Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2017) 25: 976 – 994
© TÜBİTAK
doi:10.3906/elk-1503-127

Implementation of a web-based service for mobile application risk assessment

Asım Sinan YÜKSEL[1,*], Mehmet Erkan YÜKSEL[2], Ahmet SERTBAŞ[3], Abdül Halim ZAİM[4]

[1]Department of Computer Engineering, Süleyman Demirel University, Isparta, Turkey
[2]Department of Computer Engineering, Mehmet Akif Ersoy University, Burdur, Turkey
[3]Department of Computer Engineering, İstanbul University, İstanbul, Turkey
[4]Department of Computer Engineering, İstanbul Commerce University, İstanbul, Turkey

Received: 16.03.2015 • **Accepted/Published Online:** 15.03.2016 • **Final Version:** 10.04.2017

Abstract: The Android operating system has increased in popularity and has been increasing its shares in the smart phone market. Users can carry out their daily work such as paying bills, being social, and sharing photos through mobile applications. These applications have access to sensitive information about the user, such as location, contacts, call logs, and SMS messages. However, the users have no knowledge of the applications or the personal information these applications have access to. Even if an application is not malware or does not have malicious behavior, it can compromise the security and privacy of the user by accessing the permissions and gathering sensitive personal information. In this study, we have designed and implemented a prototype of a novel fuzzy risk inference system that serves as a web-based service. The system analyzes the risks related to Android-based mobile applications and performs risk scoring by taking several features into account. The system presents the user with the risks of exposure before the installation of applications on the user's device and serves as an intelligent decision support system.

Key words: Mobile security, mobile risk analysis, mobile application security, security risk analysis, risk assessment, decision support systems, information systems

1. Introduction

Smart phones, tablets, or any other smart devices operate on mobile operating systems. Among these operating systems, Android obtained the highest market share. According to data for May 2014, which was published by Google in the Google I/O conference, over 900 million Android devices have been activated [1]. More than 25 million applications have been downloaded from the Google Play Store as of July 2014. This open-source platform has become the center of attention as applications are developed and uploaded to the Google Play Store easily by any smart phone user who signs up for a developer account. Thus, security and privacy risks of this platform increased and are still increasing.

In the literature, there have been many studies [2–11] to present solutions for users' security and privacy problems and make the Android operating system more secure. Although these studies offer insights for our study, they lack certain security and privacy-centric features such as risk analysis, risk scoring, and user-centric services. In our previous study [12], we analyzed these proposed solutions in detail and created a mobile security taxonomy. We also mentioned the need for a user-friendly, device-independent, web-based approach.

Accordingly, we examined the web-based analysis services that are different from the proposed solutions

*Correspondence: asimyuksel@sdu.edu.tr

mentioned in our previous study and have similar features with our proposed solution in this study. Dexter [13] is one of these tools, which was developed as a software framework that analyzes Android-based applications and has a web-based interface. The developed tool extracts information from normal and malicious applications as much as it can and presents the results on the web in different formats. Dexter was developed as a web application by using the Python programming language. XML caching is used because of performance concerns related to SQL databases.

Another tool is Andrubis [14]. It is a fully automatic Android application analysis tool that combines static and dynamic analysis methods on the system and Dalvik virtual machine levels. It has a web interface that can be accessed online and it allows the uploading of mobile applications. It is capable of analyzing 3500 samples per day. To date, 1,000,000 Android applications have been analyzed in total. AVC Undroid [15] is a web-based free-of-charge service that examines Android applications with a static analysis technique. Registered users can analyze larger files, comment on the analyzed applications, and achieve higher priorities for analysis. Besides applications that users upload, it can gather Android applications from the background. Results are for information only and do not assist in making any decision on the file. The service does not guarantee that applications are dangerous or malicious. Similar to Andrubis, Mobile-Sanbox [16] can perform web-based Android application analysis and trace method calls in applications by using static and dynamic analysis techniques. It can also examine compiled application files for suspicious activity. CopperDroid [17] performs analysis by employing a dynamic analysis technique. The developed tool uses a virtual machine-based approach. By analyzing system calls and employing simulation methods, it analyzes malicious applications and modifies their behaviors. Users can analyze applications by uploading APK files via web interface. Analysis results can be shown in various formats and contain information such as network traffic, modified normal and executable files, and system call traces.

In this study, we adopted a hybrid approach and created a service-based solution that examines the source code, works on permissions, and performs fuzzy logic-based risk scoring by taking permission information and scanning results of state-of-the-art antivirus software tools into account. Therefore, a novel web-based system for mobile application risk analysis and assessment is designed and developed to remedy the shortcomings in the previous studies and to provide a user-centric solution to security problems. The service is implemented as a device-independent, cloud-based, and free service that requires no additional software to be installed on the user's devices. As a result, the system serves as an intelligent decision support system where the users will have knowledge of the risks related to the applications before downloading and be aware of the threats related to their personal security and privacy.

2. System design

Architectural design is considered as a part of the system to ensure that the system meets users' requirements. In this design, software operations and modules are described using figures. This model will provide a better understanding of the architecture and our goals. Figure 1 shows the system architecture and explains the main components of our system and how these components interact to achieve the goals we set. The designed mobile security risk analysis service consists of four main modules. These are permission extraction, permission analysis, risk scoring, and risk reporting modules. These modules are described in the following sections.

2.1. Permission extraction module

Android applications are installed using a file with an ".apk" extension. The file labeled as "AndroidManifest.xml" in this APK file contains the application permissions. This file acts as a control mechanism that tells

