A SECURITY RELATED AND EVIDENCE-BASED HOLISTIC RANKING AND COMPOSITION FRAMEWORK FOR DISTRIBUTED SERVICES

by

Nahida Sultana Chowdhury

A Dissertation

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Doctor of Philosophy

Department of Computer and Information Science at IUPUI

Indianapolis, Indiana

May 2021

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. Rajeev R. Raje, Chair

Department of Computer and Information Science

Dr. Mihran Tuceryan

Department of Computer and Information Science

Dr. James Hill

Department of Computer and Information Science

Dr. Yuni Xia

Department of Computer and Information Science

Approved by:

Dr. Shiaofen Fang

Dedicated to my father

ACKNOWLEDGMENTS

It has been an honor to be a Ph.D. graduate student at the Department of Computer and Information Science (CSCI) at IUPUI (Indiana University-Purdue University, Indianapolis). I would like to take this opportunity to recognize and appreciate the people who have provided a great deal of support and assistance throughout my research at IUPUI.

I would first like to express my gratitude to my advisor, Dr. Rajeev R. Raje, for his kind guidance and supervision in every step throughout my research. This research would have never been accomplished without his assistance and dedicated involvement.

I would also like to thank my research committee members, Dr. Mihran Tuceryan, Dr. James Hill, and Dr. Yuni Xia, for being part of my Dissertation Committee and providing invaluable guidance.

I am also grateful to my collaborators at Software Engineering and Distributed Systems (SEDS) lab at room number SL 116 (especially Saurabh, Ayush, and Umesh) for being there to engage in discussions and for sharing their valuable expertise.

I would also like to acknowledge the Department of Computer and Information Science faculty, the department staff (especially Rachel Molina and Suzy Leonard), and the School of Science IT staff for their dedicated support. I am also grateful to the Graduate Coordinator, Nicole Wittlief, and the Office of International Affairs (OIA) for their guidance and assistance.

My late father's words to go after my dream has been a continuous encouragement through my ups and downs. I remember my father Sahid Ullah Chowdhury today. He would have been the happiest person seeing my achievements. My sister, Raziya, ensured I stay in the course of life and studies. I could not survive without her. I am deeply indebted to my mother and my husband Lincoln for their constant support and continuous encouragement throughout my work.

Lastly, I would also like to thank all my family members and friends who, directly or indirectly, have supported me unconditionally in this venture.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

SERS	Security-related and Evidence-based Ranking Scheme
E-SERS	Enhanced Security Related and Evidence-based Ranking Scheme
Apps	An application downloaded to a mobile device
QoS	Quality of Service
ML	Machine Learning
NB	Naive Bayes
GNB	Gaussian Naive Bayes
SVM	Support Vector Machine
KNN	K-Nearest Neighbor
LR	Logistic Regression
DT	Decision Trees
RF	Random Forest
APK	Android application package
JAR	Java Archive
ICFG	Inter-Procedural Control Flow
IMEI	International Mobile Equipment Identity
API	Application Programming Interface
NLP	Natural Language Processing
NLTK	Natural Language Tool Kit
NLU	Natural Language Understanding
AI	Artificial Intelligence
DST	Dempster–Shafer Theory
SL	Subjective Logic
AUC	Area Under Cover
TP	True Positive
TN	True Negative
FP	False Positive
FN	False Negative
TPR	True Positive Rate

FPR	False Positive Rate
TNR	True Negative Rate
FNR	False Negative Rate
ODAS	Online Document Arrangement System
WFIP	Weather Forecast from IP Address

ABSTRACT

The number of smart mobile devices has grown at a significant rate in recent years. This growth has resulted in an exponential number of publicly available mobile Apps. To help the selection of suitable Apps, from various offered choices, the App distribution platforms generally rank/recommend Apps based on average star ratings, the number of installs, and associated reviews — all the external factors of an App. However, these ranking schemes typically tend to ignore critical internal factors (e.g., bugs, security vulnerabilities, and data leaks) of the Apps. The AppStores need to incorporate a holistic methodology that includes internal and external factors to assign a level of trust to Apps. The inclusion of the internal factors will describe associated potential security risks. This issue is even more crucial with newly available Apps, for which either user reviews are sparse, or the number of installs is still insignificant. In such a scenario, users may fail to estimate the potential risks associated with installing Apps that exist in an AppStore.

This dissertation proposes a security-related and evidence-based ranking framework, called SERS (Security-related and Evidence-based Ranking Scheme) to compare similar Apps. The trust associated with an App is calculated using both internal and external factors (i.e., security flaws and user reviews) following an evidence-based approach and applying subjective logic principles. The SERS is formalized and further enhanced in the second part of this dissertation, resulting in its enhanced version, called as E-SERS (Enhanced SERS). These enhancements include an ability to integrate any number of sources that can generate evidence for an App and consider the temporal aspect and reputation of evidence sources. Both SERS and E-SERS are evaluated using publicly accessible Apps from the Google PlayStore and the rankings generated by them are compared with prevalent ranking techniques such as the average star ratings and the Google PlayStore Rankings. The experimental results indicate that E-SERS provides a comprehensive and holistic view of an App when compared with prevalent alternatives. E-SERS is also successful in identifying malicious Apps where other ranking schemes failed to address such vulnerabilities.

In the third part of this dissertation, the E-SERS framework is used to propose a trust-aware composition model at two different granularities. This model uses the trust score computed by E-SERS, along with the probability of an App belonging to the malicious category, as the desired attributes for selecting a composition as the two granularities. Finally, the trust-aware composition model is evaluated with the average star rating parameter and the trust score.

A holistic approach, as proposed by E-SERS, to computer a trust score will benefit all kinds of Apps including newly published Apps that follow proper security measures but initially struggle in the AppStore rankings due to a lack of a large number of reviews and ratings. Hence, E-SERS will be helpful both to the developers and users. In addition, the composition model that uses such a holistic trust score will enable system integrators to create trust-aware distributed systems for their specific needs.

CHAPTER 1. INTRODUCTION

Current mobile applications (“Apps”) markets (“AppStores”), such as the Google PlayStore, Apple AppStore, Amazon AppStore, and Windows Phone App Store, have over 5 million Apps (as of the 3rd quarter of 2020) in total [1] supporting almost every kind of service that we need in our daily life, presented in Figure 1.1. In these AppStores, for any category (e.g., photo editor), there are many similar Apps offered by different developers.

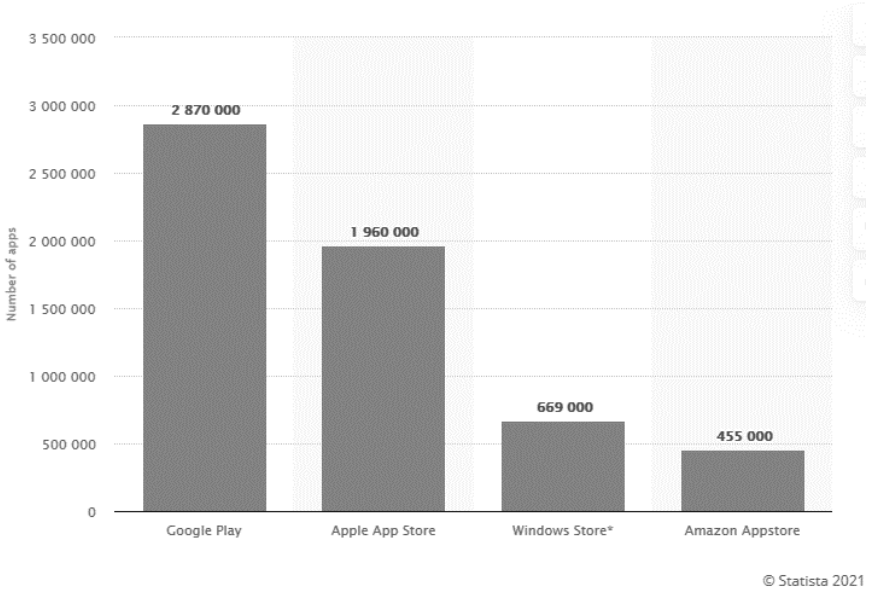

Figure 1.1. Number of apps available in leading AppStores as of 3rd quarter 2020 [1]

Too many options for the desired functionality make the App selection process challenging. One of the traditional ways to address this challenge is to look at the average star rating score (out of 5) provided by the host AppStore. Other prevalent approaches include manual/automatic reading/analyzing of reviews, monitoring top lists, and experimenting with other metrics such as the numbers of installs, updates, and downloads. These AppStores allow the user to assess an App by providing reviews and a star rating on a scale from 1 to 5. These AppStores use the weighted average star ratings score to promote specific Apps [2]. A high average rating indicates a better App while comparing similar Apps. Many studies have indicated that App ratings and associated reviews correlate positively with downloads and sales of Apps ([3] [4] [5] [6] [7]) - a high number of ratings lead to high number of downloads of an App. We carried out a simple informal survey to assess this observation – our survey audience contained Computing students and professionals.

We sent our survey to a balanced sample of the general population in the Computing domain. The survey was conducted anonymously as we did not request the users to provide their demographic data. We asked the following questions: "*In general, what is the most important factor that users considered to assess an App before downloading*?" – we received 130 responses. The response summary is given below [8]:

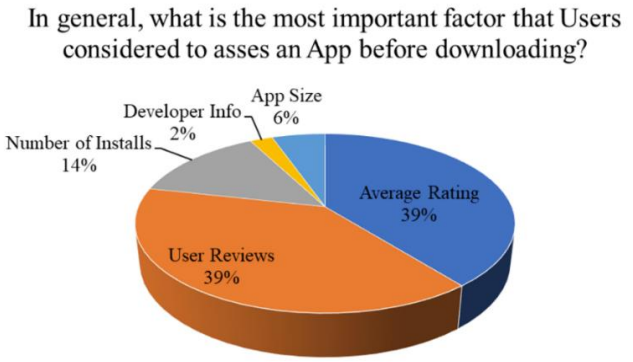

Figure 1.2. Survey response on App evaluating factors.

The survey response reveals that to pick the right App to download users mostly prefer to look at the Average Rating (39%) or User Reviews (39%). In [9], Lim et al. has conducted a comprehensive survey of App's users across the world which also shared the similar findings. If a user is more cautious about their selection, then they may look at the reviews given by other users. The number of reviews is large for many Apps and it makes the task of selecting a particular App very tedious. As the two factors (reviews and rating score) are important for a user to select an App, developers try to manipulate these factors in different ways. These approaches have drawbacks, which are listed below.

Limitation of the Average Star Rating Score. According to [10], in May 2019, 55.5% of Apps had 4.2 stars on average and 44.5% of the Apps had less than 3 stars ratings in the Google PlayStore. A typical user always prefers the highest rated App. The five-star rating system is biased, as the average rating is often prejudiced by users' two extreme choices of either five stars or one-star [11]. In the dataset that we have collected for this research, we noticed the same trend, where 77% of total reviews are rated with either five stars or one star. Also, the user ratings are often biased, and many times do not reflect the actual effectiveness of an App. Therefore, to select an App from available choices, the average star rating is not enough - a comprehensive analysis of all available evidences is needed.

Subjectivity of Reviews. Reviews are also not suitable, as a single metric for ranking and selection, due to reasons such as the poorly written structure, self-promotion of Apps by the companies and developers, and in some cases, developers requesting friends/family to give poor reviews to competing Apps [12]. This problem is even more complex with newly published Apps, for which either the user reviews or ratings are missing, or the number of downloads is still insignificant. In such situations, the users may not fully comprehend the risks associated with using a particular App.

The abovementioned metrics, due to these reasons discussed, do not address issues related to security risks (such as data leakage, insecure data storage, poor authorization, and sensitive information disclosure) associated with Apps. Many mobile Apps provide personalized services (e.g., *sms* services) to the users. These kinds of services usually ask users for explicit permissions to obtain personal information (e.g., contact details). For a less careful user, a wrong setting of permissions may result in potential risks associated with the unintended disclosure of their sensitive data. Recently, experts from the security domain highlighted the fact that many Apps in popular AppStores are not safe to use as they have shared user sensitive data with third parties. These findings indicate that millions of mobile phone users could be at risk. Some of the prominent sources that describe different malicious Apps are reported in [13] [14] [15] [16] [17] [18]. Once a user's data is compromised, it results in significant hardship to that user while trying to contain the impact of such an exposure. These issues indicate a need for a comprehensive scheme that will encompass various factors, including the `trust' about the behavior of the Apps. Using such a scheme, before downloading an App, users will be able to know the associated risk factors and their severity. This will help the user to pick a 'trustworthy' App from many choices. In literature, researchers have defined *Trust* in different ways. Some of the definitions of '*trust'* are:

- 'An entity can be trusted if it always behaves in the expected manner for the intended purpose [19].'
- 'Trust is the belief that certain events occur in the trustee under certain conditions [20].'
- 'Trustworthiness is a quality that is measured by the reputation of the service, (in the context of service selection problem). It reflects how the service managed to deliver the advertised QoS in past interactions [21].'

- ‘Trust is the truster’s opinion about a trustee based on the evidence, which is collected from the experiences of the trustee (e.g., truster’s experience about the trustee’s quality attributes and evaluated based on the trust policy and with the awareness of the context) [22].’

To incorporate the trust factor in the App selection and ranking process, in this dissertation, we have proposed an approach that computes a rating score based on the trustworthiness of an App named SERS (Security-related and Evidence-based Ranking Scheme). The trust of an App in the SERS is defined as “*the ability of an App not to disclose any critical data*” [23]; it is a modification of the prior definition that has been used in [24] [25]. We compute the trust of an App from internal and external evidence generated using that App's internal artifacts (e.g., code) and external artifacts (e.g., user reviews). Internal evidence indicates the developers’ view of the App, while the external evidence indicates the users’ view of the App.

In the SERS, to generate the trust score for an App, we apply the principles of the theory of evidence [26], Subjective Logic [27], static code analysis, static taint analysis, and Natural Language Processing (NLP). To examine the acceptance of such an approach we conducted another survey with the same audiences mentioned earlier. We asked the following questions: “*Which one of the following ranking schemes could be the right fit to evaluate an App*?” – we received 130 responses. The survey outcomes (shown in Figure 1.3) indicate that a combined ranking scheme (43.8%) is more acceptable than other existing ranking schemes, such as based on average user ratings, users’ review sentiments, and other internal and external factors. The SERS considers the comprehensive nature of an App than the other existing choices and thus, provides a better ranking of similar Apps.

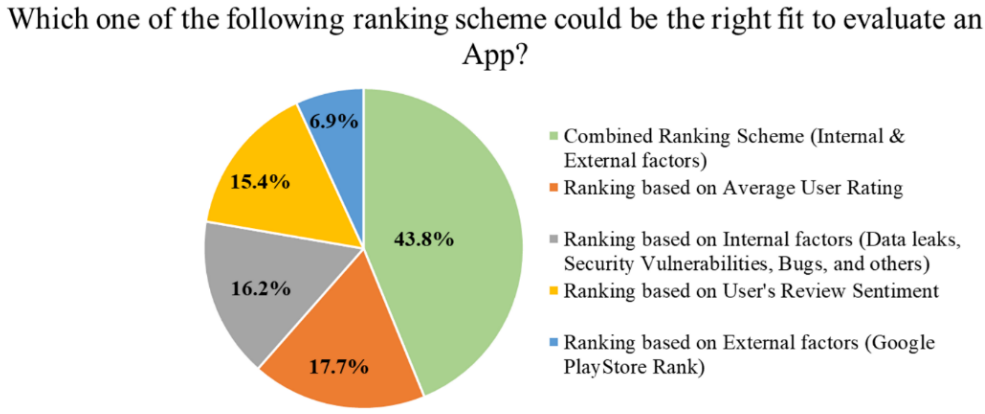

Figure 1.3. Survey on Ranking Schemes.

The SERS scheme is then formalized so that it can support any number of external and internal sources to generate different types of evidence. This enhancement of the SERS scheme is referred to as the E-SERS (Enhanced SERS). In addition, the E-SERS considers the temporal aspect of external evidence and the reputation of evidence sources.

Further, we have created a Web-based prototype that implements the E-SERS and has again empirically validated our approach in the context of the Google PlayStore. Our experiments indicate that the E-SERS is a more reliable alternative; in terms of security, privacy, and code quality; than any prevalent rating techniques that focus only on either the internal or the external perspective of an App. Besides, E-SERS is successful in identifying the malicious Apps where other ranking schemes failed to detect such vulnerabilities. Having a holistic approach also benefits newly published good Apps which follow proper security measures but struggle in AppStore ranking due to lack of large number of reviews and ratings.

In the last part of this dissertation, we have proposed a trust-aware service composition framework for mobile ecosystems at two levels of granularity. This framework uses the trust scores generated by E-SERS as the objective parameter. We have empirically evaluated this framework and existing models using two case studies.

Our experiments indicate that the proposed trust-aware service composition model performs better in execution time and can generate the better trustworthy service binding scheme than others.

1.1 Goal and Research Hypotheses

The overall goal of this dissertation is to quantify an App's trust using various available evidence (e.g., bugs, security vulnerabilities, user reviews, etc.) and develop a trust-aware composition framework to compose such trustworthy Apps to generate a trust-aware distributed system. Hence, two specific hypotheses of this research are:

- Quantifying the trustworthiness of an App, using the security and privacy-related vulnerabilities and considering its holistic view, will provide a better ranking scheme while comparing similar Apps.
- Such quantification of trust will enable the creation of a better and trust-aware composition model to generate a distributed system from selected Apps.

This research goal is achieved by identifying and analyzing the security vulnerabilities of an App and quantify its trustworthiness using principles of subjective logic, static code analysis, static taint analysis, and NLP. The hypotheses are empirically evaluated using prototypes created based on SERS and E-SERS.

1.2 Contributions

The contribution of this dissertation are as follows:

1) This dissertation proposes a security-related and evidence-based scheme, SERS, and its enhancement, SERS, to quantify the trust of an App. SERS and E-SERS use formalism and provide a holistic view of the trust of an App.
2) Both SERS and E-SERS are empirically validated using publicly accessible Apps from the Google PlayStore and their outcomes are compared with prevalent ranking techniques such as those based on the average star ratings and the Google PlayStore Rankings.
3) This dissertation also proposes and empirically evaluates a trust-aware composition model for creating an ensemble of Apps selected using the E-SERS.

1.3 Organization

The rest of the dissertation structure is given below:

In Chapter 2, we present related literature. Chapter 3 provides an overview of the necessary background theories. Chapter 4 describes the proposed SERS and E-SERS in detail. Chapter 5 discusses the experimental results and analysis of E-SERS, along with a short overview of our E-SERS Web-based prototype. Chapter 6 outlines the approach for classifying an App as malicious or benign using its data flow features. Chapter 7 presents the proposed trust-aware composition framework, which uses the approach presented in Chapter 6. Finally, Chapter 8 concludes the dissertation by summarizing the contributions, possible limitations, and future directions of this research.

CHAPTER 2. RELATED LITERATURE

There exist few related approaches which model trust-related attributes. Relevant works based on the principles we applied to our framework are presented here. In this dissertation, the related works are described in the following sections. Sections 2.1 to 2.4 are customized contents that have been taken from our previously published papers [23] [24] [25].

2.1 Sentiment Analysis

It is a popular approach to assess reviews and propose recommendations in many areas - e.g., products, and movies [28][29][30]. Sentiment analysis has been also applied to AppStore reviews to manage and advance Apps in a few research efforts [31][32]. Sangani et al. [33], have applied the review-to-topic mapping approach where a list of topics helps the developer to identify the most demanded feature by the users to be on the top of the rank list. A similar effort has been made by Pagano and Maalej [34] and Palomba et al. [35], where they examined the types of user feedback and unveiled how developers monitor user reviews to update in terms of users' rating.

Only a few research efforts have attempted to quantify the trust tuples based on the reviews of Apps [36][37]. The main difference between our and these research works is they have not used a combined view of the Apps. We have adopted a similar theory, based on the Subjective Logic, for merging internal and external evidence. Besides, based on the trust score, we form the rank ordering of the Apps.

2.2 Static Code Analysis

Static code analysis tools are capable of detecting potential logical inconsistency, run-time errors (e.g., dereferencing a null pointer), and security violations (e.g., SQL injection) in an application. This analysis can be performed at different levels such as the binary code, bytecode, and source code. FindBugs [38] and Jlint [39] both are open-source and static byte-code analyzers for Java. FindBugs can identify more bug types than Jlint (e.g., unreachable code). Khalid et al. [40] applied FindBugs to discover which categories of bugs occur more often in low rated Apps

rather than in high rated Apps by investigating the associations among each category of bugs in an App and the corresponding App rating. In our research, we have employed FindBugs to recognize different categories of bugs in terms of bug confidence levels (high, medium and low) and bug ranks (1 to 20). The bug rank describes the severity of the bug and the confidence level symbolizes the trust of the tool about the bug existences.

Functional testing is one of the popular approaches for predicting bugs in a mobile App. For example, Espresso [41], implemented by Google, is able to determine the synchronization issues which can lead to unsafe thread interactions. But it works on an emulator and results in identifying limited performance issues (such as display screen size, memory limitation, etc.). Another helpful tool is Monkey [42], which appears with the Android software development toolkit. It can only generate user interface events where the users have to define the desired number of events. Bug Rocket, is an automated testing tool, provided by Ma et al. [43], that associates distributed testing settings with testing automation based on reverse engineering procedures. Other existing tools to test mobile Apps include SwiftHand [44], EvoDroid [45], and Dynodroid [46].

2.3 Data Flow analysis

The Android security model is a permission-based access control system. Here, an App may request to access the security and privacy-sensitive data in their manifest file. Several research attempts have confirmed that permissions play an important role (i.e., [47]) to identify the malicious activities of Apps. Research efforts on permission-based risk include DroidRanger [48], and DroidRisk [49]. DroidRisk deals with the occurrences and the number of permissions an App demands. Sarma et al. [47] and Gates et al. [50] assigned high-risk values to permissions that are severe and not often asked by the Apps in the same category.

Examining the permissions alone to assess the risk is not sufficient enough because -- all the requested permissions may not be utilized during execution and without counting the possibility to send the sensitive information [51]. To overcome this constraint, we consider the faulty data flows only and their corresponding permissions. A similar approach is suggested by Mirzaei et al. [52], where it fuses a probabilistic model to predict the existence of data flows with the influence of flow is in benign and malicious Apps.

Static taint analysis tools are usually based on sensitive API calls tracking. Current static taint analysis tools for Android Apps are: FlowDroid [53], TaintDroid [54], AndroidLeaks [55], DroidSafe [56], and others. Compared with others, FlowDroid can identify a high number of data leaks while keeping the false positives rate low. Therefore, we have used FlowDroid in this research.

The static taint analysis tools are useful for detecting data leaks, but they do not cover all the security vulnerabilities. There are several tools to detect common security vulnerabilities. There is also a significant number of research for detecting different categories of bugs, such as Androbugs [57], QARK [58], JAADAS [59] are some other tools that are available to detect security vulnerabilities.

Our research is complementary to most of these efforts. Our aim is to quantify the trust of an App in terms of data confidentiality, not to detect any kind of malware.

2.4 Traditional Methods for App Ratings

Five most popular App-stores (Apple's AppStore, GooglePlay Store, Amazon AppStore, Windows Phone Store, and Blackberry AppWorld) use rating mechanisms known as the store rating. The store rating of an app is represented as a number of stars from 1 to 5 and is aggregated from individual user ratings. For example, in the Google PlayStore, the store rating of an app is the cumulative average of all individual user ratings over all the versions.

2.5 Ranking of Apps

There has not been any work on the ranking system which focus on both indirect and direct trust artifacts of Apps. All the existing research effort [60][61] on the ranking scheme are either based on an internal view or external view. In [60] Zhu et al. presented a hybrid ranking principle which is a combination of risk scores and overall rating. The risk factor is established based on the permission requested by the App and risk value is determined by examining each of the dangerous permissions App request. As we described earlier, using permissions alone to estimate risk has serious limitations and is inaccurate. As, Apps are usually over-privileged, and many permissions requested in the manifest might not be utilized throughout the execution. Therefore, we developed the risk assessment matrix based on only those permissions which contribute towards the malicious

activities or security flaws. Similarly, another ranking model is provided by Cen et al. [61] where a crowdsourcing ranking approach is performed to solve the app risk assessment problem from users' comments. However, user's comments are subjective and may vary from user to user. Thus, in our approach, we focus on both functional and non-functional perspectives of an App instead of focusing only one.

2.6 Malware App Detection through Static Analysis

A substantial amount of research efforts has been carried out in identifying Android malicious Apps using different machine learning techniques [62] [63] [64]. In [65], the author has utilized supervised machine learning algorithms (Support Vector Machine (SVM) and K-Nearest Neighbors (KNN)) to perform the classification of Apps into benign or malicious. The feature set was generated based on the App's manifest.xml file. The experimental findings have revealed 79.08% average accuracy to identify malicious Apps. Similarly, DERBIN [66] and Droidmat [67] detect malware through analyzing App's manifest.xml file. The DroidAnalyzer [68] also adopted static analysis approaches to detect malware Apps. It uses permissions, dangerous APIs, and keywords related to malicious behaviors to identify malicious Apps.

The work in [64] is also dedicated to anomaly detection of malicious Apps. The feature vectors were generated from system data (such as network data). Later, classified them with different machine learning algorithms (SVM, KNN, Random Forest (RF), Naïve Bayes (NB)). Here, the RF classification algorithm performs better than others.

In our work also we have adopted a static analysis approach. Here, the malware App is identified based on the data flow features. We have employed multiple classifiers, namely, SVM, KNN, RF, NB, Logistic Regression (LR), and Decision Tree (DT). LR and RF perform better than others, with 88% accuracy of classifying benign and malware Apps.

2.7 Service Composition

In service-oriented computing environments, manually discovering the composition results is always challenging due to the vast available services. Therefore, automated service composition is most desirable. In this dissertation, we first define a trust model then, based on trust, we have proposed an automated service composition framework. Trust plays a very significant role in

service composition. During the selection process, a service filter is utilized to ensure the trust of services selected for composition and finally return the most trustworthy binding scheme.

Recently, automatic service composition brought a lot of attention, but all of them are based on QoS properties [69] [70] [71] [72] which aim to determine optimal binding schemes in an automated way based on the user's request. However, there are few efforts have been made where trust-oriented service composition effort has been introduced [73] [74] by considering external factors (such as QoS, rating) for the composition. In our approach, the trust score is based on internal and external factors, giving a comprehensive service view.

CHAPTER 3. BACKGROUND PRINCIPLES

This chapter presents a summary of the background principles. The background section comprises an outline of related principles applied to this. Here, we present background about Static analysis tool, Static taint analysis tool, principles of Subjective Logic, Key Concepts related to Android, and rank correlation techniques – that are related to this dissertation work.

