


## Article

# Web-Based Android Malicious Software Detection and Classification System

İbrahim Alper Doğru \*  and Ömer KIRAZ

Department of Computer Engineering, Gazi University Faculty of Technology,  
Teknikokullar Ankara 06500, Turkey; omer.kiraz1@gazi.edu.tr

\* Correspondence: iadogru@gazi.edu.tr; Tel.: +90-312-202-8591

Received: 20 August 2018; Accepted: 10 September 2018; Published: 12 September 2018



**Abstract:** Android is the most used operating system (OS) by mobile devices. Since applications uploaded to Google Play and other stores are not analyzed comprehensively, it is not known whether the applications are malicious software or not. Therefore, there is an urgent need to analyze these applications regarding malicious software. Moreover, mobile devices have limited resources to analyze the applications. In this study, a malicious detection system named “Web-Based Android Malicious Software Detection and Classification System” was developed. The system is based on client-server architecture, static analysis and web-scraping methods. The proposed system overcomes the resource restriction issue, as well as providing third-party service support by means of client-server architecture. Based on the performance evaluation conducted in this research, the developed system’s success rate is 97.62% on benign and malicious datasets.

**Keywords:** Android; malware detection; static analysis; mobile security

## 1. Introduction

The use of mobile devices has been increasing in the world. Mobile device users can easily perform banking, shopping, web page navigation, gaming, and similar transactions without computers. According to first quarter report of International Data Corporation (IDC) Company of 2017, the global smartphone market has grown 3.4% and Android’s market share in the smartphone market has reached up to 85% [1]. According to the G Data H1/2016 report, worldwide use of the Android platform was determined to be 68% in the first half of 2016 [2]. Again, according to this report, 1,723,265 new malicious application examples were identified. Yet again, the amount of new malicious software is increasing, as well as the complexity of the malicious software introduced to prevent the detection of malicious software. Google Play Protect, announced at Google I/O 2017, runs embedded on Play Store and scans the applications before and after their installations [3,4].

Android apps have become the target of malicious software developers because the Android platform has a free and open-source operating system, and whenever an application is added on the Google Play Market, the app is not examined in detail [5]. Beginning from version 6.0 Marshmallow, the operating system warns users and demands confirmation when an application requests a dangerous permission at run time. Additionally, unlike previous versions, you have the option to turn off the permissions in the latest version. These features are included on Android 6.0, so users of older Android versions are not able to use these features. Since 38.6% of Android users use Android 5.1 and older as of 2018 (Data collected during a 7-day period ending on 16 April 2018) [6], solutions covering all Android users are needed. It is, therefore, essential to develop systems that detect whether applications on the Google Play Market and other application markets are benign or malicious software. Yet, the resources of mobile devices are inadequate for such a detection system.

In this study, an Android malware detection system was developed to detect malicious applications through client-server architecture, static analysis and web-scraping methods. All transactions are conducted on the server side in consideration of the limited resource of mobile phones. Hence, there is no load generated on the client side. Virus Total, which is a highly trusted virus/malware scanner tool, was integrated into the developed system and used in the calculation method. The robustness of the developed system was evaluated using 5545 malicious and 1173 benign Android applications. The developed system was compared with existing static analysis methods as a basis for a discussion of its pros and cons.

In this article, the following topics are discussed in sequence: the related literature is reviewed in the second section; the structure of the developed system and the used tools are shown in the third section; the obtained outcomes of the performance evaluation are shared in the fourth section; the studies to be conducted in the future and the results of the research are given in the last section.

## 2. Related Studies

There are three methods found in the literature used as analysis methods for Android malicious software. These are the static, dynamic and hybrid analysis methods.

### 2.1. Static Analysis Method

In this analysis method, Android applications are analyzed using the features found in the .apk file before being installed on the device. DroidMat uses a static analysis method to determine whether applications are benign or malicious [7]. Permissions in each application's AndroidManifest.xml file, other components (activity, service and receiver) and API (Application Programming Interface) calls in the application's bytecodes are used as features. Using the Apktool [8] tool, AndroidManifest.xml and smali files of benign and malicious applications are obtained. The permissions, activities, service and receiver components, intent and API calls in these files are considered as features. In a DREBIN study, a method for Android malware detection was introduced that allows malicious applications to be identified directly on the smartphone [9,10]. Based on the developed method, many features were collected from the application code and AndroidManifest.xml file. APK Auditor is a system for detecting malicious Android-based malware [11]. APK Auditor uses a static analysis method to characterize and classify Android applications as to whether they are benign or malicious [11]. The developed system consists of three main components: an Android client, a signature database and a central server [11,12]. DroidOL addresses the issue of malware detection and suggests a new online machine learning-based framework [13]. The DroidOL system consists of three stages. In the first stage, static analysis is performed on a certain number of applications to obtain inter-procedural control-flow graphs [13]. In the second stage, the sub-graph properties of the inter-procedural control-flow graphs are extracted using the Weisfeiler-Lehman kernel, and the applications are represented as feature vectors [13]. In the last stage, an online passive aggressive classifier is used and trained to detect malicious software with these vectors [13]. In the ASE study, an integrated static detection system with four filtering layers was proposed, including MD5 (Message Digest 5) detection of characteristic values, detection of combination of malicious permissions, detection of hazardous permissions and detection of hazardous intent [14]. Wang et al. proposed a system to manage a large application market effectively and efficiently in order to categorize malicious and benign applications [15]. The alarm is triggered if the application is identified as malicious by using a combination of multiple classifiers. In the study of Sokolova et al., a system was proposed that characterizes normal behaviors for each application category and emphasizes expected permission requests. Moreover, category patterns and central permissions are obtained using graph analysis metrics [16]. The models obtained are evaluated by the performance of the application classification based on the categories developed [16]. Anwar et al. proposed a static system for mobile botnet detection [17]. This system considers the static features of Android applications, including MD, permissions, recipients, and background services [17]. Arslan et al. made a static approach that detects malicious and benign software by calculating

the number of the permissions that applications request on the code side [18]. The Dex2jar [19], and Apktool [8] tools were used to perform reverse engineering on the applications [18]. In the AndroDialysis study, the intentions in the AndroidManifest.xml file were used as a distinguishing feature in identifying malware [20]. In the study conducted, it was determined that the intentions were more effective than the permissions in the detection of malicious software [20]. However, it is considered that intentions cannot replace the permissions in the detection of malicious software [20]. In the study of Kang et al., developer information was used as an attribute [21]. It was argued that the detection of malicious software can be made more effective by comparing the application certificate serial number with pre-defined malicious certificate serial numbers [21,22]. Along those lines, Utku and Dogru developed a malicious software detection system based on well-known malicious software at the application level for mobile devices [23]. Naive Bayes and KNN machine learning algorithms were used in this study to classify permissions [23]. Atici et al. developed a static system based on machine learning algorithms and control flow graphs of Dalvik byte codes for Android malware analysis [24]. In this study, grammatical expressions consisting of control flow graphs of Android malicious software were used as an input vector [24].

## 2.2. Dynamic Analysis Method

This analysis method is based on the behavior of Android applications. Shabtai et al. developed a system that uses a dynamic analysis method to detect malicious software by network patterns of applications [12,25]. In the study, it is claimed that the network traffic patterns of different applications with the same function have similar patterns, and the results confirmed the claim [12,25]. The DroidAuditor is a behavior analysis system that monitors the behavior of apps on real Android devices and creates the graphical representations of such [26]. Andro-Profiler classifies malicious software using behavioral profiles extracted from the integrated system logs, including system calls [27]. Andro-Profiler runs a malicious application on the virtual device to create integrated system logs and analyzes the integrated system logs to create human-readable behavioral profiles [27]. In a study by Garg et al., a system is developed to detect malicious applications by looking at network activity through an observing eye on the network [28]. The developed model also has the ability to detect malicious applications using network traces, to work with different versions of operating systems, to detect unknown applications, and to detect infected applications with encrypted data [28]. In the study of Chang et al., a behavior-based malware detection system using machine learning was proposed [29]. In this study, in addition to the DroidBox [30] structure, an automatically triggered view identification program was added [29].

