds of attacks. In early days of computers, users used to suffer from different malicious attacks like viruses, spywares, trojan horses, worms, etc. But the first ransomware documented in 1989 was a new variant of trojan called AIDS (Aids Info Disk) Trojan. That trojan hid the directories and encrypted the names of the files. Then, it displayed a notification to "renew the license" of a fake software and required a payment to unlock it [1]. It is important to note, however, that even if the victim pays the requested ransom, it is not guaranteed that the captive data will be reachable again.

A pronounced trend in recent years has been shifted towards ransomware [2,3]. In 2016, due to a vulnerability in the Windows operating system, the ransomware WannaCry affected more than

150 countries and an estimated 300,000 people worldwide over a weekend [4,5]. The estimates for the potential costs from this hack was $4 billion [6]. Furthermore, Verizon's 2017 data breach investigations report 2017 [7] announced that 72% of all healthcare malware attacks in 2017 were ransomware.

It is known that ransomware mostly targets Windows computers but, as stated by SophosLabs 2018 Malware Forecast [8], this year witnessed an amount of crypto-attacks on different devices and operating systems including Android. According to the same report, Android ransomware is expected to continue to increase and dominate as the primary type of malware on Android platform in the coming year. Also, Android ransomware is especially severe because private information and photos are kept on Android mobiles. Android noticeably continues to increase its sizable lead over iOS and other operating systems in the world [9,10] as it occupied 76.61% of the market share in 2018 [11]. The share of the Android platform presented in Table 1 shows that Android dominated the market for the three recent years. As more users shift to Android devices, cybercriminals are also turning to Android to inflate their gain. This change makes Android the most growing targeted mobile platform [12] in the coming years.

Table 1. Mobile operating system market share worldwide.

Operating System	2016	2017	2018
Android	71.97%	73.54%	76.61%
iOS	18.89%	19.91%	20.66%
Other	9.14%	6.55%	2.73%

Given the critical expansion of ransomware attacks, it is imperative to develop a detection technique against them. Therefore, anti-viruses are used to shield the devices against ransomwares. In general, most of the anti-viruses use signature-based approaches to detect and defend devices against ransomware which is not sufficient and limited to only known malicious apps [13]. The signature based approaches must be complemented with more complex methods that provide detection of unknown malicious apps.

Furthermore, there is no fully trusted secure platform to download totally clean apps including Google play store as many apps passed the Google play checks [14–16]. What makes it worse, even if organizations pay the ransom, there is a big chance they would not have their data back. Like in an experienced ransomware incident nearly one in three of the organizations who paid the ransom did not recover their files [17]. Also, by paying the ransoms, the organizations could be marked by the attackers as "easy target," increasing their chances of being attacked again in the future [18].

According to the way of discrimination, studies can be categorized into two kinds: static and dynamic analysis. Also, machine learning approaches have been utilized in many works to detect malware and ransomware in more sophisticated mechanisms. Works presented in [19–21] used static and/or dynamic analysis for detecting malware while other research [22–24] focused on ransomware detection.

The available technologies defending ransomware are not enough as new ransomwares employ a combination of techniques to evade anti-virus detection. Also, the literature counts only a few studies that proposed methods to detect Android ransomware in particular [25]. These research works have approached different static and dynamic methods to detect ransomwares [23,24,26,27].

There are plenty of open-source malware datasets, however, research community is still lacking of ransomware datasets. Unfortunately, the available approaches are based on ransomware datasets with replicated samples [23,24,26], which may affect the detection results and distort their accuracy. As a result building a unique ransomware dataset without duplicates samples is a demand to be a reference for research community.

Static analysis is an effective mechanism in any Android malware or ransomware detection system [28] and API calls feature is a key static metric that is utilized to identify malicious behaviors [29–31]. Therefore, this paper provides a deep analysis of API calls to investigate the extent

of their influence on the accuracy of the detection process. In this study, we propose a feature-based system called API-based ransomware detection system (API-RDS) to provide a static analysis paradigm for detecting Android ransomware apps. Current state of knowledge was inspected to explore different approaches of ransomware detection. The API-RDS enhanced the available techniques that verify the presence of ransomware based on API packages used. Furthermore, the proposed API-RDS is designed based on a high quality unique datasets to detect the attack incident with high discriminative power for the application before it is installed and harms the mobile devices. The app under test will be classified as a benign or a malicious app, resulting a prevention of cybercriminals from encrypting/locking the assets in Android devices.

As part of the development of API-RDS, all released API-calls were traced in recent published Android apps including benign and ransomware apps. Their frequencies were reported and then analysed to provide the researchers and developers with the list of significant API calls that need to be considered in their detection systems. Additionally, API packages associated to benign or ransomware apps were analysed to highlight methods that are related to encryption and locking attacks. We elaborated specific API features of Android ransomware in details such as android.app.admin API and javax.crypto packages. Then, the approved API-calls list was tested and evaluated by applying data mining techniques after constructing unique datasets of API-calls for both clean and ransomware Android apps. API-RDS reduced the features set to improve the performance of machine learning process by eliminating weakly relevant features. The resulted prediction model showed high detection rate of ransomware apps that reached 99.4% by considering only the key API-calls that were selected in this research. The resulted prediction model was then integrated to the API-RDS components to provide the detection service. API-RDS was evaluated and its performance outperformed existing API-based approaches in terms of accuracy and complexity.

API-RDS detection service could be provided as a stand-alone service like mobile app or website. Also, API-RDS could be injected with other static/dynamic detection services to increase even the service efficiency in terms of accuracy and complexity as can be elaborated in Figure 1. Additionally, the resulted datasets including apps' samples and API-calls datasets for both benign and ransomware apps can be shared and utilized by researchers and developers.

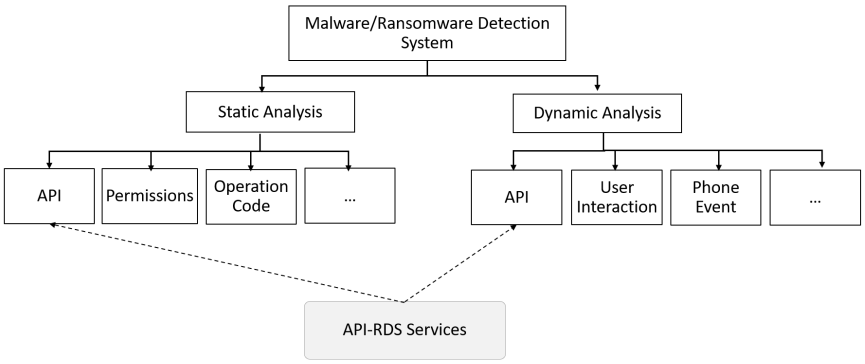


Figure 1. API-based ransomware detection system (API-RDS) services.

In summary, the main contributions of this paper are as follows:

- Propose API-based ransomware detection system (API-RDS) that distinguishes ransomware apps from benign apps before running the app and prevent the malicious executable from damaging the device.
- Provide high accuracy of ransomware detection.
- Provide efficient approach since it depends on static analysis which does not need an emulator or sandbox. Consequently, saving the cost in the deployment environment.
- Recognize the important API packages that highly influence the detection of Android ransomware.
- Create unique, labeled ransomware and benign datasets without repetitive samples.

- Reduce the complexity of the machine learning model in comparison with other state-of-the-art research work.

This paper is further outlined as follows: Section 2 is a brief review of Android apps and malicious applications analysis. Section 3 presents research publications that are related to Android ransomware detection. After that, Section 4 elaborates the methodology of API-RDS. Data collection is described in Section 5. Section 6 reveals the proposed API-based ransomware detection system (API-RDS). The evaluation of API-RDS is presented in Section 7 and Section 8 concludes the paper.

2. Background

Before we describe the ransomware framework and detection, it is essential to understand Android app and how it works. Android applications are written in java programming language and distributed as ".apk" files. APK file is a compressed file (ZIP file) that includes the following:

- AndroidManifest.xml file: defines the capabilities of the application and informs the Operating System about the other application's components. All permissions are defined in this file like accessing contacts and Bluetooth.
- Dalvik executable or classes.dex file: all java classes and methods in the application code are repacked into one single file (classes.dex).
- Several of .xml files: define the user interface of the application.
- Resources: include all external resources associated with the application (e.g., images).

2.1. API Calls

API is an acronym that stands for application programming interface. APIs provide different methods and interfaces that allow software components to communicate with each other. These APIs are also used to access the key features and data within Android devices. In general, API framework comprises of a set of API packages that contains specific classes and methods. By looking into API components calls, we may explore the behavior of an app and report its capabilities. However, in many cases, the API calls used by security attackers are hidden using cryptography, reflection or dynamic code techniques. Consequently, increasing the difficulty of analyzing the app.

2.2. Ransomware

Ransomware is a type of malware that keeps the data encrypted or locked until ransom is paid to the attacker. Usually, the cybercriminals request payment to be paid in digital currency such as bitcoins to avoid getting caught. In general, ransomwares become increasingly noticed in the last of the year 2000. Initially, the vast majority of ransomware victims were Windows desktop users. After a while, ransomware emerged to different platforms, including Android, iOS and other operating systems. Years ago, the market was dominated by misleading applications where many ransomwares masquerade. Ransomware can lurk not only in un-trusted source downloads but it can also hide in Google Play apps and can even spread through exploit kits using yet-unknown vulnerabilities. Moreover, many of ransomwares were designed to pose as anti-virus software.

There are two main types of ransomware; crypto ransomware and lock screen ransomware. The crypto ransomware encrypts the user's data and files so decrypting them requires the key used to encrypt them. While locker ransomware prevents users from accessing their data by locking the device's screen.

Ransomware attacks common scenario starts with the user downloading a fake app from Google Play store or any third party marketplace. It can also arrive as a payload either dropped or downloaded by other malware. Generally, the ransomware attack contains three main phases: gain execution, block access and the last phase is victim notification of ransom message (see Figure 2).

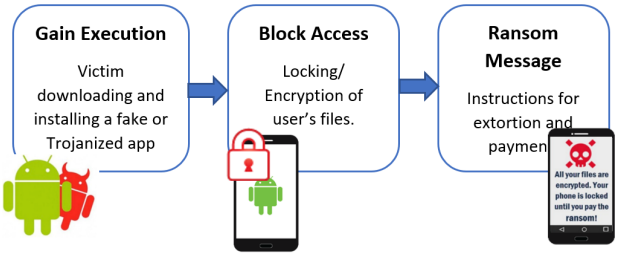


Figure 2. Ransomware phases in general.

Over the past years, ransomware attackers have invented many techniques to their malicious codes to evade detection by anti-viruses and security tools. However, ransomwares have some common characteristics that may help detection tools to expose them. In References [23,32], the authors identified a set of ransomware malicious behavior characteristics in Android environment that includes:

- Acquire admin privileges
- Detect running anti-viruses and block them
- Encrypt user's files on the device
- Steal contacts
- Initiate the camera of the device and take photos
- Lock or unlock the device
- Mute the ringtone and notification sounds
- Display threatening text messages

2.3. Malicious Application Analysis

Most of the anti-viruses use signature-based approaches to detect and defend devices against ransomware which is not sufficient for many reasons [24]. The first reason is that the attack should be captured before and already exists in the anti-viruses databases. Also, the advanced approaches of the new ransomwares such as obfuscation, dynamic code loading and self-altering are potent at avoiding anti-virus detection [33]. Even simple string renaming in the ransomware source code may prevent the anti-virus from identifying the ransomware. Moreover, signature-based detection is inefficient in terms of battery power consumption which is a scarce resource in mobile devices since it needs a comparison of each signature with all signatures in the database.

In research community, there are two standard approaches for the analysis of malicious applications; static and dynamic. Static analysis works on the principle of decompiling the application APK and inspecting the code and its related metadata. It is performed in non-runtime environment seeking for different features extracted from the manifest file and the code. These features include code flaws, signatures, permissions, API calls, and so forth. Static analysis feature detection overcomes the signature-based anti-malwares because it can detect unknown and new malicious samples. Another advantage is that static approach had reduced the cost with reasonable performance compared to dynamic approach [34]. Also, it inspects the content of the application code thus, can achieve full code coverage [35]. A lot of research [19,24,36] admits that static analysis alone is ineffective in detecting and predicting the maliciousness of an application behavior. Dynamic analysis, on the other hand, performed while an application is running on real or virtual machine to trigger the behavior of the app. Dynamic detection analysis technique can be used along with static analysis in order to capture the actual application behavior [37]. Also, the dynamic approach utilized because the static analysis is not capable of analyzing applications where the code is dynamically loaded at run time [24]. Besides, static analysis detection could be evaded by basic obfuscation techniques. Dynamic approach captures real time behaviors such as user interaction, incoming calls and memory dumps.

2.4. Machine Learning Algorithms

Many of the machine learning algorithms could be used in the context of classifying Android applications. The algorithms used in this research are discussed below [38,39]. One of the most popular classification algorithms is Random Forest. In Random Forest, the set of predictor is set randomly and root-mean-square ξ is calculated in Equation (1):

$$\xi = (O_i - \Gamma) \tag{1}$$

where O_i is observation and Γ are tree responses. The prediction ρ_ϱ for the random forest is defined as seen in Equation (2).

$$\rho_\varrho = \frac{1}{k}\Sigma_{(j=1)}^{k}\Gamma_j \tag{2}$$

where ρ_ϱ is predication based on random forest and Γ_j is jth tree-response which are randomly distributed samples; while k is a total number of runs. Another algorithm is Decision Tree (J48) which works on Gain, Entropy and pruning. Pruning Entropy E and Information Gain G are already defined in the decision tree, while pruning is calculated through finding the error rate e. If error rate e is greater than parent then pruning is done; else splitting is done. So, e will be calculated based on error rates, E and the total number of samples (N) as shown in Equation (3).

$$e = \frac{f + \frac{\xi^2}{2N} + \sqrt[\xi]{\frac{f}{N} + \frac{f^2}{N} + \frac{\xi^2}{4N^2}}}{1 + \frac{\xi^2}{N}} \tag{3}$$

whereas $f = E/N$, while ξ will be calculated through confidence level as shown in Equation (4).

$$\xi = \Phi^{-1}(c) \tag{4}$$

where c is referred as the confidence level.

