Article

Hybrid-Based Analysis Impact on Ransomware Detection for Android Systems

Rana Almohaini ¹, Iman Almomani ^{1,2,*} and Aala AlKhayer ¹

¹ Security Engineering Lab, Computer Science Department, Prince Sultan University, Riyadh 11586, Saudi Arabia; almohaini.r@gmail.com (R.A.); akhayer@psu.edu.sa (A.A.)

² Computer Science Department, King Abdullah II School of Information Technology, The University of Jordan, Amman 11942, Jordan

* Correspondence: imomani@psu.edu.sa or i.momani@ju.edu.jo

Abstract: Android ransomware is one of the most threatening attacks that is increasing at an alarming rate. Ransomware attacks usually target Android users by either locking their devices or encrypting their data files and then requesting them to pay money to unlock the devices or recover the files back. Existing solutions for detecting ransomware mainly use static analysis. However, limited approaches apply dynamic analysis specifically for ransomware detection. Furthermore, the performance of these approaches is either poor or often fails in the presence of code obfuscation techniques or benign applications that use cryptography methods for their APIs usage. Additionally, most of them are unable to detect ransomware attacks at early stages. Therefore, this paper proposes a hybrid detection system that effectively utilizes both static and dynamic analyses to detect ransomware with high accuracy. For the static analysis, the proposed hybrid system considered more than 70 state-of-the-art antivirus engines. For the dynamic analysis, this research explored the existing dynamic tools and conducted an in-depth comparative study to find the proper tool to integrate it in detecting ransomware whenever needed. To evaluate the performance of the proposed hybrid system, we analyzed statically and dynamically over one hundred ransomware samples. These samples originated from 10 different ransomware families. The experiments' results revealed that static analysis achieved almost half of the detection accuracy—ranging around 40–55%, compared to the dynamic analysis, which reached a 100% accuracy rate. Moreover, this research reports some of the high API classes, methods, and permissions used in these ransomware apps. Finally, some case studies are highlighted, including failed running apps and crypto-ransomware patterns.

Keywords: Android; ransomware; hybrid analysis; detection; dynamic analysis; static analysis; accuracy; security; attacks; VirusTotal; MobSF

Citation: Almohaini, R.; Almomani, I.; AlKhayer, A. Hybrid-Based Analysis Impact on Ransomware Detection for Android Systems. *Appl. Sci.* **2021**, *11*, 10976. <https://doi.org/10.3390/app112210976>

Received: 18 October 2021

Accepted: 17 November 2021

Published: 19 November 2021

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2021 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

For the past few years, the vast demand to obtain mobile devices has peaked. As of January 2020, more than 5.94 billion people used mobile phones worldwide (<https://datareportal.com/reports/digital-2020-global-digital-overview>, accessed on 28 September 2021), with the Android operating system (OS) taking more than 73% of the market share [1]. Part of the massive popularity of Android is its ease of use and open nature. Such characteristics have attracted many companies and developers to invest in the development of Android applications. However, users have the privilege to download and install applications from unreliable sources and untrusted third parties. Consequently, allowing users to have full control over verifying the application's integrity makes it easy for cybercriminals to shift their attention toward the Android application market [2]. Attackers develop malicious software (malware), a type of mobile application used to infect a legitimate user's device. This malware can inflict harm on the user's device in multiple ways, such as risking and damaging the confidentiality, integrity, and availability of the target's data or the device itself [3].

Due to the similarities between the mobile platform and the traditional operating systems, the characteristics of personal computer (PC) malware are no longer limited to computers. With the worldwide popularity of the Android operating system for mobile devices, malware has taken a turn and can now infect those devices. There are multiple ways to infect a device, as malware comes in various forms that include, but are not limited to viruses, worms, scareware, and more [4]. Ransomware is a kind of malware that targets the victims' devices by either locking them or encrypting the users' files, then prompts the victims to pay money to unlock the devices or recover their files [5]. Ransomware attacks, such as WannaCry, Petya, NotPetya, Cerber, and Locky, have exploded in recent years, causing significant harm to individuals, companies, and even governments [6]. Furthermore, with the emergence of cryptocurrencies, such as Bitcoin, it has become relatively easy and more secure to guarantee that the attacker cannot be tracked [7].

Consequently, it is necessary for users to have a defensive mechanism against ransomware attacks. This mechanism is referred to as the automation of the ransomware analysis process. Automated ransomware analysis is a technique that is carried out either statically or dynamically. Static analysis is a direct and fast method accomplished by studying a ransomware sample's source code without running the application [8]. However, ransomware can evade static analysis through code obfuscation techniques. As a result, dynamic analysis can overcome this weakness since it can detect the ransomware's behavior during the execution of the application [9]. Furthermore, the irreversible nature of the ransomware attack demands early detection, as most of the current solutions depend on signature-based detection. This type of detection can discover whether an application is injected with ransomware or not by obtaining information about it first, such as its pattern, structure, or behavior. Then, it matches this information with what is stored in databases to state whether it is safe or not. However, existing research works [10–12] stress that ransomware attacks are challenging to be detected and prevented. This is because many of the current approaches are neither effective nor efficient for ransomware detection. Many of them lack the dependency on dynamic analysis and/or suffer from low accuracy detection.

Therefore, this paper introduces a hybrid Android ransomware detection system that integrates the prominent influence of the dynamic analysis to the static analysis results. The goal is to benefit from the advantages of both types of analyses significantly and apply each one of them smartly and effectively. The hybrid system examines the Android app by running the static analysis first. Then, based on the results of this analysis, the system decides to pass the app to the dynamic analysis or not.

We used more than 70 antivirus engines to scan the Android app to guarantee an efficient static analysis process. Based on the detection accuracy of the engines, the system could forward the app to the dynamic analysis, which is executed in an isolated environment from the user device. Furthermore, we performed an exhaustive study to explore and test the state-of-the-art dynamic malware analysis tools to choose a suitable tool to integrate into the hybrid system.

Moreover, the features extracted during the execution of both types of analysis were compared and stored in a database. Consequently, this research highlighted the classes, methods, and permissions most used by Android Ransomware apps. Moreover, this dataset can be offered to researchers and developers to perform further analysis, using different machine learning techniques.

Hence, the main contributions of this paper can be summarized as follows:

- Present the state of the art of ransomware detection systems, whether static based or dynamic based.
- Provide a systematic review of the available dynamic analysis tools and methodologies.
- Build a new dataset of ransomware apps.
- Propose a hybrid system to detect Android ransomware by applying both static and dynamic analyses.
- Study the impact of dynamic analysis on the ransomware detection accuracy.

- Evaluate the proposed hybrid system by examining hundreds of samples belonging to different ransomware families, using various evaluation metrics.
- Showcase studies of encountered ransomware samples.
- Discuss the observations and suggest recommendations regarding the dynamic analysis process and the ransomware patterns.

The rest of this paper is organized as follows: Section 2 presents an overview of the existing works on ransomware analysis approaches. Section 3 describes the proposed hybrid-based ransomware detection approach. In Section 4, the experiment preparation and implementation are discussed. The results and discussion are presented in Section 5. Finally, the paper is concluded and future work is presented in Section 6.

