Detecting permission over-claim in android applications

Submitted By

Raj J Majethiya

20MCEI08

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INSTITUTE OF TECHNOLOGY

NIRMA UNIVERSITY

AHMEDABAD-382481

May 2022

Detecting permission over-claim in android applications

Major Project

Submitted in partial fulfillment of the requirements

for the degree of

Master of Technology in Computer Science and Engineering(INS)

Information and Network Security

Submitted By

Raj J Majethiya

(20MCEI08)

Guided By

Prof Monika Shah

NIRMA UNIVERSITY

INSTITUTE OF TECHNOLOGY

NAAC ACCREDITED ‘A’ GRADE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INSTITUTE OF TECHNOLOGY

NIRMA UNIVERSITY

AHMEDABAD-382481

May 2022

Certificate

This is to certify that the major project entitled **"Detecting permission over-claim in Android application"** submitted by **Raj J Majethiya (20MCEI08)**, towards the partial fulfillment of the requirements for the award of degree of Master of Technology in Computer Science and Engineering of Nirma University, Ahmedabad, is the record of work carried out by him under my supervision and guidance. In my opinion, the submitted work has reached a level required for being accepted for examination. The results embodied in this Major Project Part-I, to the best of my knowledge, haven't been submitted to any other university or institution for award of any degree or diploma.

Prof Monika Shah
Internal Guide & Associate Professor
CSE Department
Institute of Technology
Nirma University, Ahmedabad

Dr Sharda valiveti
Professor & PG Coordinator (M.Tech - INS)
CSE Department
Institute of Technology
Nirma University, Ahmedabad

Dr Madhuri Bhavsar
Professor & Head
CSE Department
Institute of Technology
Nirma University, Ahmedabad

Dr Rajesh Patel
Director
Institute of Technology
Nirma University, Ahmedabad

Statement of Originality

I, **Raj J Majethiya**, **20MCEI08**, give undertaking that the Major Project entitled **"Detecting permission over-claim in Android applications"** submitted by me, towards the partial fulfillment of the requirements for the degree of Master of Technology in **Computer Science & Engineering(INS)** of Institute of Technology, Nirma University, Ahmedabad, contains no material that has been awarded for any degree or diploma in any university or school in any territory to the best of my knowledge. It is the original work carried out by me and I give assurance that no attempt of plagiarism has been made.It contains no material that is previously published or written, except where reference has been made. I understand that in the event of any similarity found subsequently with any published work or any dissertation work elsewhere; it will result in severe disciplinary action.

Signature of Student
Date:
Place:

Endorsed by
Prof Monika Shah

Acknowledgements

It gives me immense pleasure in expressing thanks and profound gratitude to **Prof Monika Shah**, Associate Professor, Computer Science and Engineering Department, Institute of Technology, Nirma University, Ahmedabad for her valuable guidance and continual encouragement throughout this work. The appreciation and continual support she has imparted has been a great motivation to me in reaching a higher goal. Her guidance has triggered and nourished my intellectual maturity that I will benefit from, for a long time to come.
I would also like to thank **Dr Sharada Valiveti** PG Co-ordinator M.tech(INS), Computer Science and Engineering Department,Institute of Technology, Nirma University, Ahmedabad for her constant support during the completion of this project.

It gives me an immense pleasure to thank **Dr Madhuri Bhavsar**, Hon'ble Head of Computer Science And Engineering Department, Institute of Technology, Nirma University, Ahmedabad for her kind support and providing basic infrastructure and healthy research environment.

A special thank you is expressed wholeheartedly to **Dr Rajesh Patel**, Hon'ble Director, Institute of Technology, Nirma University, Ahmedabad for the unmentionable motivation she has extended throughout course of this work.

I would also thank the Institution, all faculty members of Computer Science and Engineering Department, Nirma University, Ahmedabad for their special attention and suggestions towards the project work.

- Raj J Majethiya
20MCEI08

Acronyms

AAB Android App Bundle

AAR Android ARchive

AD Advertisement

ADB Android Debug Bridge

API Application Programming Interface

APK Android Application PacKage

APP Application

CNN Convolution Neural Network

DBN Deep Belief Network

DEX Dalvik Executable

GPS Global Positioning System

GUI Graphical User Interface

HIN Heterogeneous Information Network

JAR Java ARchive

KNN K-Neaest Neighbour

LSTM Long Short-Term Memory

NLPNatural Language Processing

OS Operating System

RQ Research Question

SMS Short Message Service

SVM Support Vector Machines

Abstract

Android devices have become very popular nowadays due to their rich functionality and good performance. To give the users better functionality, Android has introduced a permission model allowing developers to access the resources and information of the device with the user's permission. However, few developers over-claim the permission that has nothing to do with the apps' functionality. By using over-claim permission, many android apps may steal sensitive information from users, Which is one of the leading privacy issues. Looking towards its criticality, we have started working on detecting over-claim permission. Detecting the over-claim permission is done in a variety of ways. In our proposed methodology the semantic analysis is done for the sentences of the permission description and the app description. The analysis is done using sentence transformer, sentence similarity using Space and universal sentence encoder technique. Also keyword based analysis is done to find out if the given keyword that is related to permission is present in the app description or not.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

It is tough to imagine today's life of common man without mobile phones. There are many activities performed using mobile phones in our day-to-day life, including wake-up alarms, exercise training apps, scheduling meetings, bank transactions, communicating with other people, preparing documents, personal entertainment, and to name a few. which has resulted in more usage of phones as shown in **Fig. 1.1** . The total active mobile user presently are 6.5 billion globally[2].

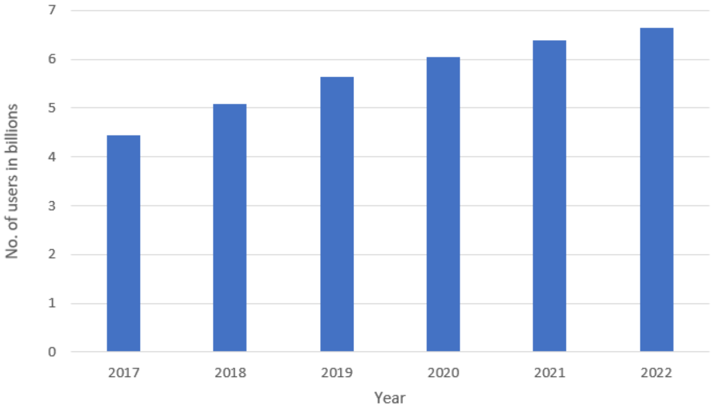


Figure 1.1: Active mobile phone users worldwide

Statistics presented in **Fig. 1.2** shows that the android operating system has the largest market share of about 74 percent [3]. The android system facilitates a permission model that allows developers to seek permission from the users to access mobile resources to perform claimed functionalities of the app. Despite that, it has been seen that many developers are seeking over-claim permission. Being unaware of the use of each permission, users grant the authorization of requested permissions. In this case, over-claim permissions may attempt to take over mobile control to perform malicious activities and access users' sensitive data to infringe on their privacy. Generally, permissions are requested at the installation time of the app, up-gradation of, and during the app's execution. Therefore Researchers have proposed various approaches to detect over-claim permissions at various stages of the app life cycle. Despite that, over-claim permission in android is still a critical problem. Therefore, this paper comprehends the basics of the Android permission model and the stages of requesting and granting permission to understand the criticality of permissions. Then after it discusses various proposed approaches to deal with over-claim permissions and compare them to identify their limitation.