The sequential minimal optimization algorithm (SMO) is a fast simple method for training support vector machine (SVM). SVM works on hyperplane represented using Bias and weighted vector. Distances on vector are calculated as seen in Equation (5).

$$d = \frac{\mid \vartheta_0 + \vartheta^T N \mid}{\parallel \vartheta \parallel} \tag{5}$$

where N represents training examples while ϑ is known as the weighted vector. ϑ_0 is referred as bias. ϑ^T is a training sample weighted vector to be taken from N.

Another classifier is Naive Bayes classifier which selects the classification Θ_t that is most likely for the features $f_1, f_2, f_3, ..., f_n$ as shown in Equation (6).

$$\Theta_t = argmax_{\Theta_j \in \Theta} P(\Theta_j) \prod P(f_i \mid \Theta_j) \tag{6}$$

where $P(f_i \mid \Theta_j)$ is an estimate using κ estimates as can be seen in Equation (7).

$$P(f_i \mid \Theta_j) = \frac{e_s + \kappa\rho}{e + \kappa} \tag{7}$$

where as

- κ is the total sample size
- e_s the total sample that have $f = f_i$ and $\Theta = \Theta_j$
- e is total number of sample where $\Theta = \Theta_j$
- ρ is priori estimator for $P(f_i \mid \Theta_j)$

3. Related Work

There is a lot of research on proposing solutions for malicious behavior detection. Some solutions adopt static analysis of malware and others resort dynamic analysis. On the other hand, some solutions decided to utilize both static and dynamic analysis to produce high detection rate of malicious incidents.

Years ago, different static approaches were proposed such as TaintDroid [40], DroidRanger [41] and RiskRanker [42] to detect malware behaviors. But, most of them relied on manually crafted detection patterns that were not able to detect new malware and caused high performance cost [43].

The authors of Reference [21] proposed DREBIN, which is considered one of the first approaches that provide malicious code detection directly on Android mobile devices. The authors used static-based machine learning system to discriminate malware from trusted applications. Linear SVM was considered for classification. Some indicator features extracted from the applications' APKs including IP addresses, sensitive API calls, permissions, and so forth. DREBIN, however, could not detect run-time loaded and complicated malicious applications [43].

Yang et al. [44] developed a prototype called AppContext that was able to detect malicious apps based on static analysis. The authors collected 633 benign apps from Google Play and 202 malware apps from various malware datasets. AppContext managed to classify the applications using machine learning in addition to the contexts that trigger security-sensitive behaviors. On the other hand, AppContext, was weak against dynamic code loading. Also, labeling each security-sensitive behavior was effort and time consuming [45].

Tam et al. [13] on the other hand, introduced CopperDroid which applied both dynamic analysis and machine learning to detect malware. Real time system calls performed by the application were captured by CopperDroid to differentiate between ransomware, malware and trusted apps. CopperDroid works by running Android application in sandbox and recording all system calls, especially IPC (inter-process communications) and RPC (remote procedure call) interactions to understand the app and detect any maliciousness behavior. However, some types of malware are smart enough to recognize the virtual environment and act as benign app which causes false positives.

Akhuseyinoglu and Akhuseyinoglu [34] proposed AntiWare which is a features-based malware detection system. The features extracted by Antiware are permissions in addition to Google Market data including developer name, download time and users' ratings. The Machine learning methods used the extracted features to test automatically Android apps and then communicate the results to the user. The main drawback of Antiware is its full reliance on Google market data and the requested permissions. The market data is not reliable as many applications are developed by different vendors every second. Moreover, permissions by its own are not efficient enough to detect malicious apps.

Recent research [19] applied dynamic classification for Android apps based on the frequency of system calls. A syscall-capture system was built to capture and analyze the behavior of system calls made by each app during the run time. The authors analyzed 100 application samples; half of them were malicious and the rest were benign. They aggregated the outputs for all applications into one dataset. Then, J48 Decision Tree and Random Forest classification algorithms were applied and they achieved accuracy level that reached 85% and 88% respectively.

Also Wang et al. [46] proposed a prototype called Droid-AntiRM that uses dynamic analysis to detect malicious apps. Droid-AntiRM identifies malware codes that utilize anti-analysis techniques. The prototype was able to identify the condition statements in Andoird applications that could trigger the malicious acts of malware. However, Droid-AntiRM could not deal with dynamic code loading, encryption, or other complicated techniques.

All of the above-mentioned works focused on malware detection. Unfortunately, there were very few studies of ransomware apps. The HelDroid tool [23] was developed in 2014 and was the first method that initiated ransomware detection in the Android platform. The goal of this tool was to analyze Android ransomware while paying attention to multiple typical behaviors of ransomware. HelDroid includes NLP (Natural Language Processing)-based text classifier features, a lightweight

smali emulation technique for detecting locking scheme and the application of taint tracking to detect file-encrypting flows. Besides, this tool works on static analysis of the ransomware using techniques like monitoring encryption calls and intimidate ransom text. They were able to identify rightly 375 ransomware on a dataset formed by 443 samples.

The main shortcomings of HelDroid are the high dependency on text classifier which is not always available and it cannot be applied on all languages especially the ones that do not have specific phase structure like Chinese, Korean and Japanese. Moreover, it can be avoided by applying encryption or any other code obfuscation techniques [47]. Finally, its detection capability depends on the training dataset [32].

Yang et al. [22] presented another detection approach and introduced some static and dynamic features that could be utilized in malware analysis in general. Features such as permissions, API calls flow and APK Structure were suggested to be used in static analysis. Whereas, access to sensitive data or paths, access to HTTP server, charge user without notification and bypass permissions features could be part of the dynamic analysis. The authors provided the design of their approach but without any implementation or testing. The authors analyzed one ransomware sample and listed the steps of APK analysis as a concept but without implementation which was not enough to prove the efficiency of the proposed design.

Zheng et al. [27] proposed a preventive countermeasure ransomware detection system called GreatEatlon. GreatEatlon is like an extension from HelDroid [23] with a concentration in crypto ransomware. The authors improved the ability of text threatening detector to scan images besides plain text. This approach has the same flaw as HelDroid, it assumes the text/image availability and it is ineffective with languages missing phase structures. For encryption detector, GreatEatlon checks AndroidManifest.xml then meta-data file for dangerous policies. After that, it scans source code which is computationally demanding. Additionally, authors performed forward and backward analyses to discover malicious reflection which was, according to authors, "not enough because in several samples the hard-coded method name was obfuscated". They used different classifier models to evaluate their approach and reported they have better performance than HelDroid.

Song et al. [48] aimed to detect abnormal usage of processor and memory usage as well as I/O rates. They recognized the ransomware behavior by dynamically monitoring processes and specific file directories. Their approach based on two modules: file monitoring and process monitoring. The file monitoring inspects input/output operations on files such as reading, writing and deleting files while the process monitoring checks processor status information for each process. However, their goal is to reduce the damage of ransomware and not to prevent it. Also, this approach is not immune against locker ransomware beside they evaluated their solution on one self-developed sample.

Likewise, Mercaldo et al. [25] suggested an approach for ransomware detection based on formal methods. In general, formal methods are used to automatically verify the correctness of a system with respect to a desired behavior. So, the authors of Reference [25] proposed converting the Java bytecode of the application into calculus for communicating systems (CCS) statements to express behavioral properties of ransomware. They tested a dataset composed of 2477 samples of real-world ransomware and good applications. This approach is manual and requires human analysts to build logic rules and identify ransomware related instructions used for classification, which is considered one of its main issues [49,50].

R-PackDroid [24] is another work that explores the ransomware detection in Android devices. R-PackDroid is a static analysis approach that is able to classify Android applications into ransomware, malware or benign based on the system API packages and using random forest classifier. R-PackDroid outperformed the previous approach (HelDroid) by detecting ransomware regardless of the application's language. Also, it succeeded to flag the applications that could not be recognized as ransomware with high confidence by the VirusTotal (https://www.virustotal.com) service.

Ferrante et al. [49] introduced a hybrid method to extinguish ransomware that combines static and dynamic analysis approaches to resist ransomware. In static analysis, they examined an app during

installation and looked for opcodes occurrences. While in dynamic approach, the behavior of the app is observed taken into account CPU usage, memory usage, network usage and system calls. They evaluated their method using 3058 mobile applications, 672 applications were containing ransomware.

Gharib et al. [51] proposed a similar approach for static and dynamic analysis. They experimentally presented a framework called DNA-Droid that relies on static analysis to classify apps into suspicious, malware or trusted. The framework achieved a good detection rate compared to HelDroid [23] and R-PackDroid [24]. Only apps that are classified as suspicious will be then examined by the dynamic analysis to determine if they are ransomware or not. Main disadvantage of DNA-Droid is applying dynamic analysis only to suspicious applications which could result in having malware that bypassed the static analysis due to obfuscation. Also, the dynamic analysis could take up to five minutes to monitor the behavior of a ransomware application.

Similar to Reference [25], Cimitile et al. [52] suggested formal methods for detecting ransomware by converting bytecode to CCS processes. They implemented a tool called Talos that invokes the CCS model checker to inspect a specific ransomware module behavior. Two formal representation for ransomware behavior performed: obtain admin privileges and encryption process instructions.

Chen and others [26] presented real-time detection tool of ransomware based on interaction with the user interface (named RansomProber). They inspected user-intent based on UI (User Interface) widgets of related activities and coordinates matching with finger movements. Besides, they observed encryption behavior based on information entropy to measure the degree of data transformation. They tested their approach on their own collected ransomware dataset and achieved a high accuracy compared with HelDroid and some anti-virus tools. However, this approach concentrates on encryption capabilities of crypto ransomware.

As can be seen, the current methods are investigating applications and looking for different malware and ransomware characteristics that help to distinguish malicious applications. The first work of detecting ransomware was in 2015 which was late compared to the advanced knowledge for inspecting Android malware in general. Unfortunately, ransomware app engages in activity that appears similar to benign applications such as encrypting files and pop up notifications (to ask for the ransom) which could be mistakenly considered not destructive [26,53]. Therefore, it is important to find out characteristics that lead to ransomware detection before the damage is done. These characteristics could use dynamic features such as launching a root exploit, sending background SMS messages, CPU usage and memory usage [22,42,48,49]. User permissions were also used to indicate malicious activities for Android applications [34,41,51]. Other methods track real time system calls [13,19] and user data [49,54].

Static analysis was also used to overcome the dynamic analysis issue that many malicious apps are aware of the emulated surroundings and therefore could decide to not exhibit malicious behaviors. Invoking API methods, for example, lockNow() and onDisable() can also be considered as suspicious behavior [23,27]. Other authors [24] tracked the occurrences of inbuilt API packages and accordingly predicted the maliciousness of application.

Table 2 shows a comparison among state-of-the-art methods in ransomware detection in terms of static or dynamic analysis, detection approach, feature set, machine learning classifier, ransomware dataset, publicly accessibility and year. It can be observed that most of the described approaches used HelDroid ransomware dataset.

Additionally, we checked some ransomware samples in HelDroid dataset and inspected that there are two samples shown in Table 3 have different hashes but exhibit identical behavior and belongs to the same ransomware according to Kaspersky [55]. Therefore, we were enthusiastic to clean the dataset from overlapping samples and verify if that would affect the result of detection when using this dataset.

Table 2. Comparison among state-of-the-art methods in ransomware detection.