## 2.3. Hybrid Analysis Method

This method of analysis is based on the usage of static and dynamic analysis methods together. Shi et al. proposed a hybrid system that combines static and dynamic analysis to detect security threats that attacks on mobile applications [31]. The system consists of two main components; namely, static and dynamic analysis [31]. In the first stage, all files contained in the .apk file are obtained using Apktool [31]. The focus of this static analysis phase is to analyze the file AndroidManifest.xml [31]. In addition, a Java application was developed to read file and text patterns during this static analysis phase [31]. Start-up activity is attained from the AndroidManifest.xml file, and this information is used in the second stage to obtain sensitive API calls [31]. In the second stage, the Auto-sign tool is used to repackage the application [31]. Dynamic analysis is performed by running the repackaged application [31]. This run is called 'symbolic run', because the application is run to get the API call path [31]. In ScanMe Mobile study, a local and cloud-based hybrid malware detection system was developed [32]. A mobile cloud-based architecture was used to perform real-time detection efficiently [32]. In a study by Singh et al., a framework for the identification of malicious applications is presented [33]. The intent of this framework is to imitate artificial user behavior and analyze the real behavior of malicious software [33]. In the study of Wang et al., a static analysis technique was used to

obtain permission usage information from API calls [34]. In addition, dynamic analysis techniques were also used to test and monitor the runtime usage behavior of applications [34].

### 3. Web-Based Android Malicious Software Detection and Classification System

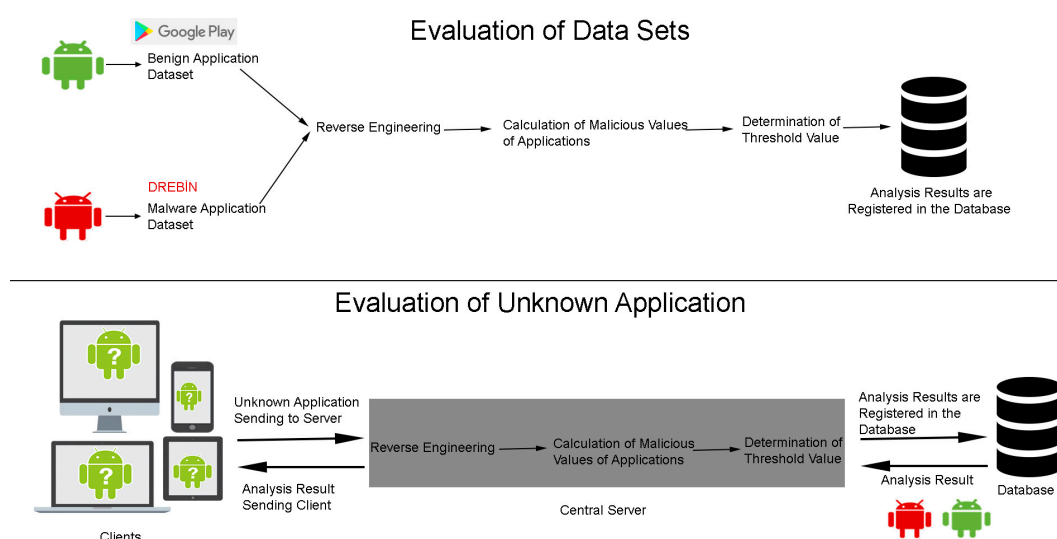
The developed system uses the static analysis method for the analysis of Android applications. The Android version used was chosen to be 4.1, due to the highest targetSdkVersion information of the applications in the Drebin malware dataset [9] being 16. The reason for this selection is that the permissions used by applications in the malicious dataset practice permissions that are present in the Android 4.1 version. To create a benign application data set, Android applications of official institutions in Turkey, other countries, and popular applications on the Google Play market were downloaded by using the APKPure [35] web page. The APKPure web page is a platform for downloading Android .apk files. By using this web page, 1331 applications were manually downloaded and a benign data set was created.

The data set created in the Drebin [9] study was used for collecting the malicious software data set, which has been shared as an open resource. This data set contains 5560 applications from 179 different malware families. These samples were collected in a period from August 2010 to October 2012 [36,37]. Since applications added to Google Play Market in the benign dataset did not go through a detailed analysis process to determine whether the applications in the benign data set were safe or not, the safety was checked using the VirusTotal API [38]. VirusTotal was used in the studies of AndroDialysis [20], Kang et al. [39], Ma et al. [40] and ICCDetector [41], which are available in the literature to ensure the safety of the applications that are used in benign data sets. 158 applications were found to be malicious by at least one antivirus program, namely the VirusTotal scanning, to sustain safer applications in the data set. As a result, a summary of the data set obtained is shown in Table 1.

**Table 1.** Summary of data set created as per VirusTotal.

Data Set Type	#	Source
Malicious	5545	Drebin
Benign	1173	Google Play market

For the static analysis methods, information obtained from the permissions that the applications use was utilized. In related works, API calls are also used as an attribute during static analysis; but these were not included in this study because it involves source code analysis. The architecture of the developed system is shown in Figure 1.



**Figure 1.** Architecture of the developed system.

The .apk files in the benign and malicious data sets were reverse engineered using the Apktool tool. The permissions found in the AndroidManifest.xml file were obtained and then the permissions that each application possessed were saved in the created database. In the third step, the permissions found individually in the malicious and benign data sets were obtained. The percentage of availability of these permissions for malicious and benign data sets was calculated. Lastly, the percentage of availability for malicious software was subtracted from the percentage of availability for benign software, and the value that we designated the malicious value was calculated. The formulated representation of this process was shown in Equation (1).

$$\text{Malicious Value of Permission} = \sum_{i=0}^n MV\_Malicious(t) - \sum_{i=0}^n MV\_Benign(t) \quad (1)$$

The total malicious value of each application was obtained by adding and evaluating the malicious values of the permissions that each application possesses. Additionally, the results of each application in the VirusTotal database were obtained using the API of VirusTotal in this step. Here, the reason for using VirusTotal to calculate the malicious values of applications is its ability to show the results of more than 50 anti-virus programs [42], and it is used in security-based studies in the literature. The studies of Ban et al. [43] and Wang et al. [15] can be given as examples of these studies. Google's subsidiary VirusTotal is a free online service that analyzes files and URLs that identify viruses, worms, Trojans, and other malicious content, as detected by virus protection engines and website scanners [44]. There are four methods for analyzing Android applications with VirusTotal: the web page, VirusTotal application installation tool, sending a mail and VirusTotal API. In this study, applications were analyzed using VirusTotal API. VirusTotal integration was added with the idea of using it for the purpose of the detection of non-new malware variants. When the literature was reviewed, it was seen that VirusTotal has been used in many studies. The md5 hash code of the application analyzed in the static analysis section of the Mobile-Sandbox study was compared with all the hashes in the VirusTotal database. If the hash is found, the "detection rate" is calculated by dividing the number of tools classifying the application as malicious by using the number of antivirus tools analyzing the application [12,45]. In the study conducted by Ban et al., VirusTotal was used to determine whether the applications used in the data set were malicious [43]. In the study of Wang et al., VirusTotal was used to generate a data set of benign applications [15]. The VirusTotal score was not evaluated for the final result, but it was used as a parameter at the point of determining whether the application was malicious or not. The VirusTotal score is calculated for each application in this study. The formulated form of this calculation is shown in Equations (2)–(4).