3.1 Static Analysis Tool

3.1.1 FindBugs

Among the other open-source static analysis tools, FindBugs is selected as it is able to analyze bytecode, can reduce the false-positive warnings [75], and is trained to identify over 400 potential bugs patterns. These bug patterns are classified into the following nine groups [76]:

- Bad Practice: code that violates recommended and necessary coding practice.
- Malicious Code Vulnerability: code that can be maliciously modified by other code.
- Multithreaded Correctness: code that could provoke problems in a multithreaded context.
- Dodgy Code: code that direct to errors.
- Correctness: code that might give different outcomes that were seemingly not what the developer expected.
- Performance: code that could be created in a different way to improve performance, indicate slow code.
- Internationalization: code that can utilize the use of encoding characters.
- Experimental: code that could pass on cleanup of database objects, steams, or other objects that require a cleaning operation.
- Security: code that can cause potential security problems.

Some sample bug patterns in nine different categories are given in Table 3.1.

Table 3.1. Samples of FindBugs warnings – Category wise (reproduced verbatim from [77])

Category	# of bug patterns	Samples
Bad Practice	84	Finalizer does not call superclass finalizer Class defines clone() but doesn't implement Cloneable Unchecked type in generic call
Malicious Code Vulnerability	15	May expose internal representation by returning reference to mutable object Field is a mutable array. Field should be both final and package protected
Multithreaded Correctness	45	Synchronization on Boolean Monitor wait() called on Condition Inconsistent synchronization
Dodgy Code	71	instanceof will always return true Useless assignment in return statement Non serializable object written to ObjectOutput
Correctness	142	Method attempts to access a result set field with index 0 Overwritten increment Comparing values with incompatible type qualifiers
Performance	27	Boxing a primitive to compare Maps and sets of URLs can be performance hogs Private method is never called
Internationalization	2	Consider using Locale parameterized version of invoked method Reliance on default encoding
Experimental	3	Potential lost logger changes due to weak reference in OpenJDK Method may fail to clean up stream or resource Method may fail to clean up stream or resource on checked exception
Security	11	HTTP Response splitting vulnerability Absolute path traversal in servlet JSP reflected cross site scripting vulnerability

For each reported bug, FindBugs assigns priorities that scale from 1 to 20. Here, 1 stand for the top priority, and 20 indicates the lowest priority bug. The warnings priority level relies on the confidence level, which is again classified into the high, medium, and low category of the tool regarding the existence of the bug.

- Confidence level High indicates that the discovered bug is definitely a real bug.
- Low indicates bugs are preferably false positives.
- Medium indicates bugs remain in the middle of these High and Low boundaries.

As FindBugs has a comparatively small number of false positives, mainly discovered bugs are considered valid bugs [76]. The execution sequence of how FindBugs works is presented in Figure 3.1, where APK presents the Android application package and JAR indicates Java Archive.

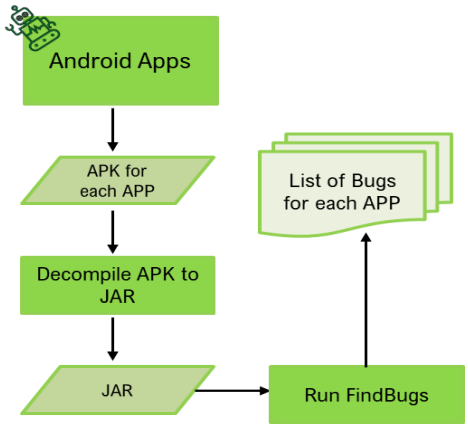

Figure 3.1. The execution sequence of FindBugs

3.2 Static Taint Analysis Tool

3.2.1 FlowDroid

FlowDroid [78], is an open-source static taint analysis framework applied to Android Apps with 86% precision and 93% recall. It traces the sensitive information associated with an App by starting at a predefined source and following the data flow until it gets a given sink. Figure 3.2 shows the overview of FlowDroid.

Android Apps are available in APK (Android Packages) format. The APK file is a compressed archive. When unzip the APK the framework explores the App for the following elements:

- lifecycle, and
- callback methods, which calls to sources and sinks.

This analyzation is achieved by evaluating various Android files, contains the

- *manifest* file,
- *dex* files; contains the executable code, and
- layout *XML* files

Then, FlowDroid generates the *main* model from the lifecycle and list of callback methods. This *main* model is then used to generate Inter-Procedural Control Flow graph (ICFG) and a call graph. From the identified sources, by traversing the ICFG the taint analysis tracks taints. The framework is designed with sources and sinks assembled by SuSi [79]. SuSi, is supervised machine-learning classifier that is able to label Android sources and sinks to permission-based Android API method signatures. The FlowDroid website [78] have the detailed list of available sources and sinks.

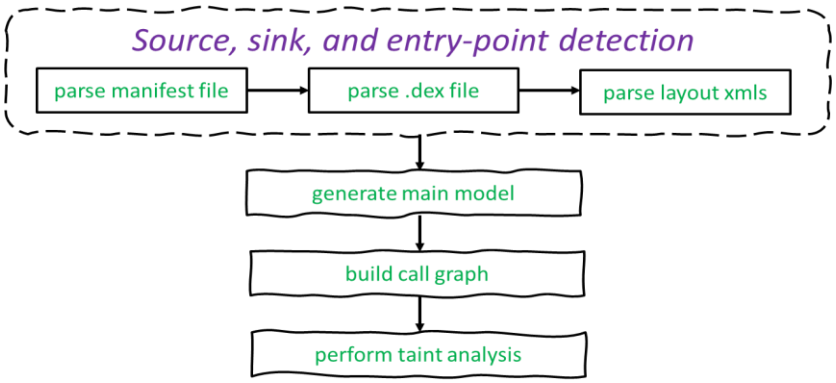

Figure 3.2. Architecture of FlowDroid (redrawn from [80])

FlowDroid records all identified information flows from sources to sinks. It provides extensive details (such as the API method's signature which attempts to read/write sensitive data from the App to third parties) regarding unauthorized leaks of confidential data. Android sources and sinks are the two main elements characterized by data flow to identify the flow path.

- An Android Source is an external resource from which data is read - for example, the *getDeviceId()* API method is a source that returns an IMEI (International Mobile Equipment Identity) into the application code.

- An Android Sink is an external resource to which data is written - for example, the *sendTextMessage()* PI method is a sink as both the message and the phone number it receives are non-constant.

FlowDroid used SuSi [4], which can categorize sources and sinks in the form of Android API method signatures. For instance, there is a category CONTACT_INFORMATION grouping all sources related to the user's whereabouts (e.g., *getContactList()*). In SuSi, the Android source APIs are categorized into 14 different categories, and similarly, SuSi categorizes the Android sink APIs into 16 different categories (given in Table 3.2).

Table 3.2. SuSi API Categories of Android Sources and Sinks

Source Categories	**Sink Categories**
ACCOUNT	ACCOUNT
LOCATION	CALENDAR
BLUETOOTH	AUDIO
CALENDAR	CONTACT
BROWSER	BROWSER
CONTACT	LOG
FILE	FILE
DATABASE	NETWORK
NFC	SMS_MMS
SETTINGS	NFC
NETWORK	PHONE_CONNECTION
SYNC	PHONE_STATE
UNIQUE_IDENTIFIER	SYNC
NO_CATEGORY	VOIP
	SYSTEM
	NO_CATEGORY

Here, the *NO_CATEGORY* points to sources and sinks grouped as non-sensitive in SuSi. We provide all the extracted source and sink APIs (gathered from FlowDroid) to SuSi to categorize and classify them into one of these above-mentioned categories.

3.3 Natural Language Processing

Sentiment analysis is a natural language processing scaling technique of determining the emotional depth in a piece of text. It is also known as “opinion mining”. Sentiment score makes it easier to understand how users feel. For example, from -1 to +1 indicates the most negative sentiment to the most positive sentiment. It is used to classify the emotion as positive, negative, and neutral. Sentiment analysis is broadly utilized in user reviews on various social media platforms (such as marketing, advertising, etc.). Our proposed E-SERS framework utilizes sentiment analysis techniques to compute external evidence (such as reviews) available in distributed platforms. In this research, we have used two libraries have been used, i.e., TextBlob, and IBM Watson NLU.

3.3.1 TextBlob

It is an open-source Python library based on NLTK (Natural Language Tool Kit) [81]. The library is used for evaluating text and determine the polarity score. Here, polarity is a value between [-1, +1] where -1 symbolizes negative, +1 means positive, and 0 indicates neutral sentiment. TextBlob analyses each word in a text and assigns a semantic score, then the score is weighted. The score is based on the polarity of each word in the sentence. Also, it returns another significant factor is Subjectivity. It is in the range of [0, 1] where 0 represents that the given sentence is objective, which implies that it is based on actual data, whereas 1 indicates that the sentence is subjective, which means that it is based on sentiments, perceptions, judgments, wishes, and affirmations of a person.

3.3.2 IBM Watson NLU

The IBM Watson Natural Language Understanding (NLU) API [82] is utilized to predict the preprocess reviews' sentiment through natural language processing. It can analyze and understand the text, including sentiment, emotion, keywords, language, entities, metadata, relations, and semantic roles. The API returns the sentiment score in the range of [-1, +1] and indicates whether a given review reflects the user's positive or negative sentiment.

3.4 Theory of Evidence

Theory of evidence [83] is also referred to as the Theory of belief functions or Dempster–Shafer theory Dempster – Shafer theory (DST). The DST framework is used for reasoning evidences with uncertainty. Here, the evidences are quantified into a tuple that consists of Belief (B), Disbelief (D), and Uncertainty (U). One of the major scopes of the Theory of evidence is to support a steady approach to fuse a diverse set of evidence from various sources.

For a given proposition, the Theory of belief functions depends on the number of evidences that are correlated to that proposition. It offers a set of procedures to fuse evidences about two similar propositions in a particular system. Other researchers have also proposed different techniques to combine a set of evidence. The work presented by this research, the E-SERS framework, uses the Dempster–Shafer theory of evidence models to aggregate and quantify an App's trust in the distributed platform.

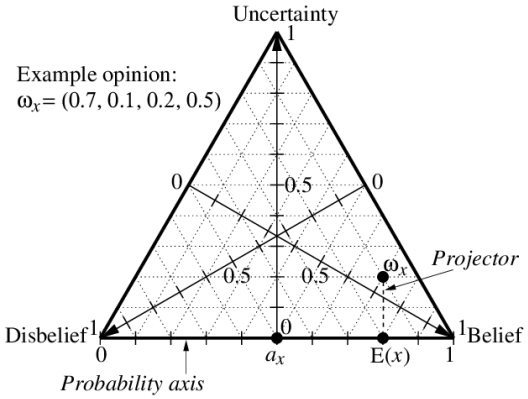

Figure 3.3. Opinion triangle (reproduced verbatim from [85])

3.5 Subjective Logic Principle

In this thesis, App's trust is quantified based on the gathered evidences. To present the trust of an App, we have applied the Subjective Logic (SL) principle. It is a probabilistic model which is introduced by Jøsang [84]. During calculations, SL considers belief and uncertainty. The Consideration of the evidences' uncertainty is the main strength of SL. It defines opinion in terms of belief ($B \in [0, 1]$), disbelief ($D \in [0, 1]$) and uncertainty ($U \in [0, 1]$). By utilizing the belief functions SL provides a set of logical operations to fuse diverse opinions.

The proposition is considered to be either true or false in the traditional probabilistic model. Due to skepticism's human nature, there is no perfect assurance whether a proposition is true or false in the real world. SL considers the factor of uncertainty which is a similar scenario to the real world. Here, the opinion is displayed on a triangle, as shown in Figure 3.3. Any point within the opinion triangle symbolizes as ω = <B, D, U> tuple. Like the traditional probabilistic model, in Subjective logic, the sum of belief, disbelief, and uncertainty is always 1. In the opinion triangle, there are three exceptional cases are:

- Absolute belief (1, 0, 0)
- Absolute disbelief (0, 1, 0)
- Absolute uncertainty (0, 0, 1)

3.5.1 Evidence to Opinion Mapping

An opinion about a proposition *p*, using source *s*, is represented as ω_p^s = <*b*, *d*, *u*, *a*>. Here, *b*, *d*, and *u* respectively denote the belief, disbelief, and uncertainty that proposition *x* can be trusted - is true, and *a* is the probability (base rate) that the proposition is correct, in absence of evidences. As proposed in [86], the formulas for calculating belief, disbelief, uncertainty, and base rate using positive and negative evidences are given below:

$$b = \frac{postive\ evidence}{total\ evidence+n} \quad (1)$$

$$d = \frac{postive\ evidence}{total\ evidence+n} \quad (2)$$

$$u = \frac{n}{total\ evidence+n} \quad (3)$$

$$a = \frac{1}{n} \quad (4)$$

The value `*n*' indicates possible outcomes about an evidence. In E-SERS, *n* is equal to 2 because an evidence can either be present or absent in the App.

3.5.2 Opinion Fusion

Apart from this opinion representation, the main strength of SL is that it supports different operators [10] for combining opinions from different sources. contains about 10 different operations, where the most important are as follows:

- Conjunction
- Disjunction
- Negation
- Ordering
- Discounting
- Consensus

3.5.2.1 Conjunction

Combine two opinions about propositions using conjunction ("AND") consists of determining a new opinion reflecting the conjunctive truth of both propositions. It must be expected that the opinion arguments in conjunction are independent. The "∧" is used to designate the conjunction operation.

Let $\omega_p = (b_p, d_p, u_p)$ and $\omega_q = (b_q, d_q, u_q)$ are opinions about two distinct trusted proposition p and q. Now, the opinion that these propositions can be trusted is symbolized by $\omega_{p \wedge q} = (b_{p \wedge q}, d_{p \wedge q}, u_{p \wedge q})$ such that

$$b_{p \wedge q} = b_p b_q \tag{5}$$

$$d_{p \wedge q} = d_p + d_q - d_p d_q \tag{6}$$

$$u_{p \wedge q} = b_p u_q + u_p b_q + u_p u_q \tag{7}$$

3.5.2.2 Disjunction

Combine two opinions about propositions using conjunction ("OR") consists of determining a new opinion reflecting the disjunctive truth of both propositions. It must be expected that the opinion arguments in disjunction are independent. The "∨" is used to designate the disjunction operation.

Let $\omega_p = (b_p, d_p, u_p)$ and $\omega_q = (b_q, d_q, u_q)$ are opinions about two distinct trusted proposition p and q. Now, the opinion that these propositions can be trusted is symbolized by $\omega_{p \vee q} = (b_{p \vee q}, d_{p \vee q}, u_{p \vee q})$ such that

$$b_{p \vee q} = b_p + b_q - b_p b_q \tag{8}$$

$$d_{p \vee q} = d_p d_q \tag{9}$$

$$u_{p \vee q} = d_p u_q + u_p d_q + u_p u_q \quad (10)$$

3.5.2.3 Negation

It is equivalent to logical "NOT" operation. Negation of an opinion consists of inverting the belief and disbelief. The "¬" is used to designate the negation operation.

Let $\omega_p = (b_p, d_p, u_p)$ is an opinion about a trusted proposition p. Now, the opinion that this proposition cannot be reliable is symbolized by $\neg\omega_p = (b_p, d_{\neg p}, u_{\neg p})$ such that

$$b_{\neg \mathrm{p}} = d_p \quad (11)$$

$$d_{\neg p} = b_p \quad (12)$$

$$u_{\neg p} = u_p \quad (13)$$

3.5.2.4 Ordering

Ordering operation is used when an agent have opinion about different propositions. Opinions can be ordered by selecting the opinion that contains the strongest belief. The "↑" is used to designate the negation operation.

Let $\omega_p = (b_p, d_p, u_p)$ and $\omega q = (b_q, d_q, u_q)$ are opinions about two distinct trusted proposition p and q. Now, the opinion that these propositions can be ordered is symbolized by $\omega_{p \uparrow q} = (b_{p \uparrow q}, d_{p \uparrow q}, u_{p \uparrow q})$ such that

IF ω_p and ω_q have different $\frac{b+u}{b+d+2*u}$ ratios then,

RETURN opinion with greatest $\frac{b+u}{b+d+2*u}$ ratio

ELSE

RETURN opinion with the least uncertainty (u)

3.5.2.5 Discounting

Discounting operation is used to compute trust transitivity. It is based on a probabilistic analysis of opinions. This helps to handle trust relations and reputation. The "⊗" is used to designate the negation operation.

Let the reputation of a source s, the r, as the opinion $\omega_s^r = (b_s^r, d_s^r, u_s^r, a_s^r)$ represent an opinion about the trusted source s and $\omega_p^s = (b_p^s, d_p^s, u_p^s, a_p^s)$ is an opinion that proposition p is

trusted based on source *s* opinion. Now, the opinion that this proposition *p* can be trusted is symbolized by $\omega_p = \omega_s^r \otimes \omega_p^s = (b_p, d_p, u_p, a_p)$ such that

$$b_p = b_s^r b_p^s \tag{14}$$

$$d_p = d_s^r d_p^s \tag{15}$$

$$u_p = d_s^r + u_s^r + b_s^r u_p^s \tag{16}$$

$$a_p = a_p^s \tag{17}$$

3.5.2.6 Consensus

This operator is used to combine two or more independent opinion about the same proposition into a single opinion. The "⊕" is used to designate the negation operation.

Let there is two opinion $\omega_p^A = (b_p^A, d_p^A, u_p^A, a_p^A)$ and $\omega_p^B = (b_p^B, d_p^B, u_p^B, a_p^B)$ about the same proposition *p*. Then the opinion is represented as $\omega_p^{A,B} = \omega_p^A \oplus \omega_p^B = (b_p^{A,B}, d_p^{A,B}, u_p^{A,B}, a_p^{A,B})$ such that

$$b_p^{A,B} = {(b_p^A\, u_p^B + b_p^B u_p^A)}/{\kappa} \tag{18}$$

$$d_p^{A,B} = {(d_p^A\, u_p^B + d_p^B u_p^A)}/{\kappa} \tag{19}$$

$$u_p^{A,B} = {(u_p^A\, u_p^B)}/{\kappa} \tag{20}$$

$$a_p^{A,B} = a_p^A \tag{21}$$

where,

$$\kappa = u_p^A + u_p^B - u_p^A\, u_p^B \tag{22}$$

3.5.2.7 Weighted Consensus

The default consensus operator, suggested by Jøsang, is not appropriate for the case of weighted opinions, as it treats opinions equally. This makes it challenging to deal with weighted opinions. Zhou et al. [87] have proposed a cumulative weighted fusion operator that is capable of dealing with fusing opinions according to their weights in a reasonable way.

Let there is two opinion $\omega_p^A = (b_p^A, d_p^A, u_p^A, a_p^A)$ and $\omega_p^B = (b_p^B, d_p^B, u_p^B, a_p^B)$ about the same proposition *p* with weights α and β respectively. Then the weighted opinion is represented as $\omega_p^{A,B} = \eta\omega_p^{\oplus(A,B)} = (b_p^{A,B}, d_p^{A,B}, u_p^{A,B}, a_p^{A,B})$ such that

$$b_p^{A,B} = \frac{(\kappa - u_p^A u_p^B)(\alpha\, b_p^A u_p^B + \beta\, b_p^B u_p^A)}{\kappa(\alpha u_p^B + \beta u_p^A - (\alpha+\beta) u_p^A u_p^B)} \quad (23)$$

$$d_p^{A,B} = \frac{(\kappa - u_p^A u_p^B)(\alpha\, d_p^A u_p^B + \beta\, d_p^B u_p^A)}{\kappa(\alpha u_p^B + \beta u_p^A - (\alpha+\beta) u_p^A u_p^B)} \quad (24)$$

$$u_p^{A,B} = {(u_p^A u_p^B)}/{\kappa} \quad (25)$$

$$a_p^{A,B} = {(\alpha\, a_p^A u_p^B + \beta\, a_p^B u_p^A - (\alpha\, a_p^A + \beta\, a_p^B) u_p^A u_p^B)}/{\alpha u_p^B + \beta\, u_p^A - (\alpha + \beta) u_p^A u_p^B} \quad (26)$$

where,

$$\kappa = u_p^A + u_p^B - u_p^A\, u_p^B \quad (27)$$

$$\eta = \frac{\alpha(u_p^B - u_p^A u_p^B) + \beta(u_p^A - u_p^A u_p^B)}{(u_p^A + u_p^B - 2u_p^A u_p^B)} \quad (28)$$

The values of uncertainty have to lies between 0 to 1, can't be exactly 0 or 1, because denominator then will be 0.

3.5.3 Trust Value Calculation

A trust score obtained from an opinion, is measured as the expected value (*E*) that indicates the probability that our proposition is true. The value of E can be calculated as follows:

$$E = b + a * u \quad (29)$$

3.6 Rank Correlation

To determine the rank correlation, we have employed Kendal tau rank distance [88]. This distance function counts the number of disagreements between two ranking lists. The distance value range is from 0 to 1; a larger distance value represents more dissimilar the two rank lists. The distance value zero indicates that the two rank lists are identical. The formula that uses to determine the Kendal tau distance is given below:

$$K(R_1, R_2) = \frac{total\ number\ of\ mismatched\ pairs}{n\,(n-1)/2} \quad (30)$$

Here, '*K*' is the distance function determined by the distance between 'R_1' and 'R_2' rank lists, and '*n*' is the list's size. For example, R_1 and R_2 contains the rank order for four items and for each pair we need to compare them, the pair comparison is given below:

Table 3.3. Example Scenario of pair comparison

	A_1	A_2	A_3	A_4
R_1	1	4	3	2
R_2	2	3	4	1

Pair	**R1**	**R2**	**Pair Count**
(A_1, A_2)	1<4	2<3	
(A_1, A_3)	1<3	2<4	
(A_1, A_4)	1<2	2>1	Mismatch
(A_2, A_3)	4>3	3<4	Mismatch
(A_2, A_4)	4>2	3>1	
(A_3, A_4)	3>2	4>1	

For the above pair comparison, we have encountered two mismatches. Therefore, the Kendal tau distance will be, $\frac{2}{4\,(4-1)/2}$ = 0.33; this indicates 33% dissimilarity among these two rank lists, R_1 and R_2.

CHAPTER 4. SYSTEM DESIGN AND EVALUATION

This chapter discusses both SERS and its enhanced version of the SERS scheme, E-SERS. The section 4.1 is largely based on our published paper [23].

4.1 SERS - Security-related and Evidence-based Ranking Scheme

SERS, as indicated earlier, focuses on the privacy and security-related internal aspect of an App and its combination with the external aspect computed from the user reviews by identifying security and privacy-related comments. Such a focus on security- and privacy-related attributes are necessary, as for a less perceptive user an incorrect setting of permissions may lead to potential risks associated with the unintended exposure of their critical data. In 2014, a survey conducted by IDG News [89] indicated that 54% of U.S. mobile App users decided not to install an App when they knew how much personal data it can collect. In addition, 30% of users uninstalled an App after knowing that the App was collecting their personal data. These statistics indicate that many users are still not aware of the risks associated with either accidental or malicious leakage of their data. Therefore, developing a comprehensive ranking scheme that considers the security and privacy concerns of the users is critical and informative to any App user. We evaluate the SERS approach on publicly available Apps from the Google PlayStore and compare our ranking with prevalent ranking techniques such as the average star ratings. The experimental results indicate the effectiveness of our proposed approach.

The section illustrates the SERS proposed approach along with the experimental results. Also, address the limitations of the SERS approach.

4.1.1 Approach and Implementation

To assess the trust level of an App in SERS, we compute two metrics, IDM (Internal Data-leak Metric) and ERM (External Review Metric), which are the rating scores based on internal and external evidences.

1) I_{DM} is computed by performing the static taint analysis of the required permissions,
2) E_{RM} is computed by performing the sentiment analysis of collected reviews.

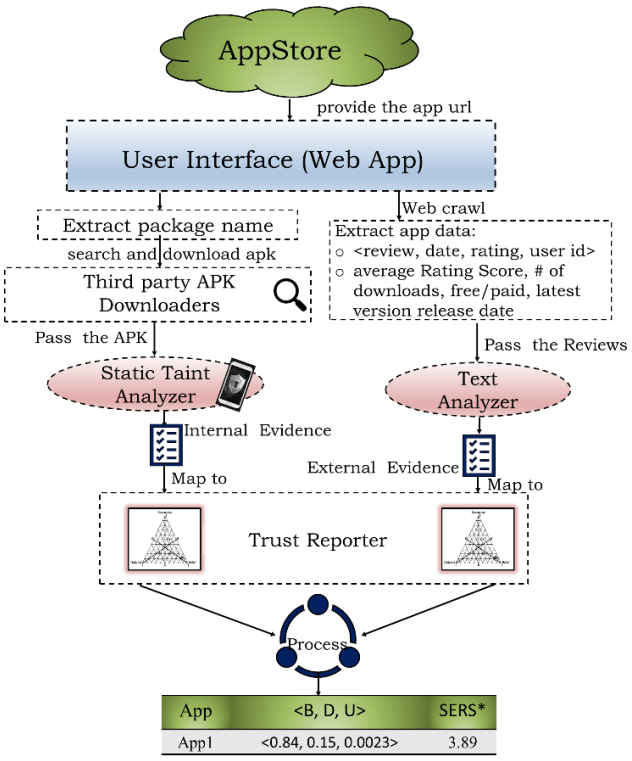

Figure 4.1. SERS Approach [23].

Each metric taken in isolation is also helpful in the selection process and so is their combination — the higher the metric the higher is the trustworthiness of an App. Figure 4.1 shows various steps involved in the SERS approach. We have developed a Web interface for the user to provide an App's URL located in Google PlayStore. Then, in an automated way, using the package name, the APK file is downloaded from the third-party APK downloaders. The APK file is passed to FlowDroid to identify any unauthorized leaks of confidential data, which is considered internal evidences. Similarly, for external evidence, using automated Web crawler extracts the most recent reviews and other details of Apps from the Google PlayStore. Then pass the reviews to a text analyzer to identify the reviews' sentiment, which is considered external evidences. The Trust Reporter combines internal and external evidence and returns a single trust score for an App using subjective logic principles. We picked the Google PlayStore as the target AppStore and identified the Top — 3 categories in the Google PlayStore — which are Communication — messenger, Entertainment —TV, and Photography — photo editor. From these categories, we selected 35 Apps and stored their details such as the APK file, recent reviews, and the corresponding user ratings in the database.