2. Related Works

This section presents previous research studies that aimed to increase Android mobile protection and mitigate ransomware threats. Android ransomware is considered an important research topic; however, there is little related work that has discussed ransomware specifically. Ransomware researchers have attempted to address the behavior of the suspected applications from multiple views, such as checking API information, extracting features, monitoring, and tracking applications activities. Almomani et al. [13] presented a new approach to detect Android ransomware which depends on machine learning techniques, using binary particle swarm optimization algorithm to tune the hyperparameters and perform feature selection. The support vector machine (SVM) and synthetic minority oversampling technique (SMOTE) were utilized for classification. The evaluated approach SMOTE- t BPSO-SVM was reported to achieve the highest Gmean, sensitivity, and specificity.

Song et al. [14] presented a tool that dynamically monitors the read and write accesses to the file systems in Android devices. The proposed tool utilizes statistical methods to detect the abnormal processes of the Android operating system based on processor usage, memory usage, and I/O rates. However, this system fails to detect ransomware variants with threatening text and performs file locking. In addition, no implementation is available for public use. In addition, Hong et al. [15] discussed their solution “Sdguard” in an attempt to enforce privacy protection as well as ransomware detection in Android mobile systems. The proposed solution monitors Android and the access control list activities by employing a supervised permission control depending on Linux discretionary access control (DAC) and using the FUSE file system. However, the system has no publicly available implementation. On the other hand, Alzahrani et al. [16] developed a lightweight hybrid framework that utilizes static and dynamic analysis under the name “RanDroid”. The developed system is specialized in detecting ransomware variants. RanDroid takes advantage of the fact that ransomware notifies victims by displaying messages on their machines. Thus, it measures the structural similarity between a predefined set of threatening images and text with the sample from the ransomware variant, consequently being able to classify the sample as ransomware or benign.

Moreover, Andronio et al. [17] presented HelDroid, a system that combines taint analysis to detect file encryption behaviors with symbolic evaluation. HelDroid identifies if the app seeks to lock or encrypt the mobile device without the user’s permission, or a threatening payment demand is displayed on the screen. However, HelDroid has an apparent vulnerability to samples that utilize evasion mechanisms. Furthermore, authors Zheng et al. [18] pursued to improve the HelDroid detection system. The improved system is called GreatEatlon, which reduces the detection speed significantly. The proposed tool includes various text classifier systems combined with taint analysis, achieving an accuracy rate of 99%. However, the system can only classify whether the application is malware or benign, with no further indication of its kind of malware. In addition, R-PackDroid [19] is another attempted solution to detect Android ransomware based on the extracted API packages information. The proposed application is a supervised machine learning-based program that utilizes the random forest classifier and executes

it on Android mobiles directly. This approach has an advantage over the previously mentioned two approaches (HelDroid and GreatEatlon) by overcoming the language barrier of the ransomware applications. A limitation of this approach is that there is a possibility of evading detection at runtime if the ransomware is equipped with strategies derived from adversarial machine learning.

Cimitille et al. [10] implemented a system called Talos that detects ransomware based on formal methods. Talos checks whether the application performs specific actions with additional instructions; therefore, it can locate the code to determine the malicious behavior. Moreover, the system is robust against obfuscation techniques. However, since the authors manually crafted the logic rules that were employed in the detection system, it was a time-consuming process. The study conducted by Gharib et al. [11] resulted in building a hybrid framework that first reviews the application statically. If the results seem to be suspicious, then it sends it to be dynamically analyzed. The static analysis was performed by implementing deep learning algorithm to extract some common features, such as API calls, app permissions, text and image classifications. At the same time, dynamic features include the sequence of API calls during the runtime to match it with the malicious behavior of ransomware. This approach, however, lacks the ability to detect malware that applies obfuscated techniques since it might avoid undergoing the dynamic analysis. Another recent real-time detection system, named RansomProber, was introduced by Chen et al. [20]. This system monitors device-sensitive files and determines the user's intention and finger movements by analyzing the UI widgets of the related operation during the running time of the application. One challenge facing this approach is the user-intent analysis since it is difficult to detect slight differences between encrypting ransomware and a benign application with an encryption function.

Alkhayer et al. [21] presented a framework for Android static analysis, "ASAF". The proposed framework models the process of analyzing the apps statically without running them by implementing the Android static parse "ASParse" tool, which extracts static features and stores them according to a defined schema. Dan et al. [22] proposed a lightweight detection system that is based on six extracted features from multiple sources to identify the locker ransomware behavior. The authors then used four classifiers—support vector machine (SVM), decision tree (DT), random forest (RF), and logistic regression (LR)—in order to classify whether the application is benign or not. Furthermore, a proactive permissions-based ransomware detection system was introduced by Alsoghyer et al. [23]. The system tests different classifiers to detect the ransomware before it is invoked—to ensure high accuracy detection. Additionally, Yang et al. [24] illustrated the design of a detection system by discarding the system message log of the racked ransomware; however, the study did not implement the proposed approach.

Table 1 shows an overview of the existing ransomware approaches that were mentioned in this section, summarized by the approach type, features used in the analysis, and availability. After in-depth analysis of the literature, it was apparent that there were certain limitations in the previous studies. We considered that little related work had discussed Android ransomware specifically. These works mainly applied the static analysis detection approach. Furthermore, since ransomware can evade static analysis through code obfuscation techniques, dynamic analysis becomes essential. However, the current dynamic analysis tools are focused mainly on malware rather than ransomware. Additionally, some tools are not mature enough, are outdated, or are only available for commercial use.

Therefore, there is a need to propose a hybrid system that investigates the impact of integrating both static and dynamic analyses to efficiently and accurately detect Android ransomware and protect Android users from being victims of this type of attack. This hybrid system should include several existing static analysis approaches and check their accuracy in discriminating ransomware apps from other benign apps. Based on the static analysis results, it will be decided to apply further dynamic analysis for these apps. The selection of the proper tools to run the dynamic analysis should be deeply studied. At the end, the main goal is to be able to detect ransomware with accuracy and less cost.

Table 1. An overview of the existing ransomware approaches.