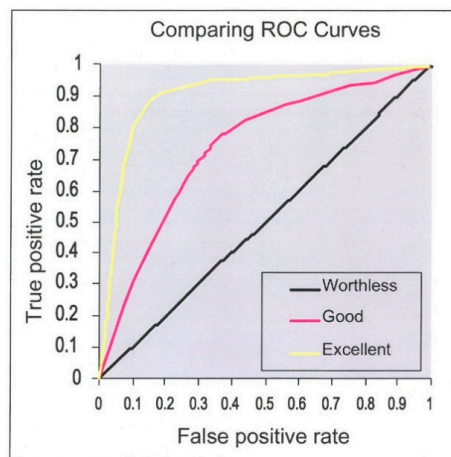
$$\text{Total (MV)} = \sum_{i=0}^{\text{number of permission}} \text{Malicious Value of Permission (t)} \quad (2)$$

$$\text{VirusTotalScore} = (\text{The Number of Antivirus Programs that Find The Application} / \text{Malicious/Total Antivirus Program Number}) * 100 \quad (3)$$

$$\text{Final Malicious Value} = (\text{Total (MV)} + \text{VirusTotalScore}) / 2 \quad (4)$$

A threshold value was determined using the final malicious values of the applications in the benign and malicious data sets by using the Receiver Operating Characteristics curve. The receiver operating characteristics curve is drawn as a graph calculated for different threshold values, with the true positivity (sensitivity) rates on the vertical axis and the false positivity (1-specificity) rates on the horizontal axis [46]. Figure 2 shows the curve of receiver operating characteristics represents excellent, good and worthless tests.





**Figure 2.** Comparison of receiver operating characteristics curves [47].

When Figure 2 is assessed, the area remaining under the curve represents an excellent test. If the area under the curve is 0.5, it represents a worthless test. In the fifth step, the applications were determined as being benign or malicious software based on this threshold value. The Accord.Statistics.Analysis [48] library was used to calculate the threshold value through the receiver operating characteristics curve. The algorithm of the piece of code used to determine the threshold value by using this library, as shown in Figure 3. A web-based client was developed for evaluating unknown Android applications. The server made by the Android application was subjected to some steps; namely, reverse engineering, computing of the malicious value of the application, and determining of the threshold value respectively. Then, the results of the analysis were saved in the database and sent to the client.

1. An object named ROC is defined using ReceiverOperatingCharacteristic in the Accord.Statistics.Analysis library.
2. Using the Compute method of the defined ROC object, the ROC curve with 100 coordinate values is calculated.
3. The for-cycles are generated as many as the number of calculated ROC curve coordinates.
4. True positivity (sensitivity) value of each coordinate value is calculated through the formula  $(\text{RocPoints}[n].\text{Sensitivity} \times 100)$ . However, the false positivity (1-specificity) value is obtained through formula  $((1 - \text{roc.Points}[n].\text{Specificity}) \times 100)$
5. When the true positivity value of greater than 90 and the false positivity value smaller than 15, they are added to the list holding the ROC objects which contains variables that hold true positivity, false positivity, and threshold values.
6. This process is repeated until the end of the for-cycles.
7. A new for-cycle is generated as the number of members in the list.
8. For each member in the list, value  $(100 - (\text{true positivity})) + (\text{false positivity})$  is calculated and the value of the first member of the list is assigned as the variable "min".
9. The values of the next members are compared to the value in the variable min, and if the value of the member is less than the value in the variable min, it is assigned as the variable min.
10. This process is repeated until the end of the for-cycles, and the threshold value of the member in the variable min is set as the threshold value.

**Figure 3.** The algorithm used to calculate the threshold value.

When Figure 3 is considered, in order to use the ReceiverOperatingCharacteristic object, it is essential that sequences allow the distinguishing of the benign and malicious applications, and the sequences that hold the total values possessed by the benign and malicious applications need to have been generated. These generated sequences were added as input to the ReceiverOperatingCharacteristic method. In the next step, a threshold value curve with 100 coordinate values was calculated. True positivity (sensitivity) and false positivity (1-specificity) scores of the generated coordinate axes were calculated. Those with a true positive rate greater than 90 and a false positive rate of lower than 15 were added to a list. The (100 - true positivity value) and false positivity values of the members in this list were added. From among the members in the list, the one with the lowest value for this was designated as the threshold value.

### 3.1. Obtaining of Application Information

Android apps are installed on phones together with .apk archive files. The information contained in these .apk files is obtained using ready-made tools or libraries. In this study, version 2.2.2 of Apktool was used to obtain the information in the AndroidManifest.xml file. On the C# Windows Form application, an application which gets the permission information of Android applications using Apktool has been developed. This application uses the System.Diagnostics.ProcessStartInfo class in the C# library to run the command that Apktool uses for reverse engineering. Since there are 6876 Android applications in total in the data set, parallel processes were used in order to make the reverse engineering process faster. The application name, target SDK version, minimum SDK version, and permissions were used, and file names that are available in the AndroidManifest.xml file of these applications were registered in the database after the reverse engineering was finished for each application in the benign and malicious data sets. In addition, malicious software family information was registered in the database for the applications in the Drebin malicious data set.

### 3.2. Malicious and Benign Application Assessment Process

In this section, the evaluation process of benign and malicious applications in the datasets is discussed. First, the singular permissions available in the malicious and benign data sets were found. This resulted in 1323 different permissions to be found. Then, the number of occurrences of each permission in the benign and malicious data sets was obtained. By using the number of occurrences obtained, the rate of occurrences in the benign and malicious data sets were calculated. Finally, the rate of occurrence in the benign data set was subtracted from the rate of occurrence in the benign data set in order to obtain the malicious value for each permission.

### 3.3. Web Application

The developed Android malware detection system in this study runs according to the client-server architecture. Because the transactions are conducted on the server side, there is no load generated on the client side. On the client side, only the analysis result is shown. The most important reason for performing server-side operations is that Android phone clients have limited resources, such as battery, processor, and RAM. The capabilities of the developed web application include Google Play integration, Apktool, VirusTotal, web scraping for APKPure, apkleecher [49] and apkbucket [50]. The application search interface of the developed web application is presented in Figure 4.

When looking at Figure 4, there are two options on the application splash screen: application search and application download. In the search for application option, you will be able to search for words on Google Play. A web-scraping method was used to do this. When the search button in the application is clicked, the searched word is added at the end of <https://play.google.com/store/search?q=link>, then &c=apps and the web page is web-scraped. To perform web scraping, it is necessary to find the XPath in the information to be obtained. In this application, the information, obtained from Google Play, is the official name and package name of the application being searched. For example, if the searched word is “football”, the link to be used for web scraping should be <https://play.google.com/store/search?q=football&c=apps>.

[//play.google.com/store/search?q=football&c=apps](https://play.google.com/store/search?q=football&c=apps). To get a list of related applications according to the searched word for this link, there is a need to find the XPath. To find the XPath, the relevant link is entered into an Internet browser and the mouse is right clicked on the opened page. The XPath is obtained by selecting the location of the application information on the incoming screen. An image showing this operation is shown in Figure 5.