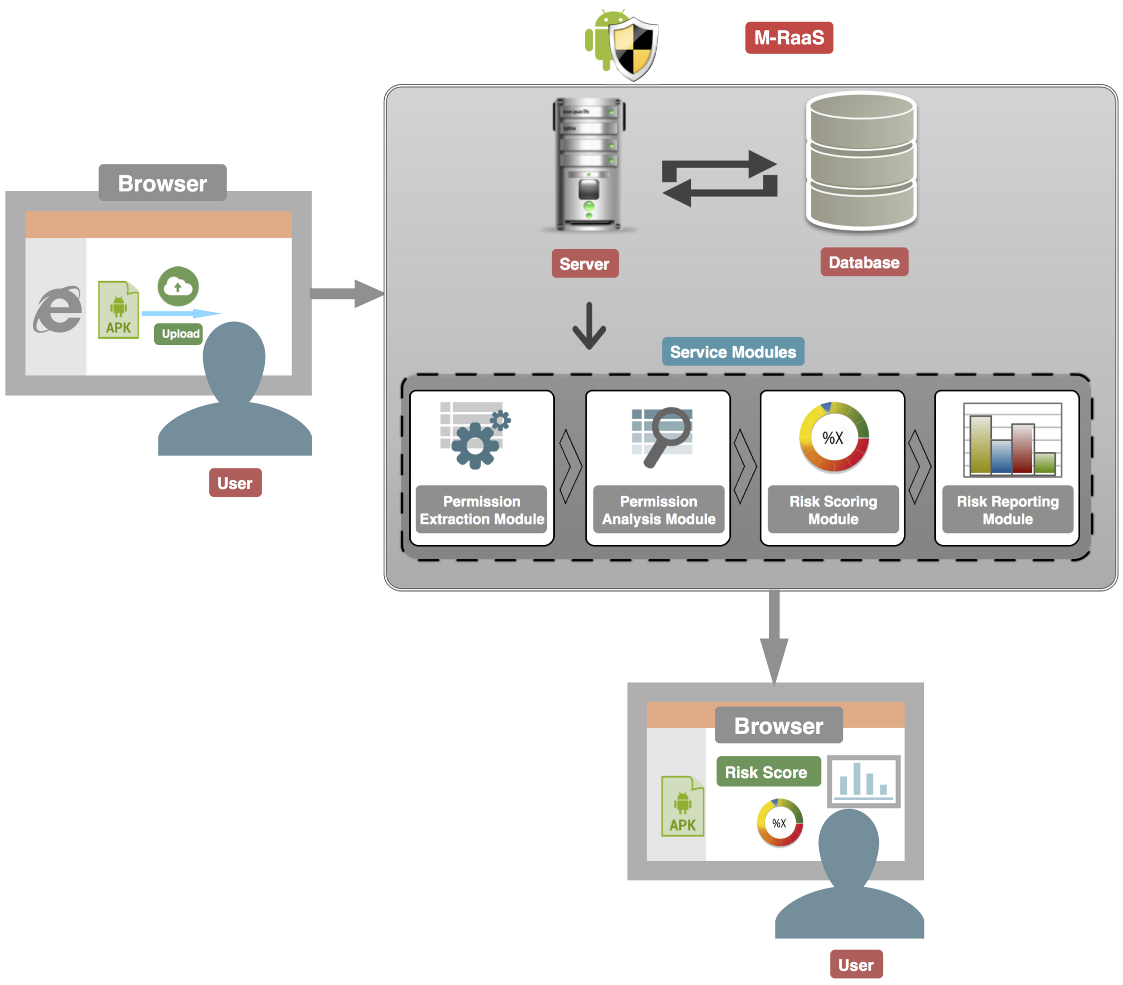


Figure 1. System architecture.

the system what needs to be done with the top-level components of the Android operating system. In this module, our main purpose is to extract the permissions in this file (AndroidManifest.xml) for risk scoring. The application installation file (.apk file) that a user uploads to the system or the application that is returned from a search result will be the input in this module. The output of this module is the permissions that are contained in the application file that was successfully uploaded to the system. Thus, the extracted permission list will be the input of the permission analysis module. A unique identity will be assigned to each application to prevent duplicate versions of the application from being installed on the system. This information will be stored in the database by establishing the Application-Permissions relation.

2.2. Permission analysis module

The "AndroidManifest.xml" file that is obtained in the permission extraction module is an XML file in binary format and its content cannot be understood. This file needs to be converted to a readable, typical XML file. This module works in two steps. In the first step, the binary XML file is converted to a typical readable XML file. In the second step, permissions are extracted from the file using "uses-permission" tag information. Extracted information (READ_CONTACTS, INTERNET, etc.) will be the input for the risk scoring module.

2.3. Risk scoring module

In this module, the risk score is produced by using the permissions obtained in the previous module and antivirus results obtained from the VirusTotal service. VirusTotal is a free web service that analyzes files and URLs and identifies malicious contents by scanning them through antivirus engines and website scanners. All the antivirus engines are regularly updated and have the latest version of the signatures. The main features of VirusTotal are: free unbiased service; scanning through multiple popular, state-of-the-art antivirus tools; containing multiple file and URL characterization tools; virus signatures that are regularly updated; and provision of detailed results from each scanner and an API for the research community. All these features make the VirusTotal service ideal for researchers. Therefore, we chose to use VirusTotal for saving time from installing various antivirus tools and licensing costs, and because of its characteristics [18].

Depending on the obtained risk score, the installed application will be classified as "X%" risky. Here, the closer the application risk value is to 100, the higher the risk related to the application. In addition to the risk score, the system also shows whether the analyzed application is malware or not according to our malware database and results obtained from VirusTotal service.

2.4. Risk reporting module

The purpose of this module is to present the user with clear information about the application to be installed. The risk score obtained in the previous modules and the extracted permissions will be graphically displayed. Additionally, the screen will display the permissions the application uses, what the permissions are being used for, unnecessary permissions, dangerous shell commands, and the potential dangers if the application is installed with these permissions.

3. Design of the fuzzy logic-based risk scoring algorithm

The risk scoring system is designed as a process consisting of 5 steps. These steps are:

1. **Listing permissions and identifying their influences:** In this stage, permissions that are most frequently used by malware are found and their influence degrees are calculated. For this purpose, the mobile malware datasets used in [19] and [20] are used as the most comprehensive datasets. The first dataset includes 1260 malwares in 49 different categories and the second dataset contains 5560 malwares in 179 categories. As a result of our analysis study, the most frequently used permissions by malware were identified. Table 1 shows the frequency of usage and the probability values of the top ten permissions for the sake of brevity. The linguistic values equivalent to the probability values are presented in Table 2.

2. **Identifying linguistic risk values and categories:** In this step, the linguistic and numerical risk values as well as the categories of the permissions that cause risks are identified. Taking the permission explanations in the Android system into account, four risk categories and linguistic and numerical influence values for each category are identified. These are listed in Table 3. These risk categories are created according to the protection levels assigned to the permissions in the Android operating system. Each permission that is listed in Table 1 is included in one of these categories. Using the values in Tables 2 and 3, the risk-influence-possibility matrix in Table 4 is obtained. This table will be used to identify the risk values of permissions in the fuzzy logic-based risk scoring process. "Risk intensity" is obtained by multiplying each permission's influence and probability values. According to values obtained by using Table 4, risk intensities of permissions are produced. The top ten permissions that are most used by malware and the risk intensities associated with them are shown in Table 5.

Table 1. Top ten permissions and their frequencies of usage and risk probabilities.

Permissions	Frequency	Risk probability (%)
INTERNET	1219	97
READ_PHONE_STATE	1166	93
ACCESS_NETWORK_STATE	1022	81
WRITE_EXTERNAL_STORAGE	834	66
ACCESS_WIFI_STATE	804	64
READ_SMS	790	63
RECEIVE_BOOT_COMPLETED	688	55
WRITE_SMS	658	52
SEND_SMS	540	43
RECEIVE_SMS	486	39

Table 2. Linguistic and numerical probability values.

Probability (%)	Linguistic value	Numerical value
75–100	High	4
50–75	Average	3
25–50	Normal	2
0–25	Low	1

Table 3. Risk categories and influence values.

Risk ID	Risk category	Linguistic value	Influence values
1	Dangerous	Very high	25
2	SystemOrSignature	High	15
3	Signature	Moderate	10
4	Normal Risk	Low	5

Table 4. Risk-influence-probability matrix.

Probability	Influence			
	Low (5)	Moderate (10)	High (15)	Very High (25)
Very high (4)	20	40	60	100
High (3)	15	30	45	75
Moderate (2)	10	20	30	50
Low (1)	5	10	15	25

3. **Identifying rules:** The fuzzy rules table is created in this step by taking the linguistic and numerical values, risk categories, malware detection ratio, unnecessary permission usage ratio, and dangerous shell command usage ratio into account. This table defines the fuzzy sets in conditional expressions. A single risk value is obtained by examining all of the permissions and extra features related to the malicious behavior by executing these rules.

4. **Inference:** For the risk inference, the most used and well-known technique, the Mamdani fuzzy model, is used [21]. We applied the most frequently used composition technique, which is the "min-max" fuzzy relation, for the inference. The system works as follows: first, the membership degrees are identified according to the crisp values. These degrees are input to the "min" operator. The fuzzy sets are obtained

Table 5. Top 10 permissions used by malware and risk intensities.

Permissions	Risk intensity
INTERNET	High
WRITE_EXTERNAL_STORAGE	High
READ_SMS	High
READ_PHONE_STATE	High
WRITE_SMS	Moderate
SEND_SMS	Moderate
RECEIVE_SMS	Low
ACCESS_COARSE_LOCATION	Low
READ_CONTACTS	Low
ACCESS_FINE_LOCATION	Low

by applying the operator for each rule. These fuzzy sets are then input to the max operator. To reach the final values, the output fuzzy set must be defuzzified.

5. **Defuzzification:** In this step, crisp values that are obtained from a fuzzy set are converted to a final value. To obtain the final value, a defuzzification method such as the centroid of area, smallest of maximum, or mean of maximum is applied. In this study, we have adopted the most frequently used method, which is the centroid of area. Risk categories defined in Step 3, malware detection ratio, unnecessary permission usage, and dangerous shell command usage ratios constitute the inputs of our fuzzy system. The fuzzy logic-based risk scoring module creates a risk score between 0 and 100 by assessing fuzzy inference rules, the fuzzy sets, and the inputs. Values close to 100 represent higher risk applications. If the system is sure about the maliciousness of the application according to the antivirus scanning results, 100 is assigned as the risk score.