4.1.1.1 Computation of I_{DM} based on Internal Evidences

As indicated above, once we have downloaded the APK file of an App, we conduct a four-step analysis on it to compute the corresponding I_{DM}:

1) Identify threat sources and sinks pair - extract the data leakage information and the corresponding APIs using a taint analysis tool, FlowDroid.
2) Identify potential risk of permissions - map the Source and Sink APIs to permission identifiers to determine the severity of the data leak.
3) Determine risk - estimate the risk level based on the potential impact and likelihood.
4) Compute I_{DM} - apply the Subjective Logic principles to compute the trust tuple for the App and convert it to I_{DM}.

These steps are described below.

1) Identify threat sources & sinks pair. FlowDroid [80] is a static taint analysis tool for Android Apps. It tracks sensitive information associated with an App by starting at a predefined source and following the data flow until it reaches a given sink. It then returns precise information about which data may be leaked and where. Android sources and sinks are the two main points characterized in data flow to define the direction of flow.

An Android Source is an external resource from which data is read - for example, the *getDeviceId()* API method is a source which returns a IMEI (International Mobile Equipment Identity) into the application code.

An Android Sink is an external resource to which data is written for example, the *sendTextMessage()* API method is a sink as both the message and the phone number it receives are non-constant.

SuSi [78], is a supervised machine-learning tool that is able to categorize sources and sinks in the form of Android API method signatures. For instance, there is a category "*CONTACT_INFORMATION'"* grouping all sources related to the user's whereabouts (e.g., *getContactList()*). In SuSi, the Android source APIs are categorized into 14 different categories: *ACCOUNT, BLUETOOTH, LOCATION, BROWSER, CALENDAR, CONTACT, DATABASE, FILE, NFC, NETWORK, SETTINGS, UNIQUE_IDENTIFIER, SYNC, and NO_CATEGORY*. Similarly, SuSi categorizes the Android sink APIs into 16 different categories: *ACCOUNT, AUDIO, BROWSER, CALENDAR, CONTACT, FILE, LOG, NETWORK, NFC, PHONE_CONNECTION, VOIP, PHONE_STATE, SMS_MMS, SYNC, SYSTEM, and*

NO_CATEGORY. Here, the *NO_CATEGORY* refers to sources and sinks classified as non-sensitive in SuSi. We feed all the source and sink APIs (obtained from FlowDroid) to SuSi so as to identify and categorize them into one of these categories.

For an App, the SERS approach uses FlowDroid to extract all the data flows from sensitive sources to sensitive sinks, which may lead to data leakages. The result of this data flow analysis is a set of API method pairs that indicate the usage of sensitive data within the App. The result is in the following form: *Source (S)* → *Sink (SN);* it indicates the sensitive data flow. For each data leak, FlowDroid returns the API method's name that tries to read/write sensitive data from the application to third parties. From the analysis report, provided by FlowDroid, we extract all faulty sources and sinks APIs and pass them to the next phase to calculate the severity (impact) of the flaws in the application.

2) Identify potential risk of permissions. In Android, all sensitive data can be accessed through the specific APIs by receiving permissions from the user. The purpose of permission is to protect the privacy of sensitive data (such as contact information). Based on the data confidentiality, the system might grant the permissions automatically or prompt the user to approve. Android has divided these permissions into several protection levels that affect whether runtime permission requests are required or not. Potential risks using of the permissions are characterized into:

- **Normal permissions**: these are lower risks that do not request the user's explicit approval. This is the default value.
- **Signature permissions**: these are granted without user's approval only if the application is signed with the device manufacturer's certificate.
- **Dangerous permissions**: these give applications an access to private user data or control over the device that alert the user to potentially insecure or especially expensive operations. Hence, user's confirmation is required before proceeding.

From the Android site [90], we have collected 91 permission identifiers (Dangerous: 26, Signature: 29, Normal: 36) and stored them into a MySQL database.

In SERS, we have used PScout [91], to conduct the mapping from API calls to permissions. PScout applies static code analysis on the Android source code and extracts the function to permission mappings. After this step, we have all the required information, a single method may request for multiple permission accesses.

3) Determine risk. In this phase, to assess the quantitative risk of Android permissions, we follow the NIST guideline [92], [93]. According to these guidelines, risk assessment is defined as:

$$R(P) = L(P) \times I(P) \quad (31)$$

where *P* is the requested permission; *R(P)* is the risk of *P*; *L(P)* and *I(P)* are the likelihood and the impact of *P*, respectively. For estimating impact levels of permissions, we consider three classes of permissions, Dangerous, Signature and Normal. Although it is problematic to estimate the exact level of harm caused by permissions, if it is requested by a malware, it is certain that Dangerous permissions are more detrimental than two other permissions. Thus, we assign the following values based on the NIST guidelines:

- The probability assigned to the likelihood of each threat is — 1.0 for Dangerous, 0.5 for Signature, and 0.1 for Normal.
- The value assigned to their impact is — 100 for Dangerous, 50 for Signature, and 10 for Normal.

By applying the Formula (31), the level of risk assessment scale is divided into following three different categories (Table 4.1):

- High (>50 to 100): it means that the vulnerability is exposed and exploitable, and its exploitation is expected to have severe impact.
- Moderate (>10 to 50): it means that based on the exposure of the vulnerability and the ease of exploitation, the severity of impact could result from its exploitation.
- Low (1 to 10): it means that the vulnerability is of minor concern and expected to have non-significant or negligible impact.

Table 4.1. Quantitative 3 × 3 Risk Assessment Matrix

Likelihood (Source/Sink)	**Level of Impact (Source/Sink potential risk of permission)**		
	Normal (10)	Signature (50)	Dangerous (100)
Dangerous (1.0)	Low (10)	Moderate (50)	High (100)
Signature (0.5)	Low (5)	Moderate (25)	Moderate (50)
Normal (0.2)	Low (1)	Low (5)	Low (10)

Again, if a single method requests for multiple permissions then risk level can be defined as:

$$\sum_i R(Pi) = L(P_i) \times I(P_i) \quad (32)$$

where $i = 1, 2, ..., n$ and n is the total number of requested permissions.

4) Compute I_{DM}. In SERS, any evidences that confirms the data confidentiality is considered as a positive evidences and one that suggests a violation is a negative evidence. Two different cases are possible in terms of the presence of data leaks reported by FlowDroid.

Case I: No data leaks found. Along with the analysis report generated by FlowDroid, we also keep track of the runtime log file. From that log file, we extract the number of total Sources (ST) that exists in an App's code. If there are no leaks, then ST is considered as a positive evidence.

Case II: Data leaks found. The positive evidences are calculated by subtracting the number of faulty Sources (SF) from ST; where SF indicate those sources, which are involved in information leakage. The negative evidences are computed using the formulae (31) and (32). Furthermore, if both Source and Sink are categorized into *NO_CATEGORY*, then we classify the flow as a neutral evidences since this category is considered as non-sensitive by SuSi.

As indicated earlier, here, we use the definition of trust of an App, as "the ability of an App to not disclose any confidential data"'. Trust of an App is quantified as a tuple of *b, d, u*) using the principles of Subjective Logic introduced by Jøsang [84] (details given ins Sec. 3.5); where *b* indicates belief, *d* indicates disbelief and *u* represents the uncertainty about an App's behavior related to the sensitive data. The *b, d, u* values are computed using formula (1) to (3).

Once we have such (*b, d, u*) tuples for similar Apps, these Apps can be rank ordered using the formula of the Ordering operator (see Sec. 3.5.2.4). This formula (33) uses the notion of probability expectancy, which depicts the value of I_{DM} of each App. We normalize our ratings to the scale of 5 to conform to the ratings used by the Google PlayStore — such a normalization allows the comparison of our rankings with the PlayStore's rankings.

$$I_{DM} = \frac{b+u}{b+d+(2 \times u)} \times 5 \quad (33)$$

4.1.1.2 Computation of E_{RM} based on External Evidences

1) Data Collection and Pre-processing. As indicated, we have collected a dataset of 35 Apps from three different categories. We picked Apps in every category that offer similar functionality and have a decent number of user reviews. The user reviews were scraped from the

Google PlayStore using a self-developed tool. Each review contains the creation time, the reviewer id, a text description of the review content, and the corresponding rating. The dataset we prepared for our evaluation is presented in Table 4.2.

Table 4.2. Statistics of collected users review dataset.

Total number of mobile Apps	*35*
Total number of crawled reviews	*112, 500*
Average number of reviews per App	*2500*
Average words per review	*11.88*
Collection time	*April 27 to July 19, 2019*

After collecting the dataset, we preprocess the reviews, as most reviews are in the form of unstructured text. These preprocessed reviews are then acted upon by the sentiment analysis tool (called TextBlob [81]) to predict the sentiment of the user while creating that review.

2) Calculation of the Sentiment value of Reviews. We provide each preprocessed review as input to the sentiment function of TextBlob. The function returns two properties: polarity and subjectivity. Polarity is in the range of [-1, +1] and indicates the sentiment (positive/negative) of text. Subjectivity lies in the range of [0, 1] and indicates whether a given text is Subjective or Objective.

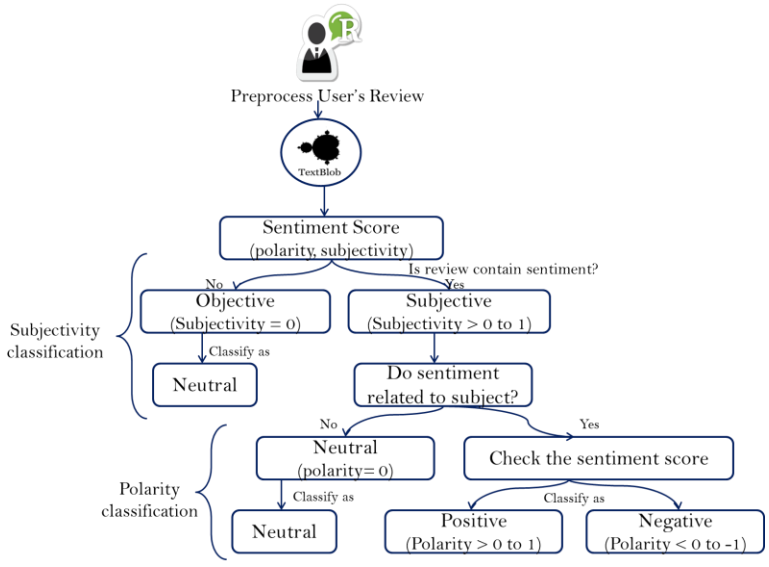

Figure 4.2. Mapping Sentiment score to evidence.

3) Mapping Sentiment values to evidences and Compute E_{RM}. To map the sentiment values to evidence, we have applied a similar algorithm described by Gallege [37]. However, that technique does not consider the boundary case [0, 0], which amounts to approximately 10% of total reviews. In our mapping approach (shown in Figure 4.2), we enhance Gallege's algorithm by considering the boundary case as a neutral evidences. After the mapping is completed for each App, we generate the total number of positive, negative, and neutral evidence. By providing these values to Formulae (1), (2), and (3), we can compute the trust tuple for each App. From the trust tuple, using the formula (33), similarly we compute the value for E_{RM} for each App.

4.1.1.3 Quantify SERS Ranking Scheme

Once we have the trust tuples for the external and internal aspects of an App, we combine them using the consensus cumulative weighted fusion operator [87] into a single tuple. As we are concerned about privacy and security-related evidences, more than the users' reviews, we put a higher weight on the internal evidences than the external evidences; the assigned weights are 70% and 30% respectively (these are parameters which could be adjusted as needed). This resultant tuple considers all available evidences and thus, provides a better quantification of trust associated with each App than the basic average star ratings provided by the Google PlayStore. The combined tuples allow us to rank-order similar Apps using the ordering operator mentioned in Formula (33). Finally, the SERS ranking generated by our algorithm is compared, using the Kendall Tau Distance method [88] that considers the number of pair-wise variances between two ranking lists (such as Average rating, E_{RM}, and I_{DM}) — similar to the approach presented in [24]. Distances of 0% and 100% represent the same and opposite rankings respectively.

4.1.2 Framework Evaluation

To conduct the experiments, we set the FlowDroid parameters to *FLOWSENSITIVE*, *CONTEXTSENSITIVE* alias, and the maximum access path length of 10. These settings sacrifice the amount of memory and speed for precision. As a result, the list of data leaks identified by FlowDroid may have less false positives as well as less false negatives. The machine that is used to perform the sensitive data flow analysis has 64 GB dedicated RAM for this task. Additionally, we set the time-out for analyzing one single Android App to 8 hours.

During the data flow extraction phase, we obtained 7782 sensitive data flows in total. Some source and sink API methods from sensitive categories appear, in these flows where a source tries to access sensitive unique identifiers, including *DeviceID* and *SubscriberID*. In [49], authors identify that sources that are categorized into *NETWORK_INFORMATION* and *UNIQUE_IDENTIFIER* are more inclined to occur in malware Apps than in benign Apps. Similarly, that study indicates that malware Apps are more prone of to use short message service (SMS) as sinks to leak data to third parties — a similar scenario is found in our experiments. Also, we observed that, in our dataset, the number of Source APIs is approximately twice than the number of Sink APIs (for each App).

4.1.2.1 Comparing different Ranking Schemes

Four different kinds of ranking schemes can be achieved using the outcome of our experiments. These are:

1) Ranking based on I_{DM} computed by the static taint analysis (internal aspect).
2) Ranking based on E_{RM} computed by the sentiment analysis (external aspect).
3) Ranking based on (I_{DM}, E_{RM}) computed by combing the internal and external aspects (SERS). In SERS, the weight of external and internal evidences can be adjusted based on user preferences.
4) Ranking based on the Average Star Ratings - default in the Google PlayStore.

Below, we describe different scenarios for comparing the above-mentioned ranking schemes - similar to our approach advocated in [24]. Table 4.3 represents the rating scores of Apps in the communication (messenger) category based on these four schemes. These Apps are selected based on the different range of popularity (such as most popular, popular, and less popular) in terms of the number of installs (given in column 1). Three insights we can establish from the data given in Table 4.3:

- Firstly, if we consider only the traditional star ratings of all these apps, as a general app user would do, we find that there is hardly any difference (of 0.6 where 4.6 is the highest rating and 4 being the lowest rating) between these fifteen Apps. Whereas, the number of installs for each App varies a lot. This highlights the fact that traditional star rating does not accurately reflect the trust of an app.

- Secondly, a less popular app such as *App10* which has only 1Million+ downloads witnesses as a more secure app than the other popular Apps. So, the proposed ranking scheme will help the user to go with the more secure app instead of just follow the traditional rating.
- Lastly, it shows that rank-orders vary from one scheme to another. Therefore, we did an empirical investigation to find the reasons behind this behavior. Table IV shows the computed Kendall Tau distances for four such comparisons of Apps from three different categories. Therefore, in Table 4.3 the Tau distance is expressed as a range.

Table 4.3. Apps Rating based on Different Ranking Schemes [23].

App ID	Traditional Star Rating (out of 5)	Average Rating of collected reviews (out of 5)	E_{RM} Rating (out of 5)	I_{DM} Rating (out of 5)	SERS Rating (out of 5) E_{RM} -30%, I_{DM} -70%	SERS Rating (out of 5) E_{RM} -50%, I_{DM} -50%	SERS Rating (out of 5) E_{RM} -70%, I_{DM} -30%
App1 (1Billion+)	4.1	3.08	4.09	4.22	3.89	3.66	3.43
App2 (1Billion+)	4.6	3.82	4.44	3.98	3.98	3.95	3.92
App3 (10Million+)	4.5	3.56	4.21	4.08	3.93	3.83	3.72
App4 (50Million+)	4.5	3.25	4.23	4.46	4.10	3.86	3.62
App5 (10Million+)	4.5	3.99	4.37	4.92	4.65	4.46	4.28
App6 (100Million+)	4.3	2.35	2.81	4.51	3.86	3.43	3.00
App7 (1Billion+)	4.1	3.52	4.22	4.37	4.12	3.95	3.79
App8 (100Million+)	4.4	3.28	4.32	4.53	4.53	4.52	4.51
App9 (1Billion+)	4	2.5	3.79	3.24	2.53	2.51	2.51
App10 (1Million+)	4.1	2.78	3.61	4.99	4.63	4.29	3.85
App11 (10Million+)	4.6	3.62	4.27	4.32	4.27	4.21	4.10
App12 (50Million+)	4.6	3.33	3.91	4.04	3.52	3.43	3.38
App13 (500Million+)	4.2	2.42	3.51	3.92	3.58	3.31	3.00
App14 (100Million+)	4.2	2.21	2.79	4.98	4.41	3.92	3.34
App15 (5Million+)	4.1	2.77	3.32	1.68	2.40	2.58	2.68

1) Average Ratings vs E_{RM}. Conceptually, the reviews should be consistent with the star ratings, as emphasized by the Kendall Tau variance (Table 4.4) which is between 16% to 22% when we compare rankings obtained by E_{RM} and average star ratings - demonstrating that these

two rankings are fairly similar to each other. After investigating the review sentiments and the corresponding ratings, we found some mismatches. For example:

Table 4.4. Distance Between Different Ranking Schemes.

App categories	Average rating & E_{RM} (Distance %)	Average rating & I_{DM} (Distance %)	I_{DM} & E_{RM} (Distance %)	SERS & Average rating (Distance %)
Communication (messenger)	16	42	49	19~39
Entertainment (online TV)	9	49	58	15~41
Photography (photo editor)	22	27	40	22~36

"*I enjoyed it very well its my first time to used it*" - the user provided a text with positive sentiment; however, for this review, the user provided a rating of 1, reflecting a mismatch.

"*Chat heads suck on marshmallow, when i permit the draw over other Apps for the chat heads it works one time and then the next time i open a chat head it asks me to permit that option, AGAIN! can u please fix your app?*" - the user provided a rating of 5; where TextBlob returned a negative sentiment for this review, again reflecting a mismatch.

Such mismatches indicate that the star ratings in many cases are not a true reflection of the associated reviews' text.

2) Average Ratings vs I_{DM}. Table 4.4 indicates that the Kendall Tau distance between I_{DM}-based and average rankings is between 27% to 49%. We found two Apps (*App2* and *App6*) that have opposite orderings - e.g., *App2* has a rank of 2 out of 15 based on the user ratings; while its rank is 12 based on the I_{DM} score. The opposite is true for the *App6* - a rank of 14 based on user ratings but a rank of 5 based on the I_{DM} score. These cases are grouped into two categories - *Good to Bad* and *Bad to Good*.

Good to Bad. The users rated the *App2* as having a good rank, but the I_{DM}-based score ranked it very low. A few supportive (first two) and a few critical (last two) user comments are given below. In selecting the reviews, we picked only those reviews which contained security-related terms such as privacy, security, spy, spam, malicious, and leaks. The primary focus of the paper is

to rank apps based on privacy and security-related features and hence, all other reviews, for each app, which do not focus on such features/terms are ignored in our analyses.

"Simply the best one, the easiest one to use, the best for security!"

"Best messaging app. Privacy."

"no privacy Auto background allows your camera to take pictures as well as your conversations without any regards to your personal privacy this application is no more than the FBI snooper"

"EXCESSIVE PERMISSIONS!! DOES NOT REQUIRE DEVICE ID!! DEV DOES NOT RESPECT YOUR PRIVACY!!!STOP VIOLATING MY PRIVACY!"

The above supportive reviews are not that informative whereas the critical reviews give more details about users' concerns. From the reviews of *App2*, we have collected a total of 22 reviews that match with one of the keywords mentioned above. In these 22 reviews, we found 7 reviews to be critical and 15 reviews to be supportive. Such a disproportion reflects why the App is having a high user rating. If the user solely focuses on functional aspects of Apps instead of aesthetical aspects (e.g., setting or customize background) then I_{DM}-based ranking is more acceptable than the user ratings-based ranking. During the security flaws inspection, we notified that the data leak associated with *App2* deals with the Dangerous permission access (such as READ_PHONE_STATE).

Bad to Good. The users rated the *App6* as bad, but the I_{DM}-based score ranked it good. Two supportive comments and two critical comments that contain the privacy and security-related words are given below:

"This is a fantastic messenger texting app when you haven't been tempo banned without prior notice/warning or opportunity to respond! While I love the fact you only share user names & not personal phone numbers & it's great for personal privacy & good for sharing pics! It's not very fair in how it'll cut off your communication with no justification. Only use this product if you don't have to rely upon it."

"love it, complete privacy."

"Need encryption security"

"its not opening bad app msg not sending while opening the app error occur and chance of your privacy risk and mobile"

Similarly, for *App6*, we have collected total of 19 reviews that match with one of the privacy and security-related keywords. In these 19 reviews, we found 11 reviews were critical and 8 reviews are supportive – leading to a lower ranking. During the computation of I_{DM}, we noticed that the data leaks associated with *App6* deal with only the Normal permission access. Thus, *App6* should have less impact associated with such flaws than flaws found in *App2*.

3) I_{DM} vs E_{RM}. Table 4.4 indicates that the distance between I_{DM} and E_{RM} - based rankings is between 40% to 58%, which is quite similar to the previous scenario. The issues that were discussed in the previous two sections also hold for this section.

4) SERS vs Average Ratings. Previous three scenarios have illustrated the fact that rankings based on partial evidences result in significantly different orderings. Thus, there is a need to combine internal and external evidences to provide a holistic view. To achieve this, we combine the internal view tuple and the external view tuple using a weighted consensus operator by giving a higher weight to the internal evidences (mentioned in Sec. 4.1.1.3). However, in our experiments, we present the different variations of weights that reflect the distance variation with the average rating. The experimented weight combinations are: 30% weight to E_{RM} and 70% weight to I_{DM}; equal weight to both E_{RM} and I_{DM}; 70% weight to E_{RM} and 30% weight to I_{DM}.

As seen from Table 4.4, the distance between the average star rankings and the SERS rankings varies from 15% to 41% for different categories - and it is lesser than other distances. Higher weight towards E_{RM} reduce the distance with the average rating, similar is true for the opposite case. Also, the SERS ranking lies between the two extremes (i.e., average and E_{RM} versus average and I_{DM}) represented by columns 2 and 3 in Table 4.4.

User ratings, as described earlier, are not always a true reflection of reviews. Reviews such as "*it's really good*" or "*very good experience*" with a rating of one star highlight the mismatch and do not reflect the correct sentiment. Similarly, reviews such as "*wp....does not supported properly*" with a user rating of five stars, again, do not present the right emotions of the user. Such mismatches lead to improper average ratings. As a result, the average ratings cannot be considered as a proper ranking scheme.

Also, reviews are mostly unstructured text. In those cases, where a review such as "*nyc app*" with rating 5 indicates a positive sentiment, while TextBlob considers it as neutral evidence. TextBlob also fails sometimes due to the lack of AppStore specific domain knowledge. Thus, it is neither fair nor sufficient to only use the star ratings or its combination with the external narratives.

Conversely, when we deal with internal evidences there is no possibility of such ambiguity as it entirely focuses on the functional aspects of an App. Although there is a possibility to return many false positives in such analysis. Therefore, when we combine both the internal and external evidences and generate a rank list, it not only encompasses all perspectives but also helps in overcoming such idiosyncrasies associated with reviews.

Table 4.5. Categorize the Reviews that focus on Security Issues.

App ID	# of Reviews (address security issue)	# of Positive Reviews	# of Negative Reviews	# of Neutral Reviews
App5	3	2	1	0
App7	11	7	2	2
App3	14	6	5	3

We selected and investigated two Apps that have an identical ranking based on SERS and average ratings – *App5* and *App7* have a rank of 2 and 5 respectively (out of 15 Apps) based on both the average ratings and the SERS score. We extracted only those comments that address the security-related issues. We read those reviews and categorized them as positive, negative, and neutral. We also picked one App that has a distinguishable ranking based on SERS and average ratings - *App3* has a rank of 4 based on user rating; while its rank is 9 based on SERS. We have applied a similar approach (as mentioned above) - to extract security related reviews and labeled them in positive, negative and neutral categories (see Table 4.5). However, a small number of privacy and security-related reviews is an indication that typical users are not aware of these internal factors. Such a discrepancy highlights the fact that users' views of security and privacy are significantly different from the traditional definitions of security and privacy found in research literature. This difference in perception, of the users, makes the holistic view of trust more important than just considering a subset of evidence while comparing similar Apps. A few reviews which do mention some privacy and security-related features are indicated below - however, these reviews are identified as indicating neutral sentiments.

"*Please update profile photo access to the only people we want our dp to be shown ..like as status privacy...*"

"*plzz try to add some new security like to hide the chats archieved chats doesn't help much soo plzzz*"

"*actually it will be more nice if you follow security policies provided by WhatsApp messenger. excluding contacts, posting statuses. etc*"

SERS achieves a holistic rank ordering of similar Apps and generates insights, using both structured and unstructured contexts, associated with Apps. The empirical investigations show that the SERS reflect the complete nature of an application than the other existing alternatives.

4.1.3 Limitations

Although, SERS considers a holistic view of an App, it does have some limitations. The limitations are:

- The framework only supports a single source for the internal evidences and the same for the external evidences.
- The primary limitation of any static analysis tool is it may return false-positive warnings.
- In the SERS, to determine risks associated with user-given permissions, we have used a 3×3 risk assessment matrix, where the likelihood and impact factors of the risks were assigned based on judgment.

Therefore, to address these limitations, we have proposed an enhanced version of the SERS scheme, named E-SERS. The E-SERS have the following additional features compared to the SERS scheme.

- The E-SERS provides a formalism to the SERS so that it can support any number of sources for generating the necessary evidence for a given App.
- To overcome the false-positive warnings related concern, in E-SERS, we have considered the tools' reputation score. This reputation is based on the performance of the tool addressed. This helps to reduce the impact of false-positive issues or any others.
- After creating the SERS, we experimented on 2555 malicious Apps to formalize risks and their impacts. We have used a 4×4 risk assessment matrix in the E-SERS based on that additional experimentation's quantitative outcomes.
- In addition, the E-SERS scheme also incorporates temporal and reputational aspects associated with user reviews. The E-SERS assigns a temporal weight to each review of an App. These weights are chosen such that recent reviews are given a higher weight than the

distant reviews. We also include, in the E-SERS, a reputation factor for each review based on its helpfulness as indicated by other users.

From here onwards, we will discuss the E-SERS. The following section introduces the E-SERS scheme in detail.

4.2 E-SERS – Enhanced Security-related and Evidence-based Ranking Scheme

In this section, Sec. 4.2.1 contains the E-SERS architecture and its essential elements. Then, we present the algorithms in the form of pseudo-code. Besides, we discussed the computation of direct trust and indirect trust artifacts.