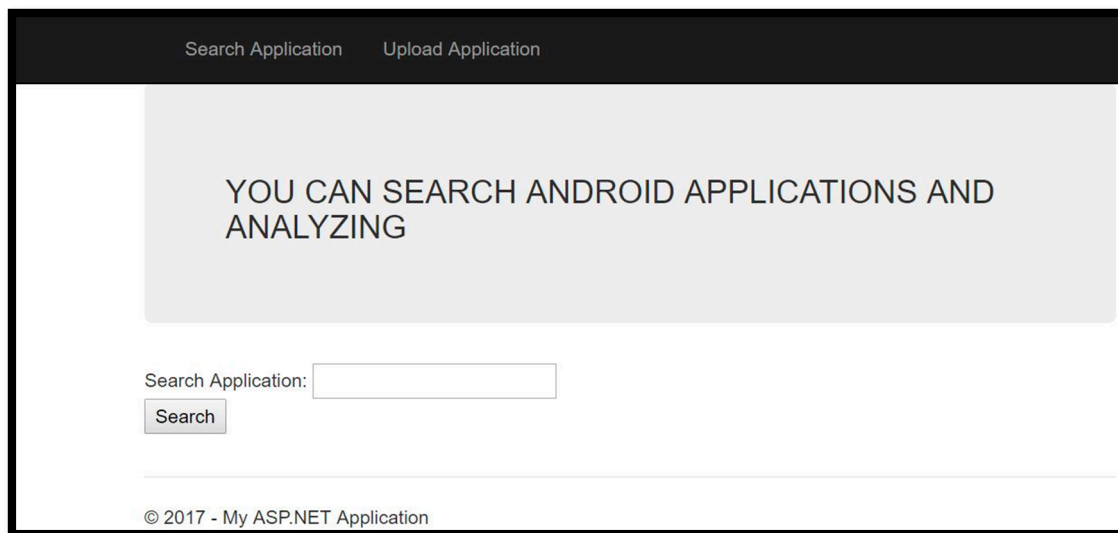


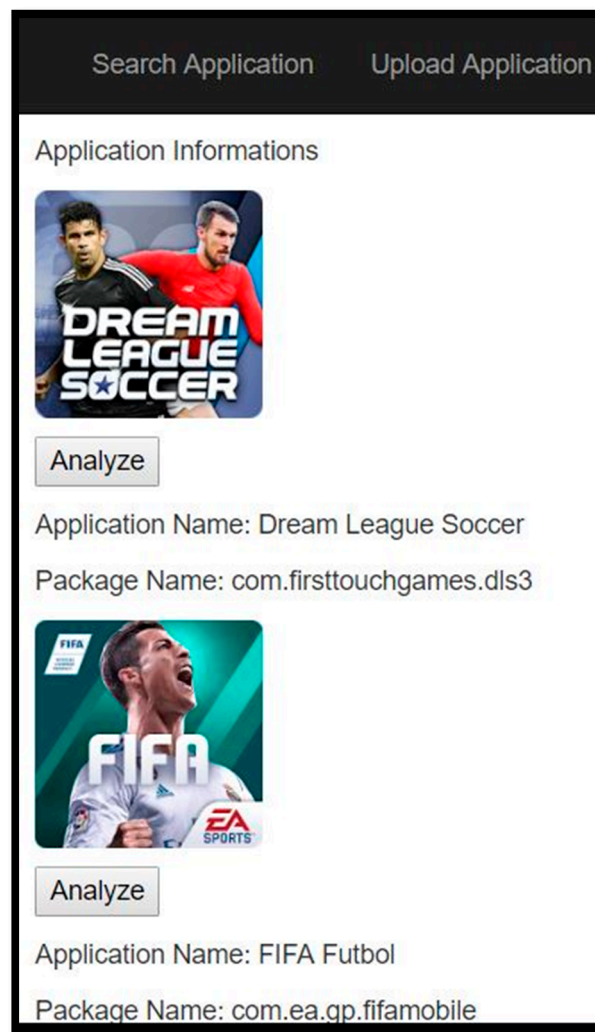
Figure 4. The application search interface of the developed web application.

```
<div class="id-card-list card-list two-cards">
  <div class="card no-rationale square-cover apps small" data-docid=
    "com.ea.gp.fifamobile" data-original-classes="card no-rationale square-cover apps
    small" data-short-classes="card no-rationale square-cover apps tiny" data-thin-
    classes="card no-rationale square-cover apps small">
    <div class="card-content id-track-click id-track-impression" data-docid=
      "com.ea.gp.fifamobile" data-server-cookie=
      "CAIaNQocEhoKFGNvbS5lYS5ncC5maWZhbW9iaWx1EAEYAzITCJiC5IPpktQCFcyu3wodT5YADUIA"
      data-uitype="500">
      <a class="card-click-target" data-server-cookie=
        "CAIaNQocEhoKFGNvbS5lYS5ncC5maWZhbW9iaWx1EAEYAzITCJiC5IPpktQCFcyu3wodT5YADUIA"
        data-uitype="500" href="/store/apps/details?id=com.ea.gp.fifamobile"
        aria-hidden="true" tabindex="-1"></a>
      <div class="cover">
        <div class="cover-image-container">...</div>
        <a class="card-click-target" href="/store/apps/details?id=com.ea.gp.fifamobile" aria-label="
          FIFA Mobile Futbol ">
          <span class="movies preordered-overlay-container id-preordered-overlay-
          container" style="display:none">...</span>
          <span class="preview-overlay-container" data-docid=
            "com.ea.gp.fifamobile"> </span>
        </a>
      </div>
```

Figure 5. HTML labels used to determine the XPath in Google Play.

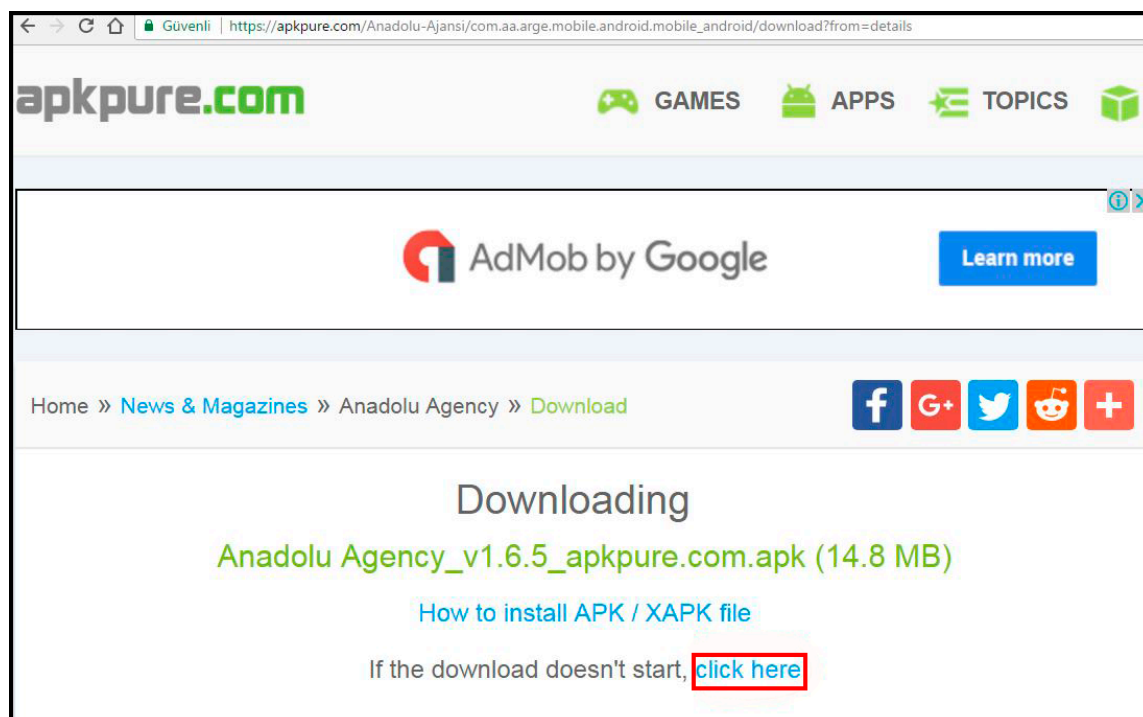
When Figure 5 is analyzed, the line marked with red is the section where the application information is located. The mouse is right-clicked on this marked section, then the Copy path is clicked, and finally the XPath is obtained by selecting Copy XPath. Using the obtained XPath, the image, package name, and application name information that the applications have will be obtained. With the Search Applications option, Android applications can be searched for by application name, application package name, or any desired words. In Figure 6, a search result based on the word football is shown.