3.1. Mathematical model of the risk scoring system

The parameters of the mathematical model are defined as below.

- *DRS:* Total risk score of permissions in Dangerous (DR) risk category.
- *SSRS:* Total risk score of permissions in SignatureOrSystem (SSR) risk category.
- *SRS:* Total risk score of permissions in Signature (SR) risk category.
- *NRS:* Total risk score of permissions in Normal (NR) risk category.
- *MRS:* Malicious risk score of the application.
- *UPUR:* Unnecessary permission usage ratio of the application.
- *SCUR:* Shell command usage ratio of the application.
- *BRS:* Behavioral risk score of the application (MRS+UPUR+SCUR).
- k : Name of the risk category (DR, SSR, SR, NR) that will be processed.
- *RT:* Risk type (Very High (VH), High (H), Moderate (M), Low (L)).

- P_n : nth permission.
- $RI_k\left(P_n\right)$: Risk influence value of nth permission in category k.
- $RP_{RT}\left(P_n\right)$: Risk probability value of nth permission with risk type RT.
- $RIV\left(P_n\right)$: Risk intensity value of nth permission, $\left(RI_k\left(P_n\right) \times RP_{RT}\left(P_n\right)\right)$.

In our model, the first step is the calculation of a behavioral risk score (BRS) that is the combination of MRS, UPUR, and SCUR. First, MRS is calculated according to the report obtained from the VirusTotal service. If the application is identified as malware by a majority (51%, 28 tools out of 55) of the antivirus tools, the application is marked as malicious and 100 is assigned as the risk score. If detection ratio is below 51%, the detection ratio is assigned as the MRS. We calculate the unnecessary permission usage ratio (UPUR) and the shell command usage ratio (SCUR) as follows:

- *UPUR:* This is the ratio of how many permissions were not used although the application defined them in its manifest file. The application may also use different permissions that are not defined in the manifest file. Our system punishes the applications whose unnecessary permission rate is greater than 27% and increases their risk scores accordingly. This 27% is an average value and obtained by analyzing the applications that we downloaded from Google Play Store and updated frequently.
- *SCUR:* This is the ratio of how many dangerous Linux shell commands were used in the application. For the purpose of identifying embedded shell commands, we created a complete list of Linux shell commands that can be used by Android-based applications. The ratio is the number of shell commands divided by the number of available commands in our list that is normalized to 100%.

In the second step, DRS, SSRS, SRS, and NRS are calculated.

- For $k = DR$, DRS is calculated as follows:

$$RIV\left(P_n\right) = RP_{VH}\left(P_n\right) * RI_{DR}\left(P_n\right) \tag{1}$$

$$DRS = \sum_{i=0}^{n} RIV\left(P_n\right) \tag{2}$$

- For $k = SSR$, SSRS is calculated as follows:

$$RIV\left(P_n\right) = RP_{H}\left(P_n\right) * RI_{SSR}\left(P_n\right) \tag{3}$$

$$SSRS = \sum_{i=0}^{n} RIV\left(P_n\right) \tag{4}$$

- For $k = SR$, SRS is calculated as follows:

$$RIV\left(P_n\right) = RP_{M}\left(P_n\right) * RI_{SR}\left(P_n\right) \tag{5}$$

$$SRS = \sum_{i=0}^{n} RIV\left(P_n\right) \tag{6}$$

- For $k = NR$, NRS is calculated as follows:

$$RIV\left(P_n\right) = RP_L\left(P_n\right) * RI_{NR}\left(P_n\right) \tag{7}$$

$$NRS = \sum_{i=0}^{n} RIV\left(P_n\right) \tag{8}$$

DRS, SSRS, SRS, and NRS are sent to our first Mamdani-based fuzzy system as input parameters. These inputs are then fuzzified according to the formula for the following triangular membership function:

$$\left\{ \begin{array}{ccc} 0 & , & x < a \\ \frac{x-a}{b-a} & , & a \leq x \leq b \\ \frac{c-x}{c-b} & , & b \leq x \leq c \\ 0 & , & x > c \end{array} \right\} \tag{9}$$

Rules are executed by using "IF-THEN" clauses and applying MIN and MAX composition. This can be formulated as below:

$$\mu_{A\cap B}\left(x\right) = MIN\left(\mu_A\left(x\right), \mu_B\left(x\right)\right) \tag{10}$$

$$\mu_{A\cup B}\left(x\right) = MAX\left(\mu_A\left(x\right), \mu_B\left(x\right)\right) \tag{11}$$

Let K_1 and K_2 be two fuzzy relations defined in $X*Y$ and $Y*Z$. The $MIN-MAX$ composition $(K_1\,OK_2)$ of K_1 and K_2 is a fuzzy set and defined as below:

$$K_1OK_2\left\{\ \left[\ (x,\, z)\,,\ MINMAX\left(\mu_{K_1(x,\,y)},\ \mu_{K_2(xy\,z)}\right)\ |\, x \in X,\, y \in Y,\, z \in Z\right]\ \right\} \tag{12}$$

In the last step, BRS and the value obtained from the previous state are sent as inputs to our second Mamdani fuzzy system, and the fuzzy rules are executed. This value is a fuzzy value and defuzzified to produce a numerical risk value by applying center of gravity (CoG), a well-known defuzzification method. In this method, the CoG of the area that is produced by the membership function is found and a crisp value of the fuzzy value is obtained, which is the final risk score. The equation below shows the application of the CoG method.

$$CoG\left(RT\right) = \frac{\sum \mu_{RT(x)} * x}{\sum \mu_{RT(x)}} \tag{13}$$

4. System implementation

We have developed a prototype of our system that runs on the Amazon Elastic Compute Cloud (Amazon EC2). Amazon EC2 provides resizable storage and computing capacity in the cloud, and it provides easy-to-use configuration utilities and virtual servers to make cloud computing easier. We have deployed a virtual Ubuntu Trusty 14.04 server on the Amazon EC2 free tier (automatic scaling is enabled) and assigned an elastic IP to access the web interface. The risk scoring module runs as a background service and allows the risk reporting module to call its procedures remotely. The risk scoring module is the heart of the system and contains our fuzzy logic implementation.

4.1. Risk scoring module

In this module, antivirus results [18], permissions and their details [22, 23], unnecessary permission usage, and shell command usage ratios are the inputs for the risk scoring process. The mathematical model that we created in the design phase is employed and risk information is generated. Figure 2 shows the algorithm that generates the risk information.

```
Input: APK file
Output: AndroidManifest file
extractPermission(apkFile):
  zipFile=readAsZip(apkFile)
    FOR EACH i IN zipFile.contents():
      IF i = "AndroidManifest.xml" THEN
        androidXML[i]=XMLConverter(zipFile.read(i))
      ENDIF
    ENDFOR
classicalXML[i] = parseXML(androidXML[i].readBuffer())
RETURN classicalXML
XMLConverter(data):
  androidxml =AndroidXMLParser(data)
  buffer= u''
  WHILE True AND androidxml.isValid():
    type = androidxml.next()
    IF  type= XML_START THEN
      buffer+= u'<?xml version="1.0" encoding="utf-8"?>\n'
    ELSE IF type= START_TAG THEN
      buffer+= u'<' + getAttribute(androidxml.getAttribute()) + androidxml.getName() + u'\n'
      buffer+= androidxml.getXMLNameSpace()
      FOR EACH i IN axml.getAttributeCount():
        buffer+= "%s%s=\"%s\"\n" % ( getAttribute(androidxml.getProperty(i)),
        axml.getPropertyName(i),  getPropertyValue(i))
      ENDFOR
      buffer+= u'>\n'
    ELSE IF type = END_TAG THEN
      buffer+= "</%s%s>\n" % ( getAttribute( axml.getAttribute()), androidxml.getName())
    ELSE IF  type= TEXT THEN
      buffer+= "%s\n" % androidxml.getText()
    ELSE IF  type = XML_END THEN
      break
  ENDWHILE
RETURN buffer
```

Figure 2. Algorithm that generates risk information.