Work	Static or Dynamic	Approach	Feature Set	Machine Learning	Ransomware Dataset	Publicly Accessible	Year
Andronio et al. [23]	Static	- Look for threatening messages in text - Analyzes dynamically allocated strings - Examine app ability of locking and encrypting the device	Threatening text, BIND_DEVICE_ADMIN permission, API methods (lockNow(),onKeyUp() and onKeyDown()), FLAG_SHOW_WHEN_LOCKED and trace of encryption process	Natural language processing (NLP)	Own collected ransomware samples (HelDroid)	Yes	2015
Yang et al. [22]	Static and dynamic	Suggest some malware and ransomware indicators	Permissions, API methods invoking flow, access to critical paths, malicious domain access and charges through sms and calls	Not stated	Not stated	No	2015
Zheng et al. [27]	Static	- Look for threatening messages in images - Forward and backward analyses to observe any malicious reflection - Inspect abuses of the device administration API to detect uses of cryptographic APIs	Threatening messages in images, meta-data policies, package name, URLs, file types and their count, number of permissions, activities and services, use of obfuscation, Reachability (operations on SMS) and API methods (Invoke()onEnable() onDisable())	Decision trees (J48), Random forests, Support vector machine (SVM), Stochastic Gradient Descent (SGD), Decision Tables (DT) and rule learners (JRip, FURIA, LAC, RIDOR)	Contagio Mobile dataset And VirusTotal	Yes	2016
Song et al. [48]	Dynamic	Monitor processes and specific file directories	File input/output events, processor status info(processor share, memory usage, I/O count and Storage I/O count for each specific process)	Not stated	One self-developed ransomware sample	No	2016

Table 2. *Cont.*

Work	Static or Dynamic	Approach	Feature Set	Machine Learning	Ransomware Dataset	Publicly Accessible	Year
Mercaldo et al. [25]	Static	Apply formal methods for checking the ransomware behavior.	Ransomware behavior (did not specify)	Not stated	HelDroid and Contagio Mobile dataset	No	2016
Maiorca et al. [24]	Static	Look for API call invoked in the executable code	234 api call in their dataset	Random forest	HelDroid and VirusTotal	Yes	2017
Ferrante et al. [49]	Static and dynamic	Detect obcodes frequencies of some characteristics from the execution logs of an application	obcode occurrences, CPU, memory and network usage and system calls	Decision Trees (J48), Naïve Bayes, and Logistic Regression	HelDroid	No	2017
Gharib et al. [51]	Static and dynamic	- Look for threatening messages in text - Look for specific images/logos - Detect permissions and API calls statically - Detect dynamically API calls and compare it with already defined behavior (DNA) in databas	Threatening text, number of nude images and specific logos and API calls	Random forests, Support vector machine (SVM), Naïve Bayes, AdaBoost (AB) and Deep Neural Networks (DNN)	HelDroid, Contagio Mobile dataset, VirusTotal and Koodous	Yes	2017
Cimitille et al. [52]	Static	Apply formal methods for checking the ransomware behavior.	Obtain admin privileges and encryption process flow	Not stated	HelDroid and Contagio Mobile dataset	No	2017
Chen et al. [26]	Dynamic	Identify user interface differences between benign and ransomware apps with coordinates of the user's finger movements.	User interface and information entropy of files before and after encryption	Not stated	HelDroid and own collected samples	No	2018

Table 3. Same Ransomware Sample with different hashes.

Sample	MD5	SHA256
Sample-1	67bde6039310b4bb9 ccd9fcf2a721a45	4d3de2103f740345aa2041691fde0878d7 d32e9e4985adf6b030d2e679560118
Sample-2	fb14553de1f41e3fc dc8f68fd9eed831	2e1ca3a9f46748e0e4aebdea1afe84f101 5e3e7ce667a91e4cfabd0db8557cbf

4. Methodology

The scope of our research is to study how API packages are utilized by attackers to encrypt files or locking the mobile device. This research examined these API calls after removing identical instances to see if we could find differences between ransomware and clean apps concerning these API packages calls.

The methodology in this paper has four stages; Data collection, Proposed API-based ransomware detection system (API-RDS), Evaluate API-RDS and finally Provide API-RDS Services. Figure 3 shows the overall workflow of the methodology with the corresponding system structure and components. The main functionalities of these components include:

- Inspect current ransomware detection methods
- Collect new Android apps [both benign and Ransomware]
- Check the latest API calls released by Android community
- Scan all API calls in the Android apps and calculate its frequency
- Filter the collected API calls values by removing the duplicates and reducing the features set.
- Analyse the API calls as: never requested /requested/top 30 requested
- Analyse the used and the highly used API-calls for both benign and ransomware apps from security perspectives
- Decide on the list of API-calls that should be listed as features in the new established datasets.
- Construct both benign and ransomware datasets considering the approved features and their values in all apps under study
- Examine the approved API-calls by running data mining techniques to build predictive models using data mining techniques and then check API-calls' impact in detecting ransomware apps and their efficiency in terms of complexity and detection accuracy
- Approve the best model with the best performance to be the predictive model in the API-RDS
- Evaluate the API-RDS and compare its performance with recent related work
- Offer the services of API-RDS including ransomware detection system and the constructed datasets to users, researchers and developers

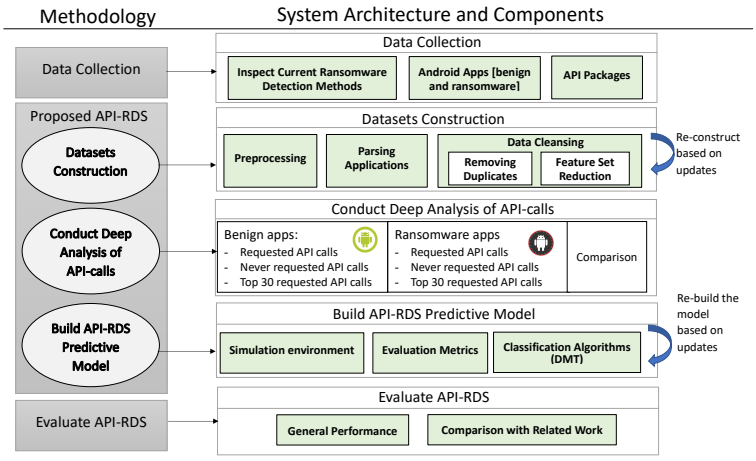


Figure 3. Flow of the methodology.

The following sections explain in details each stage with all its components.

5. Data Collection

Data collection stage involved three steps; inspect current ransomware detection methods, collecting android applications and specify API-packages calls that will be tested.

5.1. Inspect Current Ransomware Detection Methods

Here we inspected the current state-of-the-art approaches including substantive findings, as well as theoretical and methodological contributions to Android malware/ransomware protection approaches. The goal was to shed key differences among them. Particularly, we investigated the ransomware features utilized by these approaches to identify the existence of ransomware behavior. Also, we pointed out common ransomware datasets used in their experiments. This step discussed and summarized in Related Work Section 3.

5.2. Android Applications

Benign and ransomware apps were collected as apk files from different sources. Benign apps were downloaded from Google play store (https://play.Google.com/store) from top downloaded apps (free apps). These apps belong to different categories such as shopping, games and social communication. Google Chrome extension called APK downloader was used to download apps from Google Play store.

Collecting ransomware samples was one of the main parts of our research. We were constrained with the number of ransomware apps available. If we found more ransomware apps, we might end with a larger dataset as adding more benign apps was not an issue. Ransomware samples were collected from four sources: HelDroid project [23], samples provided by Chen et al. (RansomProper project) [26], Virus Total and Koodous (https://koodous.com/apks). We captured samples from HelDroid project that provides hashes for different ransomware apps from different families without providing the samples itself. Thus, we downloaded the ransomware apps by searching their hashes in koodous project which is a collaborative platform for Android malware and ransomware [56]. There are 672 ransomware samples exist in HelDroid project; however, we successfully downloaded 345 samples. The remaining 327 samples were either not available in Koodous or duplicated versions of the same exact hash value (32 samples). These samples collected from December 2014 to June 2015.

The authors of RansomProper project [26] provided us with a newer collection of ransomware samples which includes 2258 samples from 15 different families. These families cover many ransomware features such as locking screen, encrypting files and threatening messages [23,26].

Another source of ransom samples was VirusTotal service with 694 samples submitted to this service between 2017 and 2018. VirusTotal provided us with a collection of malicious Android apps. These apps were not categorized into a specific malware category such as banking malware, spyware or ransomware. To identify only ransomware applications, we determined the samples that were detected by at least five anti-virus engines (AV) with labels containing the name of a ransomware family or the keyword 'ransom'.

The last source of ransom samples was Koodous with 40 ransomware samples downloaded from their website. During apps installation, we made sure that all samples downloaded through Koodous held the tag (tag:ransomware) to ensure these are ransomware samples. Also, these samples were tested in VirusTotal to confirm that they are identified as ransomware by at least five anti-virus engines. The following are the initial datasets:

- Dataset-R: It included 345 ransomware samples from HelDroid, 2258 from RansomProper project, 694 samples from Virus Total and 40 from Koodous.
- Dataset-B: It included 519 samples from Google play store.

Details on ransomware samples in Dataset-R and Dataset-B showed in Tables 4 and 5.

Table 4. Dataset-R.

Source of Hash	Source of Apk File	Number of Samples			
		Collected	After Decompiling	Total	Total after Removing Duplicates
HelDroid project	Koodous	345	304	2959	500 (17% of total)
RansomProper project	RansomProper project	2258	2025		
Virus Total	Virus Total	694	590		
Koodous	Koodous	40	40		

Table 5. Dataset-B.

Source of Apk File	Number of Samples			
	Collected	After Decompiling	Total	Total after removing Duplicates
Google play store	519	500	500	500 (100% of total)

5.3. API-Packages Calls

In this paper, API packages belong to Oreo Android release (API 27) were considered. There are 199 API packages in this release. A description of these APIs is available in Android documentation (https://developer.android.com/reference/packages).

API calls occurrences have been shown to represent the application's ransomware behavior effectively [23,24] and therefore we adopted API packages occurrences as discriminating features to detect ransomware presence to be the input to the data mining experiments. We consider these API packages as the set of characteristic features of ransomware to detect the patterns of ransomware behavior and distinguish it from benign apps as in Reference [24].

6. Proposed API-Based Ransomware Detection System (API-RDS)

To propose the ransomware detection system (API-RDS), we went through three steps: datasets construction, conducting deep analysis of API calls and building API-RDS predictive model.

6.1. Datasets Construction

Dataset-R and Dataset-B samples that were collected in the data collection stage, need to be processed to be adequate for data mining tools. Dataset construction process started by applying reverse engineering to the android applications to access the source code of the apps then parsing the code to obtain API calls occurrences. Finally, the data cleansing phase was implemented to remove duplicate apps and reduce the dataset volume. Figure 4 shows the phases of datasets construction.

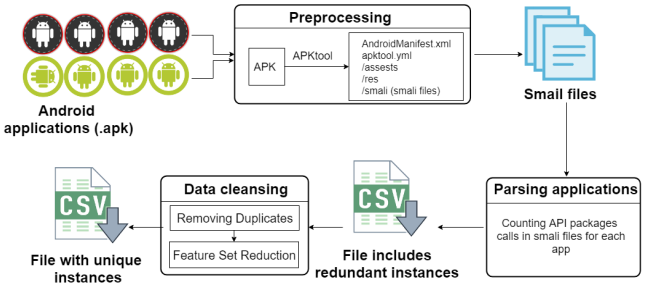


Figure 4. Dataset construction phases.

1. **Preprocessing/reverse engineering:**
   All apps were downloaded as apk files which is a single zipped file that holds all of that application's code (.dex files), resources, assets, certificates and manifest file. So, we need to decompile apk file to be able to discover the behavior of an Android application and reveal its components, structure and methods. Apktool was used to decompile all .apk files. This step is called reverse-engineering where the output of Apktool is a collection of smali code files. Smali code is an understandable form of code in smali language similar to Java that we used Notepad++ to read. Apktool was installed from Github website (https://ibotpeaches.github.io/Apktool/). In this paper reverse engineering and decompilation terms are used interchangeably.
   In fact, many errors found while decompiling apps samples with Apktool, especially for ransomware samples.
   In Dataset-R as in Table 4, 2959 ransomware samples were successfully decompiled among 3337 apps (total of samples from all sources).
   A few benign apps had errors compared to ransomware apps while decompiling using Apktool. In particular 19 apps in Dataset-B experimented decompilation error. Accordingly, all corrupted apps were removed to avoid faulty output during parsing phase that may affect the final results.
2. **Parsing applications/Feature extraction:**
   Scanning or parsing comprises searching for API packages calls in smali files resulted from Apktool. API packages calls being searched are previously identified in the data collection stage. We looked for the exact matching of API packages name strings in each ".smal" file for each application and counted their occurrences. Algorithm 1 presents the pseudo code of the parsing phase. A crawling python script was developed based on this algorithm to scan all applications' smali files and count the occurrences of the set of API packages then save their values in ".csv" file. Frequency feature extraction method was followed to count the occurrences of a string term in a sequence (smali file in our case) [57]. We counted the occurrences of 199 API packages belongs to API 27, so initially there were 199 features in the dataset. Mathematically, we can define a finite set $S = \{x_1, x_2, ..., x_n\}$ to represent API packages frequencies for each application where x_i is the frequency of the ith API package in that application [58].

Algorithm 1 Count occurrences of API packages of one application

Input: appFolder(Decompiled APK application folder)
Output: Feature set (occurrences of each API call in smali files)

```
APIList=['android', 'android/accessibilityservice', 'android/accounts', 'android/animation',
'android/annotation', 'android/app', ...]          ▷ Array of 199 API packages
featureSet[ ]← 0                                   ▷ Array of occurrences of 199 API packages
for each smaliFile in appFolder do                 ▷ Read only .smali files
    x ← 0
    content← openRead("file.smali")
    for each APIpackage in APIList do
        occurrence ← count of APIpackage keyword in content
        featureSet[x]← featureSet[x] + occurrence
        x ← x + 1
    end for
end for
save featureSet[ ] of this app in csv file
```

3. **Data Cleansing:**
   Data cleansing phase includes two stages. One to have a high quality dataset with unique instances and the other is reducing the features set to speed the training and to have more accurate results.