**Figure 6.** User interface of the developed system showing the search results for the word “football”.

When Figure 6 is analyzed, the results are listed based on the searched word “football”. Analyses were conducted by selecting the chosen applications from the list. If the application has already been analyzed, the result is shown on the client side. If there was no analysis made earlier, the application must be downloaded first to the server. For downloading, the application is downloaded from the APKPure site first. A link such as “<https://apkpure.com/UploadName/packageName/download?from=details>” is created for this process. The most important thing to note here is whether there is a special character in the application name. Special characters in application names need to be replaced with a hyphen “-” character. This link is used for web scraping on the relevant site. “// \* [@ id=\\” download\_link \\\”]” XPath is used for web scraping. This link should result in the “click here” link, as shown in Figure 7, on the resulting web page.



**Figure 7.** A sample .apk download link on APKPure web page.

If the application cannot be downloaded, the apkleecher site is used as a second alternative. For this process, a link like “<http://apkleecher.com/download/dl.php?dl=PackageName>” is created. This link is used for web scraping on the relevant site. The XPath “//body//script[3]” is used for downloading the link.

If the application cannot be downloaded, apkbucket is used as a third alternative. This process has three steps. In the first step, a link, such as <https://apkbucket.net/search?s=PackageName>, is created. This link is used for web scraping on the relevant site. The XPath “/html/body/div/div/div/div/div[1]/div[1]/div/div[3]/a” is used for the web scraping process. In the second step, the XPath “//\*[@id=“downloadSection”]/p[2]/a” is used for the web scraping process. In the last step, the XPath “/html/body/div/div/div/div/div[1]/div[1]/div/div[2]/p[2]/strong/a” is used for getting download the link.

After the application is downloaded, reverse engineering is performed using Apktool. With reverse engineering, the name of the application, the package name, the permissions it uses, the targetSdkVersion and minSdkVersion information are registered in the database. After this information is obtained, information stored in the VirusTotal database is obtained using the VirusTotal API. There are two main constraints: the size of the application should not be larger than 32,766 kilobytes, and the ability to make four queries per minute. If the Android application size is larger than 32,766 kilobytes in the VirusTotal database, the registers will be used. However, if the reverse is the case, the relevant sections will be assigned a −2 value to distinguish them in the database of the relevant application, and the analysis result will only be shown by making an evaluation based on the malicious score of the application. If the results of files with a size of smaller than 32,766 kilobytes are in the VirusTotal database, the registers will be used. However, if not, the application is scanned using the VirusTotal API. This scanning process depends on the usage intensity of VirusTotal and the file size. Therefore, whether the process is finished or not is checked every 3 min using the scanning number assigned by VirusTotal. If this process takes more than 3 min, the result of the analysis is shown only by evaluating the malicious score of the application, in order not to keep the user waiting for too long. On the other hand, when the result from VirusTotal is obtained, it is registered in the database. For the 48 applications whose sizes were larger than 32,766 kilobytes in the benign and

malicious data sets, the VirusTotal web page was visited and the application was analyzed by manual downloading. Using the calculation method described above, the final malicious value of the application can be obtained. If this value is greater than the threshold value determined by the receiver operation characteristics curve, the application is considered to be malicious. If it is smaller than threshold value, the the application is considered to be benign and the result is shown to the client.

In the Install application option, the Android application on the client side is downloaded to the server for analysis. The Android application in the local file system is loaded on the server, and then the process, starting with the reverse engineering phase, mentioned in the application search option is implemented. After the subprocesses described in the application search and application install process are finalized, the analysis results are shown in Figure 8.

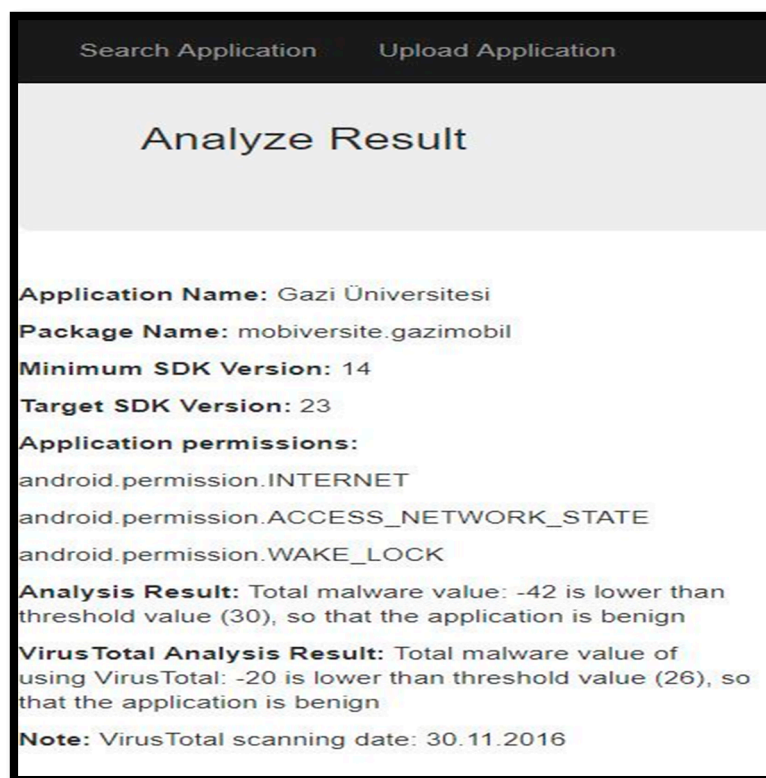


Figure 8. User interface showing the results of the application analysis.

Examining Figure 8, the result of the analysis shows the name of the application, the package name, the version of Android it runs on, the permissions it requests, and two analysis results. The calculation method used by the first analysis result in the evaluation based on the permissions found in the benign and malicious data sets. The calculation method used by the second analysis result, however, is the calculation that VirusTotal has added to the calculation used in the first analysis method. Furthermore, the user is provided with the date information of the scan made on VirusTotal.

#### 4. Results and Discussion

In this study, 5545 malicious applications from the Drebin data set and 1173 benign applications manually downloaded from the Google Play market were employed. In the developed web application, users can analyze any application based on the model created by using the information of the applications in the datasets. The permissions that the applications in the benign data set use and number of uses are shown in Table 2.

**Table 2.** 10 permissions used most in benign data set.

Permission Name	Total
android.permission.INTERNET	1306
android.permission.ACCESS_NETWORK_STATE	1142
android.permission.WRITE_EXTERNAL_STORAGE	930
android.permission.WAKE_LOCK	774
com.google.android.c2dm.permission.RECEIVE	679
android.permission.ACCESS_FINE_LOCATION	533
android.permission.VIBRATE	520
android.permission.ACCESS_COARSE_LOCATION	492
android.permission.GET_ACCOUNTS	473
android.permission.ACCESS_WIFI_STATE	462

When Table 2 is examined, the most commonly used permission is the INTERNET permission, found in 1173 benign data sets. Thus, almost all applications require the use of the Internet. The permissions that the applications in the malicious data set use and number of uses are shown in Table 3.