After generating the risk information, we apply our fuzzy logic-based risk-scoring algorithm and generate risk scores. The algorithm that creates the necessary risk inputs is shown in Figure 3. The rules were generated by taking the risk categories and malware features into account. After the rule creation process, the membership functions are applied. Figure 4 shows the process that produces the fuzzy membership functions. The last step in our fuzzy model is generating the output membership function and applying fuzzy inference. Figures 5 and 6 show these processes.

4.2. Risk reporting module

This module consists of two main pages. The first page serves as a home page and handles the searching and uploading of Android applications. The second page is where the risk scores and the risk details are displayed.

After the successful upload or search of an Android application, it communicates with the risk scoring module and passes the file name/search string as an argument. The risk scoring module runs the analysis script with the provided argument and returns the results to the reporting module as a JSON string. It parses this information and shows it in a user-friendly format. Figure 7 shows a sample risk report. The prototype of the service can be accessed via the service URL http://muras.asimsinanyuksel.com/.

5. Results

The majority of the proposed solutions in the mobile security field that we analyzed in our previous study are not user-centric or they have not been submitted to the service of users. The solutions that are developed as antivirus software cause too much overload on users' devices. Other solutions offer security by modifying the Android operating system. However, these solutions cannot operate independently of the device and are not officially published.

The web-based services that we analyzed in this study are developed for analyzing Android applications and actively working. Therefore, we compared them with our Mobile Application Risk Analysis Service (Muras) that we developed according to the certain features they have. We accessed the actively working services, used our malware dataset and Google Play Store datasets, tested the services, and examined the results according to certain criteria. The criteria and their descriptions that we used in our comparison are as follows:

- ***Risk scoring:*** Does the system assign risk scores to applications based on a certain approach?
- ***Permission risk levels:*** Does the system show the risk levels of permissions that are defined in an application?
- ***Report:*** Does the system produce risk-related, easy-to-understand reports?
- ***APK upload:*** Does the system allow uploading of APK files?
- ***Search:*** Does the system provide a searching mechanism to search and autoanalyze Google Play Store applications without uploading a file?
- ***Unnecessary permission usage detection:*** Is the system capable of detecting permissions that are defined but not actually used?
- ***Shell command detection:*** Is the system capable of detecting dangerous Linux shell commands?
- ***Antivirus integration:*** Is the system integrated with state-of-the-art antivirus tools?

Table 6 shows the comparison results. Dexter, Anubis, Mobile-Sandbox, and CopperDroid can analyze uploaded APK files and produce reports. However, they lack certain features such as risk scoring, permission risk levels, and searching through Google Play Store applications. Unlike others, AVC Undroid shows permission risk levels. Although it does not show a risk value based on a score, it assigns various colors to applications as scores. However, how these colors are assigned or what algorithm is used is not explained. Although the system we developed does not analyze applications in virtual machines by employing the dynamic analyzing method used in CopperDroid, Mobile-Sandbox, and Andrubis devices, such a method can be integrated with our system easily and will be integrated as a future work.

If a mobile application is not identified as malware by antivirus tools, it does not mean it cannot harm the user. Applications can also damage users by acquiring personal information via permissions without showing

```
Input: N/A
Output: Risk inputs
1:  generateRiskInputs() :
2:      fuzzy_system = FuzzySystem()
3:      input_dangerous_risk = FuzzyInputVariable()
4:      input_normal_risk = FuzzyInputVariable()
5:      input_signature_risk = FuzzyInputVariable()
6:      input_signatureorsystem_risk=FuzzyInputVariable()
```

Figure 3. Algorithm that generates risk inputs.

```
Input: Risk inputs
Output: Membership functions
1: createMembershipFunctions:
2:     fuzzy_system.variable["dangerous_risk"] = input_dangerous_risk
3:     input_dangerous_risk.membership [LOW_RISK] = TriangularMF("Low Risk Range")
4:     input_dangerous_risk.membership[AVERAGE_RISK] = TriangularMF ("Average Risk Range")
5:     input_dangerous_risk.membership [HIGH_RISK] = TriangularMF("High Risk Range")
6:     input_dangerous_risk.membership [VERY_HIGH_RISK] = TriangularMF("Very High Risk Range")
7:     fuzzy_system.variable["normal_risk"] = input_normal_risk
8:     input_normal_risk.membership[LOW_RISK] = TriangularMF("Low Risk Range")
9:     input_normal_risk.membership [AVERAGE_RISK] = TriangularMF("Average Risk Range")
10:    input_normal_risk.membership [HIGH_RISK] = TriangularMF("High Risk Range")
11:    input_normal_risk.membership [VERY_HIGH_RISK] = TriangularMF("Very High Risk Range")
12:    fuzzy_system.variable["signature_risk"] =input_signature_risk
13:    input_signature_risk.membership [LOW_RISK] = TriangularMF("Low Risk Range")
14:    input_signature_risk.membership [AVERAGE_RISK] = TriangularMF("Average Risk Range")
15:    input_signature_risk.membership [HIGH_RISK] = TriangularMF("High Risk Range")
16:    input_signature_risk.membership [VERY_HIGH_RISK] = TriangularMF("Very High Risk Range")
17:    fuzzy_system.variable ["signatureorsystem_risk"] = giris_ signatureorsystem_risk
18:    input_signatureorsystem_risk.membership[LOW_RISK] = TriangularMF ("Low Risk Range")
19:    input_signatureorsystem_risk.membership [AVERAGE_RISK] = TriangularMF ("Average Risk Range")
20:    input_signatureorsystem_risk.membership [HIGH_RISK] = TriangularMF ("High Risk Range")
21:    input_signatureorsystem_risk.membership VERY_HIGH_RISK] = TriangularMF ("Very High Risk Range")
```

Figure 4. Algorithm that generates fuzzy membership functions.

```
Input:  Defuzzification method,minumum and maximum values,membership function
Output: Fuzzy output
1: generateFuzzyOutput:
2:   total_risk = OutputVariable (
3:                  defuzzificationMethod=COG(),
4:                  info="Total Risk Score",
5:                  minimum=0.0,maksimum=100.0,)
6:   total_risk.membership[LOW] = MembershipValue("Low Value")
7:   total_risk.membership [AVERAGE] = MembershipValue("Average Value")
8:   total_risk.membership [HIGH] = MembershipValue ("High Value")
9:   total_risk.membership [VERY_HIGH] = MembershipValue("Very High Value"))
10:  fuzzy_system.variable["total_risk"] = total_risk
```

Figure 5. Algorithm that generates output membership function.

malicious behavior. Although our service-based solution is not developed as antivirus software that identifies malicious applications, it can make use of antivirus solutions and produce results that can assign high risk scores based on several features. Its integration with state-of-the-art antivirus tools yields high performance