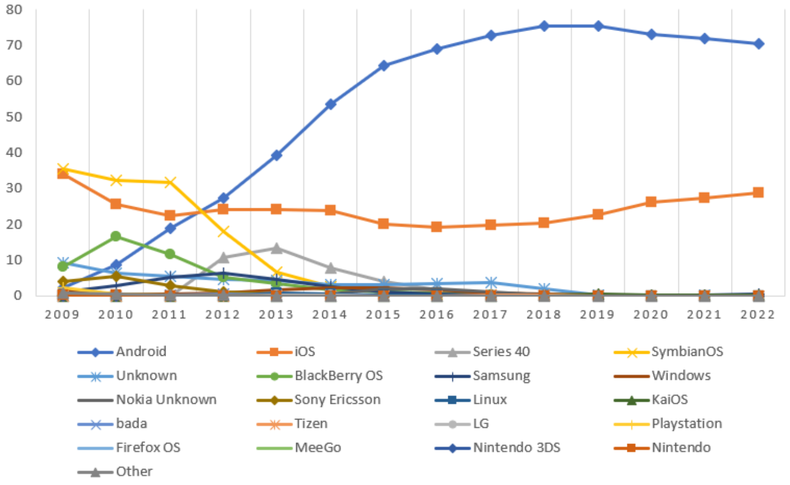


Figure 1.2: Market share of different mobile operating system

Chapter 2

Background and concept

2.1 Android permission model

The Android permission model refers to the authorization method that the developer must follow for using the mobile device's resources. The list of permission can be extracted from the in manifest.xml. After introducing android 6.0 (API level 23), the permission model concept got changed [4]. Before Android 6.0, the application has to make a declaration of permission used when installing the application. The access is given at installation time. However, after introducing android 6.0, the permission model has been changed to Run Time Permission Model. The application can ask for permission anytime when needed to perform that functionality while using the app. The permissions are categorized by the different levels of security based on how users' data and resources it uses.

2.2 Permission categories

Permissions are crucial in Android as it is used to fulfill the application's functionalities. The permissions used in Android are categorized into three groups based on the security level and right to access the resources. [5].

- Normal Level permission This type of permission has a very low risk and does not affect user's privacy. In this type of permission, no dialogue box appears and the application can directly get permission from the device. This permission includes ACCESS_NETWORK_STATE

ACCESS_WIFI_STATE

INTERNET

- Signature Level permission Signature level permissions are install time permissions, which has a condition that must be fulfilled. The app requesting these permissions must be signed with the same signature of the app that has already defined signature level permission. Example:

  BIND_JOB_SERVICE

  SYSTEM_ALERT_WINDOW

  BIND_VPN_SERVICE

- Dangerous level permission Dangerous level permission carries a high level of risk that may compromise the user's privacy. Dangerous permission does not always cause privacy problems if appropriately used according to the app's functionality. There are many dangerous level permission declared by android as shown in **table 2.1**.This permission are asked during running of application.

2.3 Over-claim permission

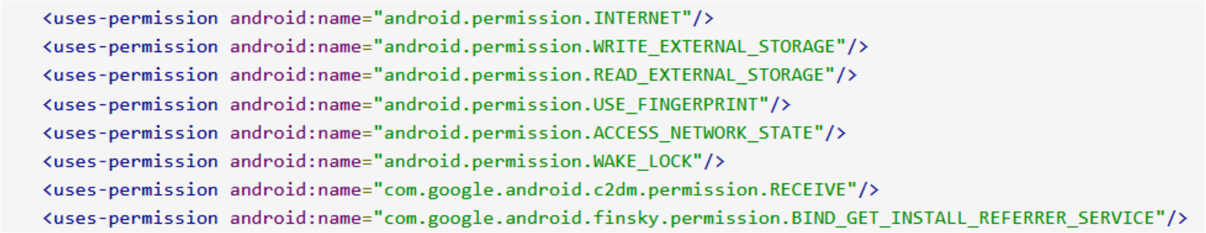
Permissions are used to provide better functionality to the users. However, some developers may ask for more permission than required to fulfill the app's functionality. The extra asked permission is called over-claim permission which may be used to infringe user's privacy. Over-claim permission may not always be dangerous level permission.

2.4 Over-claim permission detection approaches

To handle the issues of over-claim permission in Android applications, many researchers have contributed their work, including different analysis and detection approaches. The different analysis techniques are Static, Dynamic, and Semantic analysis. Other than these three methods, some of the researchers have adopted hybrid analysis, which includes the use of more than one analysis method. To analyze the permission used should be extracted from android.manifest.xml file. **Fig. 2.1**.Later Different analysis methodologies can be used to analyze the permission usage and detect over-claim permission.

Sr. no.	**Permission name**
1	ACCEPT_HANDOVER
2	ACCESS_BACKGROUND_LOCATION
3	ACCESS_COARSE_LOCATION
4	ACCESS_FINE_LOCATION
5	ACCESS_MEDIA_LOCATION
6	ACTIVITY_RECOGNITION
7	ADD_VOICEMAIL
8	ANSWER_PHONE_CALLS
9	BLUETOOTH_ADVERTISE
10	BLUETOOTH_CONNECT
11	BLUETOOTH_SCAN
12	BODY_SENSORS
13	BODY_SENSORS_BACKGROUND
14	CALL_PHONE
15	CAMERA
16	GET_ACCOUNTS
17	NEARBY_WIFI_DEVICES
18	POST_NOTIFICATIONS
19	PROCESS_OUTGOING_CALLS
20	READ_CALENDAR
21	READ_CALL_LOG
22	READ_CONTACTS
23	READ_EXTERNAL_STORAGE
24	READ_MEDIA_AUDIO
25	READ_MEDIA_IMAGES
26	READ_MEDIA_VIDEO
27	READ_PHONE_NUMBERS
28	READ_PHONE_STATE
29	READ_SMS
30	RECEIVE_MMS
31	RECEIVE_SMS
32	RECEIVE_WAP_PUSH
33	RECORD_AUDIO
34	SEND_SMS
35	USE_SIP
36	UWB_RANGING
37	WRITE_CALENDAR
38	WRITE_CALL_LOG
39	WRITE_CONTACTS
40	WRITE_EXTERNAL_STORAGE

Table 2.1: Dangerous permission list [1]



```
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.USE_FINGERPRINT"/>
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
    <uses-permission android:name="android.permission.WAKE_LOCK"/>
    <uses-permission android:name="com.google.android.c2dm.permission.RECEIVE"/>
    <uses-permission android:name="com.google.android.finsky.permission.BIND_GET_INSTALL_REFERRER_SERVICE"/>
```

Figure 2.1: Android.manifest.xml file

2.4.1 Static analysis

In this method, analyzing of the APK file is done. The application's source code is extracted by decomposing the APK or DEX files. After decompilation, the android.manifest.xml file is used to extract the permissions requested in the app. As demonstrated in **Fig. 2.1**, permissions are placed in android.manifest.xml. The permission usage is checked to identify the permission over-claim problem based on the permission extracted. The flow is described in **Fig. 2.2**.