4.2.1 Architecture

The conceptual architecture of the E-SERS is illustrated in Figure 4.3 (discussed in this section) and the details of the E-SERS system flow is presented in Figure 4.4 (See Sec. 4.2.3). The four basic components of E-SERS, and the notations that we use throughout the dissertation, are as follows:

1) App's artifacts (AA). The AA are categorized into "Internal or Direct Trust Artifacts (DTA)" and "External or Indirect Trust Artifacts (ITA)". DTA indicate the internal evidences that are observed first-hand. These evidences can be gathered from the APK files, source code, and jar files of an App. On the other hand, opinions, such as ratings and reviews, that are based on users' experiences contribute towards the indirect trust of the App. The ratings and reviews, along with review dates, are the ITA.

2) Evidence Sources. The evidence source set $S = S_1, S_2, ..., S_N$ is divided into two mutually exclusive subsets, S_{DT} and S_{IT}, which denote the list of sources that are related to the DTA and the ITA respectively. For an App X, each evidence source (S_i) generates a set of evidences, indicated as $EV_X^{S_i} = \{ev_1, ev_2, ..., ev_n\}$. Each evidence, ev_i, can be positive, negative, or neutral. Different tools are used for extracting various types of evidences. Static analysis tools can process DTA to extract evidences such as bugs details, data leakage information, security vulnerabilities, and good practices used. NLP tools can generate external evidences such as review sentiments, review reputation, etc., from ITA. All the evidences generated by these tools are passed on to the Evidence Processors for extracting opinions from these evidences.

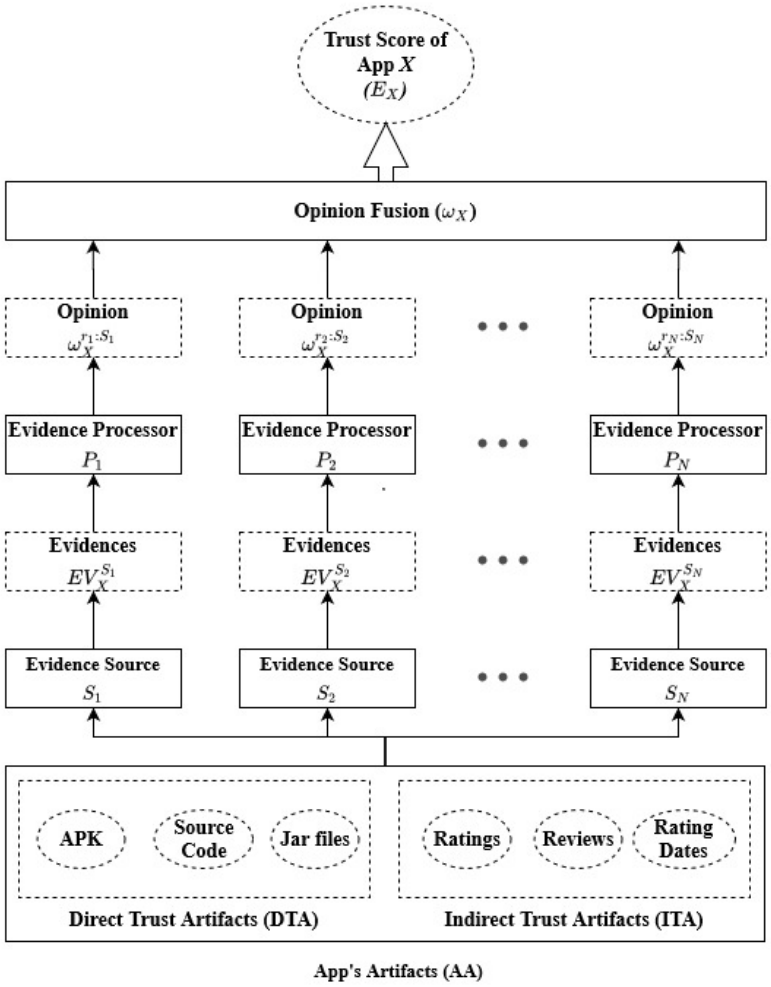

Figure 4.3. E-SERS Architecture

3) Evidence Processor. Each S_i has an associated Evidence Processor P_i. P_i maps the set of evidences, $EV_X^{S_i}$, to an opinion $\omega_X^{S_i}$. Here, $\omega_X^{S_i}$ represents the opinion about the App X based on the evidence source S_i. However, each source may produce a different number of evidence. Therefore, before fusing the different source opinions, we need to normalize them in such a way that evidences from a trusted source get more influence than others. To do so, we have introduced the source reputation into E-SERS. The reputation of each source, $\omega_{S_i}^{r_i}$, is combined with the opinion of $\omega_X^{S_i}$ to compute the weighted opinion $\omega_X^{r_i:S_i} = \omega_{S_i}^{r_i} \otimes \omega_X^{S_i}$. Similar to the technique suggested in [84], we use the discounting (or weighted) operator ($\otimes$) to represent the degree of trust about an evidence source. We have assigned a reputation score to each of the evidence sources based on their performance on appropriate benchmarks.

4) Opinion Fusion. Opinions from different sources, $\omega_X^{r_1:S_1}$, $\omega_X^{r_2:S_2}$, ... $\omega_X^{r_N:S_N}$, are combined into a single opinion (ω_X), using different subjective logic operators, to create a combined opinion. We use the consensus operator ($\oplus$) for combining independent opinions about the same App, *X*, into a single opinion ω_X. The operator returns the cumulative fusion of opinions [85] assuming that opinions are independent. However, the default consensus operator suggested by Jøsang, is not appropriate for the case of weighted opinions, as it treats opinions equally. This makes it challenging to deal with weighted opinions. Zhou et al. [87] have proposed a cumulative weighted fusion operator that is capable of dealing with fusing opinions according to their weights in a reasonable way. We use the cumulative weighted fusion operator to combine based on weights provided by the user. This combined opinion is used to generate a trust score for the App.

4.2.2 Trust Algorithm

For an App, the trust score is evaluated based on the evidences collected from direct trust and indirect trust sources. For both views, different kinds of mechanisms are applied to identify a certain type of defect or vulnerability. On top of these evidences, other factors are applied to assess the quality of that evidence. Such factors include a weight based on the temporal aspects of reviews, bug confidence, and bug or flaw severity. Additionally, we assess the reputation of tools to evaluate the trust of an App. The proposed evidence-based trust algorithm in E-SERS is shown below (Algorithm 1). The algorithms to calculate the direct and indirect trust scores are presented below as well (Algorithm 2 and Algorithm 3).

Algorithm 1 Evidence-based Trust Score for an App X

procedure calculateTrustScore (DTA_X, ITA_X, S_{DT}, S_{IT}, α, β)

$\omega_X^{\oplus S_{DT}} \leftarrow$ *create_internal_opinion*(DTA_X, S_{DT})

$\omega_X^{\oplus S_{IT}} \leftarrow$ *create_external_opinion*(ITA_X, S_{IT})

#apply Weighted fusion operator to combine the opinions - formula (23) to (28)

$\omega_X^{\oplus (S_{DT}, S_{IT})} \leftarrow$ weighted_fusion($\omega_X^{\oplus S_{DT}}$, $\omega_X^{\oplus S_{IT}}$, α, β)

#apply formula (29)

$E_X \leftarrow E(\omega_X^{\oplus (S_{DT}, S_{IT})})$

return normalized $||E_X||^2$ to scale 5 in accordance with the user rating

In Algorithm 1, we pass the input artifacts and sources along with the user desired weights for both views of an App, *X*. Then, it calls *create_internal_opinion* method to generate the DTA based opinion, $\omega_X^{\oplus S_{DT}}$. Similarly, we invoke *create_external_opinion* method to generate the ITA based opinion, $\omega_X^{\oplus S_{IT}}$. Once, we have both $\omega_X^{\oplus S_{DT}}$ and $\omega_X^{\oplus S_{IT}}$ then by applying formula (23) to (28) opinions are merged into a single opinion, $\omega_X^{\oplus (S_{DT}, S_{IT})}$. Finally, the trust value of *X*, E_X, is calculated by applying the formula (29) and is normalized to scale of 5.

Algorithm 2 Computation of opinion from Direct Trust Artifacts

```
procedure create_internal_opinion (DTAX, SDT)
    for Si ∈ SDT do
        positive_evidence ⟵ null
        negative_evidence ⟵ null

        Si:ev(X) ⟵ generate internal evidences(X)
        for e ∈ Si:ev(X) != null do
            if e is a positive_evidence
                inc positive_evidence
            end
            else
                inc negative_evidence
            end
        end
        apply formula (1) to (4) to determine (b, d, u, a), ω_X^{Si}
        # reputation matrix of SDT is presented in Table 4.8
        evaluate reputation (ri) of Si based on F1-score, ω_{Si}^{ri}

        apply formula (14) to (17) to calculate weighted opinion of Si;
    ω_X^{ri:Si} = ω_{Si}^{ri} ⊗ ω_X^{Si}
    end
    apply formula (18) to (22) for fusion of different opinion from the different source,
ω_X^{⊕SDT}
```

Algorithm 2 outlines the steps that are required to process the input DTA_X to direct trust-based trust tuple, $\omega_X^{\oplus S_{DT}}$; details are given in Sec. 4.3.1. Evidences generated from S_{DT}, are classified into either positive or negative evidences based on their behavior towards the App. Afterwards, formulas (1) to (4) are applied to compute the trust tuple. Formula (14) to (17) are used to combine the source's reputation with $\omega_X^{S_i}$ to compute $\omega_X^{r_i:S_i}$. Once we have the opinion

from each source then using formula (18) to (22) merge them into a single opinion, $\omega_X^{\oplus S_{DT}}$; where each opinion weights are considered as equal.

Similarly, Algorithm 3 presents the steps that are needed to map the input ITA_X to the indirect trust-based trust tuple, $\omega_X^{\oplus S_{IT}}$; details are provided in Sec. 4.3.2. The key difference with Algorithm 2 is, in Algorithm 3 for each evidence, the review reputation and the temporal weight is used to determine the influence of the evidence on an App. For both Algorithm 2 and 3, the time complexity is *O(MN)*, where M is the total number of sources, and N is the number of evidences generated from each source.

Algorithm 3 Computation of opinion from Indirect Trust Artifacts

```
procedure create_external_opinion (ITAX, SIT)
    for Si ∈ SIT do
        positive_evidence ⟵ null
        negative_evidence ⟵ null

        Si:ev(X) ⟵ generate internal evidences(X)
        for e ∈ Si:ev(X) != null do
            weight[e] ⟵ temporal_weighte * review_reputation_weighte
            if e is a positive_evidence
                inc positive_evidence by weight[e]
            end
            else
                inc negative_evidence by weight[e]
            end
        end
        apply formula (1) to (4) to determine (b, d, u, a), ωX^Si
        # reputation matrix of SIT is presented in Table 4.10
        evaluate reputation (ri) of Si; ωSi^ri
        apply formula (14) to (17) to calculate weighted opinion of Si;
    ωX^(ri:Si) = ωSi^ri ⊗ ωX^Si
    end
    apply formula (18) to (22) for fusion of different opinion from the different source,
ωX^⊕SIT
```

4.2.3 Framework Evaluation

We have created a prototype based on the E-SERS and have empirically evaluated it in the context of the Google PlayStore. It's an experimental approach where we have adopted both

quantitative and qualitative methods of evaluations. To compute DTA and ITA, quantitative evidences are generated using scientific and well-accepted tools. Again, for the investigation of the discrepancy between existing ranking schemes with ours, we follow the qualitative evaluation method. To do that we have observed the reviews and user sentiments, which are subjective evidences.

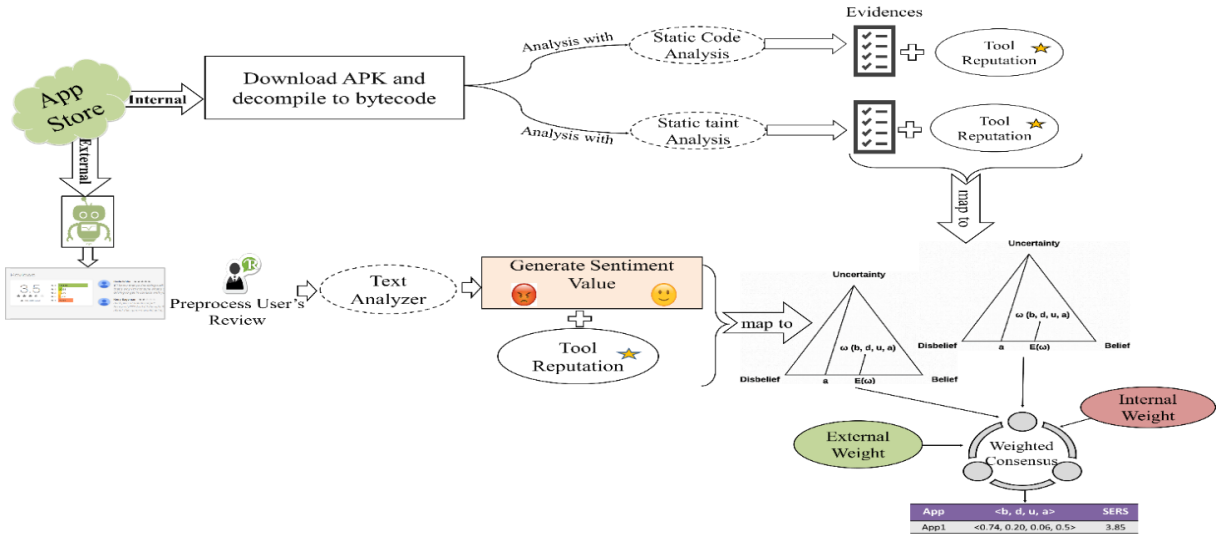

Figure 4.4. E-SERS System Flow Diagram.

For the user involvement, we have developed a Web interface for the prototype where the user is required to provide an App's URL in the Google PlayStore. Then, in an automated way, we obtain the package name and applying the package name, its APK (Android Package Kit) file is downloaded using the third party APK downloaders (e.g., APKPure [94]). The APK file is used as input to compute the DTA of that App. Similarly, to compute ITA, we have implemented an automated Web crawler that fetches the newest and relevant reviews and other details of an App (such as average rating score, reviewer id, creation date, num of likes, and the corresponding rating) from the Google PlayStore. The system flow diagram is presented in Figure 4.4. We identified 5 categories in the Google PlayStore - which are Shopping, Travel, Insurance, Finance, and News. From these categories, we selected 25 Apps and stored their corresponding details in the database.

This section is mostly about the computation of direct trust and indirect trust artifacts in details. Sources (tools) that are incorporated to generate the evidences are discussed here. For DTA we have considered three sources that are capable to generate a different set of internal evidences of an App. Again, for ITA single source is utilized to generate the external evidences. Then the

mapping mechanism for each source evidence to opinion is manifested here. In addition to that, the way to combine both internal and external opinion is illustrated here with appropriate notation.

4.2.3.1 Computation of Direct Trust

To generate the DTA of an App available in the Google PlayStore, we have utilized a static analysis tool called FindBugs, and a static taint analyzer tool called FlowDroid. These tools are selected based on their availability, as all of these tools are open-source code. Additionally, other researchers have frequently used these tools in their reported studies [52][95]. These two tools provide different perspectives about the internal aspect of an App,

- **FindBugs** detects general Java coding bug patterns; the FindBugs' opinion is represented as $\omega_X^{S_1}$.
- **Flowdroid** identifies sensitive data leaks; the FlowDroid' opinion is expressed as $\omega_X^{S_2}$.

4.2.3.1.1 Mapping Evidences of S_1 to $\omega_X^{S_1}$

Similar to our past work [24] [25], in this study, the high confidence bugs reported by FindBugs are counted as negative evidences and the low confidence bugs are counted as positive evidences. The medium confidence bugs are considered as uncertain evidences, which are equally distributed between the positive and negative evidences. As the high confidence means that the identified bug is certainly a real bug. Low confidence bugs are ideally false positives and medium confidence bugs lie in between these two extremes. To quantify the evidences as a (b, d, u, a) tuple, we have applied the formulas (1), (2), (3), and (4). For example, for a particular bug rank, if we received 100 high confidence bugs, 10 medium confidence bugs, and 256 low confidence bugs then using the above formulae, the trust tuple will be (0.70, 0.28, 0.02, 0.5). The approach, shown in Figure 4.5, encompasses different phases.

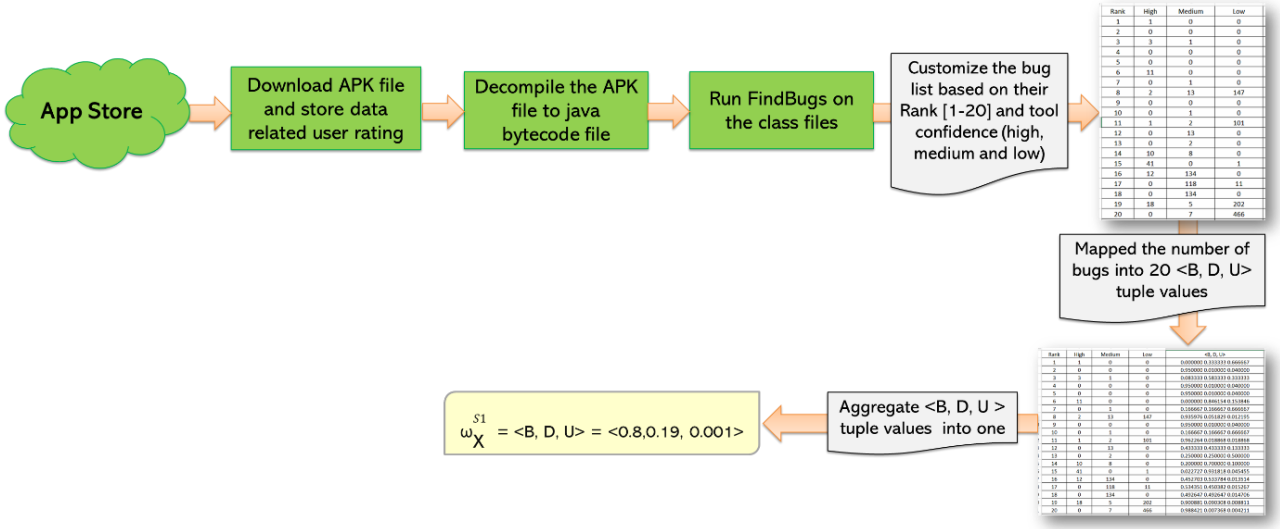

Figure 4.5. Evidence Mapping generated by FindBugs

As mentioned, FindBugs produces twenty possible categories of evidences for an App based on different bug priorities. After we compute twenty (*b*, *d*, *u*, *a*) tuples which indicate different opinions about the trust of the same App, these can be merged using the consensus operator to create a single opinion. In our approach, we have applied the weighted consensus operator, formula (23) to (28), to combine individual tuples into a single opinion. Here, the weights are set using the FindBugs assigned priorities, where the top priority bugs are assigned a higher weight than the low priority bugs.

4.2.3.1.2 Mapping Evidences of S_2 to $\omega_X^{S_2}$

Like the SERS scheme [23], we have also introduced a four-step analysis for mapping sensitive data leaks to trust tuples, where steps 1, 2, and 4 remain the same — only the approach to assess the risk has changed. That process is:

1) Initially, with the help of FlowDroid, we identify all the sensitive sources and sink pairs that are responsible for leaking the critical information. Here, an Android Source is an external resource from which data is read - such as, *getDeviceId*(); which returns an IMEI (International Mobile Equipment Identity). An Android Sink is an external resource to which data is written - such as, *sendTextMessage*(); both the message and the phone number it receives are non-constant.

2) The list of sources and sinks is given as an input and, in our current implementation of E-SERS, is taken from the SuSi [78]. With the help of the SuSi classifier, we classify sources and sinks into different categories. In SuSi, source APIs are classified into 14 different categories (e.g., *ACCOUNT*, *UNIQUE_IDENTIFIER*, *LOCATION*, *CONTACT*, *NETWORK*, etc.). Similarly, sink APIs are categorized into 16 distinct categories (e.g., *FILE*, *LOG*, *NETWORK*, *NFC*, *SMS_MMS*, etc.). The use of SuSi categories is easy to understand than the specific source-sink pairs.
3) We assess the risk factors associated with permissions that are given to sensitive APIs. As Android allows the APIs to access sensitive data after receiving the permissions from the user, permissions play a substantial role in determining the risk of the data leaks. Android has divided these permissions into different protection levels that affect whether runtime permission requests are required or not. Potential risks using the permissions are characterized by Normal, Signature, and Dangerous. The permission identifiers mapped to the corresponding APIs using PScout [91], which is a technique to conduct the mapping from API calls to permissions identifier. The NIST guideline for Risk management of information technology system [92] [93] is followed to assess the quantitative risk associated with the Android permissions.

The determination of these risk levels is subjective. This argument can be justified in terms of the probability assigned for each threat likelihood level and a value assigned for each impact level. The risk assessment matrix is given in Table 4.6. By applying the Formula (31), the risk assessment scale is divided into three different categories: *High*, *Moderate*, and *Low*, similar to the SERS scheme.

The 4 × 4 risk assessment matrix, shown in Table 4.6, contains four levels of likelihood and impact. Based on the permission that is requested, the level of impact is classified into four different categories:

- Catastrophic (identifiers that fall into the Dangerous permission identifiers category),
- Critical (identifiers that fall into the Significant permission identifiers category),
- Marginal (identifiers that fall into normal permission identifiers category), and
- Negligible (identifiers that do not belong to any of the permission identifiers categories).

Table 4.6. Quantitative 4 × 4 Risk Assessment Matrix

Likelihood (Source/Sink)	**Level of Impact (Source/Sink potential risk of permission)**			
	Catastrophic (100)	Catastrophic (50)	Marginal (20)	Negligible (10)
Frequent (1.0)	High (100)	Moderate (50)	Moderate (20)	Low (10)
Probable (0.5)	Moderate (50)	Moderate (25)	Low (10)	Low (5)
Remote (0.2)	Moderate (20)	Low (10)	Low (4)	Low (2)
Improbable (0.1)	Low (10)	Low (5)	Low (2)	Low (1)

The source and sink categories are placed into different likelihood categories based on their appearance. We have selected three malware datasets for this observation; one from VirusShare [96] and two others from Drebin [97] which contain in total reported 2555 malicious Apps. Both VirsuShare and Drebin are repositories that contain malware samples. For these Apps, we have applied Flowdroid and stored the source and sink categories that have been reported. If any of the source/sink categories appear in these three-malware datasets then those are classified as belonging to Frequent; if it appears in 2 of the observed datasets, then it belongs to Probable class; if it appears in only one dataset will be classified into the Remote class; and the rest of the categories are considered as Improbable. The source and sink distribution to different likelihood categories is given below:

Table 4.7. Likelihood categorization based on appearance.

Likelihood	Source Category	Sink Category
Frequent (1.0)	ACCOUNT_INFORMATION LOCATION_INFORMATION NETWORK_INFORMATION NO_CATEGORIES UNIQUE_INFORMATION	LOG NETWORK NO_CATEGORIES SMS_MMS
Probable (0.1)	DATABASE_INFORMATION FILE_INFORMATION	ACCOUNT_SETTINGS FILE CONTACT_INFORMATION
Remote (0.2)	CONTACT_INFORMATION NFC UNIQUE_INFORMATION	CALENDAR_INFORMATION SYSTEM_SETTINGS
Improbable (0.1)	Rest of the Source Categories	Rest of the Sink Categories

4) Any evidence that confirms the data confidentiality is considered as positive evidence and one that involves in information leakage is a negative evidence. Along with the analysis report generated by FlowDroid, we also keep track of the runtime log file. From that log file, we extract the number of total Sources (*ST*) that exists in an App's code. If there is no leak, then *ST* is considered as a total number of positive evidence. Again, if data leaks are found then the positive evidences are calculated by subtracting the number of faulty Sources (*SF*) from *ST*; where *SF* indicate those sources, which are involved in information leakage. Once the evidences are generated then the formulae (1), (2), (3) and (4) are applied to compute the (*b*, *d*, *u*, *a*) tuple that reflects the opinion $\omega_X^{S_2}$.

4.2.3.1.3 Evidence Processor and Fusion of Opinions based on Direct Trust

Before combining the opinions $\omega_X^{S_1}$, and $\omega_X^{S_2}$, the reputation of the sources is considered as a trust tuple $\omega_{S_1}^{r_1}$, and $\omega_{S_2}^{r_2}$, as discussed in Sec. 3.5.2.5. We have used two approaches to evaluate the reputation of the tools: reputation based on the existing literature and reputation based on experimentation on benchmarks. For tool *S1* and *S2*, we have adopted the first approach and gathered the precision, recall, and F-measure from the existing literature. In [98], to evaluate the FindBugs, a test suite is used formed by merging reliable resources such as IBM Haifa Research Lab and CERT. Again, to assess the FlowDroid DroidBench [99]; a micro bench-mark suite; is utilized in [80]. The reputation-related results for all three tools are presented in Table 4.8. Here, Precision (*p*) indicates the rate of false positives and Recall (*r*) estimates the false negatives against true positives. The reputation tuple is based on the *F1-score* as it is the weighted mean of Precision and Recall and gives a better measure of the wrongly classified instances than the Accuracy metric [100]. F1-score is considered as the value of belief and the rest is assigned to disbelief. Here, the uncertainty remains zero based on the assumption that the benchmarks are formulated by the domain experts, so there is no scope to have the ambiguity. Based on existing literature, the reputation scores of *S1* and *S1* are *(0.51, 0.49, 0, 0.5)* and (0.89, 0.11, 0, 0.5) respectively.

Table 4.8. Reputation of S_{DT}

Source ($S_{DT} \subset S$)	Precision (p)	Recall (r)	F1-Score (2*p*r/(p+r))	Reputation <b, d, u, a>
FindBugs (S1)	1	0.34	0.51	<0.51, 0.49, 0, 0.5>
FlowDroid (S2)	0.86	0.93	0.89	<0.89, 0.11, 0, 0.5>

Then, each source reputation ($\omega_{S_1}^{r_1}$, $\omega_{S_2}^{r_2}$) is combined with the corresponding opinions ($\omega_X^{S_1}$, $\omega_X^{S_2}$) for each tool using the discounting operator (addressed in formula (14) to (17)) to generate discounted opinions($\omega_X^{r_1:S_1}$, $\omega_X^{r_2:S_2}$) for each evidence source (tool). As an example of the calculation, let us assume that for an App *X*, the evidence-based opinion tuples of S_1, and S_2 are $\omega_X^{S_1}$ = (0.7, 0.2, 0.1, 0.5), and $\omega_X^{S_2}$ = (0.6, 0.25, 0.15, 0.5) correspondingly. After weighting these opinions with their consequent reputations $\omega_{S_1}^{r_1}$, and $\omega_{S_2}^{r_2}$ the values of $\omega_X^{r_1:S_1}$, and $\omega_X^{r_2:S_2}$ become

(0.36, 0.1, 0.52, 0.5), and (0.54, 0.22, 0.24, 0.5) respectively. Finally, the formula (18) to (22) are applied to fuse the weighted opinions that returns the opinion based on direct trust, $\omega_X^{\oplus S_{DT}}$ = (0.59, 0.24, 0.17, 0.5).