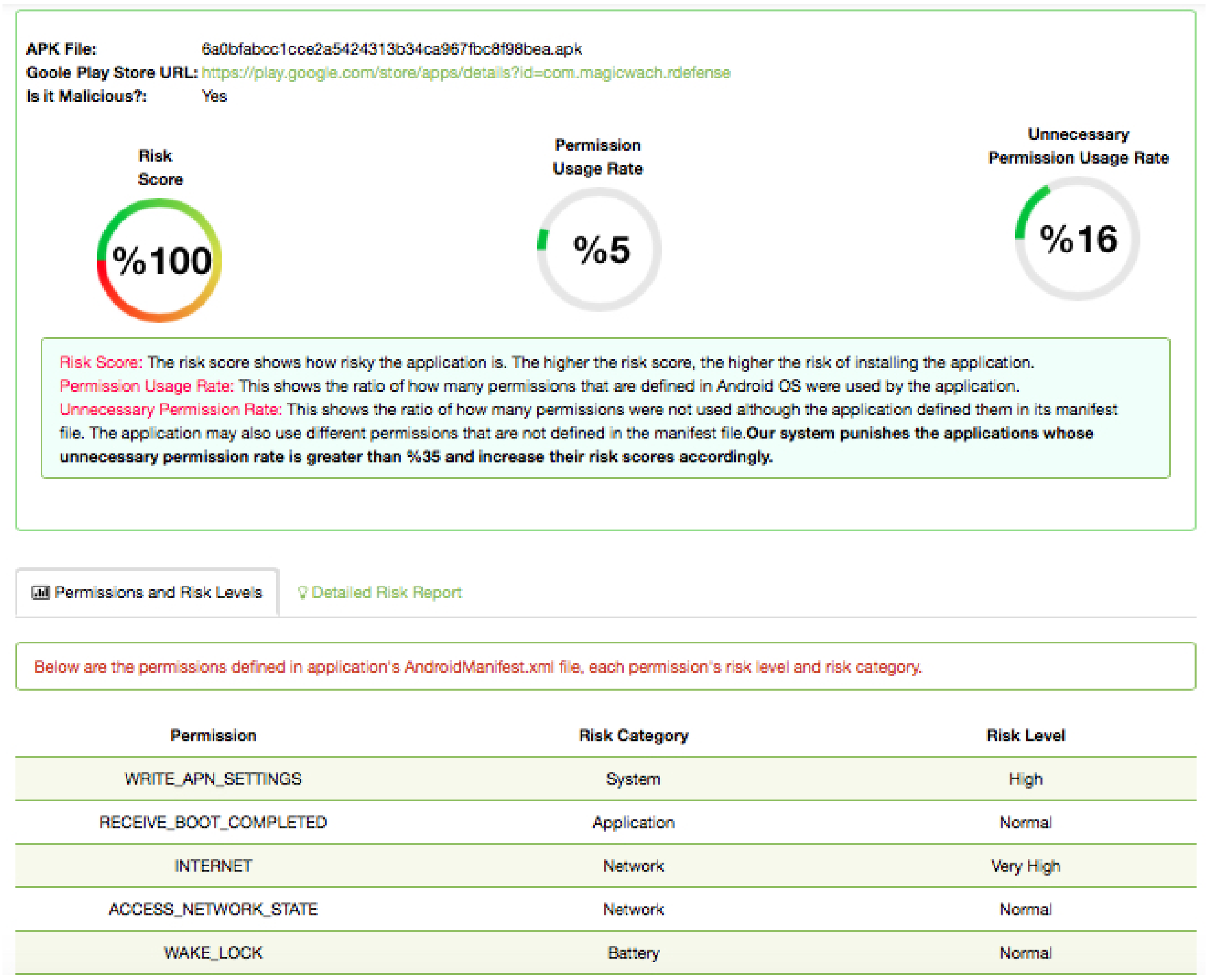
```
Input: Risks, file name, permission list
Output: Risk score
1:  evaluateRisks(risks,fileName,permission_list) :
2:      result = {"total_risk" : 0.0}
3:      input["dangerous_risk"] = risks[ DANGEROUS_RISK ]
4:      input ["normal_risk"] = risks[ NORMAL_RISK ]
5:      input ["signature_risk"] = risks[ SIGNATURE_RISK ]
6:      input ["signatureorsystem_risk"] = risks[ SIGNATUREORSYSTEM_RISK ]
7:      fuzzyCalculate (input=input, output = result)
8:      risk_score = result[ "total_risk" ]
9:      risk_level = result[ "risk_level" ]
10:       result=RiskResult()
11:     result.riskLevel=risk_level
12:     result.riskScore=risk_score
13:     result.isMalware=findMalware(hash(fileName))
14:     result.permissions=permission_list
15:  RETURN result
16: fuzzyCalculate(input,output):
17:     reset() # 1st step
18:     fuzzify(input) # 2nd  step
19:     fuzzyInference() 3rd step
20:     output=defuzzify() # 4th  step
21: RETURN output
```

Figure 6. Fuzzy inference algorithm.

and reliable risk scores besides its detection of malicious software. When it is compared with similar solutions, it is seen that these features do not exist and our system provides more and different features.

Our fuzzy logic-based security risk analysis and scoring system presents a new approach in the mobile security field. The advantages of our study over other studies are that our study can work independently of the device and it can determine which applications have high risks thanks to the fuzzy logic model and cloud-based antivirus tool [18] that determines whether applications are samples of malware. The system that has been developed is an online service, and any user who accesses the system via a browser can run a risk analysis by searching the Google Play Store or by uploading the “.apk” file of the downloaded application to our system. Additionally, the users see clear and detailed information about the applications that they are about to install. The users are also informed about the degree of the risks to which they are about to be exposed before they install the applications on their devices. Therefore, the users are able to make better decisions about whether to install the applications or not by seeing the risks that threaten their personal security and privacy. Below are the contributions and advantages of our system that we have developed:

- Device-independent operation
- Fuzzy logic-based risk analysis and scoring
- Informs the user before being exposed to risk
- Antivirus integration
- Detects unnecessary permissions
- User-centric
- Provides clear reports
- Free of charge and cloud-based
- Linux shell command detection



APK File: 6a0bfabcc1cce2a5424313b34ca967fbc8f98bea.apk
Goole Play Store URL: https://play.google.com/store/apps/details?id=com.magicwach.rdefense
Is it Malicious?: Yes

Risk Score: The risk score shows how risky the application is. The higher the risk score, the higher the risk of installing the application.
Permission Usage Rate: This shows the ratio of how many permissions that are defined in Android OS were used by the application.
Unnecessary Permission Rate: This shows the ratio of how many permissions were not used although the application defined them in its manifest file. The application may also use different permissions that are not defined in the manifest file.**Our system punishes the applications whose unnecessary permission rate is greater than %35 and increase their risk scores accordingly.**

Below are the permissions defined in application's AndroidManifest.xml file, each permission's risk level and risk category.

Permission	Risk Category	Risk Level
WRITE_APN_SETTINGS	System	High
RECEIVE_BOOT_COMPLETED	Application	Normal
INTERNET	Network	Very High
ACCESS_NETWORK_STATE	Network	Normal
WAKE_LOCK	Battery	Normal

Figure 7. Sample risk report.

Table 6. Comparison of similar solutions with Muras.

Analysis service	Risk scoring	Permission risk level	Report	APK upload	Search	Unnecessary permission detection	Shell command detection	Antivirus integration
Muras	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Dexter	No	No	Yes	Yes	No	No	No	No
AVC Undroid	No	Yes	Yes	Yes	No	No	No	No
Andrubis	No	No	Yes	Yes	No	No	No	No
Mobile-Sandbox	No	No	Yes	Yes	No	No	No	No
Copper Droid	No	No	Yes	Yes	No	No	No	No

Besides the contributions and advantages, we also tested the effectiveness of our approach. For this purpose, we created two datasets. The first dataset consists of known malicious applications [19,20]. The second dataset is obtained from the official Google Play Store. We developed a spider tool to download the top 100 free Android applications in 27 categories to create the second dataset. After creating each dataset, we ran our risk scoring script in the background, obtained the risk score of each application, and finally calculated the average risk scores. Figures 8 and 9 show our findings.