Work	Year	Static	Dynamic	Features Used	Available
[17]	2015	✓		Natural Language Processing (NLP) features.	✓
[24]	2015		✓	Critical Path and Data, Malicious Domain Access, Malicious Charges, Bypassing the Android Permission.	
[14]	2016		✓	File input/output events, Processor status information, Processor usage, I/O rate, and Memory usage.	
[15]	2017		✓	Permission control based on Linux DAC mechanism.	
[18]	2016	✓		API Calls.	✓
[19]	2017	✓	✓	System API packages.	✓
[10]	2017	✓		Model checking.	
[11]	2017	✓	✓	API Calls Hooking, API Calls, Memory, Network, Context, Image, Meta Data, Permissions.	
[16]	2018		✓	Utilizes DroidBot UI-guided test input generator and API Packages.	
[20]	2018		✓	UI widgets, users' finger movements.	
[22]	2019		✓	Text, Administrator, Window Property, System Operation, Priority, Permission.	
[23]	2020	✓		Permissions, API level 27.	✓
[21]	2020	✓		Static features in general.	✓
[25]	2020	✓		Permissions and API call of Android level 27.	✓
[26]	2021	✓		Permissions and API call of Android level 30.	✓
[13]	2021	✓		Permissions and API call of Android level 30.	✓

3. The Proposed Hybrid-Based Analysis for Ransomware Detection

This section presents the proposed hybrid Android ransomware detection system. This system utilizes both static and dynamic analyses to ensure efficient ransomware detection with high accuracy. The hybrid system starts running the static analysis first, as the dynamic analysis is more critical, complex, and time consuming. Based on the static analysis results, it is decided whether to proceed with the dynamic analysis or not. More than 70 well-known antivirus engines execute the static analysis. During this type of analysis, different features are extracted and saved. Moreover, the percentage of engines that classify the app as Ransomware is calculated. The app is forwarded to the dynamic analysis based on a specific threshold that the system admin can choose, considering the number or/and the types of antivirus engines. During the dynamic analysis, the app runs on a virtual or isolated environment, such as a sandbox, to check its behavior and functionalities and confirm whether the app is ransomware. At the same time, several features are extracted and later compared with the ones generated from the static analysis. All features resulted

from both types of analysis feed the construction of a dataset that can be used for further analysis by different ML (machine learning) or DL (deep learning) techniques. These techniques can build predictive models to be integrated with the ransomware detection systems (RDS) to detect ransomware apps with high accuracy. The proposal is that only the app's information is sent from the Android device user to the RDS. If this app was analyzed before, then the analysis result is transferred directly to the user. If not, the app is examined on the cloud (statically and dynamically), and then the analysis results are passed to the Android device user. Moreover, the apps' database is updated at the RDS server-side.

Figure 1 shows the overall flow of the proposed hybrid detection system. The details of the static and dynamic phases are elaborated in the following subsections.

Figure 1. The proposed hybrid-based system for ransomware detection.

3.1. Systematic Review of Dynamic Analysis Tools

An initial step prior to implementing hybrid analysis is to perform a systematic review of the existing tools. Figure 2 illustrates the overall methodology of the performed systematic review. This section aims to present state-of-the-art tools used to analyze Android malware apps dynamically and check their availability.

3.1.1. Searching for Tools Related to the Research Topic

The process started by searching for papers related to the research topic directly by reviewing the search results of the searched keywords and indirectly by checking the references listed in the collected academic papers and security blogs. The direct search was conducted using specific keywords (Android malware, dynamic analysis, ransomware, detection, and dynamic analysis tools). We limited the search scope to the last five years unless mentioned in the paper explicitly that it offers a dynamic analysis tool.

Figure 2. The dynamic analysis tools selection methodology.

Furthermore, the search was extended indirectly by looking through the referenced papers resulting from the first search round. The collected papers and tools were organized in an excel file with the paper name and URL link to access the full paper. Then, the screening process started by scanning the papers or the posted blogs to determine whether or not they relate to the dynamic analysis of Android malware tools and whether they are available for use or not.

3.1.2. Screening the Search Results

The Android research field has a wide range of tools that are known in the security field, such as NowSecure [27] and App-Ray [28]. This section lists several tools mentioned in previous research papers or that are well-known in the security community to be used in the dynamic analysis for Android applications. The authors of [29] proposed SandDroid, a malware detection and classification method based on support vector machines (SVM), among other classifiers. SandDroid utilizes a set of specific features extracted from the AndroidManifest file. However, SandDroid statically analyzes the APKs in the sandbox instead of performing dynamic analysis. Another dynamic analysis tool developed by Nviso (<https://apkscan.nviso.be/>, accessed on 1 September 2021) is Nviso APK Scan, which statically and dynamically scans Android applications for malware. Unfortunately, the original sandbox, which acts as a base to perform the dynamic analysis, no longer supports the recent development of Android applications, resulting in missing some critical parts during the dynamic analysis [30]. An additional online automated malware analysis tool is Hybrid-Analysis.com (<https://www.hybrid-analysis.com/>, accessed on 5 September 2021). Even though the tool was marketed as performing dynamic analysis on malware, this is only true for platforms other than Android OS [31].

On the other hand, some dynamic analysis tools are no longer supported. For example, the development and maintenance for CobraDroid, a customized Android architecture malware detection tool, have not been supported since 2015 [32]. Burguera et al. in [33] proposed “Crowdroid”, a framework to check the application behavior to detect malware in the Android application. However, the tool is not available online. The authors in [34] introduced AndroTotal, which is a community tool that provides a malware repository that automatically scans Android apps against a random set of malware detectors. However, upon checking the tool, it is no longer available. Other dynamic tools have a certain limitation on the Android version, such as CuckooDroid, which only supports Android 4.1 and is based on Cuckoo 1.2 [35]. The Cuckoo sandbox was further utilized by the Sandbox Pikker tool, which provides a free automated malware analysis [36]. Another example of incompatibility issues raised by dynamic tools is TaintDroid [37]. The authors of [37] analyzed network traffic to examine the abnormal behavior of Android applications. However, the tool has not been supported since 2013.

Mobile Security Framework (MobSF) is an automated application designed to conduct pen testing and security assessment, perform static/dynamic malware analysis for mobile apps (APK, IPA, and APPX), and also test REST APIs for the most common mobile platforms: Android, iOS, and Windows [38]. The framework was built as a web service with a GUI that includes a dashboard to display the analysis results. An emulator and an API that automates the process of analysis are combined in the tool. The analysis method involves decompiling the mobile application’s source code, then running the mobile application dynamically. This framework utilizes unique features upon performing the analysis and vulnerability test, such as mobile app component detection and configuration detection, such as activities, services, etc. For Android OS, it performs Manifest file analysis, decompilation of the source code, permission analysis, and classification. Subsequently, the dynamic analysis is performed after the static analysis by deploying an emulator integrated into the tool, providing an interactive testing environment. The dynamic analyzer must be installed on the host device and not a virtual machine. Even though the MobSF tool does not classify the mobile app, it provides other integrated tools, such as APKiD, that can assist in detecting and classifying an app. Finally, MobSF considers the worst case based

on regex-based checks applied in the static analysis, and it is up to the analyst to verify the findings' messages [38].

3.1.3. Installing and Examining the Collected Tools

The outcome of the previous phase was a list of collected tools. Each tool was examined to determine whether they were working or not, what kind of information was included in the generated reports, the used features (if available), and when they were last updated. Table 2 summarizes the output results for the first stage of our proposed approach. Nineteen tools are listed along with their type of analysis, extracted features, performance, category, availability, and whether it was published or mentioned in a research paper. Initially, the tool's availability, documentation, tutorials, or customer service help were checked, and whether it was for commercial use or free; if it is for commercial use, do they provide a free trial to test them? Lastly, we checked if it was open for individuals or only if the users belonged to a company. The tools that matched these criteria were downloaded to be tested. As a result, the MobSF tool was selected as a dynamic analyzer for our hybrid system.