4.2.3.2 Computation of Indirect Trust

4.2.3.2.1 Data Collection and Pre-processing

This section describes an overview of our dataset crawler and required data preprocessing phases. We selected 25 Apps from the Google PlayStore in 5 different categories. We picked Apps in each category to generate ITA, that offer similar functionality and have a reasonable number of user reviews, the average number of scraped reviews are 3,070 (per App), details are given in Table 5.3. For each App, we scraped three different data items: 1. App's basic details (such as user rating, category, total number of reviews, total number of installs, etc.), 2. Newest reviews, and 3. Most relevant reviews; using an in-house tool. Google PlayStore characterizes the App's reviews into three different categories are: Newest, Most relevant, and Rating. For our evaluation, we are only interested in the Newest and Most relevant datasets. Figure 4.6 depicts the architecture of our crawler. The components of our crawler are presented below:

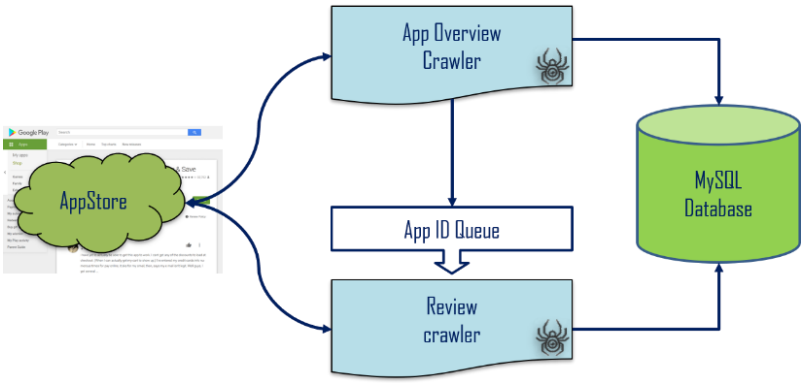

Figure 4.6. Architecture of Data Collection Phase

- **App Overview Crawler** fetches App's basic details and stores into the database. Additionally, it places the App id to the App id queue.
- **App ID queue** receives the App id extracted by an App Overview Crawler. Each time the App Overview Crawler fetches a single App id from the AppStore it stores that id into the queue.

- **Review Crawler** is designed for crawling the Most Relevant and Newest review pages, as we are interested in collecting the helpful (*liked*) and recent reviews by the users. The helpfulness score of the Most Relevant reviews is used to calculate the reputation, described later (Sec. 4.2.3.2.4). Each parsed review contains the reviewer id, creation time, review text, number of likes, and the corresponding rating. After collecting the dataset, we preprocess the reviews, as most reviews are in the form of unstructured text. Reviews are converted to Unicode and then stored into the database. Before passing the reviews for sentiment analysis, they are decoded from Unicode using the built-in API supported by the Unicodedata library [101]. This API helps to remove umlauts, accents, and other similar features.

4.2.3.2.2 Mapping Sentiment Value to Opinion Model

The IBM Watson Natural Language Understanding NLU) [82] tool is used to predict the sentiment of the preprocess reviews through natural language processing. It is capable of analyzing and understanding the text, including sentiment, emotion, keywords, language, entities, metadata, relations and semantic roles. The API returns the sentiment score in the range of [-1, +1] and indicates whether a given review reflects the positive or negative sentiment of the user. The NLU opinion is represented as $\omega_X^{S_3}$.

4.2.3.2.3 Conversion of Sentiment Score to Subjective Logic-based Tuples

We have followed a similar conversion scheme with boundary cases as described by Gallege [37] while mapping the NLU opinion to $\omega_X^{S_3}$. However, that technique utilized Linear Regression model to predict the trust tuple whereas we have adopted Random Forest Regression model. As we have noticed, mean absolute error [102] is higher for Linear Regression than Random Forest regression model. The following table (Table 4.9) contains the boundary cases for converting textual sentiments (i.e., 1 dimension, [polarity [-1, +1]) to Subjective Logic-based tuples (i.e., 3 dimensions (belief [0, 1], disbelief [0, 1], and uncertainty [0, 1]). Here, (0, 1, 0) represents the extreme disbelief and similarly, (1, 0, 0) represents the extreme belief about a review. Afterwards, these boundary cases are fed into a Random Forest Regression model to predict *b* and *d*; since *b*, *d*, *u* is linearly dependent such that $u = 1 - (b + d)$.

Table 4.9. Sentiment score map to <b, d, u>.

Sentiment Score	<b, d, u>		Sentiment Score	<b, d, u>
-1	(0, 1, 0)		+1	(1, 0, 0)
-0.75	(0, 0.75, 0.25)		+0.75	(0.75, 0, 0.25)
-0.5	(0, 0.5, 0.5)		+0.5	(0.5, 0, 0.5)
-0.25	(0, 0.25, 0.75)		+0.25	(0.25, 0, 0.75)

4.2.3.2.4 Determining the Reputation of Reviews

To determine the reputation of reviews, research have mostly applied reviewer-centric approach [103] [104], which is composed of features related to the reviewers' behavior. However, typically an AppStore does not provide the reviewer's details. Therefore, the reviewer-centric approach is not applicable for the AppStores such as the Google PlayStore. Hence, we propose a review-centric approach to determine the review reputation. The "Most relevant" category contains the set of reviews that were agreed (*liked*) by the other users. In our approach, this dataset is used to establish the reputation of any new review. The features of the most relevant reviews that used are: `num of likes' that the review received and the 'sentiment score'. Next, the mapping mechanism mentioned above is applied to convert the sentiment score of the review's dataset into (*b*, *d*, *u*) tuples. The (*b*, *d*, *u*) tuples of Most Relevant reviews are clustered (using k-means [105]) into different clusters (C_1, C_2, ..., C_N); N is the total number of clusters. Finally, the average number of 'total likes' (L) for all reviews ($\forall_r$) that belong to a cluster C_i is used as a weight for that cluster computed as:

$$W_{C_i} = \frac{\sum_{\forall r \in C_i} L_r}{\sum_{\forall r \in C_i} r} \quad (34)$$

Once the weight is determined for each cluster, we predict the cluster membership for reviews in the Newest dataset. Based on the cluster determination, the corresponding weight is assigned to the review. A high value of the weight represents a highly reputed review, and a low value denotes lower importance to that review (probably fake reviews). Thus, this relational discovery between "Most relevant" and "Newest" reduces the influence of fake reviews.

4.2.3.2.5 Determining the Temporal Weight

In our approach, recent reviews are assigned higher weights than older reviews. After a certain period interval, each App developer releases a new version that makes the older reviews less useful. However, in a new release, developers typically try to fix a limited number of bugs and update a few features. Hence, it is necessary not to fully ignore the previous reviews of the App. Considering that we have introduced a temporal weight for each review that helps to reduce the impact of older reviews. The weight is determined by Hawkes Processes; a Self-exciting Spatio-temporal point processes model [106] [107]. In this model, we feed the timestamps of reviews from the newest reviews' dataset. Then the Hawkes Processes model learns to exponentially weight reviews going back in time and returns the corresponding weight for each timestamp. Figure 4-5 shows how the temporal weights are assigned to recent reviews. For our convenience, we have normalized the temporal weights to a scale of 10.

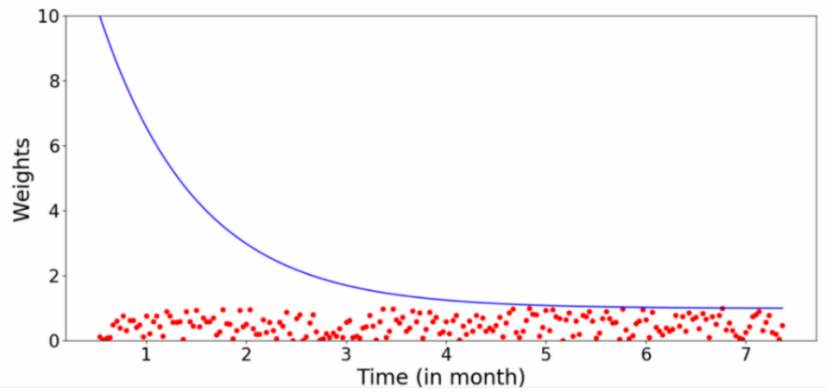

Figure 4.7. The line represents the exponential temporal weighted values, and the dots indicate the occurrence of the reviews over the time (the timestamp difference is in month

4.2.3.2.6 Computing Opinion of Indirect Trust

Three elements are required to determine $\omega_X^{S_3}$: The review sentiment score, the temporal weight, and the weight of the review reputation. Once, we have both the weights then by multiplying [108] them, we can compute the total weight for a review. Here, any review that has the sentiment score >0 and <=1 is considered as a positive evidence and one that indicates sentiment score between 0 and -1 is negative evidence. Once the evidences are generated then

formulae (1), (2), (3) and (4) are applied to compute the (*b*, *d*, *u*, *a*) tuple that indicates the opinion $\omega_X^{S_3}$.

Once we have the opinion of the tool S_3 then we need to evaluate the reputation of the NLU tool (watson). To do so, we have used an experimental approach. Existing literature provides watson's F1-score for a different dataset (i.e., movie reviews [109], and Twitter comment data set [110]). Therefore, to assess Watson, we have created a benchmark based on our collected reviews. The benchmark contains 2000 manually labeled reviews, where each positive and negative review category comprises of 1000 reviews. To achieve this, we asked a team of 4 domain experts to manually label the sentiment (either positive or negative) of each of 750 reviews, which is a total of 3000 reviews. Then from the labeled dataset, we randomly picked 1000 positive reviews and 1000 negative reviews each. To ensure the quality of labels we exchange the reviews with one another and cross-verify repeatedly. If any confusion is occurred, then based on the majority judgment the review is labeled. The confusion matrix for this dataset is given in Table 4.10.

We calculate the Precision (*p*) = *TP/(TP+FP)* and Recall (*r*) = *TP/(TP+FN)* values which are 0.89 and 0.85 accordingly. So, the *F1-score = 2* pr/(p+r)* of the NLU tool is 0.87. Thus, the reputation of S_3 is: $\omega_{S_3}^{r_3}$ = (0.87, 0.13, 0, 0.5). Next, the formula (14) to (17) are applied to compute the $\omega_X^{r_3:S_3}$. To compute the opinion of indirect trust, we have used a single source (watson) to generate evidences, hence, the fusion of opinions is not required here ($\omega_X^{r_3:S_3} \Leftrightarrow \omega_X^{\oplus S_{IT}}$).

Table 4.10. Tool NLU - Confusion Matrix.

	Positive (Actual)	Negative (Actual)
Positive (Predicted)	853 (TP)	99 (FP)
Negative (Predicted)	147 (FN)	901 (TN)

4.2.3.3 Evidence Processor and Opinion Fusion

Once we have the opinions for the direct trust ($\omega_X^{\oplus S_{DT}}$) and indirect trust ($\omega_X^{\oplus S_{IT}}$) of an App, we combine them into a single opinion; using the cumulative weighted fusion operator mentioned in the formula (23) to (28). The direct trust-based evidence likely to have less ambiguity as it solely focuses on the functional perspectives of an App. So, we assign a lower weight to the $\omega_X^{\oplus S_{IT}}$ than to the $\omega_X^{\oplus S_{DT}}$; the assigned weights are 30% and 70% respectively. These weights can be adjusted as user desires. This resultant opinion, $\omega_X^{\oplus (S_{DT}, S_{IT})}$, counts all available evidence and thus, provides a more reliable quantification of trust associated with each App than the basic average star ratings provided by the Google PlayStore. The $\omega_X^{\oplus (S_{DT}, S_{IT})}$ allows us to calculate the trust score (E_X) using the formula (29), which is normalized to a scale of 5. The value of E_X helps to rank-order similar Apps. The ranking generated by E-SERS is compared using the Kendall Tau Distance method [88] that considers the number of pair-wise variances between two ranking lists – the approach presented in [8]. Distances of 100% and 0% represent the opposite and identical rankings respectively.

CHAPTER 5. E-SERS VALIDATION

In this chapter, first, we describe the dataset used in our experiments. Then, we discuss the outcomes obtained by employing the DTA (Sec. 5.1) and ITA sources (Sec. 5.2). In Sec. 5.3, we present the impact of weights and how that influences the E-SERS and in Sec. 5.4, we have shown the comparison of different ranking schemes based on empirical investigation. Finally, Sec. 5.5, discuss our web prototype.

In our study, Google PlayStore Apps from the shopping, travel, insurance, finance, and news categories. The rationale for selecting Apps from these categories is that these categories have been identified by NowSecure in their research effort [13] [111]. NowSecure [112] is the mobile App security software company trusted by most industry leaders, such as Verizon, Lenovo, and others. There are other solutions that exist solely to detect the harmful viruses that may be present in (such as Google Play Protect [113], AVG [114], Norton Mobile Security [115], and others) Apps and these alternatives return risk warnings. However, the warnings generated by these alternatives are not quantifiable. NowSecure, on the other hand, generates a quantifiable result, called a risk score, for an App. In our study, we have investigated the association between NowSecure and our DTA-based insights. However, NowSecure, being an enterprise assistance solution, is a paid service. As we do not have a subscription to NowSecure's paid service, we could not gather any evidences about the Apps in our data set. Hence, instead of investigating the quantitative relation between the two approaches, we outline a qualitative relation below.

From each category, five different Apps were identified for our experiments. In each category, we selected one App that was used by NowSecure in their study. After that, we identified four other Apps that are "similar in functionality" (as indicated by the Google PlayStore) to that App and have a reasonable number (average number of reviews per App is 2,100) of user reviews. The details of our approach are indicated in Table 5.4. These selected Apps belong to different ranges of popularity (such as the most popular, popular, and less popular) in terms of the number of installs. During SERS evaluation, we have addressed the correlation between the traditional star rating, popularity (number of installs), and trust of an App (see Sec. 4.1.2.1).

The dataset that we prepared for our evaluation is collected from July 23 to October 29, 2019. However, in the following discussions, we do not disclose the App's details (such as App's package name or any specific id) and keep App's id anonymous (e.g., *App1*).

5.1 Findings Generated by DTA Sources

The number of data leaks identified by Flowdroid for each category of Apps along with the reported sources and sinks categories are presented in Table 5.1. Source and sink APIs that belong to *NO_CATEGORY* is not reported here, as they refer to non-sensitive data flows in SuSi [78]. In [49], authors identify that sources that are categorized into *NETWORK_INFORMATION* and *UNIQUE_IDENTIFIER* are more likely to occur in malware Apps than in benign Apps. In addition, that study indicates that malware Apps are more prone of to use short message service (SMS) as sinks to leak data to third parties - such scenarios are found in our test dataset too. For **News** category Apps, we noticed the source API belong to the *UNIQUE_IDENTIFIER* category and the quantity of sink APIs that refers to *SMS_MMS* is comparatively much higher than the other categories.

Table 5.1. Data leaks details generated by FlowDroid.

App Category	# of Data Leaks	Source Categories	Sink Categories
Shopping	664	LOG (239) SMS_MMS (186) NETWORK_INFORMATION (17) FILE (6) LOCATION_INFORMATION (2)	SMS_MMS (93) NETWORK (24) FILE (5) CALENDAR_INFORMATION (4) CONTACT_INFORMATION (3)
Travel	881	SMS_MMS (68) LOG (63) FILE (8) NETWORK_INFORMATION (3) CALENDAR_INFORMATION (2) ACCOUNT_SETTINGS (1)	SMS_MMS (46) FILE (10) CALENDAR_INFORMATION (2) ACCOUNT_SETTINGS (1) NETWORK (1)
Insurance	635	SMS_MMS (186) LOG (155) FILE (9) ACCOUNT_SETTINGS (5) NETWORK_INFORMATION (4) CALENDAR INFORMATION (2)	SMS_MMS (73) NETWORK (16) ACCOUNT_SETTINGS (3) CALENDAR_INFORMATION (3) FILE (2)
Finance	1237	LOG (161) SMS_MMS (63) NETWORK_INFORMATION (13) FILE (2)	SMS_MMS (86) NETWORK (9) CALENDAR_INFORMATION (5) LOG (2)
News	1399	LOG (114) SMS_MMS (80) UNIQUE_IDENTIFIER (14) FILE (9) NETWORK_INFORMATION (8) ACCOUNT_SETTINGS (3)	SMS_MMS (157) NETWORK (18) LOG (13) FILE (6) ACCOUNT_SETTINGS (4) CALENDAR_INFORMATION (3) CONTACT_INFORMATION (1)

FindBugs identified more than 300 programming mistakes and suspicious coding paradigms (such as de-referencing of null pointers) using its simple analysis techniques than the deep analysis techniques. In Table 5.2, we have presented the high confidence warnings (for each bug pattern) for different App categories. The Apps in **News** category have outnumbered other categories of Apps for high priority warnings that reported from the **Malicious Code Vulnerability** category.

Table 5.2. High priority warnings for each bug category generated by FindBugs

	Shopping	**Travel**	**Insurance**	**Finance**	**News**
Bad Practice	176	172	278	332	223
Style	248	534	1123	1685	1126
Malicious code vulnerability	131	463	448	260	1102
Correctness	391	932	1058	1473	1755
Performance	38	51	88	96	102
Multithreaded correctness	15	10	28	35	46
Internationalization	204	421	398	483	502

An interesting insight from the direct trust-based result is that all three tools (FlowDroid, FindBugs) show that the Apps in the **News** category are more vulnerable than other categories. A similar observation has been reported by NowSecure where they indicated that almost all local news Apps (in their dataset) leaked user data. Whereas 40% of them had severe security vulnerabilities that could lead in sensitive information being compromised.

5.2 Findings Generated by ITA Sources

As indicated, we have collected a dataset of 25 Apps from five different categories. The dataset of the associated user reviews is described in Table 5.3. The matrix of average words (per review) denotes that, the Most Relevant reviews are always more detailed than the reviews are in the Newest category.

Table 5.3. Statistics of Collected User Review Dataset.

Newest Review Dataset			**Most Helpful Review Dataset**	
Total number of crawled reviews	52,519		Total number of crawled reviews	24,299
Average Number of reviews per App	2100		Average Number of Reviews per App	970
Average Words per Review	14.8		Average Words per Review	22.3

Figure 5.1 presents the sentiment scores for each review in our dataset where every point denotes the score for an individual review. The box plot shows the median, first, and third quartiles and minimum and maximum sentiment scores for individual rating scale 1 to 5. However, a significant amount of outlier is evident for the ratings of 1, 2 and 5.

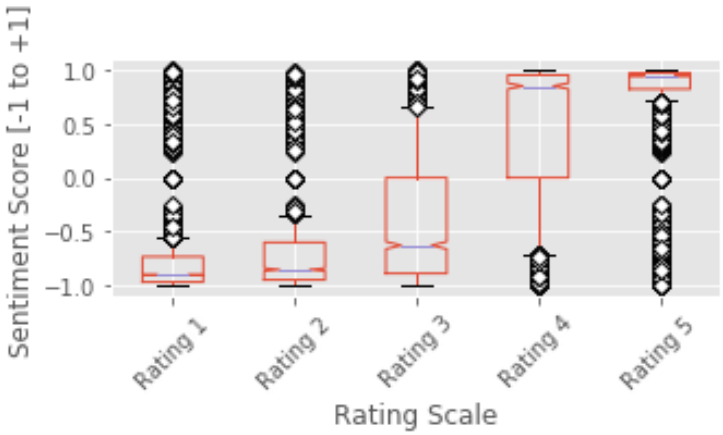

Figure 5.1. User given Rating Score vs Review's Sentiment Score.

After examining the review sentiments and the corresponding ratings, we found some mismatches. For example:

"*Don't care for this app. Too confusing, even when it works.*" - the user provided a rating of 5; where Natural language understanding tool returned a negative sentiment for this review, reflecting a mismatch.

"The website is a lot better" - the user provided a text with positive sentiment; however, the user provided a rating of 1, again reflecting a mismatch.

We also performed a review-based evidence analysis (presented in Figure 5.2) between the Newest and Most Relevant reviews datasets.

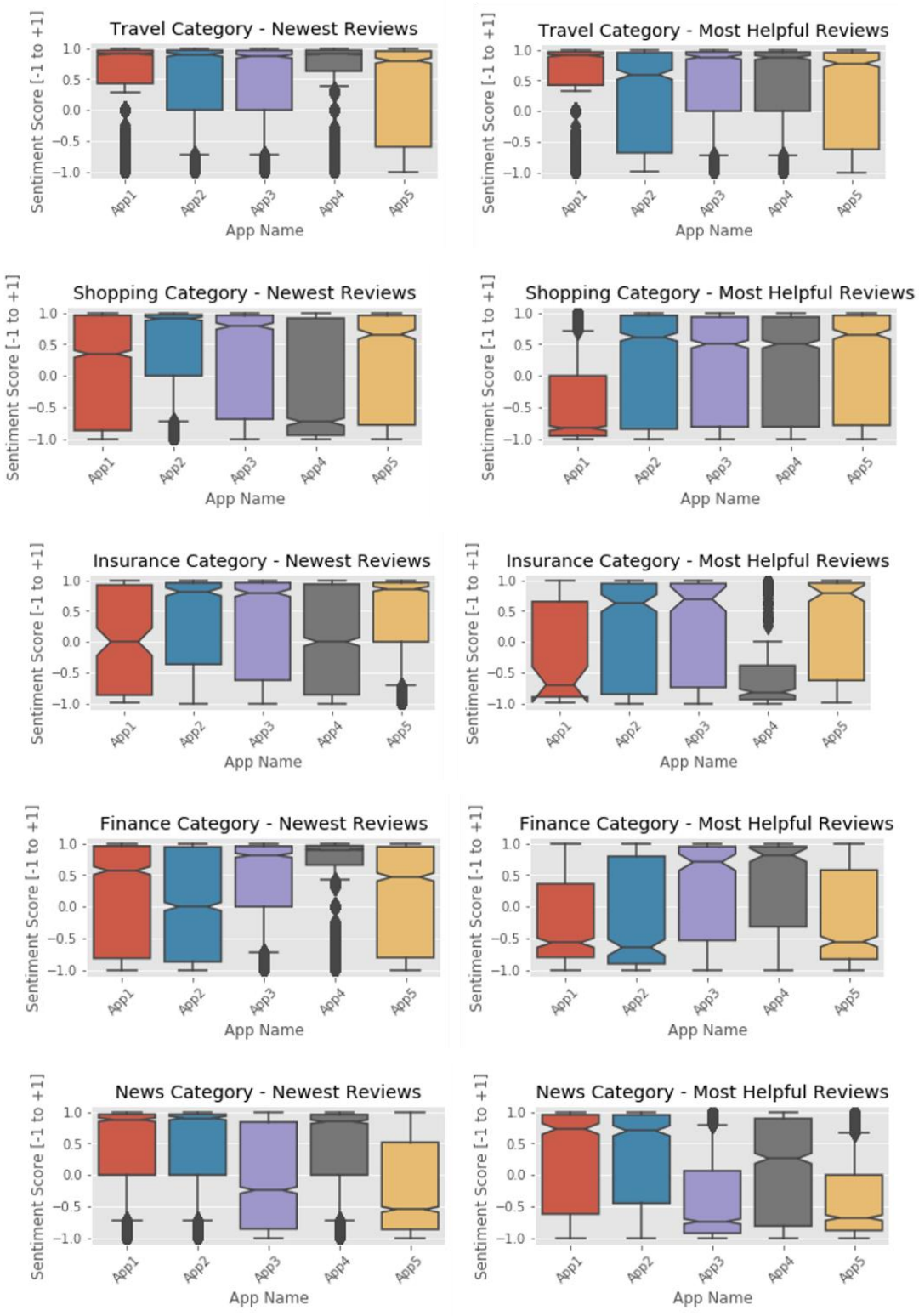

Figure 5.2. Review based evidence analysis.

This investigation highlighted the disagreement between these review datasets. For every category, there is a clear mismatch of evaluation based on these two datasets. For example, Newest reviews of *App2* in the shopping category mostly present positive sentiment whereas Most relevant feedback indicates a mix of positive and negative sentiment. However, we have noticed a significant difference in the **News** category. Here, the sentiment score for each App's Newest reviews dataset deviates from a high to low sentiment score for the Most relevant reviews dataset. For example, the sentiment score of *App3* in the News category deviates from [0.75, -0.25] to [0, -0.25]. This indicates that in the News category, users are experiencing similar difficulty (such as ads, malware, bugs, etc.) that previously highlighted by others. Some reviews with a high number of likes in News Category are presented below:

- "*This was my favorite news app but now my phone has a mind of it's own, I have adds pop up randomly while I'm trying to make a phone call or text or anything else, I can no longer post comments on news stories, it's becoming harder and harder to just read a article. This News break app is becoming very broke. I will uninstall if they do not fix things soon.*"; - the number of likes for this comment is **2324** and sentiment score is **-0.599476**.
- "*Used to be 5 stars until ads started popping up. There are ads running continuously on the top of the screen. Now there are pop up ads. When the ad finishes you are not brought back to the page the ad took you away from. I have to delete this app because its ruined now. How do I explain this to all the people that I told how good this messed up app is? Sucks.*"; - the number of likes for this comment is **1765** and sentiment score is **-0.909597**.

From the above explanation, it can be assumed that to look only at the Newest review dataset is not an ideal option. As it fails to unfold the detail behaviors about an App from the user point of view. Therefore, the user should observe the Most relevant reviews as well. However, in the most of those cases, a review's sentiment score leads to a more negative score for the Most Relevant review dataset. We can infer that the reviews in the Most Relevant category tend to have more negative sentiment than the Newest category, which reflects that the users are more inclined to like criticism rather than appreciation of an App. Overall, users give 'like' or write reviews to present their dissatisfaction or problems that they are facing.

Table 5.4. Number of reviews relate to bug and security scope.