**Table 3.** 10 permissions used most in malicious data set.

Permission Name	Total
android.permission.INTERNET	5332
android.permission.READ_PHONE_STATE	4939
android.permission.WRITE_EXTERNAL_STORAGE	3722
android.permission.ACCESS_NETWORK_STATE	3677
android.permission.SEND_SMS	2992
android.permission.RECEIVE_BOOT_COMPLETED	2671
android.permission.ACCESS_WIFI_STATE	2430
android.permission.RECEIVE_SMS	2135
android.permission.WAKE_LOCK	2128
android.permission.READ_SMS	2081

When Table 3 is analyzed, the most commonly used permission in the malicious data set is the INTERNET permission, as in the case of the benign data set. The INTERNET, READ\_PHONE\_STATE, WAKE\_LOCK, WRITE\_EXTERNAL\_STORAGE, ACCESS\_WIFI\_STATE, and ACCESS\_NETWORK\_STATE permissions are common in the 10 permissions most commonly used in both the malicious and benign data sets. It is considered that it is natural for the permissions INTERNET, ACCESS\_WIFI\_STATE and ACCESS\_NETWORK\_STATE to be common in both benign and malicious data sets, since these permissions authorize the application to use the Internet. In addition, READ\_PHONE\_STATE, WRITE\_EXTERNAL\_STORAGE, SEND\_SMS, RECEIVE\_SMS, and READ\_SMS permissions are common in the permissions that Android has identified as dangerous permissions, and in the 10 most used permissions in the malicious data sets. Table 4 shows some of the results obtained as a result of the transactions carried out.

**Table 4.** Permissions with the highest and lowest malicious value.

Permission Name	Number of Occurrences in Malicious Data Set	Number of Occurrences in Benign Data Set	Rate of Occurrences in Malicious Data Set (%)	Rate of Occurrences in Benign Data Set (%)	Malicious Value
android.permission.READ_PHONE_STATE	4939	369	89	28	61
android.permission.SEND_SMS	2992	33	54	2	52
android.permission.READ_SMS	2081	27	38	2	36
android.permission.RECEIVE_SMS	2135	51	39	4	35
android.permission.RECEIVE_BOOT_COMPLETED	2671	296	48	22	26
com.android.launcher.permission.INSTALL_SHORTCUT	1414	55	26	4	22
android.permission.WRITE_SMS	1239	8	22	1	21
android.permission.CAMERA	227	274	4	21	−17
android.permission.READ_EXTERNAL_STORAGE	333	301	6	23	−17
android.permission.ACCESS_NETWORK_STATE	3677	1142	66	86	−20
android.permission.WAKE_LOCK	2128	774	38	58	−20
com.google.android.providers.gsf.permission.READ_GSERVICES	1	289	0	22	−22
android.permission.GET_ACCOUNTS	443	473	8	36	−28
com.google.android.c2dm.permission.RECEIVE	380	679	7	51	−44



When examining Table 4, the READ\_PHONE\_STATE permission is the one with the highest malicious value. This permission is followed by SEND\_SMS, READ\_SMS, RECEIVE\_SMS and RECEIVE\_BOOT\_COMPLETED permissions. READ\_PHONE\_STATE, SEND\_SMS, RECEIVE\_SMS, READ\_SMS, and READ\_CONTACTS are also permissions in common with the permissions that Android has identified as dangerous permissions.

The confusion matrix is used to calculate the evaluation results. Accuracy, error rate, true positive rate, false positive rate, true negative rate and false negative rate equations are shown below.

$$\text{Accuracy: } (TN + TP) / \text{Total} \quad (5)$$

$$\text{Error Rate: } 1 - \text{Accuracy} \quad (6)$$

$$\text{True Positive Rate: } TP / (FN + TP) \quad (7)$$

$$\text{False Positive Rate: } FP / (TN + FP) \quad (8)$$

$$\text{True Negative Rate: } TN / (TN + FP) \quad (9)$$

$$\text{False Negative Rate: } FN / (FN + TP) \quad (10)$$

True Positive (TP), number of malware applications that are classified truly as malware. False Negative (FN), number of malware applications that are classified incorrectly as benign. False Positive (FP), number of benign applications that are classified incorrectly as malware. True Negative (TN), number of benign applications that are classified truly as benign.

In this study, different methods were used to evaluate benign and malicious data sets. The first method is the evaluation based on 197 permissions that the applications possess in the data sets that are available in Android version of 4.1. In this method, while the malicious values of the applications are calculated, the permissions, except for these 197 permissions, are ignored. The malicious values of the apps are calculated by adding the malicious values of the Android 4.1 permissions that the applications possess. The results obtained based on this evaluation are shown in Table 5.

**Table 5.** Results of evaluation based on the permissions in Android version 4.1.

	Number of Correct Detection	Number of False Detection	Rate of Correct Detection (%)	Rate of False Detection (%)
Benign Data Set	1106	67	94.29	5.71
Malicious Data Set	5136	409	92.62	7.38
Overall	6242	476	92.91	7.09

When Table 5 is examined, the correct detection rate of the benign data set according to the relevant evaluation is 94.29%. The correct detection rate in the malicious dataset is 92.62%. The overall correct detection rate is 92.91%, and the overall false detection rate is 7.09%.

The second method is an evaluation based on the permissions used by the applications in the benign and malicious data sets. In this method, the malicious values of the applications are calculated by adding the malicious values of the permissions the applications possess. The result obtained based on this evaluation is shown in Table 6.

**Table 6.** Evaluation results based on the permissions the applications in the data sets possess.

	Number of Correct Detection	Number of False Detection	Rate of Correct Detection (%)	Rate of False Detection (%)
Benign Data Set	1129	44	96.25	3.75
Malicious Data Set	5164	381	93.13	6.87
Overall	6293	425	93.67	6.33

When Table 6 is examined, the correct detection rate of the benign data set according to relevant evaluation is 96.25%. The correct detection rate in the malicious dataset is 93.13%. The overall correct detection rate is 93.67%, and the overall false detection rate is 6.33%.

The third method is the evaluation in which the malicious values of the applications are calculated by using the VirusTotal values of the applications and the first method. That is, it is the evaluation made using VirusTotal values of the applications in the benign and malicious data sets and 197 permissions and applications in Android version 4.1. The obtained results based on this evaluation are shown in Table 7.

**Table 7.** Evaluation results based on the permissions in Android 4.1 and VirusTotal.

	Number of Correct Detection	Number of False Detection	Rate of Correct Detection (%)	Rate of False Detection (%)
Benign Data Set	1136	37	96.85	3.15
Malicious Data Set	5406	139	97.49	2.51
Overall	6542	176	97.38	2.62

When Table 7 is examined, the correct detection rate of the benign data set according to relevant evaluation is 96.85%. The correct detection rate in the malicious dataset is 97.49%. The overall correct detection rate is 97.38% and the overall false detection rate is 2.62%.

The fourth method is the evaluation in which the malicious values of the applications are calculated using the VirusTotal values of the applications and the second method. That is, it is the evaluation made using the VirusTotal values of the permissions and the applications that the applications in benign and malicious data sets use. The results obtained based on this evaluation are shown in Table 8.

**Table 8.** Evaluations results based on the permissions in the data sets and VirusTotal.

	Number of Correct Detection	Number of False Detection	Rate of Correct Detection (%)	Rate of False Detection (%)
Benign Data Set	1151	22	98.12	1.88
Malicious Data Set	5407	138	97.51	2.49
Overall	6558	160	97.62	2.38

When Table 8 is examined, the correct detection rate of the benign data set according to the relevant evaluation is 98.12%. The correct detection rate in the malicious dataset is 97.51%. The overall correct detection rate is 97.62%, and the overall false detection rate is 2.38%. Among these four methods, the method with the highest performance rate is the method using the permissions in benign and malicious data sets together with VirusTotal. The use of VirusTotal in the calculation method increased the success of the system as seen in the results given above. Moreover, evaluation based on permissions on the specified Android version was the least effective method. It is believed that this is the result of the exclusion of special permissions and the permissions released after Android version 4.1. The comparison of the developed system with studies using the static analysis method in the literature is shown in Table 9.