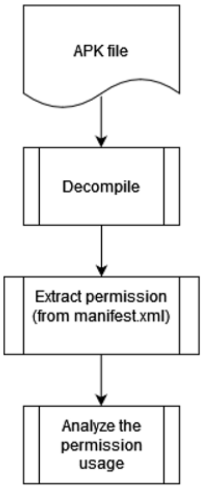

APK file
Decompile
Extract permission (from manifest.xml)
Analyze the permission usage

Figure 2.2: Static analysis

2.4.2 Dynamic analysis

The Permissions requested at run-time can be analyzed by dynamic analysis. The permissions asked during the run time are saved in the activity manager's services check permission method. This run time permission can be dynamically tested using any tool by running the app in real-time. For example, if the user requests to perform a function that requires specific permission, the permission is requested at that time to fulfill the app's requirement. Further, this requested permission is included for analysis as shown in **Fig. 2.3**.

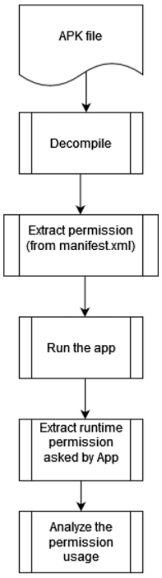

APK file
Decompile
Extract permission (from manifest.xml)
Run the app
Extract runtime permission asked by App
Analyze the permission usage

Figure 2.3: Dynamic analysis

2.4.3 Symantic analysis

This technique examines the metadata provided by the developer and check for overclaim permission. In this method, permission asked by the developer is cross-checked with the app's description, whether there exists any related information in the description by the developer for using that particular permission or not. Further machine learning algorithms related to semantic analysis are used to check for the semantic meaning of the permission as shown in **Fig. 2.4**

2.4.4 Hybrid analysis

In this model multiple analysis methods are used together. The run time permission asked during the running of the application is also analyzed in real-time, based on which permission can be identified as over-claim permission by generating the list of appropriate permissions using semantic analysis based on the app's description. All three different analysis is used in the hybrid analysis.

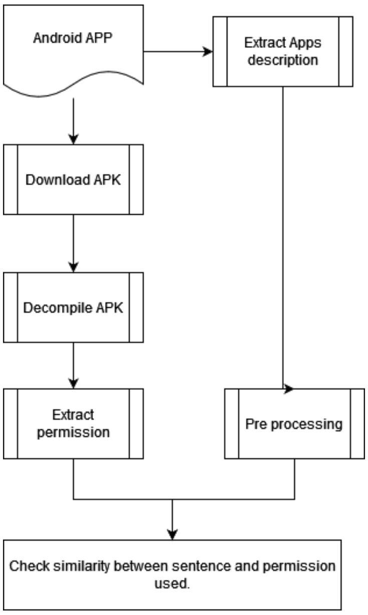


Figure 2.4: Semantic analysis

Chapter 3

Literature survey

This section contains a survey of papers to identify the methods adopted to detect the over-claim permission. The gaps and limitations of different methods are also explained, providing a better understanding of what work has been done and how to conduct future research. To start the survey, papers that include detection of over-claim permission are selected. Further, there are many different methods and techniques to figure out the usage of over-claim permission. Many researchers share the common domain and have related work based on different methods. For example, many papers contain the work related to identifying over-claim permission in Third-party libraries. Identification of over-claim permission for only a particular application category has also been made. Our survey has been diversified based on identified research topics, and a comparison of methods used for each of the topics is presented in this section.

3.1 Analysis of Third-party Library

The android gives feature to developer to use any third party library. for example advertisement library. Therefore it created another challenge to detect over-claim permission used by the third party. The authors of [6, 7, 8, 9, 10, 11] have worked on detection of permission over-claim in third party libraries which is presented in **table 3.1**. The comparison is made based on whether compile-time permission is used for analysis or permission used at run time in analysis. The compile-time permission refers to the Static analysis process, and run time permission refers to Dynamic analysis. Another challenge is to identify permission added during the up-gradation of the Android application, which is discussed in [12]. Further, the limitations of each method are also summarized in the table

ref.	Year	CT	RT	Limitations
[6]	2021	Y	N	Identification of certain permission
[7]	2021	Y	N	All third party library app falls under un-trusted domain
[8]	2021	N	Y	Manual usage of tool
[9]	2021	Y	N	Check only dangerous permission
[10]	2021	Y	N	Manually identification
[11]	2017	N	Y	Only identify add library
[12]	2022	Y	Y	Check for permission change only after update

Table 3.1: Comparison of over-claim permission detection in Third-party library

3.2 analysis of malware apps

In many of the works, the researchers are using the data that contain malware Applications. Malware app those which per from activities like taking control of users' devices, modifying the device settings, and infringing on users' privacy. The cause or the reason for this also includes the extra declared permissions. The authors of [13, 14, 15, 16, 17] have worked on identifying apps as malware or not based on the already known malware families. The comparison is shown in **table 3.2** which includes whether the analysis carried out is for the compile-time permission that is static analysis, or run-time permission that is dynamic analysis. Also, the limitations of each paper are discussed. From which it is identified that there are limited known malware families and unknown malware apps which do not belong to any category can not be identified accurately.

ref.	Year	CT	RT	Limitations
[13]	2021	Y	N	Only identify Known Malware
[14]	2018	N	Y	Limited known malware apps family
[15]	2017	Y	N	Over fitting of data
[16]	2020	Y	N	Time consuming
[17]	2019	Y	N	Permission categorizing is not official

Table 3.2: Over-claim permission detection using malware analysis

3.3 Category wise application

There are over 50 different categories in which the Android apps have been categorized based on their functions. For example, the most popular categories are Shopping, food drinks, productivity, Books, Lifestyle, Entertainment, music, Education, and to name a few [18]. Instead of detecting over-claim permission for all categories of application, the authors of [19, 20, 21, 6] have only focused upon a single category of application. **table 3.3** contains the comparison of this work. The comparison of these methods is based on whether the permission used in the analysis is compiled time or run time. The common technique used by researchers is that a data set is created that includes applicable permission. Based on this, over-claim permission is identified. Further, the limitations of each work are discussed.

ref.	Year	CT	RT	Limitations
[19]	2021	Y	N	Less input malware app
[20]	2021	Y	N	Only statistical study
[21]	2021	Y	N	Only carried out for diabetics related app
[6]	2021	Y	N	Identify certain set of permission

Table 3.3: Summary of category based over-claim permission detection approaches

3.4 Tools for APK decompilation

From the research articles that have been surveyed, it is identified that to carry out the analysis of permissions used in-app decompiling is done. This is done using many decompilation tool. The **table 3.4** contains the complete survey of different tools used in different research works.

Sr. no	Tool name	Format	Extraction	Interface
1	Bytecode viewer	APK,DEX	Offline	GUI
2	JADx	APK,DEX,AAR,AAB	Offline	Command line
3	APK tool	APK	Offline	GUI
4	classy shark	APK,JAR,DEX,AAR	Offline	GUI
5	Andro guard	APK,DEX	Offline	GUI
6	AndroPyTool	APK,DEX	Offline,Runtime	GUI
7	MobSF	APK,DEX	Offline,Runtime	GUI
8	Enjarify	APK,DEX,JAR	Offline	GUI
9	Drozer	APK,DEX	Offline	Command line
10	ADB	APK	Offline	GUI
11	APK easy tool	APK,DEX	Offline	Command line

Table 3.4: comparison of different tool to decompile .APK

3.5 Analysis using semantic methods

Many researchers have adopted the semantic analysis method to check permission usage. **table 3.5** represents the comparison of different approaches and algorithms that uses semantic analysis to identify app as malware. The article [22] have work related to identifying app as malware based on the feature set. Semantic analysis is used to create a feature set for permission with similar descriptions but uses different resources. In article [23] along with permission, opcodes and intents are used to generate a knowledge graph for classification of the app. The latent semantic indexing algorithm obtains a lower dimension representation of parameters to create a precise feature set. In the article, [24] API vector embedding is done using semantic methods to generate a feature set.