App Category	Newest Reviews											
	bug (%)	fix (%)	problem (%)	issue (%)	defect (%)	crash (%)	privacy (%)	security (%)	spy (%)	spam (%)	malicious (%)	leaks (%)
Shopping	9.7	33	23.1	20.7	0.1	26	0.3	2.7	0	0.6	0	0
Travel	11.2	24.7	27.9	28.6	0.2	14.2	1.6	0.2	1.6	1.6	0.2	0
Insurance	9.6	33.9	19.8	19.6	0	27.9	0.2	4.2	0.6	0.2	0	0
Finance	7.3	34.0	27.7	30.3	0	9.8	0.4	5.7	0	1.0	0.2	0
News	8.4	43.2	24.6	23.4	0.2	19.1	1.9	0.6	7.2	1.4	0.6	0
App Category	**Most Helpful Reviews**											
	bug (%)	fix (%)	problem (%)	issue (%)	defect (%)	crash (%)	privacy (%)	security (%)	spy (%)	spam (%)	malicious (%)	leaks (%)
Shopping	9.5	34.2	23.6	23.0	0.1	26.7	0.4	2.7	0	0.6	0	0
Travel	8.5	20.2	33.2	32.7	0.2	11.3	2.6	0.7	0.7	0.5	0	0
Insurance	13.5	26.5	20.6	27.1	0	25.8	0	3.9	0	0	0	0
Finance	10.3	33.2	30.6	34.7	0	7.9	0	4.9	0	0.5	0	0
News	7.6	45.7	21.4	22.8	0.3	23.2	1.9	0.4	0.3	1.2	0.3	0

App Category	Number of Bug and Security issue related Reviews	
	(Newest)	(Most Relevant)
Shopping	904	1057
Travel	437	425
Insurance	499	155
Finance	491	388
News	834	667

We also examined the number of reviews specific to bug or security concerns (presented in Table 5.4). To determine that, we created a list of keywords: bug, fix, problem, issue, defect, crash, solve, permission, privacy, security, spy, spam, malicious, and leaks. Most of the keywords are describe by Maalej et al. [116] under the bug reports review type. Besides that, we also added new keywords, which symbolize the security and data privacy concerns (such as privacy, security, spy, spam, malicious and leaks). We have followed the simple technique of *String Matching*, to automatically check if the reviews contain a certain keyword. For this, we have used SQL queries, while ignoring letter cases and binding around the keywords (e.g., using "LIKE" in SQL).

From the keyword distribution shown in Table 5.4, it is clear that users have addressed more bug-related feedback than privacy and security-related concerns – a fact that was also highlighted in Chapter 4. However, the total number of bugs, privacy and security-related reviews indicate that typical users are not aware of these internal issues. In the privacy and security-related analysis, Apps from the Insurance, Finance and News categories have higher negative feedback than others. Some examples are given below:

- "*I tried the app again, there is no way to see comments or log into my Fox account. The ads point to malicious advertising sites that are full of malware, adware, **spy**ware. Fox needs to vet their advertisers more carefully!"*

- "*Norton Security says this app is a **PRIVACY** RISK, that it collects information from my phone and sends it to an unknown location. I have now deleted it, until I learn more about what this means and what the app is actually doing. I Strongly suggest everyone else do the same*."
- "*Only reason im giving 1 star is theres not a 0 rating here! I agree with one of the comments about the app not be being **malicious** but the advertising is relentless! I uninstalled this app and I'm STILL getting CRAP ADS from marketing advertising sites!!! RIDICULOUS!!!!*"
- "*Grrr, website sucks...faulty **security** cert. And click pay bill on this thing, and yup...nothing....sigh. love American Family though, they need to invest in IT techs though! Jeez*."

For Apps that provide insurance or finance services, users are expected to trust these Apps to be responsible with a considerable amount of their sensitive data. We found that one of the most popular Apps (*App2*) in the Finance category, has been installed more than 10 million times, actively leak sensitive user information.

5.3 Rank Variation based on Weights of Internal and External Opinions

Figure 5.3 shows the influence of weights assigned to the $\omega_X^{\oplus(S_{DT},S_{IT})}$ while combining both internal($\omega_X^{\oplus S_{DT}}$) and external ($\omega_X^{\oplus S_{IT}}$) evidence in all of the selected categories. In Figure 5.3 (a), 70% and 30% weights are assigned to $\omega_X^{\oplus S_{DT}}$ and $\omega_X^{\oplus S_{IT}}$ respectively for the News category Apps. Due to this scheme, the combined ranking behavior resembles more closely to the direct trust evidence-based ranking. For both evidence, equal weights are assigned in Figure 5.3 (b). This eliminates the inclination to any of the evidence-based rankings. Similarly, in Figure 5.3 (c) we have assigned a higher weight (70%) to the indirect trust-based evidences. Due to this, the combined ranking behavior is biased towards the external evidence-based ranking. Other App categories are presented in Figure 5.3 (d) (e) (f) and (g). We adhere to the view that the direct trust evidence provides a better reflection of the App quality and, thus, we assign 70% weight to direct trust and 30% weight to indirect trust.

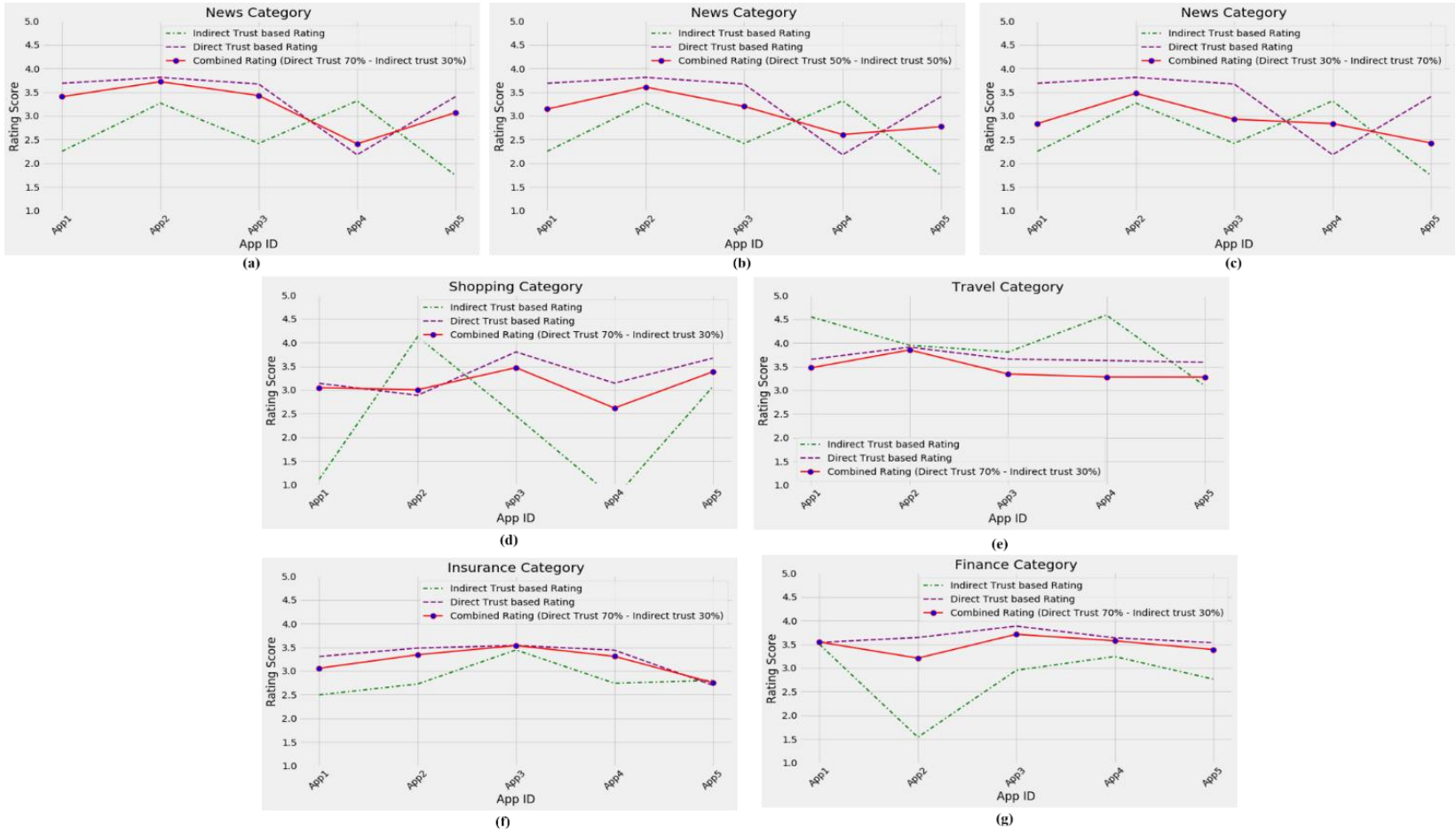

Figure 5.3. Rank Variation based on Weights of Internal and External Opinion.

5.4 Comparison of different Ranking Schemes

Five different kinds of ranking schemes are devised using the outcome of our experiments. These are:

1) Ranking based on Direct Trust Artifacts; $\omega_X^{\oplus S_{DT}}$ (internal view).
2) Ranking based on Indirect Trust Artifacts; $\omega_X^{\oplus S_{IT}}$ (external view).
3) Ranking based on E-SERS; $\omega_X^{\oplus (S_{DT}, S_{IT})}$; computed by combing the internal and external view. The weight of external and internal evidences can be adjusted based on user preferences.
4) Ranking based on the Average Star Ratings.
5) Google PlayStore Rank; from AppBrain [117] Google PlayStore rank for each App is collected.

We illustrate different scenarios for comparing the above-mentioned ranking schemes - similar to our approach described in [23]. The rank-orders differ from one scheme to another. Therefore, we did an empirical analysis to identify the reasons behind this behavior. Table 5.5 shows the computed Kendall Tau distances for four such comparisons of Apps from five different categories.

Table 5.5. Distance between different ranking schemes.

App Category	Average Rating & $\omega_X^{\oplus S_{IT}}$ (Distance %)	Average Rating & $\omega_X^{\oplus S_{DT}}$ (Distance %)	Average Rating & Google PlayStore Rank (Distance %)	E-SERS $\omega_X^{\oplus (S_{DT}, S_{IT})}$ &Google PlayStore Rank (Distance %)
Shopping	10%	50%	40%	30% ~ 50%
Travel	0%	50%	30%	30% ~ 50%
Insurance	40%	60%	30%	40% ~ 50%
Finance	30%	50%	40%	40% ~ 60%
News	20%	30%	40%	30% ~ 40%

1) Average Ratings and $\omega_X^{\oplus S_{IT}}$. In an ideal case, the reviews should be consistent with the star ratings. As indicated by the Kendall Tau variance (Table 5.5) which is between 0% to 40% when we compare rankings obtained by $\omega_X^{\oplus S_{IT}}$ and average star ratings - indicating that these two rankings are reasonably similar to each other. However, for the difference, there could be two potential reasons:

- For the collected review dataset, we have assigned two additional weight factors. One, review-centric reputation score to suppress the impact of false reviews and the other is the temporal weight to minimize the impact of old reviews. Whereas, in average rating score, all reviews are treated equally. This reason may lead to a disparity between these two ranking schemes.
- As explained earlier, a good number of mismatches is observed between the review sentiments and the rating scores. Such mismatches indicate that the star ratings, in many cases, are not a true representation of the text of the associated review.

2) Average Ratings and $\omega_X^{\oplus S_{DT}}$. Table 5.5 indicates that the Kendall Tau distances between $\omega_X^{\oplus S_{DT}}$ based and average rankings is between 30% to 60%. We have selected an App, *App4*, from the News category that has opposite orderings - *App4* has a rank of 2 out of 5 based on the user ratings; while its rank is 5 based on the $\omega_X^{\oplus S_{DT}}$ score.

App4 in News category. The users rated the *App4* as having a good rank, but the $\omega_X^{\oplus S_{DT}}$ based score ranked it very low. A few supportive (first two) and a few critical (last two) user comments are given below. In selecting the reviews, we picked only those reviews which contained bugs or privacy and security-related terms as addressed in Table 5.5. All other reviews, for each App, which do not focus on such features/terms are ignored in this investigation.

"The ability to stream CBS news content live and coverage 24/7 is a great thing. Thanks CBS for being the first major network with this feature. App works well through phone pc and Roku with sharp layout and graphics. Don't understand some of the other bad reviews with technical issues, no problem here!"

"*I've always been able to view the app w/o any problems, always clear picture n crisp clear sound.*"

"*Norton Security says this app is a PRIVACY RISK, that it collects information from my phone and sends it to an unknown location. I have now deleted it, until I learn more about what this means and what the app is actually doing. I Strongly suggest everyone else do the same.*"

"*App started sending notifications without my permission. Deleted.*"

From the reviews of *App4*, we collected a total of 84 reviews that match with one of the keywords mentioned above, which is the only 3.2% reviews of the total reviews of *App4*. This expresses that, most of the users are not conscious of internal issues. Also, among these 84 reviews,

most of the reviews reported a *crash*. During the internal evidence analysis, we found critical security vulnerabilities for this App. Through the security flaws inspection, we noticed that the data leaks associated with *App4* deal with the Dangerous permission access (such as *READ_PHONE_STATE*, *ACCESS_WIFI_STATE*).

3) Average Ratings and Google PlayStore Rank. The Kendall Tau variance (Table 5.5) is between 30% to 40%, when we compare rankings obtained from the use of average star ratings and Google Rank. For the Google PlayStore, some known factors that influence the search result ranking are: App Name, App Description, *Rating* and *Reviews*, Backlinks, In-App Purchase, Updates Downloads and Engagement, and other hidden factors [118]. However, the leading AppStores do not disclose, how the ranking factors are weighted.

To understand the correlation between average rating score and Google PlayStore rank we have conducted a simple experiment. We have fetched a dataset of 500 Apps from AppBrain [116] and Google PlayStore, which contains Google Play Ranking, rating score, number of installs, and number of reviews. The dataset is then fed, as the training set, to a machine learning model (regression with XGBRegressor [119]) to predict the App's rank. Here, the independent features are rating (f_0), number of installs (f_1), and the number of reviews (f_2). The correlation among them is presented in the Figure 5.4 - it shows that the rating, reviews, and the number of installs has impacts on App ranking.

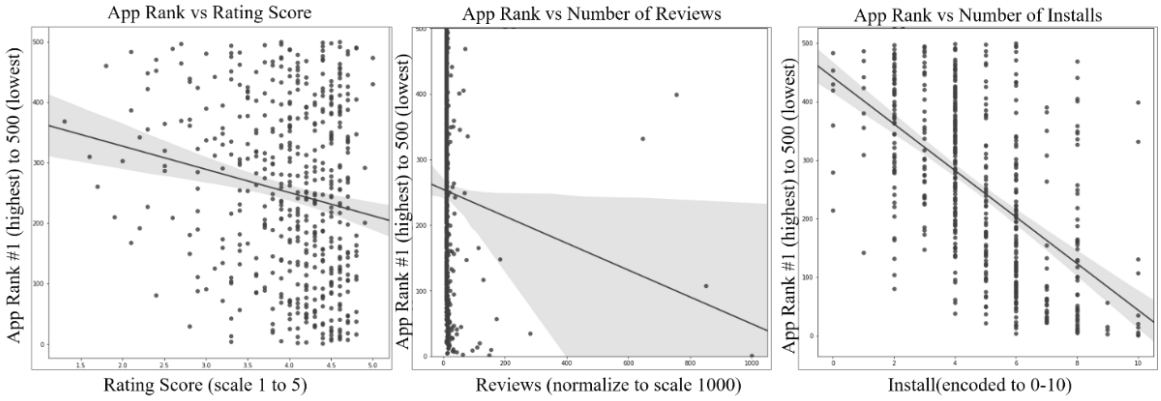

Figure 5.4. The association between App Rank and external factors (rating, number of reviews, and installs).

Figure 5.5 represents the feature importance score (F) that is named according to their index from f_0 to f_2. Here, the number of reviews and rating have a higher importance score than the number of installs. While the rating score is an influential factor for the Google PlayStore ranking

so the disparity between these two ranking schemes is not that high. We have picked an App, *App1*, from the Shopping category that has opposite positions - *App1* has a rank of 4 out of 5 based on the user ratings; while its rank is 2 based on the Google PlayStore rank. 45% of the total reviews for the *App1* contain a below 4 rating. On the other hand, the number of installs (more than 5 Million) and the number of reviews (91,857) for this App are relatively higher than for the other. So, these factors when combined with other Google PlayStore ranking factors give the App a higher rank.

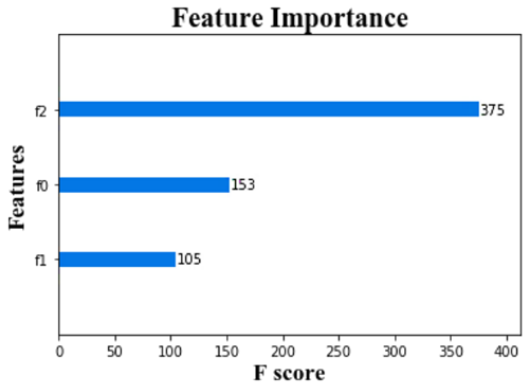

Figure 5.5. Feature Importance Bar Chart - rating (f_0), installs (f_1) and number of reviews (f_2).

4) $\omega_X^{\oplus(S_{DT},S_{IT})}$ and Google PlayStore Rank. Prior scenarios have indicated that rankings based on any partial evidence result in significantly distinct orderings. Thus, there is a need to combine direct and indirect trust-based evidence to provide a comprehensive ranking scheme. To achieve this, we combine the direct trust ($\omega_X^{\oplus S_{DT}}$) tuple and the indirect trust ($\omega_X^{\oplus S_{IT}}$) tuple using a weighted consensus operator (introduced in formula (23) to (28)) by giving a higher weight to the $\omega_X^{\oplus S_{DT}}$ (addressed in Section 4.3.3). In our experiments, we have used the following weight combinations: 30% weight to $\omega_X^{\oplus S_{IT}}$ and 70% weight to $\omega_X^{\oplus S_{DT}}$; equal weight to both $\omega_X^{\oplus S_{IT}}$ and $\omega_X^{\oplus S_{DT}}$; 70% weight to $\omega_X^{\oplus S_{IT}}$ and 30% weight to $\omega_X^{\oplus S_{DT}}$. As seen from Table 5.5, the distance between the E-SERS rankings and Google PlayStore rank varies from 30% to 50% for different categories based on the weights we have assigned - and it is lesser than other distances. A higher weight for $\omega_X^{\oplus S_{IT}}$ reduces the distance with the Google PlayStore rank, whereas a lower weight for $\omega_X^{\oplus S_{IT}}$ increases the distance.

As described earlier, the star rating or its combination with the external narratives is neither reasonable nor sufficient to assess an App. When we deal with the direct trust-based evidence, there is no possibility of any ambiguity as it entirely focuses on the internal (functional) aspects of an App — albeit, there is a likelihood of returning false positives in such analysis. Therefore, when we combine both direct and indirect trust-based evidence and generate a rank list, it not only considers all aspects but also supports in overcoming any idiosyncrasies associated with user reviews. Such a scenario is illustrated with the help of *App2* in the Shopping category. *App2* is one of the top Apps ranked by the Google PlayStore. Users' reviews and the rating score depict a similar scenario, where approximately 78% of reviews are rated 4 stars or above. Based on the review sentiment, 70% reviews reflect positive sentiment for *App2*. E-SERS assigns a lower rank to the *App2* when it is evaluated based on direct trust-based evidence. During the internal evidence analysis, we found severe security vulnerabilities for this App. Through the security flaws investigation, we found that the data leak associated with *App2* deals with the Dangerous permission access (such as *ACCESS_FINE_LOCATION*, *ACCESS_COARSE_LOCATION*) and these sensitive data are written to *SMS_MMS*. Thereby, highlighting the fact that user reviews many times fail to a grasp the real view of an App and anyone relying on only reviews or star scores may regret their selection.

5.5 Web Prototype

This section will discuss our web prototype, which has been created for user involvement and will be released to the community at large. The prototype is based on Flask Web Framework [120] and is written in Python. Following inputs are expected from users to use this prototype:

- An App's URL in the Google PlayStore
- Weights for internal and external views; both views are equally weighted by default – users can adjust the weight based on their preferences.
- Once a user provides both these inputs, then in an automated way, the prototype obtains the package name and using the third-party APK downloaders (e.g., APKPure [89]) and the package name downloads the Apps' APK file. using The APK file is used as an input to compute the DTA of that App. Similarly, to compute ITA, we have implemented an automated Web crawler that fetches the newest and relevant reviews and other details of an App (such as average rating score, reviewer id, creation date, number of likes, and the

corresponding rating) from the Google PlayStore. The backend of the prototype then calculates the E-SERS score based on the direct and indirect trust attributes. Apart from the E-SERS score, the prototype also displays following details about an App and a set of Apps using a GUI: The rating scores computed based on the internal and external views; are visually presented using the trust meter.

- The computed E-SERS score for an App is classified into three different categories: Distrusted, Neutral, and Trusted. A score below 2.5 is considered as "Distrusted", between 2.5 to 3.5 is indicted as "Neutral", and between 3.5 to 5 is labeled as "Trusted".
    - To determine these ranges, we have used VirusShare [96] and Drebin [97] [121] [122] datasets which are the repository of malware samples of Apps.
    - For these datasets, we observed that a major percentage (approximately 74%) of Apps has a trust rating score below 2.5.
- The top 5 most occurring positive and negative words in the reviews are also indicated.

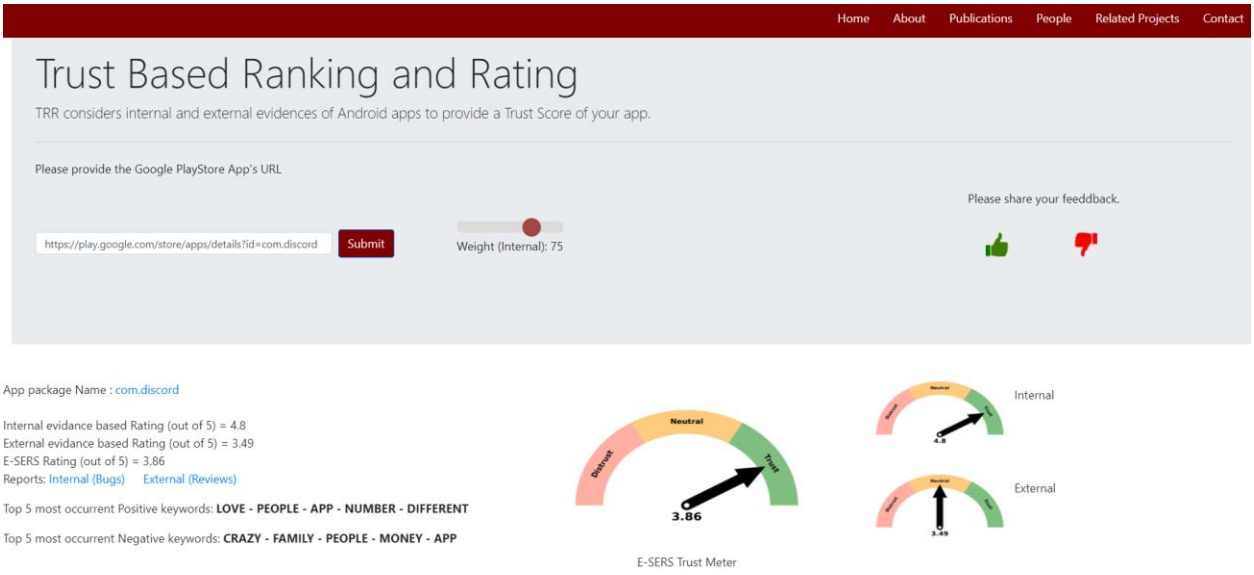

Figure 5.6. E-SERS Web Prototype.

- A Bug report generated by FindBugs for a particular App is also displayed.

The prototype is deployed at *rankings.cs.iupui.edu* – and is made accessible to the computing community. In addition, the prototype allows users to provide their feedback for future enhancements. This feedback can be provided via the 'Like/Dislike' feature. The screenshot of the web prototype is given in Figure 5.6.

In the following section, we have discussed the performance of the prototype measured as the average execution time required to handle a request.

Initially, all the necessary methods were implemented following a sequential paradigm. In that scenario, the required time to download the APK file and fetch reviews for an App on an average was 130.61sec. The average time required to compute the trust score of an App was 233.1sec. Hence, the end-to-end execution time after initiating a request until receiving a response was on average 6.1mins. In order to reduce this end-to-end time, we parallelized parts of our code using the multiprocessor [123] module, a Python package that supports concurrency. The following tasks were carried out in parallel:

- We generated both direct and indirect trust artifacts in parallel. In the input collection phase, as indicated earlier, we collect three different kinds of data for an App – the APK file, newest reviews, and most helpful reviews. We executed these tasks simultaneously.
- We also performed computations of the DTA- and ITA-based trust scores concurrently.

The execution times, after these concurrent executions, were reduced to 85.03sec (for artifact generation), and 161.2sec (for the trust score computations). Hence, after parallel executions, the end-to-end time decreased to 4.1mins – 32.7% improvement over the non-parallelized version.

CHAPTER 6. DETECTION OF MALWARE APPS USING DATA FLOW FEATURES

Once a user has selected a specific set of Apps from available choices, they may want to combine these selected Apps to form a composed system using the E-SERS. Before we discuss the composition models based on the E-SERS technique, in this chapter, we introduce an internal evidence-based malicious Apps detection technique that uses data flow features. This technique is later applied, in the next chapter, in the proposed trust-aware service composition model.

6.1 Overview

We propose a technique for categorizing Apps into malicious or benign. As described in Figure 6.1, after an App is given as an input to the FlowDroid, FlowDroid produces all identified data flows from sources to sinks. Based on the data flow details, a feature matrix is generated. These features are then passed to train the classifiers and predict whether an App is malicious or benign.