- **Removing duplicates**
  It is essential to have a high-quality dataset, and one significant aspect that may disrupt the data quality is data duplication, which affects the data mining results. After scanning and counting the number of each API package occurrences for feature extraction, we discovered that many samples have the exact same number of API packages calls. This may indicate that these are the same apps but with different hashes meaning; we have multiple instances in the dataset referring to the same Android app (or app with the same functionality). This could be a result of malware polymorphism technique which is used by attackers where they made some changes to the application in order to derive different hash in order to evade detection by signature based anti-viruses [59]. These apps eventually could have the same feature set for API calls and therefore need to be removed from dataset to avoid duplication [60]. One of the main contributions of this work is offering a clean ransomware dataset without duplicate apps. Removing duplicated data is necessary to have accurate and consistent data where irrelevant features have a negative impact on machine learning [61,62]. Therefore, we eliminated the repeated samples from the dataset. There were no identical instances in benign apps, on the contrary, many ransomware samples have duplicates. As a result, the instances in the ransomware dataset were significantly reduced. Specifically, in Dataset-R, we had 2959 ransomware samples resulted after prepossessing phase. However, after removing duplicated instances we only had 500 unique ransomware samples showed in Table 4. Meaning, almost 83% of the ransomware samples were removed due to samples duplication.
- **Feature set reduction**
  Feature set reduction is concerned about irrelevant and redundant features. Mainly, features are classified into strongly relevant, weakly relevant and irrelevant features [63]. Having fewer features reduces the time of training where a large number of features may lead to a higher computational cost [64]. Also reducing feature vectors improves the accuracy of prediction and the performance of machine learning algorithm [65]. As we count the occurrences of API packages calls in each Android application, we found that some API packages have zero occurrence in many applications. Meaning, these API packages were not requested by specific apps. If the number of occurrences of an API package call is equal to zero in all samples, that API package will be deleted from the features set. In our dataset, among 199 APIs in API 27 Android release, we found 26 API packages that had never been called in benign nor in ransomware apps. Such packages are considered irrelevant features and could be removed [66]. Therefore, we removed these packages from the features set. Figure 5 shows these eliminated packages. There are 173 API packages left after removing the irrelevant packages. Figure 6 shows a snapshot from the Dataset-R where there are 173 features which belongs to API 27 beside the app's Identification and classification features.

1. java.time.zone
2. java.time.chrono
3. java.security.acl
4. java.nio.file.spi
5 java.nio.charset.spi
6. dalvik.bytecode
7. android.view.textclassifier
8. android.test.suitebuilder
9. android.test.mock
10. android.service.vr
11. android.service.voice
12. android.service.textservice
13. android.service.restrictions
14. android.service.carrier
15. android.service.autofill
16. android.printservice
17. android.nfc.cardemulation
18. android.net.wifi.hotspot2.pps
19. android.net.wifi.hotspot2.omadm
20. android.net.wifi.hotspot2
21. android.net.wifi.aware
22. android.net.rtp
23. android.icu.math
24. android.icu.lang
25. android.companion
26. android.app.assist

Figure 5. Eliminated application programming interface (API) packages.

App ID	android	android/accessibilityservice	android/accounts	android/animation	android/annotation	android/app	android/app/admin	android/app/backup	android/app/job	...	Classification
00245F3782D8D6CAB719	0	0	0	0	25	17	0	0	0		Ransomware
002DC6186FB99273379D	85	0	1	0	747	9	0	0	0		Ransomware
004D107307A104B4F719	0	0	0	0	34	0	0	0	0		Ransomware
006BE25A70BE5B3ED038	0	29	0	2	50	8	0	0	0		Ransomware
009909D6D84CD7ADE0B	85	0	12	0	1377	9	0	0	0		Ransomware
00FA6FB96CE806D0064B	85	9	12	0	1386	5	0	0	0		Ransomware
010365C324C49058F474	0	0	0	2	12	7	0	0	0		Ransomware
0123A7B5EA28E6B9FFF7	0	0	0	2	11	0	0	0	0		Ransomware
0124FC265E03ACC4C244	0	0	0	5	45	8	0	0	0		Ransomware
01265C3ED0A4FE70AB94	0	0	0	1	10	7	0	0	0		Ransomware
016856E84CFCB17C7F70	0	0	0	1	10	5	0	0	0		Ransomware
01A198393C134932560A	0	29	0	2	50	9	0	0	0		Ransomware
01D00A39D4943CD4196	0	0	0	38	10	0	0	0	0		Ransomware
01E8CEA7DF22B1B3CC56	85	0	1	2	1329	17	0	0	0		Ransomware
0293A12B977EEC003796	0	0	0	2	10	0	0	0	0		Ransomware
033CB80BD374AC624104	0	0	0	38	10	0	0	0	0		Ransomware
035103EB0D9678284A3C	0	0	0	0	14	0	0	0	0		Ransomware
04177C48379AFFBF261A	0	0	0	0	25	17	0	0	0		Ransomware
04376A7EF229515A47D8	0	0	0	5	11	0	0	0	0		Ransomware

Figure 6. Sample from Dataset-R.

The following constructed datasets could be supplied to users, researchers and developers:

1. **Android benign dataset**
   This dataset includes samples of more than 500 benign applications downloaded from different categories in Google play store.
2. **Android ransomware datasets**
   One dataset contains the hashes of all tested 2959 ransomware applications. Also, another dataset can be provided that contains 500 hashes of unique ransomware applications.
3. **API-Packages calls dataset**
   This dataset provide API-packages calls (belong to Android API 27) occurrences in 500 benign and 500 ransomware apps.

6.2. Conduct Deep Analysis of API Calls

After constructing the datasets (Dataset-B and Dataset-R), we analyzed API packages calls to recognize the key packages that are highly differentiating benign and ransomware apps. The analysis conducted on both ransomware and benign datasets which includes 1000 samples in total. First, we separately investigated API packages occurrences in benign and ransomware apps and showed the top desired API packages. Also, we investigated the requested and never requested APIs. Finally, a comparison was made between the API packages occurrences in benign and ransomware in addition to specifying significant distinctions between them.

The dataset was previously prepared to analyze API packages calls and highlight the differences between ransomware and benign apps. This section includes statistics of API packages in ransomware apps and benign separately and then compare different extents of the API packages occurrences between ransomware and benign samples. In the experiments, 500 ransomware and 500 benign Android samples were scanned to get statistics on all API packages belong to API 27 (199 packages).

6.2.1. Benign Applications

There were 26 API packages out of the 199 packages had never been called in clean apps. Therefore, those 26 packages were eliminated from further analysis.

Some of API packages are popular in benign apps while others were rarely imported. Figure 7 shows the percentage of occurrences for 173 API packages (APIs with at least one request) used by benign apps. These figures reveal, generally, there is a high demand for many API packages.

Figure 8 illustrates the top 30 requested API package by all benign apps. As expected, the package java.lang was called in all benign apps since it includes classes that are essential to the design of Java programming language. Classes like `System`, `Math` and `String` are all belong to this fundamental package. Also, reflection is a popular java feature in Android as it allows to get information about classes and components while an app is running and modify them. The reflection API java.lang.reflect includes classes and interfaces that can be used to dynamically load code at runtime [67]. The package java.lang.reflect was called in all benign apps except one. One possible reason for this high percentage

that developers need to target devices running different versions of Android to specify what classes/methods are available before trying to use it [68].

Figure 7. Percentage of occurrences of APIs in benign apps. (**a**) API packages part-1. (**b**) API packages part-2.

Another core java package is java.util which was highly requested by benign apps with 99.6%. This is expected as this package facilitates dealing with arrays, text formatting and other legacy functionalities.

Similar to java.util, the packages android.os and android.app were requested by 99.6% of benign apps; where these packages are primary to provides essential OS services and `Activity`

class, respectively. Other packages such as android.content and android.content.res are related to content and how Android display them are also highly needed by apps with 99.6% of total benign apps.

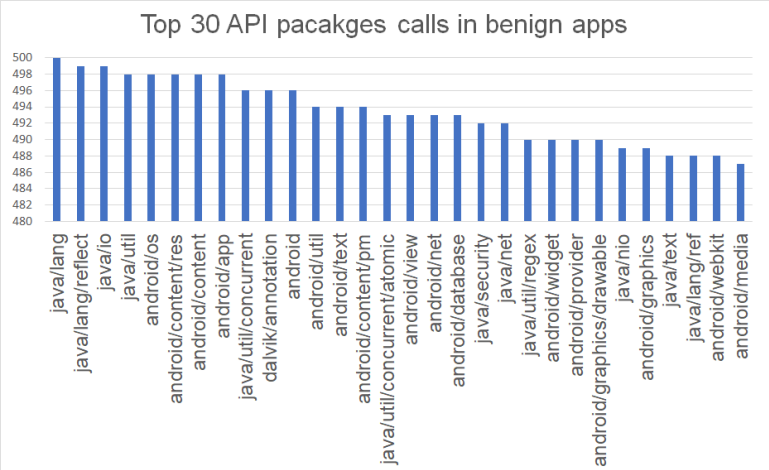


Figure 8. Top 30 API packages in benign apps.

Moreover, java.nio as well as java.io are used to read/write data and were commonly requested too by benign samples. Also, the packages java.util.concurrent and java.util.concurrent.atomic were presented in the majority of benign samples. Both packages are essential for synchronization programming. In particular, java.util.concurrent provides classes to implement multi-threaded applications and java.util.concurrent.atomic used to enable thread-safe programming.

The package java.util.regex that is used to support regular expressions was also highly requested by clean applications with 98% occurrences. Moreover, the packages android.view, android.text, java.text, android.widget, android.graphics and android.graphics.drawable were all highly requested by benign apps with more than 97% of scanned samples. This high percentage could be due to the necessity of these packages to provide classes either to style text or to provide elements to design app's screen. The package android.util was also expected to be in the list too because it involves general features such as string/number conversion methods, date/time methods, base 64 encoders and decoders and XML utilities which are common methods in Android apps. Many apps need to access contact and calendar information; therefore it is required to import android.provider API. This package existed in the top 30 requested APIs as it is called by 98% of benign apps under test.

The database package android.database was also imported in most of the benign apps (98.6%). Most often, apps handle database either internally or externally, in both cases, the apps must request the database package. If an app needs to handle data from a database or requires to load data from a remote server, it is common to use the network connection. Thus, most benign apps as well requested android.net and java.net packages. Confirming the hypothesis, benign apps under test called android.net and java.net packages with 98.6% and 98.4%, respectively.

Another package used for web surfing is android.webkit. This package is primary to display web pages inside Android application, as anticipated, it was used frequently in most of apps with 97.6%. The package android.content.pm was also in the top 30 packages because it includes the classes needed to get information about the application's permissions, activities and signatures which are commonly used by Android apps. An additional popular package that provides fundamental classes for permissions and other Manifest elements is the android package. This package was imported nearly in most benign apps (99.2%). Further, many benign apps contain audios and/or videos which require classes belong to android.media API. The tested benign apps confirmed this hypothesis with 97.4% of them requesting the package android.media. About 98.4% of benign apps requested java.security that could be due to the developers' need to secure their apps. This package includes many built-in features

that reduce security issues like code signing and encryption. Likewise, java.lang.ref was preferred by many apps as it includes classes needed to interact with the garbage in order to perform caching.

It is worth to mention that there are three main encryption API packages according to the official Android website [69]. These packages are javax.crypto, javax.crypto.interfaces and javax.crypto.spec. The basic API javax.crypto is used to access many cryptographic methods that can be utilized to obfuscate and deobfuscate the malicious payload. This package imported by the majority of benign apps with 93.8% of the total apps. Also javax.crypto.spec was highly requested in 88.4% of apps. While the package javax.crypto.interfaces called in only five apps with 1% of total apps. It is normal to observe encryption APIs in benign apps since many apps involve cryptographic activities belong to these APIs to increase the security of their apps and not to intent to harm the users and their devices.

6.2.2. Ransomware Applications

There were 68 API packages never called by ransomware apps. These APIs were eliminated from further analysis in this section. Figure 9 shows the occurrences percentage for 131 API packages (APIs with at least one request) used by ransomware apps.

(**a**) (**b**)

Figure 9. Percentage of occurrences of APIs in ransomware apps. (**a**) API packages part-1. (**b**) API packages part-2.

The top 30 API packages requested by ransomware apps were identified and listed in Figure 10. The most requested and frequently used package was java.lang with an average of 15,928.96 calls. As mentioned in benign section above Section 6.2.1, this is a common package for java programming language.

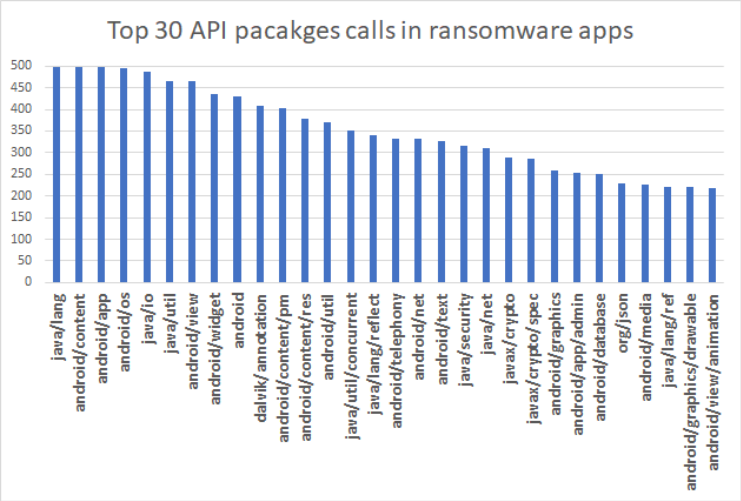


Figure 10. Top 30 API packages in ransomware apps.