Ref	Year	Approach	Algorithm
[22]	2021	Generate knowledge graph to classify app as malware	CNN
[23]	2020	Identify app as malware based on knowledge graph	CNN,KNN
[24]	2020	identify malware app using API calls	KM clustering
[25]	2020	identify app as malware based on android malware activities and features	CNN,LSTM
[26]	2019	Classify app malware app based on feature set	KNN,SVM

Table 3.5: Summary of semantic analysis for over-claim permission detection

Semantic analysis has a close relationship with Natural Language processing, which may be a reason for having a large fraction of semantic analysis of over-claim detection using NLP. Therefore, this paper gives a separate summary of semantic analysis using NLP in **table 3.6**. Every app specifies its functionalities in natural language format at App description. It is a general understanding that every permission requested by the app must match the statements given in the app description. Therefore most work [27, 28, 29, 30, 31] uses app description to find significance of usage of individual permission. This inclination toward the use of permission name analysis with apps description has been observed in the article[28, 29, 30] using keyword matching. In the article, [27] the comments given in the code are also analyzed to check the similarity with the app's description. In the article, [31] the analysis of privacy policy is done.

To analyze the permission usage in the android app, we need to extract permission from source code. The extracted APK file contains many parameters other than permissions that can be used for semantic analysis.The work-related to semantic analysis is presented in **table 3.5** and **table 3.6** which summarize the different approaches and

Ref	Year	Approach	Algorithm
[27]	2021	Check for insensitive flow using semantic analysis of comments included in code	NLP
[28]	2021	Correlate app's description with permission using word embedding algorithm	NLP
[29]	2020	create list of requested dangerous permission at run time and check if it is related to apps description	NLP
[30]	2019	Identify permission usage from description by keyword matching	NLP
[31]	2016	Check for permission usage by matching verbs related to permission with description and also privacy policy & NLP	NLP

Table 3.6: NLP based semantic analysis to detect over-claim permission

algorithms used. **table 3.7** includes the comparison of input parameters. Parameters to analyze the use of over-claim permission in the app shows the popularity of permission name followed by API call and App description. Use of permission name seems more obvious to find irrelevant permission usage in the app.

Ref.	PD	PN	AC	HL	IN	OC	APC	AD
[22]	Y	Y		Y				
[23]		Y			Y	Y		
[24]		Y	Y					
[25]		Y	Y		Y		Y	
[26]		Y	Y					
[27]		Y						Y
[28]		Y						Y
[29]		Y	Y					Y
[30]		Y						Y
[31]		Y						Y

Table 3.7: Summary of parameters used in semantic analysis

Chapter 4

Proposed methodology

Based on the review of methods used to analyze the Android permission. Our proposed methodology includes the analysis of android permission using semantic analysis. Among all the analysis techniques, the semantic analysis gives the best result when used for comparing the statements of app description and permissions description. NLP algorithms are best suitable for semantic analysis of English language sentences. Our proposed methodology contains analysis using three different NLP algorithms based on which we have to find the cosine similarity between sentences. The architecture of our proposed methodology is shown in **Fig. 4.1**.First we need permission. After that, the description of each permission is extracted using web scraping. Along with this, another script has been made in which we can directly get the app's description by entering the URL of the app. So now the Apps description and the permission description are extracted, which can be further used for the semantic analysis. The next step is to pre-process the data and clean the data of apps description and permission description in order to get better results while performing semantic analysis. Once the data is pre-processing, the next step is to perform the semantic analysis.

4.1 Decompiling APK File

Decompilation of APK is done in order to extract the permission that have been used in the app. the android.manifest.xml have these permission included in it. we need to extract the this permission. Based on this the list is created and further analysis can be carried out. The permission extracted is shown in **Fig. 4.2**

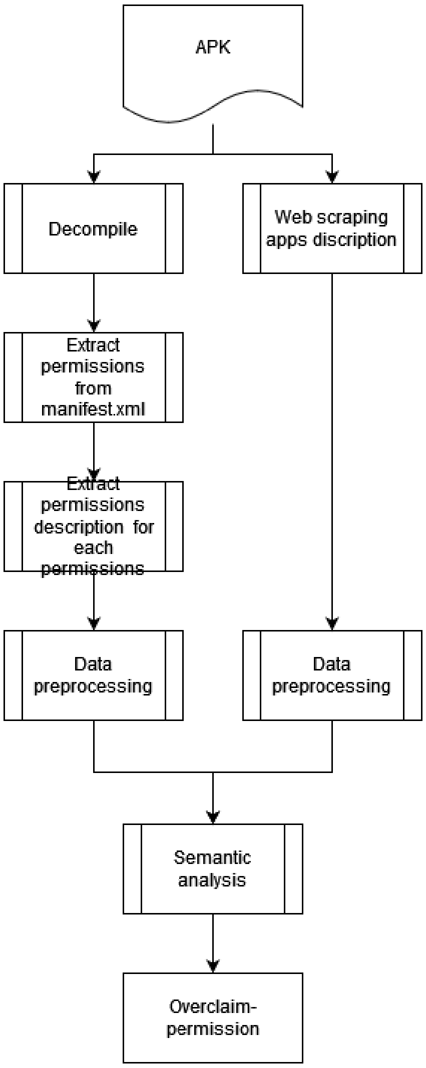
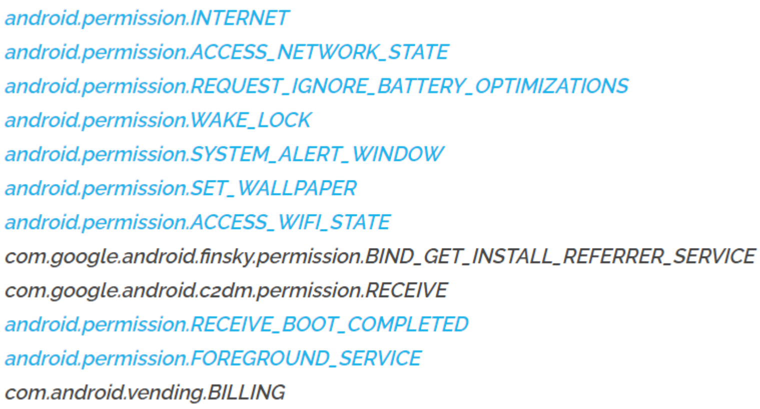

APK
Decompile
Web scraping apps discription
Extract permissions from manifest.xml
Extract permissions description for each permissions
Data preprocessing
Data preprocessing
Semantic analysis
Overclaim-permission

Figure 4.1: Proposed methodology



android.permission.INTERNET
android.permission.ACCESS_NETWORK_STATE
android.permission.REQUEST_IGNORE_BATTERY_OPTIMIZATIONS
android.permission.WAKE_LOCK
android.permission.SYSTEM_ALERT_WINDOW
android.permission.SET_WALLPAPER
android.permission.ACCESS_WIFI_STATE
com.google.android.finsky.permission.BIND_GET_INSTALL_REFERRER_SERVICE
com.google.android.c2dm.permission.RECEIVE
android.permission.RECEIVE_BOOT_COMPLETED
android.permission.FOREGROUND_SERVICE
com.android.vending.BILLING

Figure 4.2: Extracted Permission List

4.2 Extracting apps description

Our proposed methodology also includes web scrapping of apps description in order to get the data of apps description based on which further analysis can be carried out. To extract the description of the app, a script has been made in which the URL of the app should be given as input. The process is shown below **Fig. 4.3**. Permission description is also extracted along with app description.