Table 2. Summary of dynamic and hybrid analysis tools for malware applications.

S/N	Tool Name	Analysis Type	Extracted Features	Published in Other Research	Research Article	Availability	Released Year	Categ
1	SanDroid	Hybrid	Permissions, API Calls	Yes	[29]	Yes, but not stable	2013	Online/free
2	MobSF	Hybrid	Not mentioned	Yes	[38]	Yes	2015	Installation on PC/free
3	Cuckoo Droid	Dynamic	API calls and network activity	Yes	-	Yes	2014	Installation on PC/free
4	sandbox pikker	Dynamic	API calls and network activity	No	-	Yes	-	Online/free
5	Hybrid Analysis	Hybrid	API and IOC	Yes	-	No (discontinued)	2017	Public/online/free
6	Cobra Droid	Dynamic	Not mentioned	Yes	-	No (discontinued)	2013	Installation on PC/free
7	Joesand-box	Dynamic	Not mentioned	No	-	No, needs trial account and for business only	-	Online/commercial
8	Andro 4 Lab	Hybrid	Depends on each tool	No	-	Working	-	Public/installation on PC/free
9	Quixxi	Hybrid	Permissions and app vulnerability	No	-	Not tested	-	Installation on device/commercial
10	Now Secure	Hybrid	App vulnerability	No	-	Not tested	-	Online/commercial
11	AVC UnDroid	Static	Not mentioned	Yes	[39]			Online/free
12	Andro Total	Dynamic	Not mentioned	Yes	[34]	No	-	Online/free
13	SNDBOX	Hybrid	Not mentioned	-	-	Under maintenance	-	Online/free

Table 2. Cont.

S/N	Tool Name	Analysis Type	Extracted Features	Published in Other Research	Research Article	Availability	Released Year	Categ
14	Nviso APK Scan	Hybrid	Not mentioned	Yes	[30]	No	2013	Online/free
15	TaintDroid	Hybrid	Privacy-sensitive data which include location, camera, microphone and message	Yes	[37]	No longer working	2013	Online/free
16	Appray	Hybrid	Permissions, entry points activities, content providers and services	Yes	[28]	-	2013	Online/Commercial
17	Crowdroid	Static (for android applications)/dynamic	System calls	Yes	[33]	last update in 2013	2011	Online/commercial
18	Ostorlab	Dynamic	Not mentioned	Yes	[40]	2020	2018	Online/commercial
19	DroidBox	Dynamic	Hashes, network data, permissions and API	Yes	[41]	This project is abandoned	2015	Installation on PC/free

3.2. Static Analysis Phase

Static analysis scans the source code of a malicious program without executing it in an attempt to conclude its behavior and attributes. Thus, static analysis has many benefits. One of the most important benefits of applying static analysis is the discovery of the potential behavioral scenarios related to malicious software. This is due to the possibility of examining the complete source code, even the parts that are not executed, thus achieving full code coverage [42]. Another benefit of static analysis is that it is safer than dynamic analysis since it does not require the execution of the application. Furthermore, the cost of performing static analysis is less than the dynamic with an acceptable performance [43].

The static analysis phase was implemented by employing the scanning features provided by the VirusTotal platform (<https://www.virustotal.com/>, accessed on 28 August 2021) utilizing more than 70 antivirus engines [44]. Initially, the ransomware APKs were submitted to VirusTotal in order to check the total number of available engines, the total number of engines that detect the uploaded sample as malware, and the total number of engines that cannot detect the malware. Furthermore, among the detected engines, textual analysis was performed to search for keywords indicating the ransomware family type in the names generated by the detected engines, such as “Rans”, “Ransome”, “Lock”, “Locker”, and “Encode”. This step aims to determine how accurate these engines are in identifying the submitted samples as ransomware apps. Moreover, the generated static features, such as permissions, API level, size, and other information, are recorded for further investigation.

3.3. Dynamic Analysis Phase

Unlike static analysis, dynamic analysis is the process of inspecting the behavior of malicious software in a segregated run-time environment [45]. It is accomplished by instrumenting the application's code itself or by running it virtually in an isolated environment, such as a sandbox [46]. The main advantage of dynamic analysis is that all relevant information, such as the behavior, system interaction, and the effects on the host system of the analyzed application, are monitored during the execution time [47]. However, dynamic analysis suffers from some limitations, such as dormant code, where partial parts of the program code go unnoticed because they are not executed during the runtime [48]. Furthermore, dynamic analysis consumes more time and resources, which is why it is not commonly used in commercial antivirus [49].

In the dynamic analysis phase, MobSF was utilized to implement ransomware analysis. The analysis method involves decompiling the mobile application's source code, then running the ransomware application dynamically in an isolated device or an emulator virtual machine. The emulator executes the application, gathers the information and reports it back to MobSF. The purpose is to keep the original device safe from being infected. Figure 3 illustrates the high-level architecture of the MobSF dynamic analyzer. The dynamic analyzer initiates an HTTPS web proxy connection in parallel to the web traffic capturing. Additionally, it generates an Android virtual machine (VM) configured with its own agent and automatically installs the Android APK app. Furthermore, the Android VM agent records all the application data and security analysis in the background.

Figure 3. Dynamic analysis architecture.

Once the analysis is completed, all the recorded information in the VM is sent back to the dynamic analyzer for further analysis. As a final step, a report is generated. Some of the significant features included in the MobSF dynamic analyzer are the ability to take screenshots, capture web traffic, export activities, and monitor the dynamic API calls made by the application during the runtime along with the dynamic URL and email monitoring modules.

3.4. Dataset Creation

Following the feature extraction process is the dataset creation in which the parsed features are stored to be further utilized by other classifiers. Consequently, the extracted features must be in a special serialization format, such as an F-dimensional representation, where F is the total number of features [50]. In order to implement the feature serialization format, initially, a set S contains all the features under study. Then, each ransomware application is stored as a vector (v) of length $|F|$. The entries of this vector are filled with 0 if the app never uses the features. However, the vector entities are filled with c if the

app uses the feature, where c is the total number of times that a feature is reported in the ransomware app. The process of the feature sterilization is represented in Equation (1):

$$v = \begin{cases} 0, & \forall f \in F \rightarrow f \notin Rans_app \\ c, & \forall f \in F \rightarrow f \in Rans_app \end{cases} \quad (1)$$

where c is the total number of f occurring in $Rans_app$.

The constructed dataset can be further utilized for ransomware detection and classification by implementing data mining techniques [51]. Furthermore, the risk of the ransomware attack can be ranked by deploying some intelligence ranking techniques [52].

4. Experiment Preparation and Implementation

In this work, different experiments were conducted to evaluate the performance and efficiency of the hybrid system and its selected tools and demonstrate the process of examining Android apps. Subsequently, the outcomes of the experiments are reported and discussed. In order to run and test the ransomware samples, the experiments were conducted using an HP Pavilion G6 laptop equipped with Git, Python 3.8, Oracle JDK +8, and Genymotion Desktop VM along with OpenSSL and finally Wkhtmltopdf to generate the reports. Genymotion Desktop ver 3.2 is an Android emulator for developers and QA engineers to develop and automate Android applications on a wide range of virtual devices.