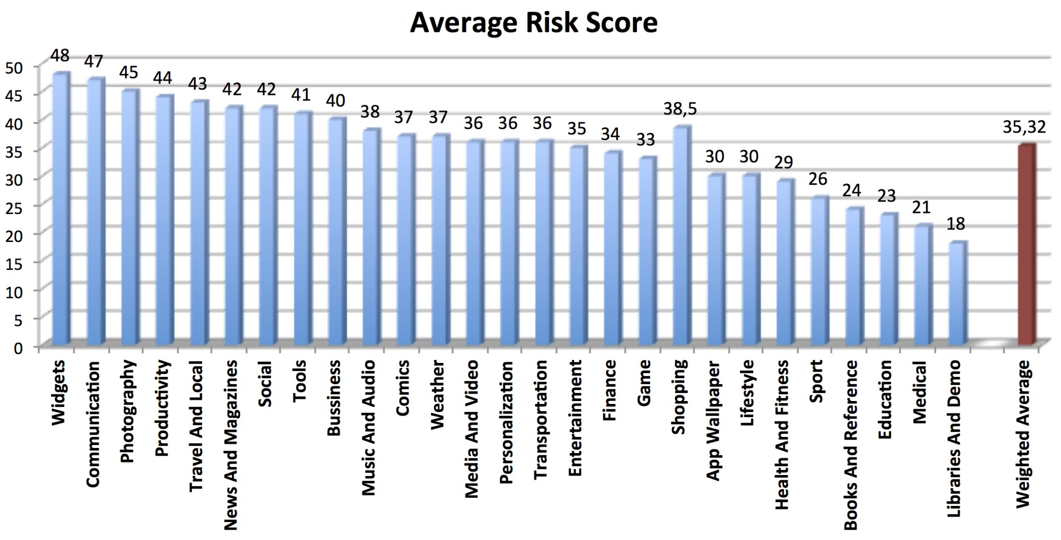


Figure 8. Average risk scores of Google Play Store applications.

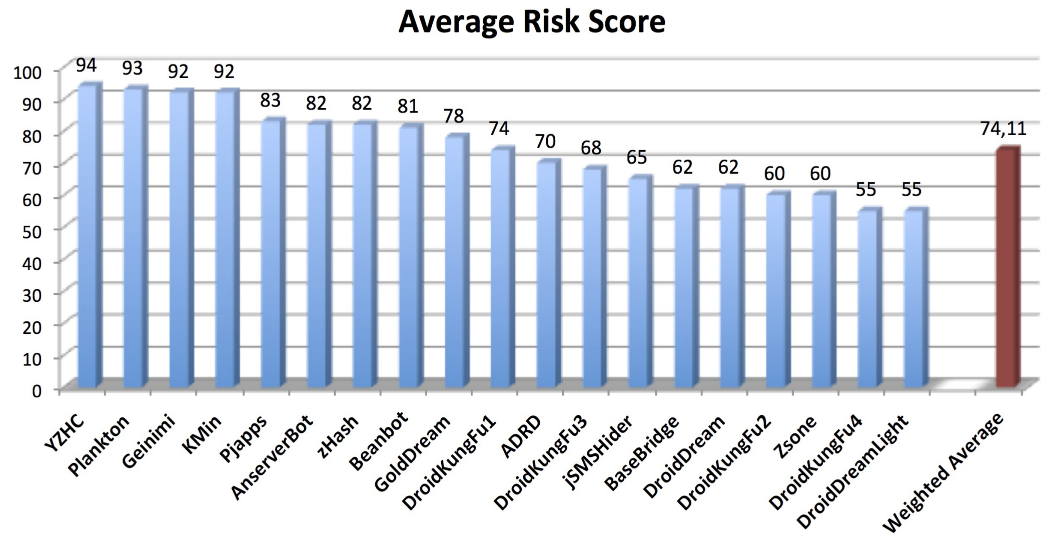


Figure 9. Average risk scores of malicious applications.

It is clearly seen from the figures that our system assigned higher scores for malicious apps than the apps that were downloaded from the Google Play Store without feeding the results from the antivirus service. While the widgets category has the highest average risk score of 48%, the DroidDreamLight category in malicious apps has the minimum average risk score of 55%.

In order to determine what kind of permissions (which risk categories) are used by Google Play Store applications and malicious applications and to see risk score distribution, we did two different analysis. Figure 10 shows four risk categories that we created and impact values of each category to the final risk scores produced by our fuzzy logic-based risk scoring algorithm. Figure 11 shows the risk score distribution for Google Play Store applications and malicious applications before applying the results obtained from antivirus tools. As seen from Figure 10, malicious applications and Google Play Store applications caused high risk scores by using permissions in the "Dangerous" risk category too much. Although there is a similar usage pattern between the "Signature" and the "Normal" risk categories, there are differences that can be seen very clearly in "Signature or System" and "Dangerous" risk categories. While the Google Play Store applications produced a score of 117 in the Dangerous risk category, malicious applications produced a score of 228. The difference in the "Signature or System" category is that more distinct and malicious applications produced an average score of approximately 6 times the Google Play Store applications' average score. When descriptions in the Android developer sites are reviewed, it is stated that permissions in the "Signature or System" category should only be used by applications in the Android operating system image and developers should not use these permissions. However, it is shown that malicious applications insistently use these permissions.

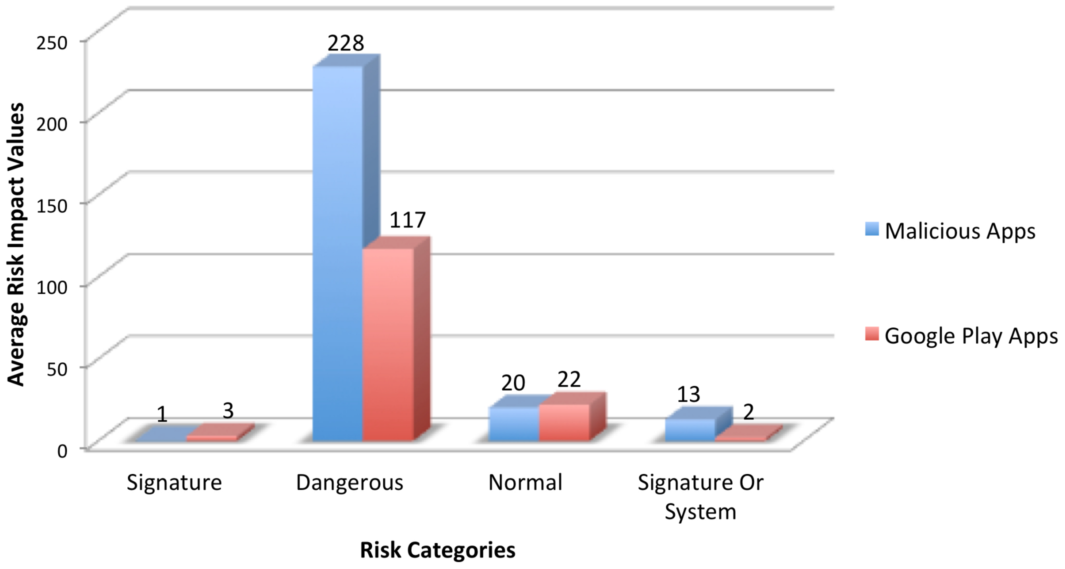


Figure 10. Average risk impact values of risk categories.

When the risk distribution in Figure 11 is observed, it is seen that there are very clear differences between Google Play Store applications and malicious applications. Store applications caused lower risk scores; on the contrary, malicious applications caused higher risk scores. While the risk scores of store applications are distributed between 0 and 60, the risk scores of malicious applications are distributed between 50 and 100. Twelve percent of store applications and 10% of malicious applications produced risk scores between 50 and 59. The most different distribution is seen between 80 and 100. While only 7% of store applications have risk scores in this range, 84% of the malicious applications have risk scores in this range. However, it is expected that the risk scores of all malicious applications should be higher than 50. In Figure 11, we see that only 5% of the malicious applications have risk scores lower than 50. Consequently, we can state that the success rate of our tool before applying the antivirus results is 95%. When we inspected the malicious files that have low risk scores, we realized that some of the APK files were not extracted successfully because of "CRC check fail error", while some of them had custom or misspelled permission names that were not in our permission database.