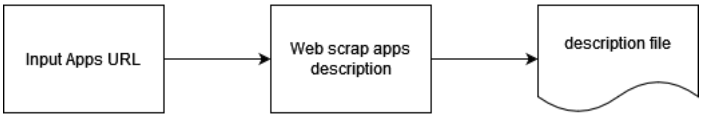


Figure 4.3: Extracting app description

4.3 Data Pre-processing

The next stage of our proposed methodology is data preprocessing. It is used to clean the sentences. which includes the removal of stop words. The list of stop words in English is already pre-defined, and we can get those sets of pre-defined stop words to check if any stop words are present in our description or not. After this, we can also remove our own defined extra words along with stop words. Further uppercase alphabets are converted to lowercase. characters in order to get the clean data that can be utilized properly for further semantic analysis as shown in **Fig. 4.4**

4.4 Semantic analysis

The final step of our proposed methodology is a semantic analysis of the sentences of app description and sentences of permission description using NLP algorithms. In this step, the permission used is checked with all the sentences of the descriptions of the apps, and based on this, the similarity score is generated. The above step is repeated for every permission, and the similarity score is generated based on this semantic analysis.

The semantic analysis is also done using various approaches to find the best and most accurate similarity score. Methods used for semantic analysis include Jaccard similarity, cosine similarity, TD-IDF (Term frequency-inverse document frequency), sentence transformer, word embedding techniques, and to name a few. From all of the above-mentioned semantic analysis techniques, Cosine similarity is the best-identified practice for finding the similarity score between 2 sentences. The working flow of the method of cosine similarity is shown in **Fig. 4.5 Fig. 4.4**

Also, there are many different approaches to finding out the cosine similarity. The approaches used in our semantic analysis are:

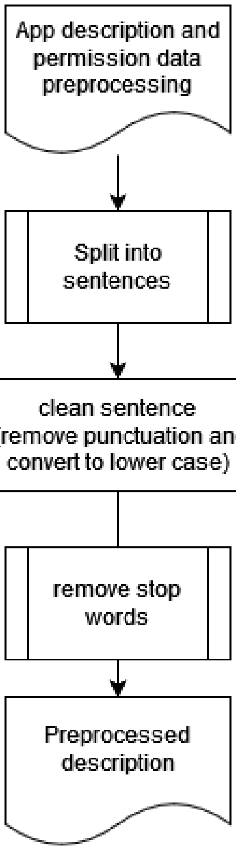

App description and permission data preprocessing
Split into sentences
clean sentence (remove punctuation and convert to lower case)
remove stop words
Preprocessed description

Figure 4.4: Pre processing of data

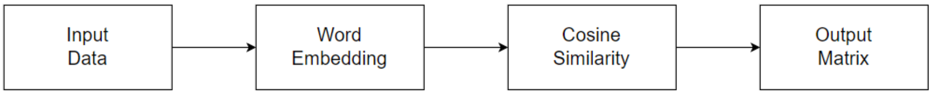


Figure 4.5: Pre processing of data

- Cosine similarity using SPACY.
- cosine similairty using MiNiLM (sentence transformer).
- Cosine similairty using Tensorhub (TFHub Universal Sentence Encoder).

4.4.1 Cosine similarity using Spacy

The semantic analysis is also done using various approaches to find the best and most accurate similarity score. Methods used for semantic analysis include Jaccard similarity, cosine similarity, TD-IDF (Term frequency-inverse document frequency), sentence transformer, word embedding techniques, and a few others. From all of the above-mentioned semantic analysis techniques, Cosine similarity is the best-identified practice for finding the similarity score between 2 sentences. Also, there are many different approaches to finding out the cosine similarity. The approaches used in our semantic analysis are:

4.4.2 Cosine similarity using MiNiLM (sentence transformer)

These techniques are called sentence transformers, which use the MiNiLm model to embed words. The model is widely known for mapping paragraphs and sentences, and the model used is 348-dimensional dense vector space. The data that we enter is converted into a tensor which is used to give a more accurate result. After converting into a tensor, the similarity is checked for each of the permission with each sentence.

4.4.3 Cosine similairty using Tensorhub (TFHub Universal Sentence Encoder)

Tensorflow hub is a sentence encoder technique that uses the pre-trained model of vectors for embedding the words. This method fits our proposed methodology because it is designed to check the semantic similarity for the sentences or small paragraphs rather than just checking the similarity between 2 words. The app description data and the permission description data are given as input, and then the first step is embedding the data.. After embedding the data, cosine similarity is checked for each of the permission sentences with each of the app description sentences. The output of the cosine similarity score is then stored in the form of a matrix.

4.4.4 Keyword based matching

Keyword-based matching is done using the rule-based matching algorithm in which we can define our custom set of words. The algorithm only words for those custom set of words and try to find the exact similar word from the paragraph. Rule-based matching has various attributes that we can give to the keyword we want to find a similar word. The attributes are: ORTH - which finds out the exact similar word, LOWER: which is used to find the word with lowercase, and many other attributes are used.

In our proposed method, keyword-based matching is done to find out the words similar to the name of the permission. In this, the permission grouping is done based on the permission that falls under the same category, for example, ACCESS_core FINE LOCATION, ACCESS_core COURSE LOCATION, ACCESS_core BACKGROUND LOCATION. All this permission falls under the same category and has similar usage. Location is the group of permission. This set of keywords is identified manually by reading the description of the apps that actually use this permission. Further, this keyword is used to detect any similar word in the description of the app that uses that particular permission. If yes, then the permission is genuinely used, and if no keyword is found, the permission can be categorized as over-claim. The working flow of this method is shown in figure. **Fig. 4.6**