The package android.telephony was in the top 30 list packages requested by ransomware apps. More than half of ransomware apps (66.6%) called android.telephony. This package includes features to keep track of phone data such as network type and connection state. Also, it can be indicative of evasion [67,70]. Figure 11 shows an example screenshots of one ransomware app (MD5: 77ADB4D5A4F8AF9B8A6D23676848C6) handling many function calls from this package. Different methods such as getDeviceId(), getNetworkOperator() and getSimCountryIso() which are all belong to the class `TelephonyManager` were found in the app. These methods could be employed to exploit information from the device and sent later to a remote server to build user profile and track the victim like in SimpleLocker ransomware family. In particular, SimpleLocker ransomware sends the IMEI number and other phone information besides the user's encrypted files to command and control server [71]. Likewise, Koler ransomware sends the IMEI of the infected device regularly to the C&C server [55]. Also, the method getDeviceId() founded in the same ransomware sample in Figure 11 belongs to the package android.telephony, which is a common method to retrieve the required IMEI number [72]. Moreover, requests to the class `TelephonyManager` could be an indicator of malicious activity, as reported in a previous study [73] in which the authors revealed that 96.7% of their malware samples imported this class.

Encryption API packages javax.crypto and javax.crypto.spec existed in the top 30 packages requested by ransomware apps. Both encryption packages were called by more than half of the ransom samples. Also, we observed that all ransom apps which requested javax.crypto also requested javax.crypto.spec and vice versa. It is important to ensure the fact that not all ransomware apps have encryption capabilities (crypto ransomware). Some ransomware apps are lockers which only block the screen of infected devices which explains the moderate percentage of encryption packages.

It is also worth noting that the API android.app.admin was in the top 30 list too with 50.8% occurrences among the ransomware apps. This package includes classes such as `DevicePolicyManager` which can be used to gain administrator privileges for example to lock screen, wipe data or initiate a factory reset of the device.

The API package org.json was also in the list of top 30. The malicious use of this API could be utilizing the class `JSONObject` to store sensitive information in the form of JSON object and leak information to an outside server [74].

```
invoke-virtual {v0}, Landroid/telephony/TelephonyManager;->
getDeviceId()Ljava/lang/String;
...
invoke-virtual {v0}, Landroid/telephony/TelephonyManager;->
getLine1Number()Ljava/lang/String;
...
invoke-virtual {v0}, Landroid/telephony/TelephonyManager;->
getNetworkCountryIso()Ljava/lang/String;
...
invoke-virtual {v0}, Landroid/telephony/TelephonyManager;->
getNetworkType()I
...
invoke-virtual {v1}, Landroid/telephony/TelephonyManager;->
getNetworkOperator()Ljava/lang/String;
...
invoke-virtual {v1}, Landroid/telephony/TelephonyManager;->
getNetworkOperatorName()Ljava/lang/String;
...
invoke-virtual {v0}, Landroid/telephony/TelephonyManager;->
getSimOperator()Ljava/lang/String;
...
invoke-virtual {v1}, Landroid/telephony/TelephonyManager;->
getSimOperatorName()Ljava/lang/String;
...
invoke-virtual {v0}, Landroid/telephony/TelephonyManager;->
getSimState()I
...
invoke-virtual {v0}, Landroid/telephony/TelephonyManager;->
isNetworkRoaming()Z
```

Figure 11. Some function calls for Telephony package in one ransomware sample.

Additionally, the package java.security.spec requested by 43.4% of the ransomware apps. This package used classes for keys and parameters specification in cryptographic algorithms. Another API package is android.annotation which includes annotations to suppress some warnings when handling feature added to new Android API levels.

6.2.3. Comparison

The API packages that appear on the top 30 API packages in both benign and ransomware apps are listed in Figure 12. These API packages called in ransomware apps with different extents than benign apps; mostly less frequency in ransomware apps. Beside these APIs, there are other API packages in the top 30 of ransomware calls that are not in the top 30 calls of benign apps and vice versa.

1. android
2. android.app
3. android.content
4. android.content.pm
5. android.content.res
6. android.database
7. android.graphics
8. android.graphics.drawable
9. android.media
10. android.net
11. android.os
12. android.text
13. android.util
14. android.view
15. android.widget
16. dalvik.annotation
17. java.io
18. java.lang
19. java.lang.ref
20. java.lang.reflect
21. java.net
22. java.security
23. java.util
24. java.util.concurrent

Figure 12. Top API packages requested by benign and ransomware apps.

When observing APIs in ransomware and benign apps separately, we could make false declarations especially when looking into the top 30 apps. Because, generally, the occurrences percentage of API packages in ransomware apps is less than benign apps. For instance org.json package was in the top 30 APIs for ransomware apps while not in the top 30 list of benign although the percentage is 45.8% in ransomware and 96.4% in benign. Therefore, we need to dig deeper and compare every API package individually.

By looking to area chart in Figure 13, we can see the API frequency of ransomware (in red) compared to benign apps(in green). The area clearly shows the high frequencies of APIs requests from benign apps compared to ransomwares. This could be due to the size difference between the

benign and ransomware samples as in general benign apps are bigger (in terms of source code) than ransomware apps.

We studied some API packages which had a significant gap in frequency between ransomware and benign and also examined the packages related to security and encryption. Figure 14 reports the distribution of some interesting API packages in terms of occurrences in both benign and ransomware apps. The most interesting package with a big gap between benign apps was android.app.admin. This API was called in almost half of the ransomware apps (50.8%) in contrast with only 8% of benign apps. It provides device administration features at the system level which might be used by attackers to exploit Android devices with the ability to lock the device. Yet, this functionality can be used for legitimate purposes such as remote device administration in enterprise scenarios [23].

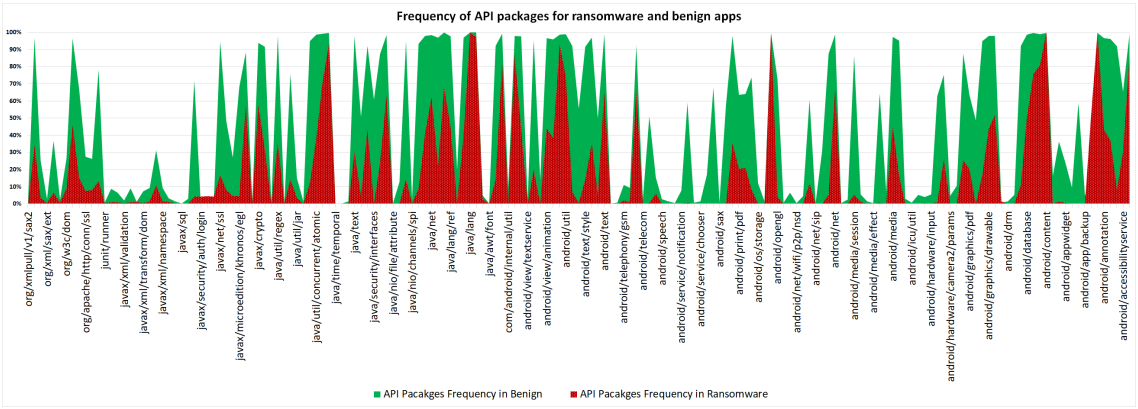


Figure 13. Compare frequency of API packages occurrences in benign and ransomware applications.

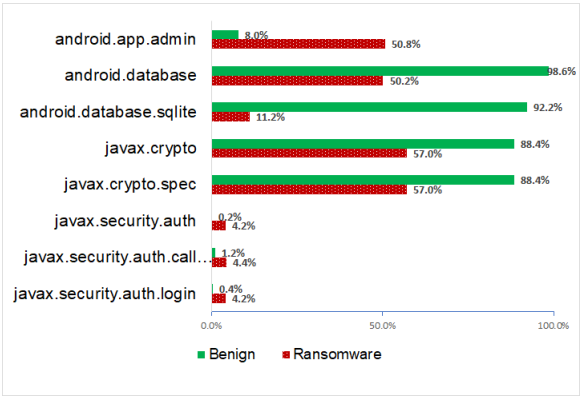


Figure 14. Some differences in API packages in terms of occurrences.

The package android.app.admin includes an instance/class called `DevicePolicyManager` which was also requested by 50.6% of ransomware samples. After digging deeper, we found that the class `DevicePolicyManager` was used on the same samples that called the android.app.admin which may indicate that the reason for calling this package is to be able to use the class `DevicePolicyManager`.

`DevicePolicyManager` class invokes many methods that could violate Android device security. This class may call lockNow(), wipeData() and other methods which were scanned and studied and come out with the results below:

- `DevicePolicyManager` class invokes the lockNow() which is a public method that locks the device immediately as if the lock period has expired. Thus, it can be used as a remote lock strategy. To command lockNow(), the app must have the tag <force-lock/> in </uses-policies> section of its meta-data. Moreover, an application with the policy <force-lock/> may exploit the built-in PIN screen to lock the device screen [75]. The force-lock tag was spotted in 28.8% of ransomware samples while found in two benign samples out of 500 (0.4%).

- Also, `DevicePolicyManager` class invokes the wipeData() which is a public method that clears all the data in the device. For this method to work, the app must have requested the policy wipe-data in its meta-data. This tag appeared in 24.2% of ransomware apps while only included in one benign app (0.2%). Furthermore, the only benign app that has the tag <wipe-data/> is called Phone Guard which is an app locker or app protector against user threats. In Phone Guard, policies tags <force-lock/>and <wipe-data/> could be used to protect users' data when the device is lost. Therefore, we may assume that, in common cases, legitimate apps do not request the method wipeData() or invoke wipe-data tag in its meta-data.

Moreover, javax.security.auth and javax.security.auth.login packages appeared in 4.2% of ransomware apps. In contrast, these packages appeared in 0.2% and 0.4%, respectively, in benign apps. The package javax.security.auth involves classes to handle authentication and javax.security.auth.login mostly used to get a pluggable authentication framework. Also, javax.security.auth.callback called in 4.4% of ransomware while in 1.2% of benign. Also, it includes `PasswordCallback` class that allows to retrieve passwords.

Encryption API packages were investigated as well to get more insights about a ransomware behavior. javax.crypto is the primary package used to access many cryptographic methods that can be used to obfuscate and deobfuscate payloads. This package was highly requested with 88.4% of the benign apps but used in 57% of ransomware apps. It is important to state that not all ransomware apps are crypto ransomware and this package is essential to access encryption methods needed by legitimate apps as well. Another encryption API is javax.crypto.spec appeared in 57% of ransomware apps where these apps also called javax.crypto. In benign apps, javax.crypto.spec imported more than ransomware with 88.4% of the total.

However, it is prevalent that crypto app needs to get file names in order to get access to each file and encrypt it. So, we investigated the method getFileNames() to see if it appears in the source code of the apps. We found that this method was requested by 6.6% of ransomware and by only three of the benign apps (0.6%). This might be a flag that this could be a crypto app but it also could be a false positive condition.

Additionally, before getting file names, crypto ransomware requires to access the root directory to search through files. One way to access to root directory is to pass the string variable "sdRootDir". Another way is to use the java method getRootPath() which belongs to java.lang API and returns the root directory as a string. We detected the existence of either the method getRootPath() or the string "sdRootDir" in 6.2% of ransomware which was less in benign apps (0.8%). We could trace a pattern for a ransomware behavior in the source code of one of the ransomware apps (MD5: 2557d58b5eecee8de620bb2b12e5423f) which belongs to SimpleLocker(Slocker) family according to VirusTotal that encrypts all data in the device's external storage. This cryptolocker app starts searching through files in directory by method getRootPath() or "sdRootDir", then create an array list of files using getFileNames() followed by encryption function and finally deleteFile() as shown in Figure 15.

```
invoke-static {}, Lcom/dcommon/fjsupport/Utils;->
getRootPath()Ljava/lang/String;

move-result-object v1

invoke-direct {v0, v1}, Ljava/io/File;->
<init>(Ljava/lang/String;)V

invoke-direct {p0, v0}, Lcom/dcommon/fjsupport/FilesEncryptor;->
getFileNames(Ljava/io/File;)V
...

invoke-virtual {v7, v8, v0}, Lcom/dcommon/fjsupport/AesCrypt;->
encrypt(Ljava/lang/String;
Ljava/lang/String;)Ljava/lang/Boolean;
...

invoke-direct {p0, v8}, Lcom/dcommon/fjsupport/FilesEncryptor;->
deleteFile(Ljava/lang/String;)V
```

Figure 15. Ransomware pattern found in source code of multiple ransomware apps.

It is worth mentioning that database handling APIs was uncommon in ransomware samples. We observed database related API packages; android.database and android.database.sqlite and found that benign apps routinely requested them. APIs android.database and android.database.sqlite called in 98.6% and 92.2% of benign apps, respectively. On the contrary, in ransomware samples, API package android.database was called in almost half of apps while android.database.sqlite called only in 11.2% ransomware apps. This observation may imply that benign apps include a database with higher probability than ransomware apps where most ransomware apps do not have or access database regularly.

Many packages were present in benign but not in ransomware, shown in Figure 16. Some of these packages requested with higher frequencies such as android.security (67.8%) and android.graphics.fonts (48.6%) while others appeared only in a few benign apps like android.media.effect and java.time.format.

1. org.xmlpull.v1.sax2
2. javax.security.cert
3. java.util.stream
4. java.util.function
5. java.time.temporal
6. java.time.format
7. java.time
8. java.nio.file.attribute
9. java.awt.font
10. com.android.internal.util
11. android.view.textservice
12. android.view.autofill
13. android.test.suitebuilder.annotation
14. android.test
15. android.telecom
16. android.speech
17. android.service.quicksettings
18. android.service.dreams
19. android.service.chooser
20. android.security.keystore
21. android.security
22. android.sax
23. android.os.health
24. android.nfc.tech
25. android.net.wifi.p2p.nsd
26. android.net.wifi.p2p
27. android.net.sip
28. android.net.nsd
29. android.mtp
30. android.media.tv
31. android.media.midi
32. android.media.effect
33. android.media.audiofx
34. android.inputmethodservice
35. android.icu.util
36. android.icu.text
37. android.hardware.camera2.params
38. android.hardware.camera2
39. android.graphics.fonts
40. android.drm
41. android.databinding
42. android.bluetooth.le

Figure 16. API packages requested by benign apps but never called by ransomware apps.