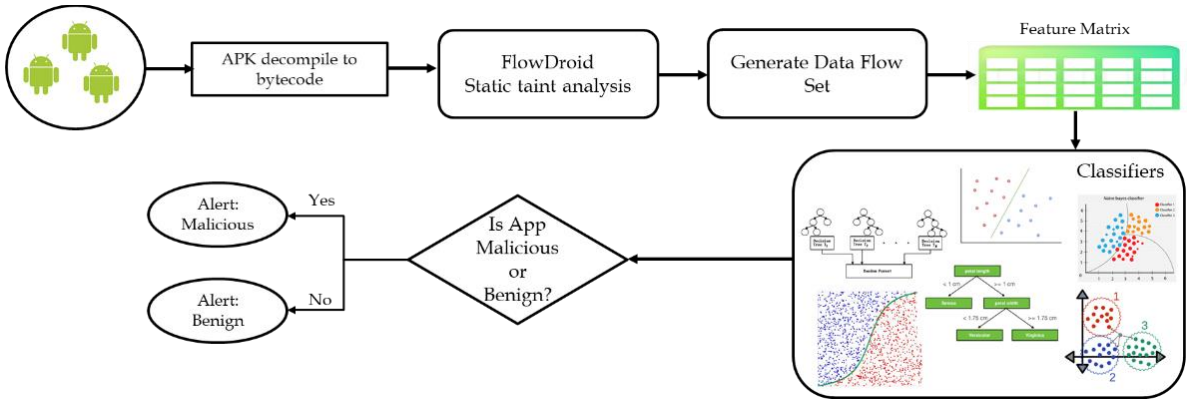

Figure 6.1. Overview of Malware App Detection Framework

6.2 Feature Extraction

The selection of features is always a critical factor to detect malware Apps. In this work, as indicated above, we extract features from Apps with a static taint analysis tool - FlowDroid. As we want to focus on security vulnerabilities, with the help of FlowDroid we identify all the sensitive sources and sink pairs that are responsible for leaking the critical information. On the

datasets that we have used in the experimentation, FlowDroid extracted 28,170 features (described in Sec. 6.4.1). Each feature is the reported Source and Sink APIs. For each App, the data is expressed by a binary vector. In the feature matrix, bit 1 represents that FlowDroid reports the particular API, and bit 0 indicates as the particular API is not reported.

6.3 Machine Learning Classification Algorithms for Detection

The App classification problem is a binary classification task resulting in two class labels – malicious and benign. Here based on the features, the Apps are labeled by either 1 or 0. Here, label '0' represents a benign App, and label '1' represents the App is a Malware App. We have picked the six most popular algorithms used for binary classification problems, which are given below. Here, the Support vector machine is particularly designed for binary classification problems and does not support multi-class classification problems.

1) Support Vector Machine
2) K-Nearest Neighbors
3) Logistic Regression
4) Naïve Bays - Gaussian Naive Bayes
5) Decision Tree
6) Random Forest

6.3.1 Support Vector Machine

Support Vector Machine (SVM) [124], is an elegant and powerful machine learning algorithm that can be used for classification and regressions fields. However, it is mostly used for classification tasks. The objective of the SVM is to find a hyperplane in an N-dimensional space, where N is the total number of features that clearly categorizes the data points.

6.3.2 K-Nearest Neighbors

K-Nearest Neighbors (KNN) [125], is a supervised machine learning algorithm that can be used to solve classification and regressions problems. The baseline assumption of this algorithm is that similar things occur close to each other, which captures the idea of similarity. It works by

determining the distances between a sample and all instances in the datasets. Then, it chooses K-nearest samples and uses the majority ensemble technique to predict which group to fit the sample.

6.3.3 Logistic Regression

Logistic Regression (LR) [126] a predictive analysis algorithm that models the probability of the class between 0 and 1, with a sum of one. It is named based on the core method that is used, the logistic function. The logistic function is also called as 'Sigmoid function'. Logistic regression supports both binary and multi-class classification.

6.3.4 Naïve Bays

Naïve Bayes (NB) [127], is a collection of classification algorithms based on Bayes' theorem. It is used for both binary and multi-class classification. This algorithm can be extended to real-valued attributes, most commonly by assuming a Gaussian distribution. The extension is called Gaussian Naive Bayes (GNB). For GNB, it is only required to calculate the input variable's mean and standard deviation for each class value.

6.3.5 Decision Tree

Decision Tree (DT) [128], is a supervised machine learning algorithm where the data is divided continuously according to a definite parameter. The tree can be characterized by two factors: decision nodes and leaves. It is a flowchart-like structure where decision nodes describe a test on a feature and leaves represent the data label. This method is used for both classification and regression problems.

6.3.6 Random Forest

Random Forest (RF) [129], is a supervised machine learning algorithm. The RF method executes in two phases; first, it creates a random forest and then makes a prediction from the first phase's RF classifier. The difference between DT and RF algorithms is that in RF, the process of

retrieving the root node and dividing the feature nodes runs randomly. This method is used for both classification and regression problems.

6.4 Evaluation

6.4.1 Datasets

We utilized the same Apps dataset that we have applied to quantify the risk assessment matrix (Chapter 4, Sec. 4.3.1.2). We have also combined a new benign Apps dataset so that the dataset contains both benign and malware Apps to validate our classification model. Benign and malware Apps were collected from the following source:

- KuafuDet [130] is a repository of benign samples of Apps. We have used the year 2016 and 2017 datasets; each contains 600 Apps.
- VirusShare [96] is a repository of malware samples of Apps. We have used the 2018 dataset, which contains 1000 samples.
- Drebin [97] [121] [122] is a repository that contains malware samples from 179 different malware families. Among the six chunks of the dataset, we used only chunks 4 and 5; chunk 4 contains 1000 samples, and chunk 5 contains 555 samples of Apps.

In our dataset, the benign Apps are labeled by 0, and the malicious Apps are labeled by 1. From the dataset, we randomly picked 80% of samples as training data, and the remaining 20% of the samples are utilized as test data.

6.4.2 Parameter of Training Model

In the empirical evaluation, we have used a computer with an Intel(R) Core (TM) i7 CPU and RAM of 16G. We utilize scikit-learn [131], a free software machine learning tool written in Python programming language.

For SVM, instead of the default *kernel* function, which is *'rbf'*, the kernel function is set to *'linear'*. We have manually specified the different kernel functions to search which parameter provides the best result. For this, we have found '*linear'* kernel function performs better than '*rbf*'. Similarly, we have found that the best possible value for the regularization parameter, *C*, is 0.25;

however, *C's* default value is 1.0. By tuning the hyperparameter *C*, the accuracy has elevated from 85% to 88%.

As for Naïve Bayes, we have picked the *GaussianNB* model in our experiments. For *DecisionTreeClassifier*, we have carried the experiment with default settings. As many of the researchers point out, in most situations, the quality of splitting and choice of splitting criteria will not cause a significant difference in the DT performance.

The number of neighbors is an essential parameter for KNN. We set the *n_neighbors* to 3; by default, the value is 5. To determine the *n_neighbors*, we have plotted a graph between accuracy rate and *n_neighbors* denoting values in a range from 1 to 10. Then we chose the *n_neighbors* value as the one having a maximum accuracy rate, which is 3.

For an imbalanced dataset, the *class_weight* is a crucial parameter. In our dataset, the number of benign Apps is much lesser than the number of the malware Apps. Hence, we have tuned the *class_weight* for both LR and RF by applying the following formulas:

$$W_{BN} = \textit{total samples / (total num of class * total num of benign Apps)} \quad (35)$$

$$W_{MW} = \textit{total samples / (total num of class * total num of malware Apps)} \quad (36)$$

where *total num of class* value is 2, either benign or malware class. Here, W_{BN} represents the weight for benign Apps, and W_{MW} represents the weight for malware Apps. After the calculation, the value of W_{BN} and W_{MW} are 1.33 and 0.8.

The *max_depth* and *n_estimators* are two other critical parameters for RF. Here, *max_depth* represents the depth of each tree, and *n_estimators* indicate the total number of trees. Usually, a deep tree and a high number of trees are better to train the data to capture more information about the data. However, these two values can significantly slow down the process of training of the model. Therefore, for both of the parameters, we need to identify the optimal values. The default value for *max_depth* is None and for *n_estimators* is 100. Initially, we use the default value of *n_estimators* and experimented with different variations of *max_depth* as shown in Table 6.1. We present, in Table 6.1, the AUC (Area Under Curve) [132] score for both with *class_weight* and without *class_weight* cases. The AUC measures the tradeoff between the false positive rate and the true positive rate. The AUC score ranges from 0 to 1; 0 indicates 100%, wrong predictions, and 1 represents 100% correct predictions.

Table 6.1. AUC Score for different *max_depth* variations

Variation of *max_depth*	**AUC Score without *class_weight***	**AUC Score with *class_weight***
max_depth = 1, *n_estimators* = 100	0.56	0.73
max_depth = 2, *n_estimators* = 100	0.67	0.76
max_depth = 4, *n_estimators* = 100	0.71	0.78
max_depth = 8, *n_estimators* = 100	0.75	0.81
max_depth = 16, *n_estimators* = 100	0.81	0.85
max_depth = 32, *n_estimators* = 100	0.83	0.86
max_depth = 64, *n_estimators* = 100	0.852	0.913
max_depth = 100, *n_estimators* = 100	0.857	0.907
max_depth = 200, *n_estimators* = 100	0.86	0.913

From Table 6.1, we can infer that for our data the optimal value for *max_depth* is 64 with *class_weight*. The visualization of *max_depth* variation vs. AUC score is presented in Figure 6.2. Here, the blue line resents the outcomes for the training data, and the red line presents the outcome for the test data.

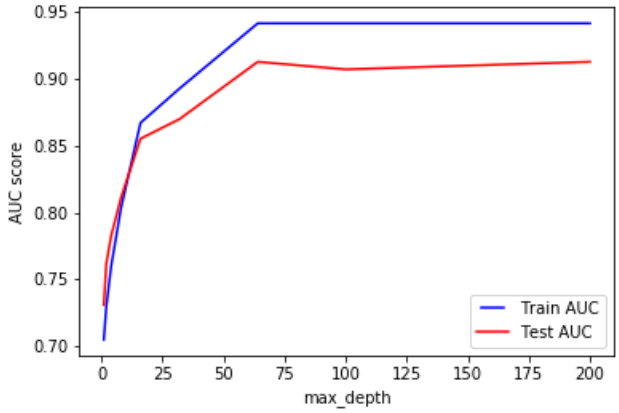

Figure 6.2. AUC Score vs *max_depth* with *class_weight*

After these explorations, we re-ran the experiment with the *max_depth* value set to 64 and with different variations of *n_estimators*, as given in Table 6.2.

Table 6.2. AUC Score for different *n_estimators* variations

Variation of *n_estimators*	**AUC Score without *class_weight***	**AUC Score with *class_weight***
max_depth = 64, *n_estimators* = 1	0.77	0.81
max_depth = 64, *n_estimators* = 2	0.82	0.86
max_depth = 64, *n_estimators* = 4	0.89	0.86
max_depth = 64, *n_estimators* = 8	0.8625	0.89
max_depth = 64, *n_estimators* = 16	0.8662	0.9158
max_depth = 64, *n_estimators* = 32	0.8664	0.9150
max_depth = 64, *n_estimators* = 64	0.8704	0.9104
max_depth = 64, *n_estimators* = 100	0.868	0.9126
max_depth = 64, *n_estimators* = 00	0.866	0.917

From Table 6.2, we inferred that for our data the optimal value for *n_estimators* is 16 with *class_weight*. Increasing the *n_estimators* beyond 16 will decrease the test performance without triggering a notable improvement in the accuracy rate. The visualization of *n_estimators* variation vs. AUC score is presented in Figure 6.3. Here, the blue line resents the outcomes for train data, and the red line presents the outcome for test data.

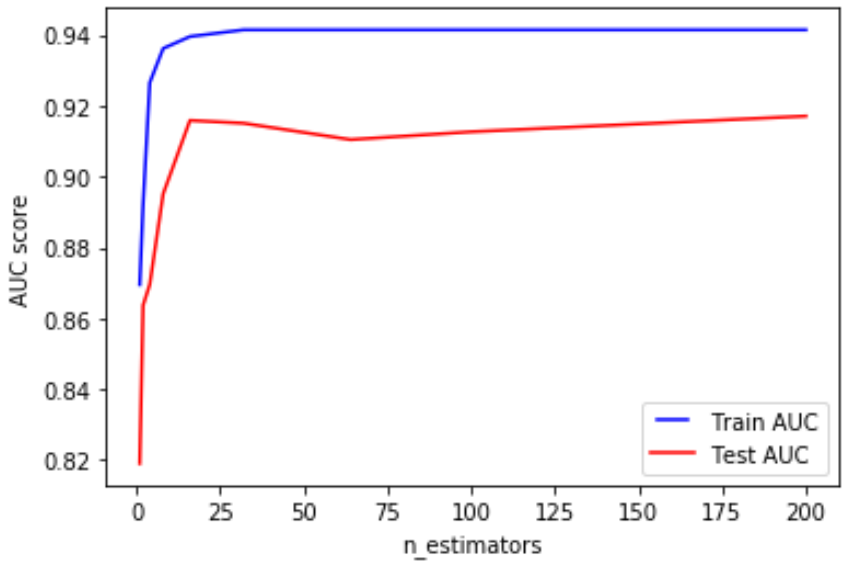

Figure 6.3. AUC Score vs *n_estimators* with *class_weight*

6.5 Experimental Result and Analysis

The following performance metrics are used to measure the results from these six different classifiers.

- True Positive Rate (TPR): It indicates the rate of Benign Apps that are successfully detected. The formula for TPR is:

$$TPR = \frac{TP}{TP+FN} \tag{37}$$

- False Positive Rate (FPR): It indicates the rate of Benign Apps that are not successfully detected. The formula for FPR is:

$$FPR = \frac{TP}{TP+FN} \tag{38}$$

- True Negative Rate (TNR): It indicates the rate of Malware Apps that are successfully detected. The formula for TNR is:

$$TNR = \frac{TN}{TN+FP} \tag{39}$$

- False Negative Rate (FNR): It indicates the rate of Malware Apps that are not successfully detected. The formula for FNR is:

$$FNR = \frac{FP}{TN+FP} \tag{40}$$

- Accuracy: It indicates the ratio of correctly identified Apps (either Benign or Malware) to the total number of tested Apps. The formula for Accuracy is:

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \tag{41}$$

- Precision: It indicates the ratio of correctly predicted Benign Apps to the total number of predicted Benign Apps. The formula for Precision is:

$$Precision = \frac{TP}{TP+FP} \tag{42}$$

- Recall: It indicates the ratio of correctly predicted Benign Apps to the total number of actual Benign Apps. The formula for Recall is:

$$Recall = \frac{TP}{TP+FN} \tag{43}$$

- F1-Score: It is the weighted average of Precision and Recall. F1 score is a better measure for uneven class distribution so we consider F1-score in our performance metrics. The formula for F1-Score is:

$$F1\text{-}Score = 2 * \frac{Precision * Recall}{Precision + Recall} \quad (44)$$

As earlier discussed in Sec. 6.4.1, we have split the dataset 80%-20%, where 80% data are used to train the classifier models and the remaining 20% of the data is used for testing. The performance of the six classifiers over test dataset is shown in Table 6.3.

Table 6.3. Performance measurement of different classification algorithms

Classifier →	**LR**	**KNN**	**GNB**	**RF**	**DT**	**SVM**
TPR (%)	96%	70%	64%	99%	90%	92%
FPR (%)	14%	2%	3%	15%	15%	14%
TNR (%)	86%	98%	97%	85%	85%	86%
FNR (%)	4%	30%	36%	1%	10%	8%
Accuracy (%)	90%	88%	85%	90%	87%	88%
Precision (%)	79%	95%	92%	78%	77%	78%
Recall (%)	96%	70%	64%	99%	90%	92%
F1-Score (%)	87%	81%	76%	87%	83%	84%

Table 6.3 shows that the accuracy and F1-score of LR and RF, were higher than the other classification methods. In terms of accuracy, the GNB classifier performed worse than others as the rate to classify the benign Apps is really low (64%). However, the TNR rate (%) for GNB and KNN classifiers was higher than others, indicating a successful rate to detect the malware Apps. The visual representation of different classification results in terms of accuracy, precision, recall, and F1-score is shown in Figure 6.4.

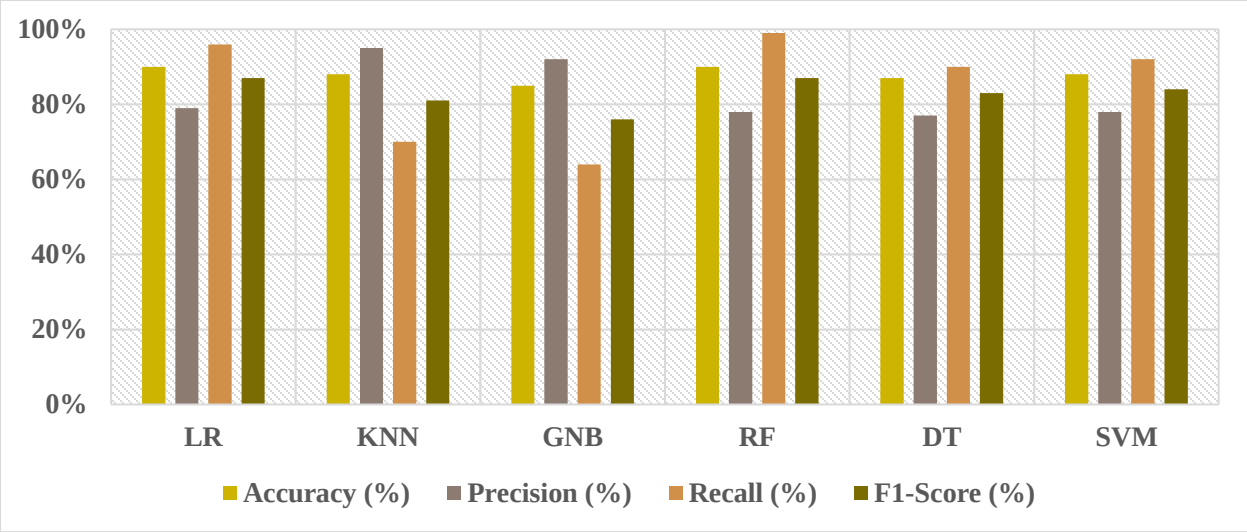

Figure 6.4. Performance comparison of different Classification models.

We conducted 10-fold cross-validation with three cross-validator repetitions to evaluate these six methods, to reduce the problems such as selection bias or overfitting. The box-and-whisker plot, indicating the accuracy of each classifier, is presented in Figure 6.5.

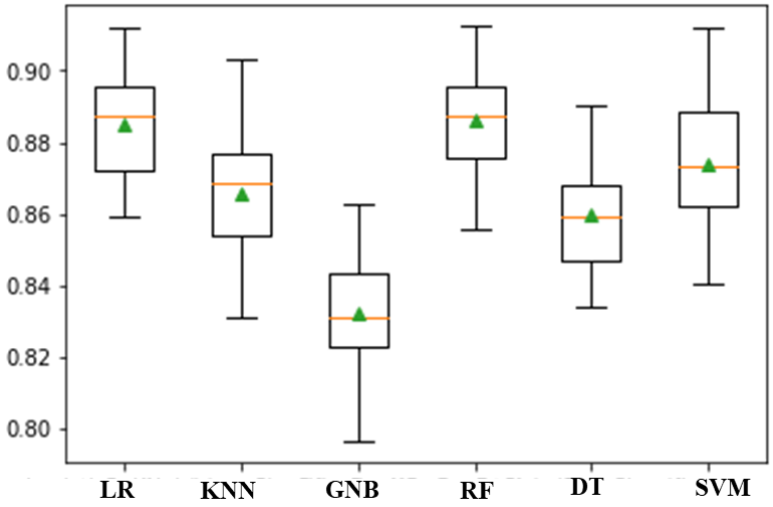

Figure 6.5. Box-and-whisker plot of accuracies for Classification models

From Figure 6.5, it is clear that LR and RF perform better on average than SVM, KNN, DT, and GNB. The execution time for each classifier is given in Table 6.4. The RF classifier performs much faster than the LR model. Despite that, the detection accuracies remain similar for both LR and RF algorithms.

Table 6.4. Execution time of different classifiers

Classifier →	SVM	KNN	LR	GNB	DT	RF
Execution time (sec)	144.64	105.77	24.03	2.15	9.6	6.52

This chapter has applied and compared six machine learning classification algorithms, LR, KNN, GNB, RF, DT, and SVM, to categorize the Android Apps into either benign or malware. The evaluation results based on 10-fold cross-validation reveal the following findings:

- The LR and RF classifiers provide a similar and higher accuracy rate than others, which is 88%.
- The KNN and GNB classifiers performed inadequately while classifying the benign Apps. The TPR for the KNN and GNB classifiers is 70% and 64%, respectively. Additionally, in the context of the TNR, both KNN and GNB techniques outperforms other classifiers. The TNR value indicates that these two models are much successful in identifying the malware Apps. Among these two techniques, the accuracy of KNN (86%) is 3% higher than GNB (83%), but for KNN, the execution time is remarkably high. So, it is a clear tradeoff between task performance and accuracy.
- The accuracy for the SVM and DT classifiers is 87% and 85% respectively, but there is a significant execution time difference between these two approaches. In terms of execution time, the SVM approach is one of the slowest classification techniques on our data set. Again, similar to the previous case, this is the tradeoff between task performance and accuracy.
- Based on the results, we infer that the GNB could be the best model to detect malware Apps at with a reduced execution time. Again, if we want to identify both benign and malware Apps successfully and need a short execution time, RF is the best classification model among these six classifiers.

Therefore, we have adopted RF to formulate the classifier model by considering the accuracy and execution time performance. This classifier model was then imposed on our proposed trust-aware composition model (described in the next chapter). The RF classifier model requires an average of 2.48 sec to return the probability values for an App belonging to either the benign or the malicious app category.

CHAPTER 7. TRUST-AWARE SERVICE COMPOSITION

Automatic service composition is a promising solution to create quality-aware distributed software systems. In this chapter, as the logical next step of our research, we tackle the challenge of trust-aware service composition in mobile ecosystems. In this chapter, "service", refers to any mobile App that is publicly available on software marketplaces. Due to the growing number of Apps and their possible combinations, we need an automatic composition technique to create a distributed system, out of a set of selected Apps, for achieving a particular function. Most of the service composition models are based on the QoS parameters, while our goal is to propose a trust-aware composition model that is based on comprehensive views of the individual Apps. We use the combined trust score as the desired attribute for any ensemble of Apps – in previous chapters, we have already discussed the quantification of the trust score, using the E-SERS approach, for an individual App.

Two prevalent composition models, that use QoS parameters for composition, are presented in the next section. After that, we describe the proposed trust-aware model. These models have been have evaluated, later on in this chapter, using the metrics of the average star rating and the trust scores.

7.1 Prevalent Composition Models

Each model presented below consists of two phases: in the first phase, the model generates possible combinations from the existing services that are available and in the second phase, it decides the final desired service composition sequence. For these two models, we use the trust score as the QoS parameter for selecting the desired composition model.

1. **Mean-Max composition model** [133]**:** In this model, all possible combinations are generated with their mean trust score values. Then only the combination that has the highest mean value will be the final trusted service composition sequence. In order to illustrate the Mean-Max service composition model, below we present an example. Assuming that, we need to compose a system by selecting one alternative each from three different categories of services – called "Service 1", "Service 2" and "Service 3". Let us further assume that for the 'Service 1' category, there is only one App option (A_{11} – trust score of 4.5); for the 'Service 2' category, there are three possible App options (A_{21}, A_{22}, A_{23} – trust scores of 4, 2, and 4.5 respectively); and finally, for the 'Service 3' category, there are two App options

(A_{31}, A_{32} – trust scores of 4 and 5 respectively). Then in the first phase, all possible combinations, for the above example, with means trust scores are as follows:

Table 7.1. Example scenario of Mean-Max Composition model.

Service 1 *(trust score out of 5)*	**Service 2** *(trust score out of 5)*	**Service 3** *(trust score out of 5)*
A_{11} (4.5)	A_{21} (4) A_{22} (2) A_{23} (4.5)	A_{31} (4) A_{32} (5)

Binding Schemes			**Mean of Trust Rating Score**
Service 1	**Service 2**	**Service 3**	
A_{11}	A_{21}	A_{31}	4.2
A_{11}	A_{21}	A_{32}	4.5
A_{11}	A_{22}	A_{31}	3.5
A_{11}	A_{22}	A_{32}	3.8
A_{11}	A_{23}	A_{31}	4.3
A_{11}	A_{23}	A_{32}	4.6

After phase 1, as indicated in Table 7.1, we have all the possible combinations of the services and associated mean rating scores based on individual trust scores. In the second phase of this model, the $A_{11} – A_{23} – A_{32}$ sequence will be selected as the final composition based on the maximum of the mean trust rating scores.

The major disadvantage of this model is that the final sequence may include malicious service with other high-rated services.

2. **Mean-Random composition Model:** In this model, during the first phase, all possible combinations are formed with their mean trust score values. Then, in the second phase, from the all possible combinations one combination is randomly picked. That random combination will be the final trusted service composition sequence. The execution will be faster than the Mean-Max model, however, the chance of selecting the optimal combination is less due to the randomized nature of selection.

7.2 Trust-aware Composition Model

In this section, we present the proposed trust-aware composition model. In this model, services are picked based on their trust scores. However, before proceeding with the composition, we have added a service filter that removes the individual services according to the different filtering conditions (perhaps, user-defined). One specific criterion that we propose for such a filter is based on the probability of a service being classified into the benign category. The decision about whether a service is benign or malicious is decided by the RF-based classification model discussed in the previous chapter. A 70% or higher percentage probability of being in the benign category is considered appropriate for a service to participate in the service composition. This boundary value (70% here) can be customized based on user preferences. This constraint will reduce of the number of potential combinations of services. After that, we can apply any of the two models mentioned earlier. Figure 7.1 shows the architecture of trust-aware composition framework.

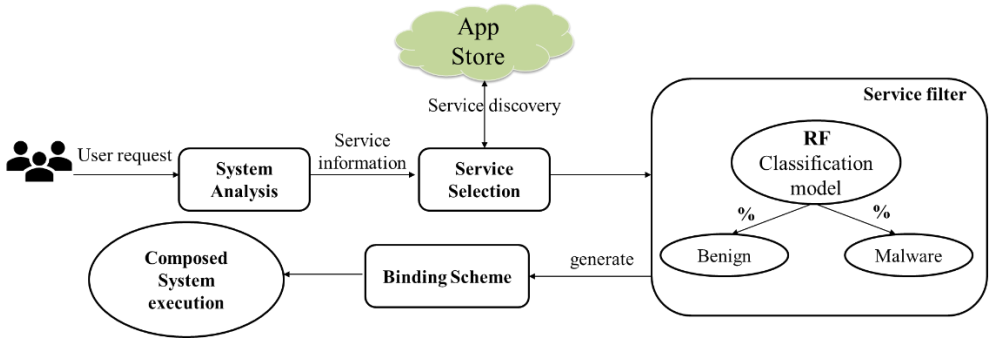

Figure 7.1. Architecture of Trust-aware composition framework (lower granularity)

7.3 Validation

The Proposed model is empirically validated using the following two case studies from different domains – these studies were selected based on the Apps availability and past literature [74]:

1) Online document arrangements system; from Productivity domain
2) Weather forecast from IP address [74]; from Weather and Tool domain

For each case study, the number of services and their interaction patterns may vary (such as sequence, parallel, etc.). During the service selection process, we have considered two additional

factors: an availability of reasonable number (as a minimum 1K) of user reviews to for the services and the feasibility of obtaining the bytecode of the services.