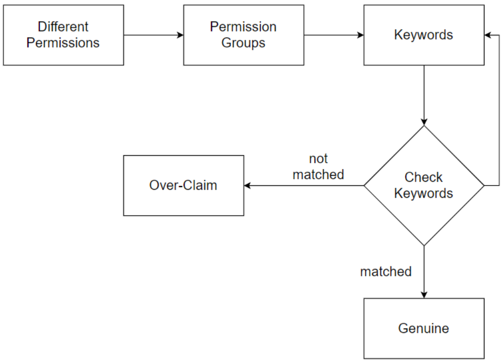


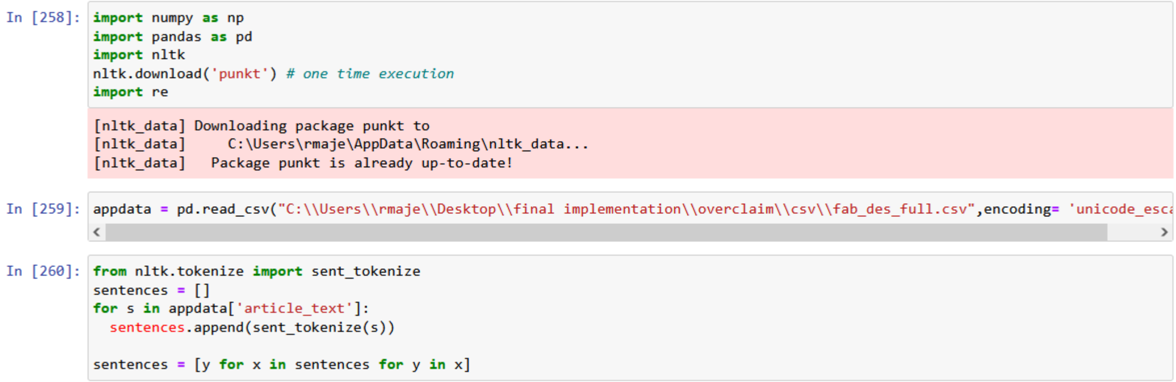
Figure 4.6: Pre processing of data

Chapter 5

Implementation

To start the implementation, I have download apps of 30 different categories. The APK files of these applications are downloaded, and then, using the decompilation tool, each app's source code is decompiled. then the permission description is extracted.

The next step is to tokenize the data. The data is stored in the CSV file in paragraph form as the downloaded app description is in the form of paragraph. The tokenization is done using NLTK (Natural language toolkit) tokenization as shown in **Fig. 5.1**.

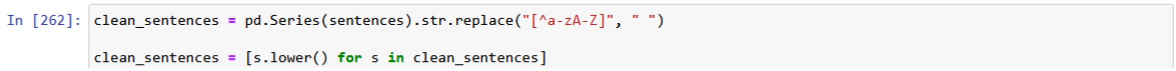


```
In [258]: import numpy as np
          import pandas as pd
          import nltk
          nltk.download('punkt') # one time execution
          import re

          [nltk_data] Downloading package punkt to
          [nltk_data]     C:\Users\rmaje\AppData\Roaming\nltk_data...
          [nltk_data]   Package punkt is already up-to-date!

In [259]: appdata = pd.read_csv("C:\\Users\\rmaje\\Desktop\\final implementation\\overclaim\\csv\\fab_des_full.csv",encoding= 'unicode_esca

In [260]: from nltk.tokenize import sent_tokenize
          sentences = []
          for s in appdata['article_text']:
            sentences.append(sent_tokenize(s))

          sentences = [y for x in sentences for y in x]
```

Figure 5.1: Tokenizing data

Along with tokenization, the cleaning of sentences is also done. Cleaning of sentences includes removing the punctuation marks, and also the words that contain Upper case alphabets are converted into lower case alphabets to increase the accuracy of the result, which is shown in **Fig. 5.2**.

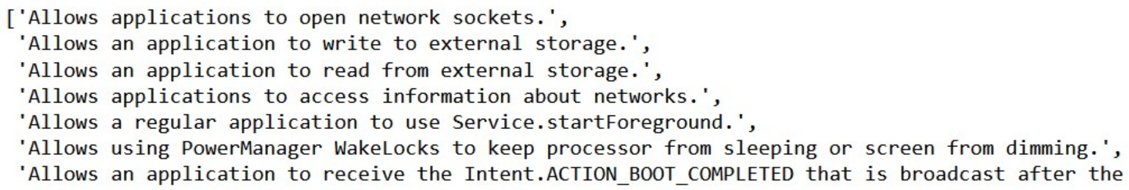
After cleaning the sentences, the stop words are removed from the sentences in order to increase the accuracy of measuring the sentence similarity. The cleaning of sentences is



```
In [262]: clean_sentences = pd.Series(sentences).str.replace("[^a-zA-Z]", " ")

          clean_sentences = [s.lower() for s in clean_sentences]
```

Figure 5.2: cleaning of sentences

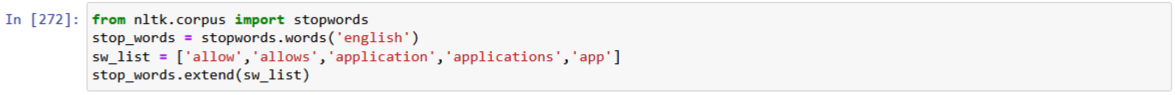
also done for the sentences of the permission description. Based on the results achieved by cleaning the sentences and removing the stop words, it has been identified that the words such as Access, Allow, Allows, Application, Applications, Required, Apps, and App are used in common in many permission descriptions, as shown in **Fig. 5.3**.



```
['Allows applications to open network sockets.',
 'Allows an application to write to external storage.',
 'Allows an application to read from external storage.',
 'Allows applications to access information about networks.',
 'Allows a regular application to use Service.startForeground.',
 'Allows using PowerManager WakeLocks to keep processor from sleeping or screen from dimming.',
 'Allows an application to receive the Intent.ACTION_BOOT_COMPLETED that is broadcast after the
```

Figure 5.3: Common words used in permission description

This can lead to false results because similar words are used in different permissions, which have different uses. For exmaple words like "allows," "an" and "application" are commonly used, which can lead to false-positive while checking semantic similarity because the meaning and functionality of this two permission are very different, but they share common words. To overcome the issue these common words are also removed along with the stop words while cleaning the sentences as shown in the **Fig. 5.4**.

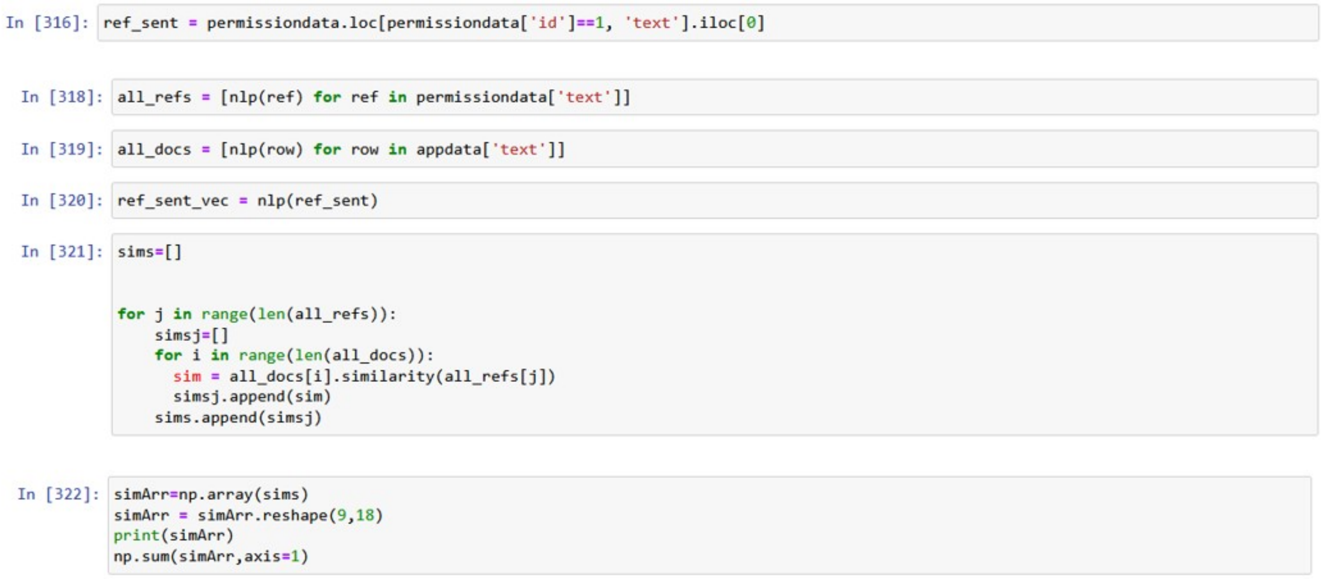


```
In [272]: from nltk.corpus import stopwords
          stop_words = stopwords.words('english')
          sw_list = ['allow','allows','application','applications','app']
          stop_words.extend(sw_list)
```

Figure 5.4: removal of extra words

After cleaning the data, all three algorithms are used to check the sentence similarity. First, by using spacy, the model for word embedding is used. Both the files containing app description and permission description are loaded, and the sentences presented in the column name 'text' in both files are used for analysis. After that, each of the permission sentences is checked with each of the description sentences, and the output of the similarity score is printed in the form of an array. The Above process is shown in the **Fig. 5.5** .