In addition to this analysis, two different analyses are done in order to observe the behavioral differences in store applications and malicious applications. The first of these analyses examines the numbers that can be considered as interesting and Linux shell commands that may cause security risks. The second analysis examines whether the applications used the permissions that they defined in their manifest files or not and how many of the permissions were actually used. In order to the perform this analysis, we stepped down to the source code and inspected if the permissions are actually called at the API level by using permission mapping [24]. Table 7 shows the Linux shell commands that the malicious applications use. Figures 12 and 13 show unnecessary

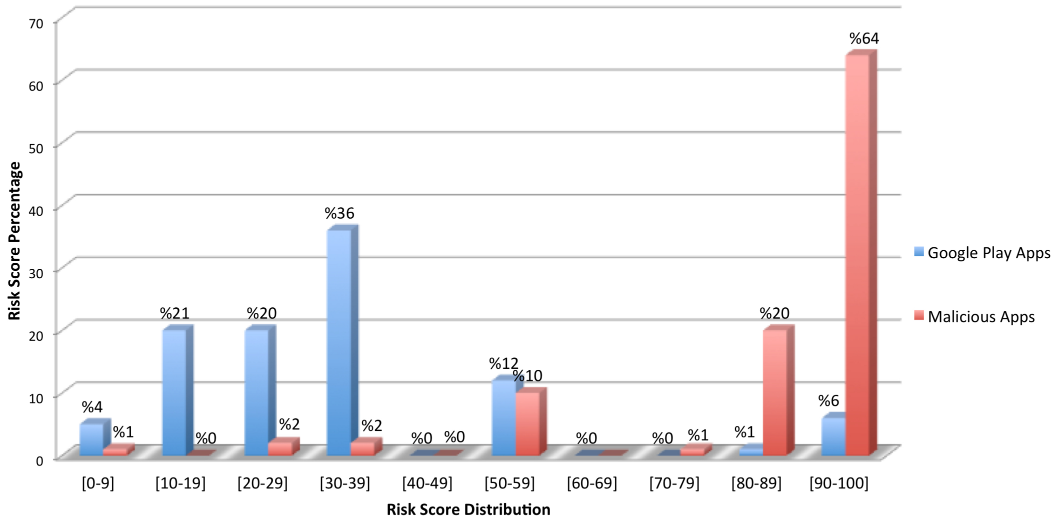


Figure 11. Risk score distribution.

permission usage by Google Play Store applications and malicious applications based on their own categories (we included the first 10 categories that have the highest ratios). When all categories were considered, it was seen that malicious applications used unnecessary permissions with a rate of 41%, while this rate is 27% for store applications.

Table 7. Linux shell commands frequently used by malicious applications.

Command	Explanation
chmod	Changes the permissions of files and directories.
sh	Executes commands from files, standard input, or command line.
su	Changes the user. As default, it provides super user privileges.
mount	Attaches a file system to another file system.
insmod	Inserts a module into the Linux kernel.
chown	Changes the owner of files.
stop vold/start vold	Starts or stops Android daemons.
cat	Reads data and outputs the contents.
setprop	Sets Android system properties.
mkdir	Creates a directory.
cp/mv	Copies/moves a file or directory.
exec	Replaces a process with another process.

Another analysis showed that store applications did not use any shell command while malicious applications used many various Linux commands. These commands may damage the users and compromise personal security and privacy when executed.

When we examined the interesting numbers used by malicious applications, it was identified that the number “48734154” was used very frequently. When a more detailed investigation was made, it was seen that this number was used for encryption purposes. After the application acquired personal information, it sent the information to the control server by applying an encryption algorithm that makes use of this number.

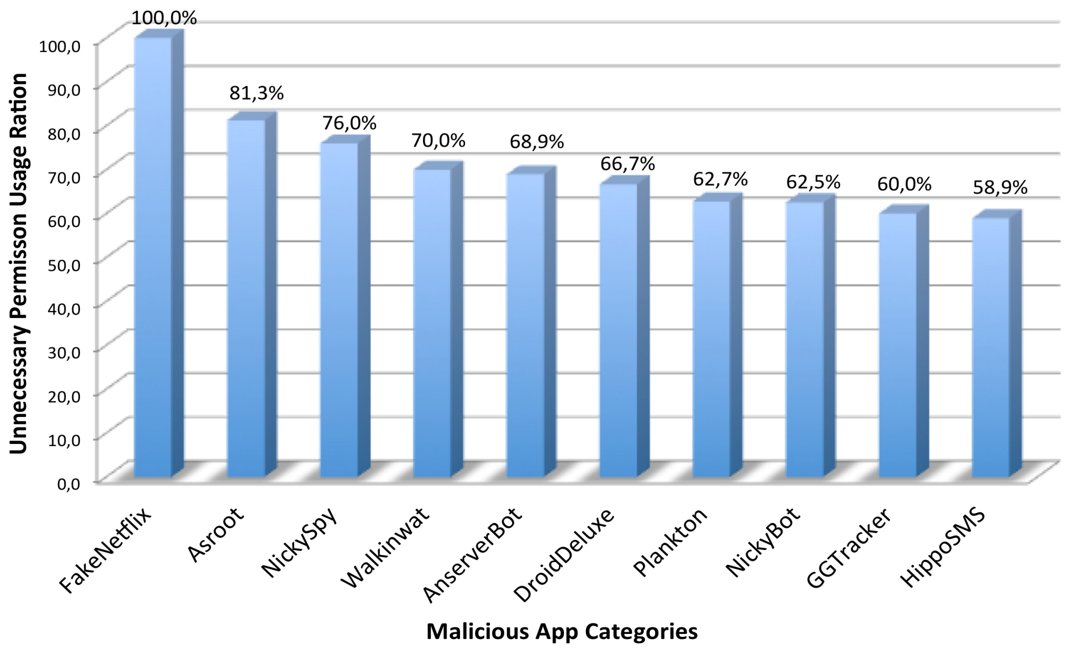


Figure 12. Unnecessary permission usage rates of malicious apps.

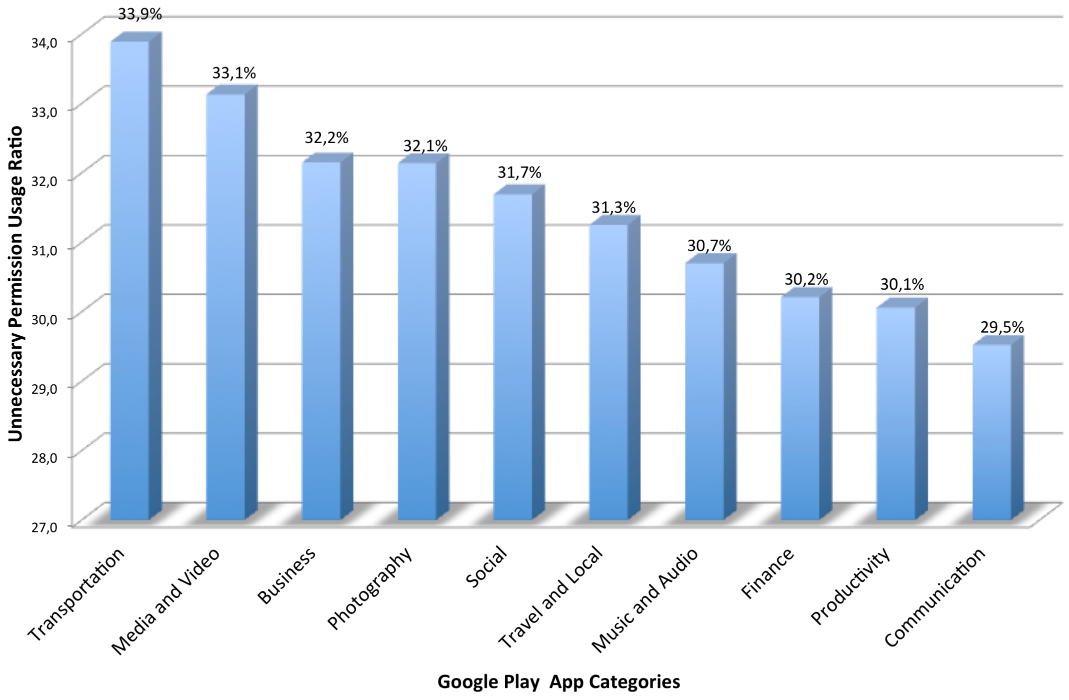


Figure 13. Unnecessary permission usage rates of Google Play apps.