6.3. Build API-RDS Predictive Model

1. **Simulation environment:**
   Weka (https://www.cs.waikato.ac.nz/~ml/weka/) version 3.8 tool was used to evaluate the proposed API-RDS. Weka provides capabilities to train and evaluate classification models given features set [76].
2. **Evaluation metrics:**
   As for evaluation metrics, we used True Positive (TP), True Negative (TN), False Positive (FP) and False Negative (FN) to measure the performance of the machine learning model. True positive and true negative refer to ransomware and benign applications that are correctly classified. In particular, true positive rate is calculated in Equation (8) where, FN indicates the number of instances in a specific class, which is classified incorrectly as the other class type. FN rate is calculated in Equation (9).

$$TPrate = \frac{TP}{TP + FN} \tag{8}$$

$$FNrate = \frac{FN}{FN + TP} \tag{9}$$

   False positive on the other hand, is the rate of instances falsely classified as a given class. False positive rate is measured in Equation (10) where TN represents all instances correctly classified as not the given class. TN is calculated in Equation (11).

$$FPrate = \frac{FP}{TP + TN} \tag{10}$$

$$TNrate = \frac{TN}{TN + FP} \tag{11}$$

Accuracy as well is the main measure to evaluate the performance of API-RDS. The accuracy is the fraction of the correctly predicted labels to the total predicted labels defined as follows:

$$accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{12}$$

Also, cost-sensitive analysis metric must be included as an evaluation metric to evaluate the classifier performance. ROC (receiver operating characteristics) is used to recognize the appropriate reflection of the classifier. It is a visualization tool that easily clarifies whether the classifier is appropriate or not by examining the performance of a binary classifier, by creating a graph of the true positives (TP) and False Positives (FP) for every classification threshold. We measured the AUC (area under the receiver operator characteristics (ROC) curve where a perfect classifier will have an AUC of 1. Thus, the closer the AUC is to 1, the higher the model's predictive power.
Another metric considered is the Kappa statistic that is used to the compliance of predictions with regard to classification values. Kappa is calculated as seen in Equation (13).

$$Kappa = \frac{ObservedAccuracy - ExpectedAccuracy}{1 - ExpectedAccuracy} \tag{13}$$

where as $ObservedAccuracy =$

$$TP_{benign} + TP_{ransomware} \tag{14}$$

and $ExpectedAccuracy =$

$$\frac{\sum(TP_{benign} + FP_{benign}) \times TotalBenign}{TotalNumberOfInstances} + \frac{\sum(TP_{ransom} + FP_{ransom}) \times TotalRansomware}{TotalNumberOfInstances} \tag{15}$$

In particular, Kappa measures the reliability of classification decisions. The best case is denoted by 1 if the classification values (ransomware or benign) are in complete agreement. On the other hand, 0 indicates the worst case.
Also, model complexity was one of the metrics used to evaluate the proposed API-RDS. Model complexity refers to the number of features included in the predictive model that will affect the size and the performance of the predictive model [77].

3. **Classification algorithms (data mining techniques):**
Random forest (RF) was chosen as a supervised learning algorithm for ransomware detection. RF is a popular classification algorithm that usually results in a good accuracy of predictions compared to other classification algorithms [78]. It was utilized in many of malware and ransomware detection research [19,24,27,51,79,80].
Also, other classification algorithms such as support vector machine (SVM), Decision trees (J48) and Naïve Bayes (NB) as well used to measure the accuracy of the proposed approach. SVM have been extensively applied in the detection of malware [21,44,79,80] and ransomware [27,51]. Here in this work, a software package was used for SVM called sequential minimal optimization algorithm (SMO) that is available in Weka tool.

In particular, Decision trees (J48) have been successfully applied to predict the existence of malicious behavior of Android apps [19,27,49]. Naïve Bayes(NB) also used for classification problems due to low computational complexity [81] as it was used in many of malware [34,82] and ransomware [49,51] detection approaches.

7. Evaluate API-RDS

First, many machine learning classifiers were applied to detect the general accuracy of the proposed model and examine the performance of API-RDS to identify unknown ransomware applications. Second, more experiments were executed to show the performance of filtered and unfiltered datasets of API-RDS compared to one of the state-of-art approaches called R-PackDroid. Figure 17 shows the evaluation structure and the evaluation metrics utilized to compare the performances.

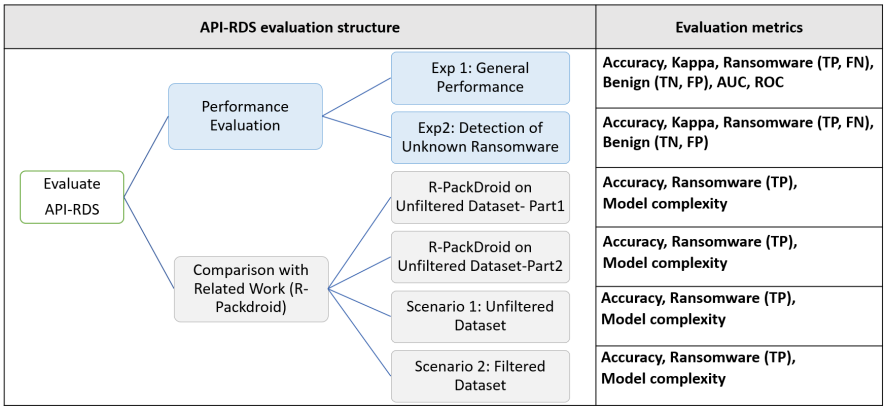


Figure 17. Application programming interface-based ransomware detection system (API-RDS) evaluation experiments.

7.1. Performance Evaluation

Data mining technique were used to detect the behavior of ransomware apps based on API packages features. Basing on two datasets, analysis was applied using different classifiers to determine whether the app is ransomware or benign.

The result of scanning ransomware and benign apps were saved in ".csv" file which later converted to arff extension so it could be processed in weka. Dataset-R consisted of 500 ransomware samples while Dataset-B included 500 benign samples. There were 174 feature vector; 173 API packages each with a corresponding label and the last feature was classification either to ransomware or benign.

- **Experiment 1: General performance**
  In this experiment, we used both Dataset-R and Dataset-B as training dataset with a total of 1000 apps (500 ransomware apps and 500 benign apps). The algorithm used was Random Forest classifier with a 10-fold cross validation testing mode. The resulting metrics as shown in Table 6 indicates a strong predictive performance from the suggested model.

Table 6. Summary of the 10-Fold cross validation accuracy metrics.

API-RDS Accuracy	Kappa	Ransomware (TP, FN)	Benign (TN, FP)
97%	0.94	490, 10	480, 20

The accuracy of identification was 97% to distinguish 1000 different ransomware and benign samples. The measured Kappa for the cross validation result was 0.94 which gives a good indication of the performance of random forest classifier used in the experiment.
In general, False Positive ratio (FP) was 2% and 4% for ransomware and benign, respectively.

In particular, 20 benign samples out of 500 (4%) were mistakenly identified as ransomware and 96% of benign apps are correctly labeled. While 98% of ransomware apps out of 500 were identified and only 10 ransomware apps incorrectly labeled as legitimate apps. The number of ransomware correctly classified (TP) is 490 and the number of benign correctly classified (TN) is 480 apps. Table 6 shows these values.

We utilized other widely used machine learning classifiers with 10 fold cross validation. The classifiers that were applied are Sequential minimal optimization (SMO), Decision trees (J48) and Naïve Bayes.

From Figure 18, we can notice that all classifiers provided high detection accuracy. Random Forest has the highest accuracy for the proposed model, followed by Decision trees (J48) with 96.6% detection rate. Also, SMO classifier achieved a close result with 96.2%. Naïve Bayes classifier exhibited the lowest result with 93.5% of accuracy. Hence, we employed Random Forest to train our model. It is worth observing that the large numbers in benign samples for many API calls (due to size difference), may improve the prediction.

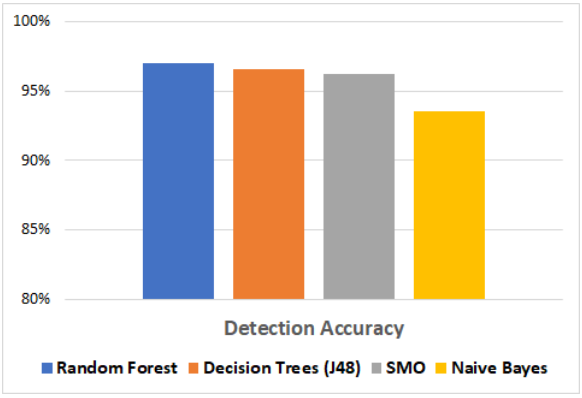


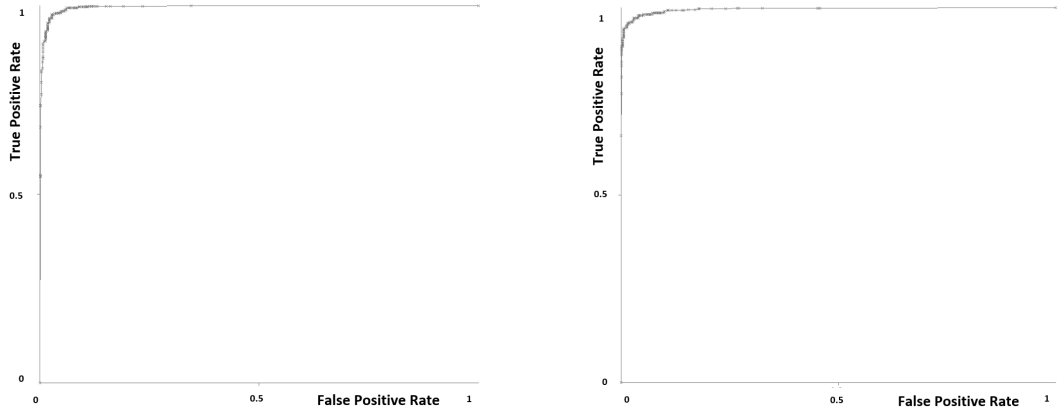
Figure 18. Accuracy of different classifiers with 10 fold cross validation.

In addition, AUC was used to assess the accuracy of the predictive models besides the False Positive ratio. Table 7 shows AUC and average of False Positives for each classifier. Generally, the classifiers achieved high AUC, more than 0.9, meaning low number of false positives. The model Naïve Bayes did not perform well on the dataset and was more prone to False positive ratio compared to other classifiers. Decision Trees (J48) gave a slightly better False Positive ratio than SMO wiht a lower AUC value. Random Forest classifier exhibited the best AUC and the lowest False positive among all classifiers. The ROC plots in Figure 19 shows the excellent AUC performance of the best case model (0.995 from Random Forest) for both benign and ransomware applications.

The high results of accuracy, Kappa and AUC are very encouraging, especially when considering a training dataset that is not homogeneous (e.g., ransomware applications from diverse sources and time frames).

Table 7. Area under the receiver (AUC) and False Positive ratio for different classifiers.

Classifier	False Positive Ratio	AUC
Random Forest	3.0%	0.995
Decision trees (J48)	3.4%	0.965
SMO	3.8%	0.961
Naïve Bayes	6.5%	0.961



(**a**) (**b**)

Figure 19. The receiver operating characteristics (ROC) curve of Random Forest model. (**a**) Ransomware apps. (**b**) Benign apps.

- **Experiment 2: Detection of unknown ransomware**
  This experiment was conducted to predict the accuracy of the model in detecting new and unseen Android apps based on API packages calls. In this experiment, the dataset was separated into training and testing datasets:

  1. Training dataset: includes 800 samples which are a combination of ransomware and benign Android apps. In particular, 400 ransomware from Dataset-R and 400 benign samples from Dataset-B.
  2. Testing dataset: a new samples that are the remaining from Dataset-R and Dataset-B with 100 unseen samples from each dataset. The ransomware samples in the testing dataset were compared with samples in training dataset to assure its uniqueness and that there are no duplicates versions. Benign apps also were new and different than samples in the training dataset. The same features set is prepared; API packages with zero occurrences that were deleted before in training dataset were deleted from this dataset as well.

  The main goal of this experiment is to measure the ability of the proposed API-RDS to detect Zero-Day and new/unseen ransomware. We applied Random Forest classifier to predict the maliciousness of the provided samples.
  Although the testing dataset was new to the model, the accuracy was 96.5% which show the high capability of the proposed model to detect ransomware activity at early stages before the damage happens.
  The accuracy matrices are shown in Table 8 are indicating that the classifier was able to discriminate between ransomware and benign apps by API calls features extracted through the source code. In particular, the API-RDS successfully identified 96% of ransomware samples and 97% of benign apps.

Table 8. Accuracy metrics.

API-RDS Accuracy	Kappa	Ransomware (TP, FN)	Benign (TN, FP)
96.5%	0.93	96, 4	97, 3

7.2. Comparison with Related Work

Other experiments were designed to present the comparison of API-RDS with the state-of-the-art approaches. The comparison was applied in terms of the dataset used, detection accuracy and complexity of the model.

The proposed API-RDS is considered the first approach that utilizes a unique dataset to build a ransomware detection system. However, it was also important to compare API-RDS to the most, recent related work. We improved R-PackDroid approach [24] which identifies Android ransomware app based on API packages occurrences collected from the source code.