4.1. Ransomware Dataset Creation

In this work, the Android ransomware dataset named “CICAndMal2017” was used [53]. It includes 10 ransomware families, where each family contains 9–11 samples of ransomware apps. In total, 100 samples were categorized based on the family as shown in Table 3.

Table 3. Summary of ransomware families used in the experiments.

No.	Ransomware Family	Sample Size
1	LockerPin	10
2	Wannalocker	11
3	Svpeng	9
4	Simplocker	10
5	RansomBO	10
6	PornDroid	10
7	Pletor	10
8	Koler	10
9	Jisut	10
10	Charger	10

Before conducting the static and the dynamic experiments, all 100 ransomware samples were run using the MobSF tool to ensure their compatibility with the proposed platforms. However, 22% of Ransomware apps succeeded in running by MobSF. Therefore, all the confirmed samples were selected as a focus group for in-depth analysis and to be used as a comparison benchmark between the chosen platforms. On the other hand, the remaining samples, representing 78%, were successfully run and analyzed statically by MobSF. Consequently, they were only included in the static analysis comparison between VirusTotal and MobSF Static Analyzer.

4.2. Evaluation Metrics

This section presents the evaluation metrics used to evaluate the proposed approach to determine the impact of the dynamic analysis in improving the detection accuracy of Android ransomware apps.

Detection accuracy: The accuracy of correctly detecting the ransomware sample and identifying it as ransomware. To estimate the accuracy of detecting a ransomware sample, we calculated the cases of the dynamic detection, which were tested using the MobSF tool to ensure that the app sample was ransomware, and compared it to the accuracy of statically detecting it using VirusTotal. The detection accuracy based on static analysis is calculated in Equation (2):

$$\text{Detection Accuracy} = \frac{\text{Total \# of Engines that detected malware}}{\text{Total \# of Engines}} \quad (2)$$

Classes and methods frequencies (ClsMth): The frequency was calculated by counting the number of times the class or method appeared when the app was dynamically analyzed. Equation (3) shows that for each class and method that belongs to a ransomware application, the frequency of ClsMth is updated.

$$\text{ClsMth Frequency} = \forall (\text{Class \& Method}) \in \{\text{RansApps}\} \rightarrow \text{ClsMth} + 1 \quad (3)$$

Permissions frequency (Per): The permissions' frequency was calculated based on the extracted results from statically analyzing the selected 22 ransomware samples by using the Static Analyzer of MobSF tool and VirusTotal. As Equation (4) elaborates, the output results of VirusTotal and MobSF were summed up to find the total frequency of the permissions of the application.

$$\text{Per Frequency} = \text{Total \# of Pers extracted by (VirusTotal + MobSF)} \quad (4)$$

5. Results and Discussion

After designing and performing the experimentation as part of our proposed approach, this section discusses the experiment results.

5.1. Detection Accuracy

In order to analyze how the experiment proceeded, we used the information recorded from the static analysis to calculate the detection accuracy of all the ransomware samples generated from VirusTotal. The static analysis falls back to almost half of the accuracy detection—ranging around 40–55%. That means, on average, half of the antivirus engines managed to detect malware samples successfully. Figure 4 illustrates the detection accuracy of the 100 ransomware apps. The results are presented for each sample during the static analysis; it can be noticed that sample 58 achieved the highest accuracy of detection with 64.91%, whereas sample 19 achieved the lowest accuracy with 36.67%. Overall, most of the VirusTotal detection and analysis engine performances were below the average, which is 50.28%. The detection accuracy of all ransomware apps are summarized in Table 4.

On the other hand, to answer our research question on the impact of the dynamic analysis in detecting ransomware, Figure 5 shows the results of the detection accuracy of the 22 samples provided from VirusTotal and MobSF. The figure shows that whenever the dynamic analysis is performed on a ransomware sample, it always performs its task. For example, this task could be locking the screen and changing the password along with the wallpaper or encrypting the files stored on the mobile phone. This probability consistently achieved a detection accuracy of 100%. In contrast, most samples had less than 50% as an average detection rate when they were statically analyzed using VirusTotal. As a result, using the dynamic analysis has a higher detection accuracy and provides better performance in discriminating ransomware apps.

Figure 4. Static detection accuracy in VirusTotal for 100 ransomware samples.

Figure 5. Detection accuracy comparison between static and dynamic analysis for VirusTotal and MobSF.

Table 4. Summary of ransomware apps used in the experiments and their detection accuracy.

ID	Total Engines	Detected Engines	Undetected Engines	Accuracy	ID	Total Engines	Detected Engines	Undetected Engines	Accuracy
1	61	31	30	50.82%	51	63	29	23	46.03%
2	59	26	21	44.07%	52	63	30	22	47.62%
3	56	26	18	46.43%	53	62	34	22	54.84%
4	62	32	24	51.61%	54	61	32	24	52.46%
5	61	27	22	44.26%	55	59	33	21	55.93%
6	61	32	24	52.46%	56	63	27	24	42.86%
7	60	29	26	48.33%	57	61	28	22	45.90%
8	56	30	21	53.57%	58	57	37	15	64.91%
9	61	35	20	57.38%	59	63	30	22	47.62%
10	61	31	21	50.82%	60	61	29	21	47.54%
11	57	27	25	47.37%	61	58	31	15	53.45%
12	63	31	23	49.21%	62	61	32	17	52.46%

Table 4. Cont.

ID	Total Engines	Detected Engines	Undetected Engines	Accuracy	ID	Total Engines	Detected Engines	Undetected Engines	Accuracy
13	62	33	24	53.23%	63	62	31	20	50.00%
14	57	29	18	50.88%	64	56	23	17	41.07%
15	62	28	25	45.16%	65	61	38	17	62.30%
16	64	30	25	46.88%	66	62	29	22	46.77%
17	62	26	27	41.94%	67	60	33	15	55.00%
18	56	29	17	51.79%	68	61	28	22	45.90%
19	60	22	31	36.67%	69	59	33	18	55.93%
20	56	29	17	51.79%	70	60	28	20	46.67%
21	61	31	18	50.82%	71	64	29	24	45.31%
22	62	30	18	48.39%	72	63	27	25	42.86%
23	61	38	18	62.30%	73	62	27	23	43.55%
24	60	27	21	45.00%	74	60	28	27	46.67%
25	60	35	19	58.33%	75	61	28	28	45.90%
26	61	37	18	60.66%	76	61	32	24	52.46%
27	61	35	20	57.38%	77	61	36	20	59.02%
28	63	29	23	46.03%	78	57	25	23	43.86%
29	63	28	23	44.44%	79	62	27	24	43.55%
30	64	27	26	42.19%	80	61	38	17	62.30%
31	62	30	21	48.39%	81	63	30	22	47.62%
32	59	33	21	55.93%	82	63	28	23	44.44%
33	61	31	25	50.82%	83	63	31	20	49.21%
34	60	26	29	43.33%	84	62	35	21	56.45%
35	61	26	24	42.62%	85	60	30	18	50.00%
36	61	28	20	45.90%	86	60	35	19	58.33%
37	62	29	22	46.77%	87	62	32	24	51.61%
38	57	32	20	56.14%	88	61	31	24	50.82%
39	55	27	17	49.09%	89	62	29	22	46.77%
40	60	30	19	50.00%	90	64	28	25	43.75%
41	63	39	16	61.90%	91	66	33	23	50.00%
42	60	31	18	51.67%	92	63	33	23	52.38%
43	63	39	16	61.90%	93	63	34	21	53.97%
44	54	35	19	64.81%	94	63	30	21	47.62%
45	54	33	21	61.11%	95	63	30	22	47.62%
46	60	32	17	53.33%	96	56	29	15	51.79%
47	62	32	18	51.61%	97	62	27	24	43.55%
48	62	32	18	51.61%	98	63	26	26	41.27%
49	62	31	20	50.00%	99	61	29	21	47.54%
50	56	33	23	58.93%	100	62	29	22	46.77%