Similarly, the sentence similarity is checked using the other two algorithms in which the app description data and permission description data is loaded into the two different



```
In [316]: ref_sent = permissiondata.loc[permissiondata['id']==1, 'text'].iloc[0]

In [318]: all_refs = [nlp(ref) for ref in permissiondata['text']]

In [319]: all_docs = [nlp(row) for row in appdata['text']]

In [320]: ref_sent_vec = nlp(ref_sent)

In [321]: sims=[]


          for j in range(len(all_refs)):
              simsj=[]
              for i in range(len(all_docs)):
                sim = all_docs[i].similarity(all_refs[j])
                simsj.append(sim)
              sims.append(simsj)

In [322]: simArr=np.array(sims)
          simArr = simArr.reshape(9,18)
          print(simArr)
          np.sum(simArr,axis=1)
```

Figure 5.5: Similarity check process

arrays, and using different models, the embedding is done. Now each of the permission sentences is checked with each of the description sentences, and the output of the similarity score is printed in the form of an array.

By executing the different algorithms, it is identified that the sentence transformer using MiNiLm gives the best accurate result compared to semantic similarity using spacy and Tensorflow hub Universal sentence encoder. To give the better understanding of the results to find out over-claim permission, a detailed comparison of only dangerous permission is done. The comparison is made based on the apps that genuinely use the dangerous permission and the apps that over-claim the same dangerous permission. Based on this comparison, it can be easily identified that the similarity score of the permission sentence with apps description sentence that uses the permission genuinely is much more higher than the similarity score of that permission sentence with the apps description sentences which over-claims the permission and has the app has not defined anything regarding that permission in the description. The comparison is presented in the **Table. 5.1** and **Table. 5.2**. From this we can clearly identify that the permission named location which is dangerous permission. This dangerous permission is used genuinely by some apps and in some apps this permission is over-claim. The similarity score for the apps that uses the permission genuinely is 0.01 is lowest and 0.4 is highest.where as for the apps that over-claim the location permission has similarity score of -0.01as lowest and 0.18 and highest. The highest score in over-claim is also lower than the lowest score for the app that uses the app genuinely Based on this similarity score we can identify that we have successfully detected the use of over-claim permission.

PN	Contact			location			storage		
	App name	similarity		App name	similarity		App name	similarity	
		Low	High		Low	High		Low	High
1	To it later	0.1	0.3	Path guide	0.15	0.4	Ente encrypt	0.1	0.3
2	Z dailer	0.1	0.4	Door dash	0.01	0.3	Dnotes	0.07	0.35
3	To do list	0.01	0.2	Weather	0.01	0.3	Libre office mate	0.05	0.4

Table 5.1: Apps that over-claim dangerous permission

PN	Contact			location			storage		
	App name	similarity		App name	similarity		App name	similarity	
		Low	High		Low	High		Low	High
1	Door dash	-0.01	0.1	genius scanner	0.01	0.1	Ente encrypt	-0.02	0.1
2				Freed audio book	-0.01	0.18	Dnotes	-0.01	0.1

Table 5.2: Apps that over-claim dangerous permission

Another method that has been used apart from cosine similarity is keyword-based matching. In this method, the keywords related to the permission name is identified by reading the description of many sentences that uses the following permission. For example, location has many permission related to a location and uses the same functionality but has a different permission name. So the grouping of permission is done based on a similar functionality. The **Table. 5.3** demonstrates the permission grouping along with the keyword related to the use of that permission which is found in the app description. Based on this, we can determine if the permission exists in the source code is described in the app's description.

5.1 Results

Based on the analyses of apps using different semantic analysis method it is identified that cosine similarity score of the sentence transformer method using MiNiLM which convert the data into tensor at the time of embedding is more accurate compared to the results of Tenserflowhub universal statement encoder and sentence similarity using spacy. Further it is identified that after removing the common words, the accuracy is increased.

The comparison of the output result of the similarity score is made for the dangerous permission to identify the threshold value, which we can use to check if the permission

Permission group	permission name	Keywords related to permission name
contact	.READ_CONTACTS	important contact, dailer, contact, call logs
location	.ACCESS_FINE_LOCATION	location, geomagnetic, multiple location, area, destination, place.
	.ACCESS_COARSE_LOCATION	
camera	.CAMERA	video, photos, front camera, camera, capture, scan document.
sms	.SEND_SMS	message, sms
	.READ_SMS	
record audio	.RECORD_AUDIO	voice, voice message.

Table 5.3: Keywords related to permission

used is over-claim or not. It is identified that the similarity score of a permission sentence that is not related to the description sentence ranges between -0.10 to 0.15/0.20. The similarity score of permission that is related to the app's description ranges between 0.10 to 0.40. The threshold value based on our results is around 0.20. So if the permission's similarity score with the app's description is below 0.20 the permission is over-claim for that app, and if the similarity score is greater than 0.20 the permission is genuinely used by the app.

Chapter 6

Challenges

There are many challenges faced at the time of performing analysis for both keyword-based matching and sentence similarity. While performing keyword-based matching, the main challenge faced is that many permission falls under the same group, so grouping should be done properly of similar kind of permission because there are common keywords for similar kind of permission. Another challenge faced while performing keyword-based matching is that contact permission was used in one of the app descriptions. While performing keyword analysis, the word contact was detected and matched. But the sentence was ”choose NO-CONTACT delivery”. It means that you can choose to have no in-person contact delivery and the sentences have no similarity with the usage of contact from a user’s device. So to overcome this, We have used sentence similarity using semantic meaning, which is achieved using NLP algorithms.

Chapter 7

Conclusion

To detect the over-claim permission, we have done the analysis using three different algorithms that find the cosine similarity score between the permission description statements and app descriptions statement. We have done this analysis by cleaning the statements by removing stop words. However, it was identified that there are some common words used in every permission description sentence which leads to less accurate results. So based on this, removing the common words is also done, and it results in a more accurate sentence similarity score. The analysis is done twice before removing extra words and after removing extra words. Based on the results of this, it is identified that the sentence transformer using MiNiLM (after removing extra stop words) outperforms other semantic similarity methods. To detect the over-claim of permission, we have done the analysis using three different algorithms that find the cosine similarity score between the permission description statements and app descriptions statement. We have done this analysis by cleaning the statements with removing stop words. But it was identified that some common words are used in every permission description sentence, which leads to less accurate results. So based on the removing of the common words is also done and it results in achieving more accurate sentence similarity score. The analysis is done twice before removing extra words and after removing extra words. Based on the results of this, it is identified that the sentence transformer using MiNiLM (after removing extra stop words) outperforms other semantic similairty methods.