When Table 9 is examined, permissions and API calls are used as attributes in general. Genom and Drebin data sets have been commonly preferred for the evaluation of the proposed tools or methods in literature. The detection rate of malicious applications was used as an evaluation criterion. The proposed tools or methods were applied to both the malicious and the benign data sets. The success rates vary between 80% and 99.2%. Our study differs from the others because users can easily analyze Android applications by using our developed web application, and we used a web-scraping technique.

**Table 9.** Comparison of static analysis methods.

Studies	Attributes	Dataset	Success Rate (%)
DroidMat [7]	Permissions and API calls	238 Malicious Data Sets 1500 Benign Data Sets	97.87%
Drebin [9]	Android Manifest File and source codes	Drebin Malicious Data Set 123,453 Benign Data Sets	94%
Kayabasi [37]	Permissions and API calls	Drebin Malicious Data Set 1400 Benign Data Sets	99.20%
APK Auditor [11]	Permissions	Drebin and Genome Malicious Data Set 1853 Benign Data Sets	88.28%
Wang et al. [15]	Permissions, intentions, API calls, hardware features and info on coding	8701 Malicious Data Sets 107,327 Benign Data Sets	99.39% for Malicious Data Set
Wu et al. [51]	DexFile and API calls	1050 Malicious Data Sets 1160 Benign Data Sets	97.66%
Arslan et al. [18]	Dexfile and Permissions	50 Malicious Data Sets	97.62% For Malicious Data Set
		25 Benign Data Sets	80% for Benign Data Set
The developed system (based on permissions in Android 4.1 and VirusTotal)	Permissions	Drebin Malicious Data Sets 1173 Benign Data Sets	97.38%
The developed system (based on the permissions in data sets and VirusTotal)	Permissions	Drebin Malicious Data Sets 1173 Benign Data Sets	97.62%

## 5. Conclusions

In this study, the proposed system was developed to detect malicious applications independently from mobile device resources. The performances of the four different calculation models on the applications were tested with benign and malicious data sets. The highest success rate of these four calculation methods was the calculation method in which the permissions in the benign and malicious data sets were used together with VirusTotal, with a rate of 97.62%. The calculation method that demonstrated the lowest success rate of 92.91% was the calculation method that made the evaluation based on the permissions in Android Version 4.1. It is seen that the reason behind this failure is that it ignores the special permissions and permissions released after that Android version.

The most successful calculation model was applied to the proposed detection systems based on the client-server architecture to analyze applications that are not known to be benign or malicious. The users are able to analyze any Android application by uploading or searching in markets without installing the application on their mobile devices. Thus, possible impacts of the malicious applications on the mobile devices can be prevented in advance.

Within the framework of this study, it is important to ensure the use of the static analysis method, as well as the usage of other features found in the .apk file, in the remediation efforts of the system. For future studies, it is planned to conduct a study which analyzes the proposed system that is supported by dynamic analysis techniques, along with the static ones, to improve the accuracy of the developed system.

**Author Contributions:** Project administration, İ.A.D.; Software, İ.A.D. and Ö.K.; Writing—original draft, İ.A.D. and Ö.K.; Writing—review & editing, İ.A.D. and Ö.K.

**Funding:** This research received no external funding.

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

## References

1. IDC. Smartphone OS. Available online: <https://www.idc.com/promo/smartphone-market-share/os> (accessed on 20 August 2018).
2. G DATA. G DATA Mobile Malware Report H1 2016. Available online: [https://file.gdatasoftware.com/web/en/documents/whitepaper/G\\_DATA\\_Mobile\\_Malware\\_Report\\_H1\\_2016\\_EN.pdf](https://file.gdatasoftware.com/web/en/documents/whitepaper/G_DATA_Mobile_Malware_Report_H1_2016_EN.pdf) (accessed on 20 August 2018).
3. Cunningham, E. Keeping You Safe with Google Play Protect. Available online: <https://blog.google/products/android/google-play-protect/> (accessed on 20 August 2018).

4. Android. Google Play Protect. Available online: <https://www.android.com/play-protect/> (accessed on 20 August 2018).
5. Google Play. Available online: <https://play.google.com/store> (accessed on 20 August 2018).
6. Android Developers. Distribution Dashboard. Available online: <https://developer.android.com/about/dashboards/index.html> (accessed on 20 August 2018).
7. Wu, D.J.; Mao, C.H.; Wei, T.E.; Lee, H.M.; Wu, K.P. Droidmat: Android malware detection through manifest and api calls tracing. In Proceedings of the Seventh Asia Joint Conference on Information Security, Tokyo, Japan, 9–10 August 2012.
8. Apktool. A Tool for Reverse Engineering Android Apk Files. Available online: <https://ibotpeaches.github.io/Apktool/> (accessed on 20 August 2018).
9. Arp, D.; Spreitzenbarth, M.; Hubner, M.; Gascon, H.; Rieck, K. DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket. In Proceedings of the Network and Distributed System Security Symposium (NDSS), San Diego, CA, USA, 23–26 February 2014.
10. Wang, Z.; Chenlong, L.; Zhenlong, Y.; Guan, Y.; Xue, Y. DroidChain: A novel Android malware detection method based on behaviour chains. *Pervasive Mob. Comput.* **2016**, *32*, 3–14. [[CrossRef](#)]
11. Kabakus, A.T.; Dogru, I.A.; Cetin, A. APK Auditor: Permission-based Android malware detection system. *Digit. Investig.* **2015**, *13*, 1–14.
12. Kiraz, O.; Dogru, I.A. Android Malware Detection Systems Review. *Duzce Univ. J. Sci. Technol.* **2017**, *5*, 281–298.
13. Narayanan, A.; Liu, Y.; Chen, L.; Liu, J. Adaptive and Scalable Android Malware Detection through Online Learning. In Proceedings of the International Joint Conference on Neural Networks (IJCNN), Vancouver, BC, Canada, 24–29 July 2016.
14. Song, J.; Han, C.; Wang, K.; Zhao, J.; Ranjan, R.; Wang, L. An integrated static detection and analysis framework for android. *Pervasive Mob. Comput.* **2016**, *32*, 15–25. [[CrossRef](#)]
15. Wang, W.; Li, Y.; Wang, X.; Liu, J.; Zhang, X. Detecting android malicious apps and categorizing benign apps with ensemble of classifiers. *Future Gener. Comput. Syst.* **2017**, *78*, 987–994. [[CrossRef](#)]
16. Sokolova, K.; Perez, C.; Lemercier, M. Android Application Classification and Anomaly Detection with Graph-based Permission Patterns. *Decis. Support Syst.* **2017**, *93*, 62–76. [[CrossRef](#)]
17. Anwar, S.; Zain, J.M.; Inayat, Z.; Karim, A.; Haq, R.U.; Jabir, A.N. A Static Approach towards Mobile Botnet Detection. In Proceedings of the 3rd International Conference on Electronic Design (ICED), Phuket, Thailand, 11–12 August 2016.
18. Arslan, R.S.; Dogru, I.A.; Barisci, N. Permisson Comparison Based Malware Detection System for Android Mobile Applications. *J. Polytech.* **2017**, *20*, 175–189.
19. Dex2jar. Tools to Work with Android .dex and java .class Files. Available online: <https://github.com/pxb1988/dex2jar> (accessed on 20 August 2018).
20. Feizollah, A.; Anuar, N.B.; Salleh, R.; Suarez-Tangil, G.; Furnell, S. AndroDialysis: Analysis of Android Intent Effectiveness in Malware Detection. *Comput. Secur.* **2016**, *65*, 121–134. [[CrossRef](#)]
21. Kang, H.; Jang, J.-W.; Mohaisen, A.; Kim, H.K. Detecting and Classifying Android Malware Using Static Analysis along with Creator Information. *Int. J. Distrib. Sens. Netw.* **2015**, 1–9. [[CrossRef](#)]
22. Goyal, R.; Spognardi, A.; Dragoni, N.; Argyriou, M. SafeDroid: A Distributed Malware Detection Service for Android. In Proceedings of the IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA), Macau, China, 4–6 November 2016.
23. Utku, A.; Dogru, I.A. Permission based Detection System for Android Malware. *J. Fac. Eng. Archit. Gazi Univ.* **2017**, *32*, 1015–1024.
24. Atici, M.A.; Sagioglu, S.; Dogru, I.A. Android malware analysis approach based on control flow graphs and machine learning algorithms. In Proceedings of the 4th International Symposium on Digital Forensic and Security (ISDFS), Little Rock, AR, USA, 25–27 April 2016.
25. Shabtai, A.; Tenenboim-Chekina, L.; Mimran, D.; Rokach, L.; Shapira, B.; Elovici, Y. Mobile malware detection through analysis of deviations in application network behavior. *Comput. Secur.* **2014**, *43*, 1–18. [[CrossRef](#)]
26. Heuser, S.; Negro, M.; Pendyala, P.K.; Sadeghi, A.-R. DroidAuditor: Forensic Analysis of Application-Layer Privilege Escalation Attacks on Android. In Proceedings of the Financial Cryptography and Data Security (FC 2016), Christ Church, Barbados, 26 February 2016.
27. Jang, J.-W.; Yun, J.; Mohaisen, A.; Woo, J.; Kim, H.K. Detecting and classifying method based on similarity matching of Android malware behavior with profile. *SpringerPlus* **2016**, *5*, 1–23. [[CrossRef](#)] [[PubMed](#)]