5.2. Classes and Methods Frequencies

Table 5 summarizes the categories and methods discovered during the 22 selected samples' dynamic analysis and their frequencies. The class that should be highlighted is *Base64*, which has the methods *encode()* and *decode()*. *Base64* was used by almost half of the samples that have encryption behavior. The *encoding()* method encrypts all bytes from the specified array using the encoding scheme and writing the encrypted results to an array output. Furthermore, in some cases, the *MainActivity* class contained the *DevicePolicyManager* class, which is used to manage policies enforced on a device. Some of the methods included in this class are the two methods *LockNow()* and *ResetPassword()*, which can be used by the ransomware app and perform the locking and resetting password actions.

Table 5. MobSF classes, methods and their frequency.

No.	MobSF Class Overview	Method	Frequency
1	android.os.SystemProperties	get	1
2	android.app.admin.DevicePolicyManager.resetPassword	-	1
3	android.view.WindowManagerImpl.addView	-	1
4	android.app.admin.DevicePolicyManager.lockNow	-	1
5	java.security.MessageDigest	update	2
6	java.security.MessageDigest	digest	2
7	com.up.net.MainActivity	-	2
8	android.telephony.TelephonyManager	getDeviceId	4
9	android.app.ApplicationPackageManager	setComponentEnabledSetting	8
10	android.util.Base64	encodeToString	9
11	java.net.URL	openConnection	10
12	android.util.Base64	decode	10
13	javax.crypto.spec.SecretKeySpec	\$init	12
14	javax.crypto.Cipher	doFinal	19
15	android.app.Activity	startActivity	27

Figure 6 illustrates the frequency of the classes discovered during the dynamic analysis of the 22 selected samples. For example, the class *android.app.Activity* and its method *StartActivity()* is reported as the top used method among most ransomware apps. *StartActivity()* is usually used whenever the app wants to launch a new activity, while the classes *javax.crypto.Cipher* and *javax.crypto.spec.SecretKeySpec* and their methods *doFinal()* and *\$init()* come in the second and third places, respectively. These classes provide cryptographic encryption and decryption features that originate from the Java cryptographic extension (JCE). The public method *doFinal()* is used to encrypt or decrypt data in a single step or complete a multi-step process. Depending on how this cipher is initialized, the data are encrypted or decrypted. The class *Crypto.Spec.SecretSpec* is a constructor used to construct a secret key from the given byte array to provide a new pass key to lock the screen.

Figure 6. MobSF classes overview vs. frequency.

5.3. Permissions Frequency

Overall, Figure 7 shows the frequency of the invoked permissions by the Android ransomware samples when they were statically analyzed using MobSF and VirusTotal. Both tools were able to detect the frequency of the permission *android.permission.WRITE_EXTERNAL_STORAGE* as the most used permission but with a slight difference of 14 occurrences in MobSF and 15 in VirusTotal. *android.permission.INTERNET* and the *android.permission.ACCESS_NETWORK_STATE* come in the second and third places, respectively, in terms of frequency, indicating that the majority of ransomware apps need to have permission to write on external devices and connect to the internet to complete the locking or encryption processes. Furthermore, the two permissions, *android.permission.READ_LOGS* and *android.permission.CHANGE_CONFIGURATION*, were detected and reported by the MobSF but never counted by VirusTotal. In general, it was observed that most of the permissions used by the ransomware apps (based on their description) were directly involved and were necessarily called to take over the device.

Figure 7. Permissions frequency comparison between MobSF and VirusTotal.

6. Case Study of Encountered Ransomware Samples

This section presents two case studies illustrating the experimentation conducted on the ransomware families. The first case discusses the failure of the MobSF tool to analyze the ransomware app. The second case presents the successful run of a crypto-ransomware app pattern flow.

6.1. A Case Study of a Failed Running App

During the installation and running of the ransomware apps in a virtual environment, the tool did not record any data results for some of the apps. However, they were working and performing the locking action of the screen and changing the wallpaper of the virtual device, as shown in Figure 8. The screenshot displays the process flow of the dynamic

analysis for a ransomware app. Starting from the beginning, where the app asks first to install an Adobe flash player app and requests specific permissions, it shows a screen trying to perform a scan. Then, the encryption happens, and the app shows a message that this device is monitored and displays instructions on how to pay the ransom to get the device back. Even though the ransomware app was successfully run, MobSF failed to generate the dynamic analysis reports.

Figure 8. Case study of MobSF failing to generate the data of a working ransomware sample.

6.2. Case Study of Crypto Ransomware Pattern

This study case highlights a pattern observed during the experimentation, where MobSF failed to detect the APIs and generate proper data analysis, although the sample was 100% a ransomware app. The observed pattern justifies the reason behind the failure

of MobSF to analyze the apps. In some ransomware apps, the app's behavior includes only one activity that triggers another app to be installed. Therefore, the ransomware app asks the user's permission to install the external app (intent). After that, the newly installed app requests permissions for superuser actions that were not declared in the Manifest file during the first installation, enabling the ransomware app to lock the device. Figure 9 shows a walk-through of one of the ransomware apps that followed this behavior.

Figure 9. A case study of ransomware apps that invokes intent to lock the victim device.

7. Conclusions and Future Work

In conclusion, this paper discussed a specific family of malware called ransomware by highlighting its characteristics and general behavior. This research discussed the impact of dynamic analysis on ransomware detection accuracy in Android systems. Initially, we determined the shortcomings of the literature, and define our study's contribution, with a detailed discussion regarding the current status and the availability of the tools utilizing static and dynamic approaches. Therefore, this paper introduced a hybrid-based system

that integrates the advantages of both static and dynamic analyses to guarantee accurate detection of Android ransomware. These two types of analyses were executed efficiently. The hybrid system starts with static analysis, and based on its detection performance, the system can decide whether to proceed with the dynamic analysis. In addition, we conducted a deep study to choose proper tools to inject into the proposed hybrid system. Subsequently, we chose VirusTotal as a static analyzer and MobSF as a dynamic analyzer. The comparison results showed that static analysis using VirusTotal achieved almost half of the detection accuracy—ranging from around 40–55%, compared to the dynamics-based analysis using MobSF that succeeded in achieving a 100% accuracy rate. We evaluated the results of the analysis detection by implementing some specific evaluation metrics, such as the ransomware detection accuracy and the methods, classes, and permissions frequencies, to determine the proposed approach's effectiveness.