R-PackDroid was designed to classify Android app to ransomware, malware or benign. In its first experiment, the testing set included 440 ransomware where R-PackDroid was able to detect 92.5% of them. The rest of the samples were identified as malware (1.8%) or as benign apps (5.7%). In its second experiment, R-PackDroid identified 68.5% of 985 ransomware samples and the rest were incorrectly identified as malware (23%) or benign (8.5%). Considering the maliciousness of application, R-PackDroid was able to detect 94.3% and 91.4% of the malicious samples in the first and second experiments, respectively. Moreover, R-PackDroid employed imbalanced dataset to discriminate ransomware from benign and generic malware which could bias the classifier performance [83].

Two scenarios were applied in order to compare with R-PackDroid work. The first one was conducted to have fair comparison with R-PackDroid by testing API-RDS on unfiltered ransomware dataset with duplicate samples. The second scenario was applied on filtered testing dataset with unique ransomware samples. The same training dataset was used in both scenarios with 400 ransomwares and 400 benign samples. In the first scenario, the testing dataset included 700 ransomware samples. API- RDS was able to detect ransomware with 99.4% of accuracy.

Moreover, API-RDS outperformed R-PackDroid in the second scenario as it correctly identified 96% of ransomware apps. There was no filtration for duplicated apps in R-PackDroid. In contrast, in API-RDS second scenario as shown in Table 9, the dataset of ransomware samples was filtered for duplicated apps and the remaining samples were almost 17% of the overall ransomware dataset.

Table 9. Dataset comparison with R-PackDroid.

Work	Raw Dataset	Filtered Dataset
R-PackDroid Part-1	4098 benign, 5560 malware and 672 ransomware	NA
R-PackDroid Part-2	4098 benign, 5560 malware and 2022 ransomware	NA
API-RDS	500 benign and 2959 Ransomware	500 benign and 500 ransomware

In addition, authors in R-PackDroid used HelDroid dataset that includes 672 ransomware samples. They used 232 ransomware in the training dataset for both experiments and used the rest 440 samples as a testing dataset in their first experiment. Although we could not acquire all the samples in HelDroid ransomware dataset (only 304 samples were obtained), we found only 63 unique samples; which means 79.3% were duplicate samples.

It is important to point that, we also found duplicates between their training and testing datasets. Moreover, API-RDS has the advantage of having a smaller features set which leads to reducing the complexity of the model by 26% compared to R-PackDroid. Table 10 shows a comparison between API-RDS and R-PackDroid.

Table 10. Compare proposed API-RDS to state of the art R-PackDroid approach

Work	Testing Dataset	Classified as Malicious	API Level	Number of Features (Complexity)	Detection Accuracy
R-PackDroid Part-1	440 (unfiltered)	415	API24	234	94.3%
R-PackDroid Part-2	958 (unfiltered)	876	API24	234	91.4%
API-RDS Scenario-1	700 (unfiltered)	696	API27	173	99.4%
API-RDS Scenario-2	100 (filtered)	96	API27	173	96%

8. Conclusions

The purpose of this study was to enhance the detection of ransomware applications in Android platform. Firstly, current ransomware detection approaches were inspected and a deep comparative analysis among them was conducted. We proposed API-based Ransomware Detection System (API-RDS), a scalable framework for static analysis of Android applications to detect ransomware apps. The proposed API-RDS system is a proactive mechanism based on data mining and machine learning. This system statically scans an Android app by inspecting API packages calls in the source code and reports the classification of seen and unseen app either to clean app or ransomware. API packages occurrences frequency were compared between ransomware and benign apps. Also, the study identified API packages that mostly differentiate ransomware app from benign , for example, android.app.admin and android.database.sqlite. In addition to the API-packages that were never used or highly used in benign or/and ransomware apps. In contrast to previously proposed systems, a high-quality unique dataset was used instead of duplicated samples that may affect the data mining results and distort their accuracy. One of the significant contributions of this work is to develop a unique and reliable ransomware dataset. Out of 2959 ransomware samples, we found nearly 17% of these samples were unique.

To demonstrate the capability of the proposed API-RDS, various experiments were conducted. The first experiment was conducted to evaluate the general performance of API-RDS. Different classification algorithms were applied where the best accuracy level achieved by Random Forest with 97% accuracy, 0.995 AUC performance, 0.94 Kappa and obtained a significantly low false negative rate. With further future improvements, the performance would be even higher. In addition, a second experiment was applied to evaluate the performance of API-RDS for detecting new ransomware apps. API-RDS successfully identified 96% of unseen ransomware samples and 97% of unseen benign apps. The overall accuracy was 96.5%.

The last experiment aimed to compare API-RDS to state-of-the-art approaches(R-PackDroid), whether the datasets were filtered or not to have a fair comparison with the related work. The best detection accuracy achieved by R-PackDroid was 94.3% whereas API-RDS attained 99.4% and 96% accuracy using unfiltered and filtered datasets; respectively. Moreover, the complexity of the classification model was reduced by 26% due to features set reduction.

It is worth mentioning that API-RDS service could be provided in terms of a mobile app, a website app or integrated with other static/dynamic detection systems. Moreover, the unique constructed datasets in this research including the apps samples themselves and the API-calls datasets for both benign and ransomware apps are available to researchers and developers.

For future work, other static metrics could be deeply investigated and added to the system. The application category could be examined to differentiate ransomware apps because many ransomware apps in our dataset belong to specific categories such as games and flash players. Intelligence solutions could also be injected to measure the significance of static metrics in the detection process. Additionally, the proposed system could be complemented by dynamic analysis to even enhance more the detection accuracy while considering the efficiency of the system.

Author Contributions: Conceptualization, I.A.; methodology, I.A.; software, S.A.; validation, I.A. and S.A.; formal analysis, I.A. and S.A.; investigation, I.A. and S.A.; data curation, I.A. and S.A.; writing—original draft preparation, S.A.; writing—review and editing, I.A.and S.A.; visualization, I.A. and S.A.; supervision, I.A.; project administration, I.A. and S.A.; funding acquisition, I.A.

Funding: This research has been supported by the College of Computer and Information Sciences at Prince Sultan University, Riyadh, KSA.

Acknowledgments: The Authors would like to thank the Security Engineering Lab (sel.psu.edu.sa) for providing the experimental environment.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Gazet, A. Comparative analysis of various ransomware virii. *J. Comput. Virol.* **2010**, *6*, 77–90.
2. Kharraz, A.; Robertson, W.; Balzarotti, D.; Bilge, L.; Kirda, E. Cutting the gordian knot: A look under the hood of ransomware attacks. In Proceedings of the International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, Milan, Italy, 9–10 July 2015; pp. 3–24.
3. Rajput, T.S. Evolving Threat Agents: Ransomware and Their Variants. *Int. J. Comput. Appl.* **2017**, *164*, 28–34.
4. Telegraph, T. WannaCry Ransomware 'from North Korea' Say UK and US. 2017. Available online: http://www.telegraph.co.uk/news/2017/06/15/wannacry-ransomware-north-korea-say-uk-us/ (accessed on 15 November 2017).
5. Guardian, T. University College London Hit by Ransomware Attack. 2017. Available online: https://www.theguardian.com/technology/2017/jun/15/university-college-london-hit-by-ransomware-attack-hospitals-email-phishing (accessed on 15 November 2017).
6. News, C. WannaCry Ransomware Attack Losses could Reach $4 Billion. 2017. Available online: https://www.cbsnews.com/news/wannacry-ransomware-attacks-wannacry-virus-losses/ (accessed on 15 November 2017).
7. Verizon. Data Breach Investigations Report. 2017. Available online: http://www.verizonenterprise.com/verizon-insights-lab/dbir/2017/ (accessed on 15 November 2017).
8. Labs, S. No Platform Immune from Ransomware, According to SophosLabs 2018 Malware Forecast. 2017. Available online: https://www.sophos.com/en-us/press-office/press-releases/2017/11/sophoslabs-2018-malware-forecast.aspx (accessed on 16 November 2017).
9. IDC. Smartphone OS Market Share, 2017 Q1. 2017. Available online: https://www.idc.com/promo/smartphone-market-share/os (accessed on 30 November 2017).
10. Telegraph, T. Android Roars Back in Strongest Growth in Two Years, as iOS Shrinks. 2017. Available online: http://www.telegraph.co.uk/technology/2016/05/17/android-roars-back-in-strongest-growth-in-two-years-as-apple-shr/ (accessed on 17 November 2017).
11. Stats, S.G. Mobile Operating System Market Share Worldwide. 2018. Available online: http://gs.statcounter.com/os-market-share/mobile/worldwide/2018 (accessed on 11 October 2018).
12. Symantec. Internet Security Threat Report. 2017. Available online: https://www.symantec.com/content/dam/symantec/docs/reports/istr-22-2017-en.pdf (accessed on 2 November 2017).
13. Tam, K.; Khan, S.J.; Fattori, A.; Cavallaro, L. CopperDroid: Automatic Reconstruction of Android Malware Behaviors. NDSS. 2015. Available online: https://core.ac.uk/download/pdf/77298524.pdf (accessed on 1 August 2019).
14. Check Point Software Technologies Ltd. Charger Malware Calls and Raises the Risk on Google Play. 2017. Available online: https://blog.checkpoint.com/2017/01/24/charger-malware/ (accessed on 1 October 2018).
15. Tee, M.Y.; Zhang, M. Hidden App Malware Found on Google Play. 2018. Available online: https://www.symantec.com/blogs/threat-intelligence/hidden-app-malware-Google-play (accessed on 4 September 2018).
16. Tee, M.Y.; Zhang, M. More Fraudulent Apps Containing Aggressive Adware Found on Google Play. 2018. Available online: https://www.symantec.com/blogs/threat-intelligence/apps-containing-aggressive-adware-found-Google-play (accessed on 4 September 2018).
17. Global, T. Telstra Cyber Security Report 2017. 2017. Available online: https://www.telstraglobal.com/images/assets/insights/resources/Telstra_Cyber_Security_Report_2017_-_Whitepaper.pdf (accessed on 20 November 2017).
18. Guardian, T. Don't Pay WannaCry Demands, Cybersecurity Experts Say. 2017. Available online: https://www.theguardian.com/technology/2017/may/15/dont-pay-ransomware-demands-cybersecurity-experts-say-wannacry (accessed on 30 November 2017).
19. Bhatia, T.; Kaushal, R. Malware detection in android based on dynamic analysis. In Proceedings of the 2017 International Conference on Cyber Security And Protection Of Digital Services (Cyber Security), London, UK, 19–20 June 2017; pp. 1–6. doi:10.1109/CyberSecPODS.2017.8074847.
20. Wong, M.Y.; Lie, D. IntelliDroid: A Targeted Input Generator for the Dynamic Analysis of Android Malware. NDSS, 2016; Volume 16, pp. 21–24. Available online: https://www.ndss-symposium.org/wp-content/uploads/2017/09/intellidroid-targeted-input-generator-dynamic-analysis-android-malware.pdf (accessed on 1 August 2019).

21. Arp, D.; Spreitzenbarth, M.; Hubner, M.; Gascon, H.; Rieck, K.; Siemens, C. DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket. NDSS. 2014. Available online: https://www.researchgate.net/profile/Hugo_Gascon/publication/264785935_DREBIN_Effective_and_Explainable_Detection_of_Android_Malware_in_Your_Pocket/links/53efd0020cf26b9b7dcdf395.pdf (accessed on 1 August 2019).
22. Yang, T.; Yang, Y.; Qian, K.; Lo, D.C.T.; Qian, Y.; Tao, L. Automated detection and analysis for android ransomware. In Proceedings of the 2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems, New York, NY, USA, 24–26 August 2015; pp. 1338–1343.
23. Andronio, N.; Zanero, S.; Maggi, F. HelDroid: Dissecting and Detecting Mobile Ransomware. In *Research in Attacks, Intrusions, and Defenses, Proceedings of the 18th International Symposium, RAID 2015, Kyoto, Japan, 2–4 November 2015*; Bos, H., Monrose, F., Blanc, G., Eds.; Springer International Publishing: Cham, Switzerland, 2015; pp. 382–404. doi:10.1007/978-3-319-26362-5_18.
24. Maiorca, D.; Mercaldo, F.; Giacinto, G.; Visaggio, C.A.; Martinelli, F. R-PackDroid: API package-based characterization and detection of mobile ransomware. In Proceedings of the Symposium on Applied Computing, Marrakech, Morocco, 3–7 April 2017; pp. 1718–1723.
25. Mercaldo, F.; Nardone, V.; Santone, A.; Visaggio, C.A. Ransomware steals your phone. formal methods rescue it. In Proceedings of the International Conference on Formal Techniques for Distributed Objects, Components, and Systems, Marrakech, Morocco, 3–7 April 2016; pp. 212–221.
26. Chen, J.; Wang, C.; Zhao, Z.; Chen, K.; Du, R.; Ahn, G.J. Uncovering the face of android ransomware: Characterization and real-time detection. *IEEE Trans. Inf. Forensics Secur.* **2018**, *13*, 1286–1300.
27. Zheng, C.; Dellarocca, N.; Andronio, N.; Zanero, S.; Maggi, F. Greateatlon: Fast, static detection of mobile ransomware. In Proceedings of the International Conference on Security and Privacy in Communication Systems, Guangzhou, China, 10–12 October 2016; pp. 617–636.
28. Wang, W.; Zhao, M.; Gao, Z.; Xu, G.; Xian, H.; Li, Y.; Zhang, X. Constructing Features for Detecting Android Malicious Applications: Issues, Taxonomy and Directions. *IEEE Access* **2019**. doi:10.1109/ACCESS.2019.2918139
29. Ma, Z.; Ge, H.; Liu, Y.; Zhao, M.; Ma, J. A Combination Method for Android Malware Detection Based on Control Flow Graphs and Machine Learning Algorithms. *IEEE Access* **2019**, *7*, 21235–21245.
30. Almomani, I.; Khayer, A. Android applications scanning: The guide. In Proceedings of the 2019 International Conference on Computer and Information Sciences (ICCIS), Aljouf, Saudi Arabia, 3–4 April 2019; pp. 1–5.
31. Almomani, I.; Alenezi, M. Android Application Security Scanning Process. In *Android*; IntechOpen: London, UK, 2019.
32. Mercaldo, F.; Nardone, V.; Santone, A. Ransomware Inside Out. In Proceedings of the 2016 11th International Conference on Availability, Reliability and Security (ARES), Salzburg, Austria, 31 August–2 September 2016; pp. 628–637. doi:10.1109/ARES.2016.35.
33. Hoffmann, J.; Rytilahti, T.; Maiorca, D.; Winandy, M.; Giacinto, G.; Holz, T. Evaluating analysis tools for android apps: Status quo and robustness against obfuscation. In Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy, New Orleans, LA, USA, 9–11 March 2016; pp. 139–141.
34. Akhuseyinoglu, N.B.; Akhuseyinoglu, K. AntiWare: An automated Android malware detection tool based on machine learning approach and official market metadata. In Proceedings of the 2016 IEEE 7th Annual Ubiquitous Computing, Electronics Mobile Communication Conference (UEMCON), New York, NY, USA, 20–22 October 2016; pp. 1–7. doi:10.1109/UEMCON.2016.7777867.
35. Quan, D.; Zhai, L.; Yang, F.; Wang, P. Detection of android malicious apps based on the sensitive behaviors. In Proceedings of the 2014 IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications, Beijing, China, 24–26 September 2014; pp. 877–883.
36. Li, Q.; Li, X. Android malware detection based on static analysis of characteristic tree. In Proceedings of the 2015 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery, Xi'an, China, 17–19 September 2015; pp. 84–91.
37. Moser, A.; Kruegel, C.; Kirda, E. Limits of static analysis for malware detection. In Proceedings of the Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007), Miami Beach, FL, USA, 10–14 Dececember 2007; pp. 421–430.