28. Garg, S.; Peddoju, S.K.; Sarje, A.K. Network-based detection of Android malicious apps. *Int. J. Inf. Secur.* **2016**, *16*, 385–400. [CrossRef]
29. Chang, W.-L.; Sun, H.-M.; Wu, W. An Android Behavior-Based Malware Detection Method using Machine Learning. In Proceedings of the 2016 IEEE International Conference on Signal Processing, Communications and Computing (ICSPCC), Hong Kong, China, 5–8 August 2016.
30. Lantz, P. An Android application sandbox for dynamic analysis. Bachelor's Thesis, Department of Electrics and Information Technologies, Lund University, Lund, Sweden, November 2011.
31. Shi, Y.; You, W.; Qian, K.; Bhattacharya, P.; Qian, Y. A Hybrid Analysis for Mobile Security Threat Detection. In Proceedings of the IEEE 7th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON 2016), New York, NY, USA, 20–22 October 2016.
32. Cole, Y.; Zhang, H.; Ge, L.; Wei, S.; Yu, W.; Lu, C.; Chen, G.; Shen, D.; Blasch, E.; Pham, K.D. ScanMe Mobile: A Local and Cloud Hybrid Service for Analyzing APKs. In Proceedings of the Research in Adaptive and Convergent Systems (RACS 2015), Prague, Czech Republic, 9–12 October 2015.
33. Singh, S.; Mishra, B.; Singh, S. Detecting Intelligent Malware on Dynamic Android Analysis Environments. In Proceedings of the 10th International Conference for Internet Technology and Secured Transactions (ICITST 2015), London, UK, 14–16 December 2015.
34. Wang, H.; Guo, Y.; Tang, Z.; Bai, G.; Chen, X. Re-evaluating Android Permission Gaps with Static and Dynamic Analysis. In Proceedings of the IEEE Global Communications Conference (GLOBECOM 2015), San Diego, CA, USA, 6–10 December 2015.
35. Apkpure. Available online: <https://apkpure.com/> (accessed on 20 August 2018).
36. The Drebin Dataset. Available online: <https://www.sec.cs.tu-bs.de/~danarp/drebin/> (accessed on 20 August 2018).
37. Kayabasi, G. Classification of Android Applications through Permission Based Static Analysis. Bachelor's Thesis, Gazi University, Institute of Science, Ankara, Turkey, December 2016.
38. VirusTotal. VirusTotal Public API v2.0. Available online: <https://www.virustotal.com/en/documentation/public-api> (accessed on 20 August 2018).
39. Kang, B.; Yerima, S.Y.; McLaughlin, K.; Sezer, S. N-opcode Analysis for Android Malware Classification and Categorization. In Proceedings of the International Conference on Cyber Security and Protection of Digital Services (Cyber Security 2016), London, UK, 13–14 June 2016.
40. Ma, L.; Wang, X.; Yang, Y.; He, J. Ultra-lightweight Malware Detection of Android Using 2-level Machine Learning. In Proceedings of the 3rd International Conference on Information Science and Control Engineering (ICISCE 2016), Beijing, China, 8–10 July 2016.
41. Xu, K.; Li, Y.; Deng, R.H. ICCDetector: ICC-Based Malware Detection on Android. *IEEE Trans. Inf. Forensics Secur.* **2016**, *11*, 1252–1264. [CrossRef]
42. Shehu, Z.; Ciccotelli, C.; Ucci, D.; Aniello, L.; Baldoni, R. Towards the Usage of Invariant-based App Behavioral Fingerprinting for the Detection of Obfuscated Versions of Known Malware. In Proceedings of the 10th International Conference on Next Generation Mobile Applications, Security and Technologies (NGMAST 2016), Cardiff, UK, 24–26 August 2016.
43. Ban, T.; Takahashi, T.; Guo, S.; Inoue, D.; Nakao, K. Integration of Multi-modal Features for Android Malware Detection Using Linear SVM. In Proceedings of the 11th Asia Joint Conference on Information Security (AsiaJCIS 2016), Fukuoka, Japan, 4–5 August 2016.
44. VirusTotal, About VirusTotal. Available online: <https://virustotal.com/en/about/> (accessed on 20 August 2018).
45. Spreitzenbarth, M.; Schreck, T.; Ehtler, F.; Arp, D.; Hoffmann, J. Mobile-Sandbox: Combining static and dynamic analysis with machine-learning techniques. *Int. J. Inf. Secur.* **2014**, *14*, 141–153. [CrossRef]
46. Tomak, L.; Bek, Y. Operation Characteristics Curve and Comparison of Regions under Curve. *J. Exp. Clin. Med.* **2009**, *27*, 58–65. [CrossRef]
47. University of Nebraska Medical Center. The Area under an ROC Curve. Available online: <http://gim.unmc.edu/dxtests/roc3.htm> (accessed on 20 August 2018).
48. Accord.NET Framework. Accord.Statistics.Analysis Namespace. Available online: [http://accord-framework.net/docs/html/N\\_Accord\\_Statistics\\_Analysis.htm](http://accord-framework.net/docs/html/N_Accord_Statistics_Analysis.htm) (accessed on 20 August 2018).
49. Apklecher.com. Available online: <http://apklecher.com/> (accessed on 20 August 2018).



50. APKBucket. Available online: <https://apkbucket.net/> (accessed on 20 August 2018).
51. Wu, S.; Wang, P.; Li, X.; Zhang, Y. Effective Detection of Android Malware Based on the Usage of Data Flow APIs and Machine Learning. *Inf. Softw. Technol.* **2016**, *75*, 17–25. [[CrossRef](#)]



© 2018 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 (<http://creativecommons.org/licenses/by/4.0/>).