Moreover, this research reported some of the most-used API classes and methods used in ransomware apps, such as *javax.crypto.Cipher*, *javax.crypto.spec.SecretKeySpec*, and its *method doFinal()* and *\$init()*, which came in second and third places, respectively. In addition to the highly used permissions generated from the static analyzer for both VirusTotal and MobSF, they were able to detect the frequency of the permission *android.permission.WRITE_EXTERNAL_STORAGE* as the most used permission. The permissions *android.permission.INTERNET* and *android.permission.ACCESS_NETWORK_STATE* came in second and third places in terms of frequency, indicating that most ransomware apps need to have this permission to write on external devices and connect to the internet to complete the locking or encryption processes. Finally, some case studies were highlighted, including failed running apps and crypto-ransomware patterns.

As future work, other dynamic-based analysis tools can be tested. New ransomware apps can be collected and analyzed. Machine learning techniques can be applied to the results' reports and features generated by both static and dynamic analyses to detect new ransomware apps. Additionally, other ransomware behavior patterns can be observed and analyzed.

Author Contributions: Conceptualization, R.A. and I.A.; methodology, R.A. and I.A.; software, R.A., I.A. and A.A.; validation, R.A. and I.A.; formal analysis, R.A. and I.A.; investigation, R.A. and I.A.; resources, R.A. and I.A.; data curation, R.A., I.A. and A.A.; writing—original draft preparation, R.A. and I.A.; writing—review and editing, R.A., I.A. and A.A.; visualization, R.A., I.A. and A.A.; supervision, I.A.; project administration, I.A.; funding acquisition, I.A. All authors have read and agreed to the published version of the manuscript.

Funding: The article processing charges (APC) was funded by Prince Sultan University (PSU).

Acknowledgments: The authors would like to thank PSU for paying the APC of this publication. Moreover, this research was done during the author Iman Almomani's sabbatical year 2021/2022 from the University of Jordan, Amman – Jordan.

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

References

1. Sharma, S.; Kumar, R.; Rama Krishna, C. A survey on analysis and detection of Android ransomware. *Concurr. Comput. Pract. Exp.* **2021**, *6*, e6272. [\[CrossRef\]](#)
2. Alzahrani, A.; Alshehri, A.; Alshahrani, H.; Fu, H. Ransomware in Windows and Android Platforms. *arXiv* **2020**, arXiv:2005.05571.
3. Chitneni, A. Study of emerging trends on latest technologies and its cybersecurity challenges. *J. Innov. Dev. Pharm. Tech. Sci. (JIDPTS)* **2020**, *3*, 1–4.
4. Garg, S.; Baliyan, N. Android Security Assessment: A Review, Taxonomy and Research Gap Study. *Comput. Secur.* **2020**, *100*, 102087. [\[CrossRef\]](#)
5. Alsoghyer, S.; Almomani, I. Ransomware detection system for Android applications. *Electronics* **2019**, *8*, 868. [\[CrossRef\]](#)
6. Bansal, C.; Deligiannis, P.; Maddila, C.; Rao, N. Studying ransomware attacks using web search logs. In Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval, Xi'an, China, 25–30 July 2020; pp. 1517–1520.
7. Meland, P.H.; Bayoumy, Y.F.F.; Sindre, G. The Ransomware-as-a-Service economy within the darknet. *Comput. Secur.* **2020**, *92*, 101762. [\[CrossRef\]](#)

8. Almomani, I.; Alenezi, M. Android application security scanning process. In *Telecommunication Systems-Principles and Applications of Wireless-Optical Technologies*; IntechOpen: London, UK, 2019.
9. Hwang, J.; Kim, J.; Lee, S.; Kim, K. Two-Stage Ransomware Detection Using Dynamic Analysis and Machine Learning Techniques. *Wirel. Pers. Commun.* **2020**, *112*, 2597–2609. [CrossRef]
10. 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. [CrossRef]
11. 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.
12. Bello, I.; Chiroma, H.; Abdullahi, U.A.; Gital, A.Y.; Jauro, F.; Khan, A.; Okesola, J.O.; Abdulhamid, S.M. Detecting ransomware attacks using intelligent algorithms: Recent development and next direction from deep learning and big data perspectives. *J. Ambient. Intell. Humaniz. Comput.* **2021**, *12*, 8699–8717. [CrossRef]
13. Almomani, I.; Qaddoura, R.; Habib, M.; Alsoghyer, S.; Al Khayer, A.; Aljarah, I.; Faris, H. Android Ransomware Detection Based on a Hybrid Evolutionary Approach in the Context of Highly Imbalanced Data. *IEEE Access* **2021**, *9*, 57674–57691. [CrossRef]
14. Song, S.; Kim, B.; Lee, S. The effective ransomware prevention technique using process monitoring on android platform. *Mob. Inf. Syst.* **2016**, *2016*, 1–9. [CrossRef]
15. Hong, S.; Liu, C.; Ren, B.; Chen, J. Poster: Sdguard: An android application implementing privacy protection and ransomware detection. In Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services, Niagara Falls, NY, USA, 19–23 June 2017; p. 149.
16. Alzahrani, A.; Alshehri, A.; Alshahrani, H.; Alharthi, R.; Fu, H.; Liu, A.; Zhu, Y. RanDroid: Structural similarity approach for Detecting ransomware applications in android platform. In Proceedings of the 2018 IEEE International Conference on Electro/Information Technology (EIT), Rochester, MI, USA, 3–5 May 2018; pp. 0892–0897.
17. Andronio, N.; Zanero, S.; Maggi, F. Heldroid: Dissecting and detecting mobile ransomware. In *International Symposium on Recent Advances in Intrusion Detection*; Springer: Kyoto, Japan, 2015; pp. 382–404.
18. 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.
19. Scalas, M.; Maiorca, D.; Mercaldo, F.; Visaggio, C.A.; Martinelli, F.; Giacinto, G. R-PackDroid: Practical on-device detection of Android ransomware. *arXiv* **2018**, arXiv:1805.09563v1.
20. 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.* **2017**, *13*, 1286–1300. [CrossRef]
21. Al Khayer, A.; Almomani, I.; Elkawlak, K. ASAF: Android Static Analysis Framework. In Proceedings of the 2020 First International Conference of Smart Systems and Emerging Technologies (SMARTTECH), Riyadh, Saudi Arabia, 3–5 November 2020; pp. 197–202.
22. Su, D.; Liu, J.; Wang, X.; Wang, W. Detecting Android locker-ransomware on chinese social networks. *IEEE Access* **2018**, *7*, 20381–20393. [CrossRef]
23. Alsoghyer, S.; Almomani, I. On the effectiveness of application permissions for Android ransomware detection. In Proceedings of the 2020 6th Conference on Data Science and Machine Learning Applications (CDMA), Riyadh, Saudi Arabia, 4–5 March 2020; pp. 94–99.
24. 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.
25. Faris, H.; Habib, M.; Almomani, I.; Eshtay, M.; Aljarah, I. Optimizing extreme learning machines using chains of salps for efficient Android ransomware detection. *Appl. Sci.* **2020**, *10*, 3706. [CrossRef]
26. Almomani, I.; AlKhayer, A.; Ahmed, M. An Efficient Machine Learning-based Approach for Android v. 11 Ransomware Detection. In Proceedings of the 2021 1st International Conference on Artificial Intelligence and Data Analytics (CAIDA), Riyadh, Saudi Arabia, 6–7 April 2021; pp. 240–244.
27. NowSecure Inc. The Mobile App Security Company | NowSecure, 2018. <https://www.nowsecure.com/> (accessed on 10 October 2021).
28. Titze, D.; Stephanow, P.; Schütte, J. *App-Ray: User-Driven and Fully Automated Android App Security Assessment*; Technical Report; Fraunhofer AISEC: Berlin, Germany, 2013.
29. Team, B.R. *SandDroid: An Apk Analysis Sandbox*; Xi'an Jiaotong University: Xi'an, China, 2014.
30. NVISO. ApkScan: Scan Android Applications for Malware. Available online: <https://apkscan.nviso.be/> (accessed on 11 October 2021).
31. Hybrid-Analysis: Free Automated Malware Analysis Service-Powered by Falcon Sandbox. Available online: <https://www.hybrid-analysis.com/> (accessed on 11 October 2021).
32. GitHub-jakev/CobraDroidBeta: Source Code for the Beta Release of CobraDroid. Available online: <https://github.com/jakev/CobraDroidBeta> (accessed on 15 October 2021).