38. Zhang, Y.; Lv, D.; Guo, R.; Dietterich, T.; Zhou, Z.; Zhang, C.; Ma, Y.; Kuncheva, L.; Rokach, L.; Zhou, Z.; et al. Data Mining: Practical Machine Learning Tools and Techniques. *J. Softw. Eng.* **1997**, *11*, 97–136.
39. Almomani, I.; Alenezi, M. Efficient Denial of Service Attacks Detection in Wireless Sensor Networks. *J. Inf. Sci. Eng.* **2018**, *34*, 977–1000.
40. Enck, W.; Gilbert, P.; Han, S.; Tendulkar, V.; Chun, B.G.; Cox, L.P.; Jung, J.; McDaniel, P.; Sheth, A.N. TaintDroid: An information-flow tracking system for realtime privacy monitoring on smartphones. *ACM Trans. Comput. Syst.* **2014**, *32*, 5.
41. Zhou, Y.; Wang, Z.; Zhou, W.; Jiang, X. Hey, you, Get off of My Market: Detecting Malicious Apps in Official and Alternative Android Markets. NDSS. 2012. Volume 25, pp. 50–52. Available online: https://www.csd.uoc.gr/~hy558/papers/mal_apps.pdf (accessed on 1 August 2019).
42. Grace, M.; Zhou, Y.; Zhang, Q.; Zou, S.; Jiang, X. Riskranker: Scalable and accurate zero-day android malware detection. In Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services, Low Wood Bay, Lake District, UK, 25–29 June 2012; pp. 281–294.
43. Affairs, S. DREBIN Android App Detects 94 Percent of Mobile Malware. Hoboken, NJ, USA. 2014. Available online: http://securityaffairs.co/wordpress/29020/malware/drebin-android-av.html (accessed on 1 December 2017).
44. Yang, W.; Xiao, X.; Andow, B.; Li, S.; Xie, T.; Enck, W. Appcontext: Differentiating malicious and benign mobile app behaviors using context. In Proceedings of the 37th International Conference on Software Engineering (ICSE), Florence, Italy, 16–24 May 2015; Volume 1, pp. 303–313.
45. Yang, W.; Kong, D.; Xie, T.; Gunter, C.A. Malware detection in adversarial settings: Exploiting feature evolutions and confusions in android apps. In Proceedings of the 33rd Annual Computer Security Applications Conference (ACSAC), Orlando, FL, USA, 4–8 December 2017.
46. Wang, X.; Zhu, S.; Zhou, D.; Yang, Y. Droid-AntiRM: Taming Control Flow Anti-analysis to Support Automated Dynamic Analysis of Android Malware. In Proceedings of the 33rd Annual Conference on Computer Security Applications (ACSAC'17), Orlando, FL, USA, 4–8 December 2017.
47. Rastogi, V.; Chen, Y.; Jiang, X. Droidchameleon: Evaluating android anti-malware against transformation attacks. In Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security, Hangzhou, China, 8–10 May 2013; pp. 329–334.
48. Song, S.; Kim, B.; Lee, S. The effective ransomware prevention technique using process monitoring on android platform. *Mob. Inf. Syst.* **2016**, *2016*, 2946735.
49. Ferrante, A.; Malek, M.; Martinelli, F.; Mercaldo, F.; Milosevic, J. Extinguishing Ransomware-a Hybrid Approach to Android Ransomware Detection. In Proceedings of the International Symposium on Foundations and Practice of Security, Nancy, France, 23–25 October 2017; pp. 242–258.
50. Milosevic, N.; Dehghantanha, A.; Choo, K.K.R. Machine learning aided android malware classification. *Comput. Electr. Eng.* **2017**, *61*, 266–274.
51. Gharib, A.; Ghorbani, A. DNA-Droid: A Real-Time Android Ransomware Detection Framework. In Proceedings of the International Conference on Network and System Security, Helsinki, Finland, 21–23 August 2017; pp. 184–198.
52. Cimitile, A.; Mercaldo, F.; Nardone, V.; Santone, A.; Visaggio, C.A. Talos: No more ransomware victims with formal methods. *Int. J. Inf. Secur.* **2018**, *17*, 719–738.
53. Kharraz, A.; Arshad, S.; Mulliner, C.; Robertson, W.K.; Kirda, E. UNVEIL: A Large-Scale, Automated Approach to Detecting Ransomware. In Proceedings of the USENIX Security Symposium, Austin , TX, USA, 10–12 August 2016; pp. 757–772.
54. Scaife, N.; Carter, H.; Traynor, P.; Butler, K.R. Cryptolock (and drop it): Stopping ransomware attacks on user data. In Proceedings of the 2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS), Nara, Japan, 27–30 June 2016; pp. 303–312.
55. Lab, K. Koler—The 'Police' Ransomware for Android. 2014. Available online: https://media.kasperskycontenthub.com/wp-content/uploads/sites/43/2018/03/08081243/201407_Koler.pdf (accessed on 11 October 2017).
56. koodous. Koodous Documentation. Available online: https://docs.koodous.com/ (accessed on 10 September 2018).
57. Imran, M.; Afzal, M.T.; Qadir, M.A. A comparison of feature extraction techniques for malware analysis. *Turk. J. Electr. Eng. Comput. Sci.* **2017**, *25*, 1173–1183.
58. Santos, I.; Brezo, F.; Ugarte-Pedrero, X.; Bringas, P.G. Opcode sequences as representation of executables for data-mining-based unknown malware detection. *Inf. Sci.* **2013**, *231*, 64–82.

59. Martinelli, F.; Mercaldo, F.; Nardone, V.; Santone, A.; Visaggio, C.A. Identifying Mobile Repackaged Applications through Formal Methods. In Proceedings of the ICISSP, Portor, Portugal 19–21 February 2017; pp. 673–682.
60. Chen, Q.; Zobel, J.; Verspoor, K. Duplicates, redundancies and inconsistencies in the primary nucleotide databases: A descriptive study. *Database* **2017**, *2017*. doi:10.1093/database/baw163.
61. Rahm, E.; Do, H.H. Data cleaning: Problems and current approaches. *IEEE Data Eng. Bull.* **2000**, *23*, 3–13.
62. Raman, K. Selecting features to classify malware. In Proceedings of the InfoSec Southwest, Austin, TX, USA, 30 March–1 April 2012.
63. Yu, L.; Liu, H. Efficient feature selection via analysis of relevance and redundancy. *J. Mach. Learn. Res.* **2004**, *5*, 1205–1224.
64. Korn, F.; Pagel, B.U.; Faloutsos, C. On the "dimensionality curse" and the "self-similarity blessing". *IEEE Trans. Knowl. Data Eng.* **2001**, *13*, 96–111.
65. Sheena, K.K.; Kumar, G. Analysis of Feature selection Techniques: A Data Mining Approach. *Int. J. Comput. Appl. ICAET* **2016**, *975*, 8887.
66. Jović, A.; Brkić, K.; Bogunović, N. A review of feature selection methods with applications. In Proceedings of the 2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), Opatija, Croatia, 25–29 May 2015; pp. 1200–1205.
67. Suarez-Tangil, G.; Stringhini, G. Eight Years of Rider Measurement in the Android Malware Ecosystem: Evolution and Lessons Learned. *arXiv* **2018**, arXiv:1801.08115.
68. Darcey, S.C.L. Learn Java for Android Development: Reflection Basics. 2010. Available online: https://code.tutsplus.com/tutorials/learn-java-for-android-development-reflection-basics--mobile-3203 (accessed on 7 August 2018).
69. Android. Package Index | Android Developers. 2018. Available online: https://developer.android.com/reference/packages (accessed on 4 August 2018).
70. Sharma, A.; Dash, S.K. Mining api calls and permissions for android malware detection. In Proceedings of the International Conference on Cryptology and Network Security, Heraklion, Crete, Greece, 22–24 October 2014; pp. 191–205.
71. Bel, S. How To Dissect Android Simplelocker Ransomware. 2014. Available online: http://securehoney.net/blog/how-to-dissect-android-simplelocker-ransomware.html (accessed on 4 October 2018).
72. Mughal, J. Find/Get Imei Number in Android Programmatically. 2016. Available online: https://www.android-examples.com/get-imei-number-in-android-programmatically/ (accessed on 11 October 2017).
73. Rosmansyah, Y.; Dabarsyah, B. Malware detection on android smartphones using api class and machine learning. In Proceedings of the 2015 International Conference on Electrical Engineering and Informatics (ICEEI), Denpasar, Indonesia, 10–11 August 2015; pp. 294–297.
74. Kelkar, S.P. Detecting Information Leakage in Android Malware Using Static Taint Analysis. Ph.D. Thesis, Wright State University, Dayton, OH, USA, 2017.
75. Saurel, S. Creating a Lock Screen Device App for Android. 2018. Available online: https://medium.com/@ssaurel/creating-a-lock-screen-device-app-for-android-4ec6576b92e0 (accessed on 15 August 2018).
76. Amos, B.; Turner, H.; White, J. Applying machine learning classifiers to dynamic android malware detection at scale. In Proceedings of the 2013 9th International Wireless Communications and Mobile Computing Conference (IWCMC), Sardinia, Italy, 1–5 July 2013; pp. 1666–1671.
77. Martín, I.; Hernández, J.A.; Muñoz, A.; Guzmán, A. Android malware characterization using metadata and machine learning techniques. *Secur. Commun. Netw.* **2018**, *2018*, 5749481
78. Touw, W.G.; Bayjanov, J.R.; Overmars, L.; Backus, L.; Boekhorst, J.; Wels, M.; van Hijum, S.A. Data mining in the Life Sciences with Random Forest: A walk in the park or lost in the jungle? *Brief. Bioinform.* **2012**, *14*, 315–326.
79. Suarez-Tangil, G.; Dash, S.K.; Ahmadi, M.; Kinder, J.; Giacinto, G.; Cavallaro, L. DroidSieve: Fast and accurate classification of obfuscated android malware. In Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy, Scottsdale, AZ, USA, 22–24 March 2017; pp. 309–320.
80. Smutz, C.; Stavrou, A. Malicious PDF detection using metadata and structural features. In Proceedings of the 28th Annual Computer Security Applications Conference, Orlando, FL, USA, 3–7 December 2012; pp. 239–248.

81. Bose, A.; Hu, X.; Shin, K.G.; Park, T. Behavioral detection of malware on mobile handsets. In Proceedings of the 6th International Conference on Mobile Systems, Applications, and Services, Breckenridge, CO, USA, 17–20 June 2008; pp. 225–238.
82. Yerima, S.Y.; Sezer, S.; McWilliams, G.; Muttik, I. A new android malware detection approach using bayesian classification. In Proceedings of the 2013 IEEE 27th International Conference on Advanced Information Networking and Applications (AINA), Barcelona, Spain, 25–28 March 2013; pp. 121–128.
83. Roy, S.; DeLoach, J.; Li, Y.; Herndon, N.; Caragea, D.; Ou, X.; Ranganath, V.P.; Li, H.; Guevara, N. Experimental study with real-world data for android app security analysis using machine learning. In Proceedings of the 31st Annual Computer Security Applications Conference, Los Angeles, CA, USA, 7–11 December 2015; pp. 81–90.