6. Conclusion

Fuzzy logic-based systems are frequently used for risk analysis and assessment in the field of computer science and in other fields when there are no available data and the knowledge is insufficient. A well-designed fuzzy logic system will help understand the risks. Therefore, the risks and their exposure degrees can be revealed, and the risks can be managed. Taking the increasing use and popularity of mobile devices into consideration, we have proved that a fuzzy-logic based system can comfortably be used to analyze and assess risks in the mobile security field.

In this study, a prototype of a fuzzy risk inference service that examines the security risks of applications is designed and developed. The system is capable of assigning risk scores to Android applications and determining if the applications have potential security and privacy risks and whether the applications are malware or not. The system is designed as a user-centric system and works independently of the devices. Any Android smart device owner can analyze an application that he/she wants to install and see the security risks of installing an application via online user-friendly reports. The users can decide whether or not to install them by taking the risk score, explanations, and malware statuses as a reference. Therefore, users will be informed before installing the applications to their phones and will be aware of the threats against their privacy and security.

We increased the risk inference system's performance by integrating well-known, popular antivirus solutions into our system and by feeding our inference engine with the scores that we produced by parsing the antivirus reports. Therefore, the final risk score is the sum of these scores. As future work, we will combine our fuzzy logic model with neural networks to build an adaptive neuro-fuzzy inference system to increase the inference performance. We will use the self-learning features of artificial neural networks and the power of the antivirus detection systems, and we will define our membership functions accordingly. By using a hybrid learning approach, our system will be able to construct data pairs for neural network training. We think that the new approach will be more powerful than our current implementation since it will have learning capabilities.

References

[1] Vilwock W, Madiraju P, Ahamed SI. A system implementation of interruption management for mobile devices. In: 16th International Conference on Computational Science and Engineering; 3–5 December 2013; Sydney, Australia. New York, NY, USA: IEEE. pp. 181-187.

[2] Beresford AR, Rice A, Skehin N, Sohan R. Mock-Droid: trading privacy for application functionality on smartphones. In: 12th Workshop on Mobile Computing Systems and Applications; 1–3 March 2011; Phoenix, AZ, USA. New York, NY, USA: ACM. pp. 49-54.

[3] Wang X, Sun K, Wang Y, Jing J. DeepDroid: dynamically enforcing enterprise policy on Android devices. In: Network and Distributed System Security Symposium; 8–11 February 2015; San Diego, CA, USA. Reston, VA, USA: Internet Society. pp. 1-15.

[4] Cozzette A, Lingel K, Matsumoto S, Ortlieb O, Alexander J, Betser J, Reiher P. Improving the security of Android inter-component communication. In: International Symposium on Integrated Network Management; 27–31 May 2013; Ghent, Belgium. New York, NY, USA: IEEE. pp. 808-811.

[5] Isohara T, Takemori K, Kubota A. Kernel-based behavior analysis for Android malware detection. In: 7th International Conference on Computational Intelligence and Security; 3–4 December 2011; Hainan, China. New York, NY, USA: IEEE. pp. 1011-1015.

[6] Enck W, Octeau D, McDaniel P, Chaudhuri S. A study of Android application security. In: USENIX Security Symposium; 8–12 August 2011; San Francisco, CA, USA. Berkeley, CA, USA: Usenix. p. 2.

[7] Enck W, Ongtang M, McDaniel P. On lightweight mobile phone application certification. In: 16th ACM Conference on Computer and Communications Security; 9–13 November 2009; Chicago, IL, USA. New York, NY, USA: ACM. pp. 235-245.

[8] Xu R, Saidi H, Anderson R. Aurasium: practical policy enforcement for Android applications. In: USENIX 2012 Security Symposium; 8–10 August 2012; Washington, DC, USA. Berkeley, CA, USA: Usenix. pp. 539-552.

[9] Kaur A, Upadhyay D. PeMo: Modifying application's permissions and preventing information stealing on smartphones. In: 5th International Conference-Confluence The Next Generation Information Technology Summit; 25–26 September 2014; Noida, India. New York, NY, USA: IEEE. pp. 905-910.

[10] Gilbert P, Chun BG, Cox LP, Jung J. Vision: automated security validation of mobile apps at app markets. In: 2nd International Workshop on Mobile Cloud Computing and Services; 28 June 28–1 July 2011; Bethesda, MD, USA. New York, NY, USA: ACM. pp. 21-26.

[11] Davis B, Sanders B, Khodaverdian A, Chen H. I-arm-droid: a rewriting framework for in-app reference monitors for Android applications. In: Mobile Security Technologies; 24 May 2012; San Francisco, CA, USA. New York, NY, USA: IEEE. pp. 33-41.

[12] Yuksel AS, Zaim AH, Aydin MA. A comprehensive analysis of Android security and proposed solutions. International Journal of Computer Network and Information Security 2014; 6: 9-20.

[13] Matenaar F, Schulz P, Galauner A, Schlösser M. Dexter: Android Analysis Framework. Available online at https://www.dexlabs.org/.

[14] Lindorfer M, Neugschwandtner M, Weichselbaum L, Fratantonio Y, Van Der Veen V, Platzer C. Andrubis - 1,000,000 apps later: a view on current android malware behaviors. In: 3rd International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security; 11 September 2014; Wroclaw, Poland.

[15] AV Comparatives. AVC Undroid. Available online at https://www.av-comparatives.org/.

[16] Spreitzenbarth M, Freiling F, Echtler F, Schreck T, Hoffmann J. Mobile-sandbox: having a deeper look into android applications. In: Proceedings of the 28th Annual ACM Symposium on Applied Computing; 18–22 March 2013; New York, NY, USA. New York, NY, USA: ACM. pp. 1808-1815.

[17] Tam K, Khan SJ, Fattori A, Cavallaro L. CopperDroid: automatic reconstruction of android malware behaviors. In: Proceedings of the Symposium on Network and Distributed System Security; 8–11 February 2015; San Diego, CA, USA.

[18] Haffejee J, Irwin B. Testing antivirus engines to determine their effectiveness as a security layer. In: Information Security for South Africa; 13–14 August 2014; Johannesburg, South Africa.

[19] Zhou Y, Jiang X. Dissecting Android malware: characterization and evolution. In: Security and Privacy Symposium; 20–23 May 2012; San Francisco, CA, USA. New York, NY, USA: IEEE. pp. 95-109.

[20] Arp D, Spreitzenbarth M, Huebner M, Gascon H, Rieck K. Drebin: Efficient and explainable detection of android malware in your pocket. In: 21st Annual Network and Distributed System Security Symposium; 23–26 February 2014; San Diego, CA, USA.

[21] Mamdani EH, Assilian S. An experiment in linguistic synthesis with a fuzzy logic controller. Int J Man Mach Stud 1975; 7: 1-13.

[22] Enck W, Ongtang M, McDaniel P. Understanding Android security. Security & Privacy 2009; 1: 50-57.

[23] Zhou Y, Wang Z, Zhou W, Jiang X. Hey, you, get off of my market: detecting malicious apps in official and alternative Android markets. In: Network and Distributed System Security Symposium; 6–8 February 2012; San Diego, CA, USA. Reston, VA, USA: The Internet Society. pp. 1-13.

[24] Johnson R, Wang Z, Gagnon C, Stavrou A. Analysis of android applications' permissions. In: Sixth International Conference on Software Security and Reliability Companion; 20–22 June 2012; Washington, DC, USA. New York, NY, USA: IEEE. pp. 45-46.