33. Burguera, I.; Zurutuza, U.; Nadjm-Tehrani, S. Crowdroid: Behavior-based malware detection system for android. In Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices, New York, NY, USA, 17 October 2011; pp. 15–26.
34. Maggi, F.; Valdi, A.; Zanero, S. AndroTotal: A flexible, scalable toolbox and service for testing mobile malware detectors. In Proceedings of the Third ACM Workshop on Security and Privacy in Smartphones & Mobile Devices, New York, NY, USA, 8 November 2013; pp. 49–54.
35. CuckooDroid. Available online: <https://cuckoo-droid.readthedocs.io/en/latest/> (accessed on 12 September 2021).
36. SandboxPikker. Available online: <https://sandbox.pikker.ee/> (accessed on 12 May 2020).
37. 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. (TOCS)* **2014**, *32*, 1–29. [\[CrossRef\]](#)
38. Dalziel, H.; Abraham, A. *Automated Security Analysis of Android and IOS Applications with Mobile Security Framework*; Syngress: Amsterdam, The Netherlands 2015.
39. Agrawal, P.; Trivedi, B. Analysis of Android Malware Scanning Tools. *Int. J. Comput. Sci. Eng.* **2019**, *7*, 807–810. [\[CrossRef\]](#)
40. Montealegre, C.; Njuguna, C.R.; Malik, M.I.; Hannay, P.; McAteer, I.N. Security vulnerabilities in android applications. In Proceedings of the 16th Australian Information Security Management Conference, Perth, Australia, 5–6 December 2018.
41. Chaurasia, P. Dynamic Analysis of Android Malware Using DroidBox. Ph.D. Thesis, Tennessee State University, Nashville, TN, USA, 2015.
42. Alzaylaee, M.K.; Yerima, S.Y.; Sezer, S. DL-Droid: Deep learning based android malware detection using real devices. *Comput. Secur.* **2020**, *89*, 101663. [\[CrossRef\]](#)
43. Bridges, R.A.; Oesch, S.; Verma, M.E.; Iannacone, M.D.; Huffer, K.M.; Jewell, B.; Nichols, J.A.; Weber, B.; Beaver, J.M.; Smith, J.M.; et al. Beyond the Hype: A Real-World Evaluation of the Impact and Cost of Machine Learning–Based Malware Detection. *arXiv* **2020**, arXiv:2012.09214.
44. Zhu, S.; Zhang, Z.; Yang, L.; Song, L.; Wang, G. Benchmarking Label Dynamics of VirusTotal Engines. In Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, CCS '20, New York, NY, USA, 30 October 2020; pp. 2081–2083. [\[CrossRef\]](#)
45. Mohanta, A.; Velmurugan, K.; Hahad, M. *Preventing Ransomware: Understand, Prevent, and Remediate Ransomware Attacks*; Packt Publishing Ltd.: Birmingham, UK, 2018.
46. Melvin, A.A.R.; Kathrine, G.J.W. A Quest for Best: A Detailed Comparison Between Drakvuf-VMI-Based and Cuckoo Sandbox-Based Technique for Dynamic Malware Analysis. In *Intelligence in Big Data Technologies—Beyond the Hype*; Springer: Berlin/Heidelberg, Germany, 2021; pp. 275–290.
47. Hasan, H.; Ladani, B.T.; Zamani, B. MEGDroid: A model-driven event generation framework for dynamic android malware analysis. *Inf. Softw. Technol.* **2021**, *135*, 106569. [\[CrossRef\]](#)
48. Lam, K.Y. A Review on Malware Variants Detection Techniques for Threat Intelligence in Resource Constrained Devices: Existing Approaches, Limitations and Future Direction. In Proceedings of the Advances in Cyber Security: Second International Conference, ACeS 2020, Penang, Malaysia, 8–9 December 2020; Springer Nature: London, UK, 2021; Volume 1347, p. 354.
49. Rhode, M.; Burnap, P.; Jones, K. Early-stage malware prediction using recurrent neural networks. *Comput. Secur.* **2018**, *77*, 578–594. [\[CrossRef\]](#)
50. Li, C.; Mills, K.; Niu, D.; Zhu, R.; Zhang, H.; Kinawi, H. Android malware detection based on factorization machine. *IEEE Access* **2019**, *7*, 184008–184019. [\[CrossRef\]](#)
51. Almomani, I.; Khayer, A. Android applications scanning: The guide. In Proceedings of the 2019 International Conference on Computer and Information Sciences (ICCIS), Sakaka, Saudi Arabia, 3–4 April 2019; pp. 1–5.
52. Dharmalingam, V.P.; Palanisamy, V. A novel permission ranking system for android malware detection—The permission grader. *J. Ambient. Intell. Humaniz. Comput.* **2020**, *12*, 5071–5081. [\[CrossRef\]](#)
53. Lashkari, A.H.; Kadir, A.F.A.; Taheri, L.; Ghorbani, A.A. Toward developing a systematic approach to generate benchmark android malware datasets and classification. In Proceedings of the 2018 International Carnahan Conference on Security Technology (ICCST), Montreal, QC, Canada, 22–25 October 2018; pp. 1–7.