Bibliography

[1] "Number of smartphone subscriptions worldwide from 2016 to 2022." https://developer.android.com/guide/topics/permissions/overview, 2021. Accessed: 2022-03-15.

[2] "Number of smartphone subscriptions worldwide from 2016 to 2022." https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/, 2022. Accessed: 2022-02-30.

[3] "Mobile operating system market share worldwide." https://gs.statcounter.com/os-market-share/mobile/worldwide, 2022. Accessed: 2022-03-30.

[4] J. Zhang, C. Tian, Z. Duan, and L. Zhao, "Rtpdroid: Detecting implicitly malicious behaviors under runtime permission model," *IEEE Transactions on Reliability*, vol. 70, no. 3, pp. 1295–1308, 2021.

[5] F. Guyton, *Feature Selection on Permissions, Intents and APIs for Android Malware Detection*. PhD thesis, Nova Southeastern University, 2021.

[6] E. Chatzoglou, G. Kambourakis, and V. Kouliaridis, "A multi-tier security analysis of official car management apps for android," *Future Internet*, vol. 13, no. 3, p. 58, 2021.

[7] P. Pande, K. Mallaiah, R. K. Gandhi, A. K. Medatiya, and S. Srinivasachary, "Fine grained confinement of untrusted third-party applications in android," in *2021 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS)*, pp. 372–376, IEEE, 2021.

[8] F.-H. Hsu, N.-C. Liu, Y.-L. Hwang, C.-H. Liu, C.-S. Wang, and C.-Y. Chen, "Dpc: A dynamic permission control mechanism for android third-party libraries," *IEEE*

Transactions on Dependable and Secure Computing, vol. 18, no. 4, pp. 1751–1761, 2019.

[9] M.-Y. Su, S.-S. Chen, T.-R. Wu, H.-S. Chang, and Y.-L. Liu, "Permission abusing by ad libraries of smartphone apps," in *2019 Eleventh International Conference on Ubiquitous and Future Networks (ICUFN)*, pp. 475–477, IEEE, 2019.

[10] Z. Wu, H. Lee, and S. U.-J. Lee, "An empirical study on the impact of permission smell in android applications," *Journal of the Korea Society of Computer and Information*, vol. 26, no. 6, pp. 89–96, 2021.

[11] J. Tang, R. Li, H. Han, H. Zhang, and X. Gu, "Detecting permission over-claim of android applications with static and semantic analysis approach," in *2017 IEEE Trustcom/BigDataSE/ICESS*, pp. 706–713, IEEE, 2017.

[12] M. Shah, "Detecting over-claim permissions and recognising dangerous permission in android apps," *International Journal of Information and Computer Security*, vol. 17, no. 1-2, pp. 204–218, 2022.

[13] Y. Du, M. Cui, and X. Cheng, "A mobile malware detection method based on malicious subgraphs mining," *Security and Communication Networks*, vol. 2021, 2021.

[14] H. Alshahrani, H. Mansourt, S. Thorn, A. Alshehri, A. Alzahrani, and H. Fu, "Ddefender: Android application threat detection using static and dynamic analysis," in *2018 IEEE International Conference on Consumer Electronics (ICCE)*, pp. 1–6, IEEE, 2018.

[15] G. Tao, Z. Zheng, Z. Guo, and M. R. Lyu, "Malpat: Mining patterns of malicious and benign android apps via permission-related apis," *IEEE Transactions on Reliability*, vol. 67, no. 1, pp. 355–369, 2017.

[16] M. S. Saleem, J. Mišić, and V. B. Mišić, "Examining permission patterns in android apps using kernel density estimation," in *2020 International Conference on Computing, Networking and Communications (ICNC)*, pp. 719–724, IEEE, 2020.

[17] A. Hamidreza and N. Mohammed, "Permission-based analysis of android applications using categorization and deep learning scheme," in *MATEC Web of Conferences*, vol. 255, p. 05005, EDP Sciences, 2019.

[18] "Number of smartphone subscriptions worldwide from 2016 to 2022." https://shoutem.com/blog/top-10-android-app-category-on-google-play/, 2021. Accessed: 2022-03-10.

[19] I. Almomani, A. AlKhayer, and M. Ahmed, "An efficient machine learning-based approach for android v. 11 ransomware detection," in *2021 1st International Conference on Artificial Intelligence and Data Analytics (CAIDA)*, pp. 240–244, IEEE, 2021.

[20] A. Șandor and G. Tonț, "Android social applications permission overview from a privacy perspective," in *2021 16th International Conference on Engineering of Modern Electric Systems (EMES)*, pp. 1–4, 2021.

[21] J. J. Flors-Sidro, M. Househ, A. Abd-Alrazaq, J. Vidal-Alaball, L. Fernandez-Luque, and C. L. Sanchez-Bocanegra, "Analysis of diabetes apps to assess privacy-related permissions: systematic search of apps," *JMIR diabetes*, vol. 6, no. 1, p. e16146, 2021.

[22] Y. BAI, S. CHEN, Z. XING, and X. LI, "Argusdroid: Detecting android malware variants by mining permission-api knowledge graph,"

[23] A. K. Singh, G. Wadhwa, M. Ahuja, K. Soni, and K. Sharma, "Android malware detection using lsi-based reduced opcode feature vector," *Procedia Computer Science*, vol. 173, pp. 291–298, 2020.

[24] J. Xu, Y. Li, R. Deng, and K. Xu, "Sdac: A slow-aging solution for android malware detection using semantic distance based api clustering," *IEEE Transactions on Dependable and Secure Computing*, 2020.

[25] X. Pei, L. Yu, and S. Tian, "Amalnet: A deep learning framework based on graph convolutional networks for malware detection," *Computers & Security*, vol. 93, p. 101792, 2020.

[26] H. Zhang, S. Luo, Y. Zhang, and L. Pan, "An efficient android malware detection system based on method-level behavioral semantic analysis," *IEEE Access*, vol. 7, pp. 69246–69256, 2019.

[27] Y. Liu, N. Xi, and Y. Zhi, "Nleu: A semantic-based taint analysis for vetting apps in android," in *2021 International Conference on Networking and Network Applications (NaNA)*, pp. 327–333, IEEE, 2021.

[28] O. Olukoya, L. Mackenzie, and I. Omoronyia, "Security-oriented view of app behaviour using textual descriptions and user-granted permission requests," *Computers & Security*, vol. 89, p. 101685, 2020.

[29] J. Liu, D. He, D. Wu, and J. Xue, "Correlating ui contexts with sensitive api calls: Dynamic semantic extraction and analysis," in *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*, pp. 241–252, IEEE, 2020.

[30] Y. Feng, L. Chen, A. Zheng, C. Gao, and Z. Zheng, "Ac-net: Assessing the consistency of description and permission in android apps," *IEEE Access*, vol. 7, pp. 57829–57842, 2019.

[31] L. Yu, X. Luo, C. Qian, and S. Wang, "Revisiting the description-to-behavior fidelity in android applications," in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, vol. 1, pp. 415–426, IEEE, 2016.