7.3.1 Online Document Arrangements System

The first case study contains a composite system called "Online Document Arrangement System" (ODAS) and is comprised of the following category of services: Virus scanner (S_1), Grammar check (S_2), PDF converter (S_3), and FAX (S_4). The composite ODAS is presented in Figure 7.2. In this ODAS, a document file is supplied as input to S_1 and S_2 services. If the input file passes the checks provided by both these services, then it is supplied to S3. After the successful execution of S_3 the resultant output file is sent as an input to the S_4 service. Then S_4 conducts the final step in the execution sequence.

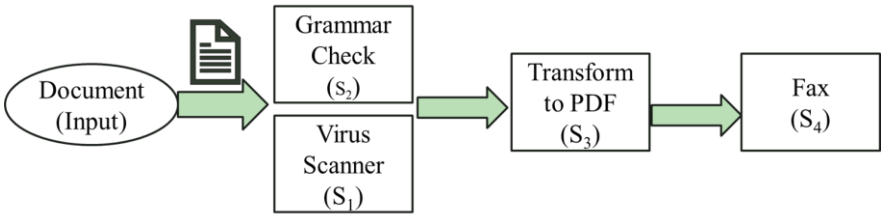

Figure 7.2. Abstract composite process - Online document arrangements system (ODAS).

In this composed ODAS, as seen in Figure 7-2, four types of services interact to achieve the desired outcome. The ODAS consists of both sequential and parallel interaction patterns. Again, for each service (S_i) there would be many available alternatives. For this case study, we have picked five different Apps (from Google PlayStore) for each service category – a total of 20 services. To compose the ODAS, using these 20 choices, the total number of possible combinations (C) are:

$$C = \prod_{i=1}^{n} X_{Si} \tag{45}$$

Here, n is the total number of services required to compose the ODAS and X_{Si} represents the number of available alternatives for each service S_i. For the OADS, n is 4 and X_{Si} for each service is 5. Hence, the total number of possible combinations, C, is 625. This indicates the explosion of alternatives associated with exhaustive combinations for any composed system. Hence, it is necessary to prune the infeasible alternatives and we are using the trust scores assigned to the services by E-SERS to prune the infeasible combinations.

Before identifying all possible combinations for the ODAS using these 20 services, we first check if a given service belongs to the benign or malicious category. This classification is based

on the service's data flow features. If a service belongs to the benign category, then only we apply the E-SERS approach and compute the trust score for that service. Hence, in the trust-aware composition model, there are two proposed levels of granularity:

1) high level – where only the trust scores for each service, computed by E-SERS, are used in applying the abovementioned two composition models, and
2) low-level – where before computing the trust score of a service, using E-SERS, the probability of deciding if a service is malicious or benign is calculated. This results in a small set of alternatives. On this small set, the abovementioned two composition models are applied.

For each service in the OADS case study, the trust score computed by the E-SERS is presented in Table 7.2.

Table 7.2. E-SERS score of each service in ODAS.

Virus scanner (S_1) *(trust score out of 5)*	**Grammar check (S_2)** *(trust score out of 5)*	**PDF converter (S_3)** *(trust score out of 5)*	**FAX (S_4)** *(trust score out of 5)*
S_{11} (2.6)	S_{21} (3.6)	S_{31} (3.4)	S_{41} (3.9)
S_{12} (3)	S_{22} (3.3)	S_{32} (3.7)	S_{42} (3.2)
S_{13} (3.9)	S_{23} (3.8)	S_{33} (4.1)	S_{43} (4.2)
S_{14} (3.4)	S_{24} (3.3)	S_{34} (2.4)	S_{44} (3.6)
S_{15} (3.1)	S_{25} (2.7)	S_{35} (3.1)	S_{45} (3.4)

1) High-level Trust-aware Model. After the computation of the trust scores, as indicated above, we apply the mean-max and mean-random composition models for the OADS, and the resultant service sequences are indicated below:

- Total possible combinations = 625
- Mean-Max composition model.
  - Binding scheme (higher granularity): S_{13} -> S_{23} -> S_{33} -> S_{43}
  - Composite trust score is 3.2.
  - Execution time: 2.9ms
- Mean-Random composition model.
  - Binding scheme (higher granularity): S_{11} -> S_{21} -> S_{31} -> S_{44}

- Composite trust score is 2.64.
- Execution time: 1.9ms

Here the execution time contains the time required to compose the system including the time needed to compute the possible alternatives. As seen from above, there is a tradeoff between the two possible system configurations for the OADS. If the composite trust score is the primary objective then the solution provided by the Mean-Max composition model is a better solution; on the other hand, if the execution time is the primary objective then the solution provided by the Mean-Random composition model is a better solution.

2) Low-level Trust-aware Model. For the Low-level Trust-aware Model, as indicated earlier, we first determine the probability of assigning a given service to the benign (B) or malicious (M) class. The probability values of each service, in the OADS, are given in Table 7.3.

Table 7.3. Service filter attribute value of each service in ODAS.

Virus scanner (S_1) (B% | M%)	**Grammar check (S_2)** (B% | M%)	**PDF converter (S_3)** (B% | M%)	**FAX (S_4)** (B% | M%)
S_{11} (68 | 32)	S_{21} (60 | 40)	S_{31} (93 | 7)	S_{41} (87 | 13)
S_{12} (39 | 61)	S_{22} (68 | 32)	S_{32} (63 | 37)	S_{42} (100 | 0)
S_{13} (93 | 7)	S_{23} (87 | 13)	S_{33} (100 | 0)	S_{43} (80 | 20)
S_{14} (82 | 18)	S_{24} (82 | 18)	S_{34} (75 | 25)	S_{44} (68 | 32)
S_{15} (56 | 44)	S_{25} (58 | 42)	S_{35} (90 | 10)	S_{45} (68 | 32)

As we have mentioned earlier, a service with 70% or higher percentage probability of being in the benign category will only be considered for service composition. Based on this standard, we filtered out two services (S_{13}, S_{14}) from S_1, two services from S_2 (S_{23}, S_{24}), 4 services from S_3 (S_{31}, S_{33}, S_{34}, S_{35}) and 3 services from S_4 (S_{41}, S_{42}, S_{43}). After performing these initial pruning operations, we obtain the following results for Mean-Max composition model – since, the Mean-random composition model selects a sequence randomly, it will not select a sequence that is more optimal than the one selected by the Mean-Max model and hence, it is not listed below:

- Total possible combinations = 48
- Trust aware composition model with Mean-Max model.
  - Optimal Binding scheme (lower granularity): S_{13} -> S_{23} -> S_{33} -> S_{43}
  - Composite trust score is 3.2.

- Execution time: 0.9ms

7.3.2 Weather forecast from IP Address

The second case study contains a composite system called "Weather forecast from IP address" (WFIP) and is comprised of services *IpToCity* (S_1), *CityToZip* (S_2), and *ZipToWeather* (S_3). The composite WFIP is presented in Figure 7.3. In this WFIP, an *IP* address is provided as input to S_1 service. S_1 generates the corresponding *City* name and then it is supplied to S_2. After the successful execution of S_2 the consequential *zipcode* is provided to the S_3 service. Then S_3 performs the final step in the execution sequence and return the weather forecast.

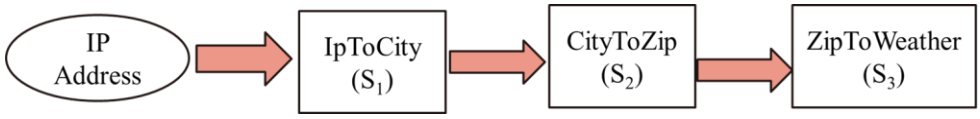

Figure 7.3. Abstract composite process - Weather forecast from IP Address (WFIP).

In this composed WFIP, three services cooperate to achieve the desired outcome by following sequential interaction patterns. Similar to the previous case study, we have picked five different Apps from Google PlayStore for each service category – a total of 15 services. For each service in the WFIP case study, the trust score computed by the E-SERS is indicated in Table 7.4.

Table 7.4. E-SERS score of each service in WFIP.

IpToCity (S_1) *(trust score out of 5)*	**CityToZip (S_2)** *(trust score out of 5)*	**ZipToWeather (S_3)** *(trust score out of 5)*
S_{11} (3.66)	S_{21} (3.43)	S_{31} (4.21)
S_{12} (3.95)	S_{22} (3.95)	S_{32} (3.43)
S_{13} (3.83)	S_{23} (4.52)	S_{33} (3.3)
S_{14} (3.86)	S_{24} (2.51)	S_{34} (3.92)
S_{15} (4.46)	S_{25} (4.29)	S_{35} (2.58)

1) High-level Trust-aware Model. After the computation of the trust scores, as indicated above, we apply the Mean-Max and Mean-random composition models for the WFIP, and the resultant service sequences are indicated below:

- Total possible combinations = 125
- Mean-Max composition model.
  - Binding scheme (higher granularity): S_{15} -> S_{23} -> S_{31}

- Composite trust score is 4.39.
- Execution time: 2.6ms

- Mean-Random composition model.
  - Binding scheme (higher granularity): S_{11} -> S_{24} -> S_{33}
  - Composite trust score is 2.15.
  - Execution time: 1.8ms

We can observe, from the above details, a pattern similar to the first case study. – i.e., the Mean-Max composition model selects a better alternative, with a higher composite trust score, for the composed system than the Mean-random model. However, similar to the first case study, the Mean-random model selects a better alternative, with a lower execution time, than the Mean-max model.

2) Low-level Trust-aware Model. For the Low-level Trust-aware Model, as indicated earlier, the probability values of each service to the benign (B) or malicious (M) class are computed, and these values are given in Table 7.5.

Table 7.5. Service filter attribute value of each service in WFIP.

IpToCity (S_1) (B% | M%)	**CityToZip (S_2)** (B% | M%)	**ZipToWeather (S_3)** (B% | M%)
S_{11} (50 | 50)	S_{21} (87 | 13)	S_{31} (75 | 25)
S_{12} (75 | 25)	S_{22} (75 | 25)	S_{32} (81 | 19)
S_{13} (75 | 25)	S_{23} (60 | 40)	S_{33} (58 | 42)
S_{14} (87 | 13)	S_{24} (58 | 42)	S_{34} (58 | 42)
S_{15} (75 | 25)	S_{25} (81 | 19)	S_{35} (75 | 25)

Services with the probability of 70% or below belonging to the benign category, again, are filtered out. By doing so, the remaining choices are: four services (S_{12}, S_{13}, S_{14}, S_{15}) from S_1 category, three services (S_{21}, S_{22}, S_{25}) from the S_2 category, and 3 services (S_{31}, S_{32}, S_{35}) from the S_3 category. After performing these initial pruning operations, we obtain the following results using the Mean-Max composition model– again, since the Mean-random composition model selects a sequence randomly, it will not select a sequence that is more optimal than the one selected by the Mean-Max model and hence, it is not listed below:

- Total possible combinations = 36

- Trust aware composition model.
    - Optimal Binding scheme (lower granularity): S_{15} -> S_{23} -> S_{31}
    - Composite trust score is 4.39.
    - Execution time: 0.74ms

7.4 Comparison of the Trust-aware Model

We compared the proposed trust-aware model (Low-level) with the Mean-Max composition model that uses average star rating as the objective parameter.

7.4.1 Case Study 1 - OADS

For each service, in OADS, the average rating score (collected from Google PlayStore) is given in Table 7.6.

Table 7.6. Average rating score of each service in OADS.

Virus scanner (S_1) *(Googple PlayStore Rating)*	**Grammar check (S_2)** *(Googple PlayStore Rating)*	**PDF converter (S_3)** *(Googple PlayStore Rating)*	**FAX (S_4)** *(Googple PlayStore Rating)*
S_{11} (4)	S_{21} (4)	S_{31} (4.4)	S_{41} (3.7)
S_{12} (4.4)	S_{22} (4.4)	S_{32} (4.6)	S_{42} (3.8)
S_{13} (4.5)	S_{23} (4.7)	S_{33} (3.9)	S_{43} (4.2)
S_{14} (4.3)	S_{24} (4.3)	S_{34} (4.4)	S_{44} (4.8)
S_{15} (4.5)	S_{25} (4.5)	S_{35} (3.3)	S_{45} (3.8)

The application of the Mean-Max composition model based on average rating score results in following outcomes:

- Total possible combinations = 625
- Mean-Max composition model based on average rating score.
    - Binding scheme: S_{13} -> S_{23} -> S_{32} -> S_{44}
    - Composite average rating score is 4.65.

We can see, from above outcomes, that the selected sequence is different from the one suggested by the Low-level Trust-aware model. According to the data given in Table 7.3, it is evident that both S_{32} and S_{44} have a comparatively low probability of being benign services and

thus, may not be ideal candidates for an inclusion in a composed system. The average rating score, in addition, is subjective and reflects a user's sentiments, and it is also known that developers can manipulate the rating scores. Hence, a composition based on an average rating score may not be a choice for an inclusion in a composed system.

Next, we formulate the Bayesian estimation of the trust value of a service provided by Li et al. [73]. The formula for trust value with n-ratings (r_1, r_2, … r_n) is given below, where each rating $\in$ [0, 1].

$$T(r_1, r_2, \ldots r_n, \delta) = \frac{\sum_i^n xi + n\delta}{2n} \quad (46)$$

Here, $\delta \in [0, 1]$ indicates the user's prior subjective belief about the trust, initially without having prior subjective information the δ value is 0.5; represents natural belief. In the OADS case study, we set the mean E-SERS score of the services in a binding scheme as the value of δ after doing normalization for each score. The rating list is based on average rating as given in Table 7.4, and the value of *n* is 4. For the average rating score also, we carried out normalization. The resulted service sequence after applying the Mean-Max composition model, based on Bayesian estimation of the trust, is given below:

- Total possible combinations = 625
- Mean-Max composition model based on average rating score.
    - Binding scheme: S_{13} -> S_{23} -> S_{32} -> S_{43}
    - Composite average rating score is 4.19.

The Bayesian estimation with average rating-based list performs better than the composed alternative solely, which is completely based on the average rating score. The final binding scheme eliminates one of the services, S_{44}, which has a low probability of being in the benign category.

We, then, apply the formula (44) for computing the trust values with a rating list is based on E-SERS score, where each rating $\in$ [0, 1] and set the δ value to default 0.5. The resulted service sequence after applying the Mean-Max composition model, based on Bayesian estimation of the trust, is given below:

- Total possible combinations = 625
- Mean-Max composition model based on E-SERS score.
    - Binding scheme: S_{13} -> S_{23} -> S_{33} -> S_{43}
    - Composite trust score is 3.2.

The Bayesian estimation using the E-SERS rating-based list performs identical as trust-aware composition model where the trust score for the final binding scheme is 3.2.

7.4.2 Case Study 2 - WFIP

For each service, in WFIP, the average rating score (collected from Google PlayStore) is given in Table 7.7.

Table 7.7. Average rating score of each service in WFIP.

IpToCity (S_1) *(Googple PlayStore Rating)*	**CityToZip (S_2)** *(Googple PlayStore Rating)*	**ZipToWeather (S_3)** *(Googple PlayStore Rating)*
S_{11} (4.1)	S_{21} (4.3)	S_{31} (4.6)
S_{12} (4.6)	S_{22} (4.1)	S_{32} (4.6)
S_{13} (4.5)	S_{23} (4.4)	S_{33} (4.2)
S_{14} (4.5)	S_{24} (4)	S_{34} (4.2)
S_{15} (4.5)	S_{25} (4.1)	S_{35} (4.1)

The application of the Mean-Max composition model based on average rating score results in following outcomes:

- Total possible combinations = 125
- Mean-Max composition model based on average rating score.
    - Binding scheme: S_{12} -> S_{23} -> S_{31}
    - Composite average rating score is 4.53.

We can again see, from above outcomes, that the selected sequence is different from the one suggested by the Low-level Trust-aware model. According to the data given in Table 7.4, it is evident that S_{12} has a comparatively lower trust score (3.95) and thus, may not be an ideal candidate for an inclusion in a composed system. Next, we apply the Bayesian estimation of the trust value formula (46). Similarly, in the WFIP case study, also we set the mean E-SERS score of the services in a binding scheme as the value of δ after doing normalization for each score. The rating list is based on the average rating given in Table 7.7, and the value of n is 3. For the average rating score also, we carried out normalization. The resulted service sequence after applying the Mean-Max composition model, based on Bayesian estimation of the trust, is given below:

- Total possible combinations = 125
- Mean-Max composition model based on average rating score.
    - Binding scheme: S_{15} -> S_{23} -> S_{31}
    - Composite average rating score is 4.4.

The Bayesian estimation with an average rating-based list performs better than the composed alternative created solely based on the average rating score. The final binding scheme eliminates one of the services, S_{12}, which has a low trust score. Then we apply the formula (44) for trust value with rating list is based on E-SERS score, where each rating $\in [0, 1]$ and set the δ value to default 0.5. The resulted service sequence after applying the Mean-Max composition model, based on Bayesian estimation of the trust, is presented here:

- Total possible combinations = 125
- Mean-Max composition model based on E-SERS score.
    - Binding scheme: S_{15} -> S_{23} -> S_{31}
    - Composite trust score is 3.4.

The Bayesian estimation using the E-SERS rating-based list performs identical as trust-aware composition model where the trust score for the final binding scheme is 3.4.

Sec. 5.4 has highlighted the rankings' disparity based on the average rating score and the E-SERS score. A similar scenario has been observed here for the composition models as well. Overall, we can conclude that our proposed trust-aware composition model in lower granularity level performs better in execution and can generate the most optimal service binding scheme than generated by other models.

CHAPTER 8. CONCLUSION AND FUTURE WORK

In this dissertation, we have proposed a security-related and evidence-based ranking framework. The preliminary scheme proposed, is called SERS, which is later enhanced to E-SERS. E-SERS computes direct trust and indirect trust scores for an App using evidence obtained from Static code analysis, Static Taint Analysis, Security Analysis, and Sentiment Analysis and aggregates the results using Subjective Logic principles and operations. It obtains a holistic rank ordering of comparable Apps and provides insights, using structured and unstructured artifacts associated with Apps available in the Google PlayStore. The empirical evaluations show that the E-SERS considers the comprehensive nature of an App when compared with the other existing choices and provides a better ranking of similar Apps. Additionally, the E-SERS, using the direct trust artifacts, overcomes the limitation of small number of reviews associated with newly published Apps.

Utilizing the E-SERS trust score, a trust-aware composition model proposed as the last part of this dissertation. The composition model has two levels of granularity. The proposed trust-aware composition model is empirically evaluated using two test cases and is compared with the techniques based on average star rating parameter and the trust score.

8.1 Contributions

The major contributions of the dissertation are summarized below:

- It proposed a security-related and evidence-based ranking scheme to compute the trust of an App.
- The proposed scheme is formalized to accommodate many sources to generate evidence for an App. It also integrates the reputation of sources, temporal aspects of external artifacts, and combines internal and external trust values using the principles of subjective logic.
- A prototype (http://rankings.cs.iupui.edu/) based on the proposed scheme is created and rigorously experimented with using real-world data sets collected from the Google PlayStore. The results of the experiments are compared with other prevalent approaches. This prototype is made available to the research community.

- An App classifier model which utilizes data flow features and employs different machine learning classification algorithms for the categorizing Apps into benign or malware. This model is used in the proposed trust-aware composing model as indicated below.
- A trust-aware composition model, with two levels of granularities, is proposed and empirically evaluated using two case studies. The results of the experiments are compared with the results obtained using the average star rating parameter and the trust score.

8.2 Threats to the Validity

There are a few threats to the validity of research presented in this dissertation. These are listed below:

- We have only utilized the text analysis of ~77,000 reviews for 25 Apps. Hence, the Apps used in this experiment might not be representative of the entire AppStore. We have made our data available to anyone to address this threat to use and build on our experiments [134].
- To carry out static data flow analysis, all code must be accessible, and any obfuscated flows cannot be identified easily. However, we are incapable to fully address the issue, as we do not have access to an App's source code. Consequently, their usage may affect our results. However, we used the standard tools that have been used in other research studies for the Android Apps.
- It is well known that static code analysis tools may return false-positive warnings. To overcome this limitation, in our approach, we have considered the reputation score of the tools. This reputation is based on the performance of the tool on benchmarks.
- For the trust-aware composition framework, the empirical study involves two simple composition case studies. However, any realistic composed system will contain a large component service set. To address this case studies were selected based on the Apps availability and past literature.
- The user feedback about the web prototype is not available yet and hence, is not considered in this study. This feedback will be a good resource for understanding the usefulness and acceptance of this research. However, as an initial effort, we conducted an informal survey where we asked the users to select their desired ranking scheme. A majority (43.8%) preferred the proposed combined ranking scheme over others.

- Finally, any comprehensive ranking scheme which incorporates the source code analysis and sentiment analysis of associated reviews can present a number of challenges, such as, the need for significant number of resources to identify vulnerabilities in an Apps. Applying principles of parallel processing can help to reduce this limitation.

8.3 Future Work

Following are a few directions for future research:

- We plan to apply E-SERS to datasets of newly published Apps which have an insignificant number of reviews or installs.
- We intend to apply E-SERS to Apps from other existing AppStores (such as Amazon AppStore).
- We want to integrate other good techniques (such as deep learning classifier models) to increase the classifying accuracy of categorizing an App into benign or malware group. To accomplish this, we will need to extend both benign and malware App's datasets.
- We will apply the proposed E-SERS framework for more complicated composite services and assess the validity of the proposed trust-aware composition model.

REFERENCES

[1] Biggest app stores in the WORLD 2020. Retrieved March 04, 2021, from https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/.

[2] How rating affects ranking in search results and top charts across platforms. Retrieved March 04, 2021, from https://www.adweek.com/digital/how-rating-affects-ranking-in-search-results-and-top-charts-across-platforms/.

[3] M. Harman, Y. Jia, and Y. Zhang (2012). App store mining and analysis: MSR for app stores. In 9th IEEE working conference on mining software repositories.

[4] D. Pagano and W. Maalej (2013). User Feedback in the AppStore: An Empirical Study. In The 21st IEEE International Requirements Engineering Conference.

[5] Z. Svedic (2015). The effect of informational signals on mobile apps sales ranks across the globe.

[6] W. Martin, F. Sarro, Y. Jia, Y. Zhang, and M. Harman (2017). A Survey of App Store Analysis for Software Engineering. IEEE Transactions on Software Engineering.

[7] A. Finkelstein, M. Harman, Y. Jia, W. Martin, F. Sarro, and Y. Zhang (2017). Investigating the relationship between price, rating, and popularity in the Blackberry World App Store. In Information and Software Technology.

[8] Survey on Ranking: https://tinyurl.com/survey-on-ranking-schemes.

[9] S. L. Lim and P. J. Bentley and N. Kanakam and F. Ishikawa and S. Honiden (2015). Investigating Country Differences in Mobile App User Behavior and Challenges for Software Engineering.

[10] Android and Google Play statistics, development resources and intelligence. (2021, March 03). Retrieved March 04, 2021, from https://www.appbrain.com/stats.

[11] M. Siegler (2009, September 22). Youtube comes to a 5-star realization: Its ratings are useless. Retrieved March 04, 2021, from https://techcrunch.com/2009/09/22/youtube-comes-to-a-5-star-realization-its-ratings-are-useless/.

[12] A. Henry (2014, February 04). Why you shouldn't trust app store reviews (and what to trust instead). Retrieved March 04, 2021, from https://lifehacker.com/why-you-shouldnt-trust-app-store-reviews-and-what-to-1515379780.

[13] A. Dellinger (2019, June 07). Many popular android apps leak sensitive data, leaving millions of consumers at risk. Retrieved March 04, 2021, from https://www.forbes.com/sites/ajdellinger/2019/06/07/many-popular-android-apps-leak-sensitive-data-leaving-millions-of-consumers-at-risk/#7bc629d0521e, 2019.

[14] J. Doevan. Android virus. Versions provided. the list of infected apps for 2021. Retrieved March 04, 2021, from https://www.2-spyware.com/remove-android-virus.html.

[15] A. Venkat, and R. Ross (n.d.). Kaspersky: Malware found hiding in popular Android app. Retrieved March 04, 2021, from https://www.bankinfosecurity.com/kaspersky-malware-found-hiding-in-popular-android-app-a-13008.

[16] M. Kan (2019, August 28). Malware discovered in popular android app camscanner. Retrieved March 04, 2021, from https://www.pcmag.com/news/malware-discovered-in-popular-android-app-camscanner.

[17] L. Tung (2019, August 28). Android Google Play app with 100 million downloads starts to deliver malware. Retrieved March 04, 2021, from https://www.zdnet.com/article/android-google-play-app-with-100-million-downloads-starts-to-deliver-malware/.

[18] Z. Doffman (2019, August 13). Android Warning: Devious malware found Inside 34 apps already installed By 100M+ Users. Retrieved March 04, 2021, from https://www.forbes.com/sites/zakdoffman/2019/08/13/android-warning-100m-users-have-installed-dangerous-new-malware-from-google-play/#5c7ed4cd22a9.

[19] S. Pearson (2002). Trusted Computing Platforms: TCPA Technology in Context. Prentice Hall PTR: Upper Saddle River.

[20] R. Yahalom, B. Klein, and T. Beth (1992). Trust relationships in secure systems - a distributed authentication perspective. Karlsruhe.

[21] N. Limam, and R. Boutaba (2010). Assessing software service quality and trustworthiness at selection time. IEEE Transactions on Software Engineering, 36(4), 559-574. doi:10.1109/tse.2010.2.

[22] Z. Yan (2008). A comprehensive trust model for component software. Proceedings of the 4th International Workshop on Security, Privacy and Trust in Pervasive and Ubiquitous Computing - SecPerU '08. doi:10.1145/1387329.1387330.

[23] N. S. Chowdhury, and R. R. Raje (2019). SERS: A security-related and Evidence-based ranking scheme for mobile apps. 2019 First IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA). doi:10.1109/tps-isa48467.2019.00024.

[24] N. S. Chowdhury, and R. R. Raje (2018). A holistic ranking scheme for apps. 21st International Conference of Computer and Information Technology (ICCIT). doi:10.1109/iccitechn.2018.8631955.

[25] N. S. Chowdhury and R. R. Raje (2017). Disparity between the programmatic views and the user perceptions of mobile apps. 2017 20th International Conference of Computer and Information Technology (ICCIT). doi:10.1109/iccitechn.2017.8281774.

[26] G. Shafer (2020). A mathematical theory of evidence. A Mathematical Theory of Evidence, 3-34. doi:10.2307/j.ctv10vm1qb.5.

[27] A. Jøsang, R. Hayward, and S. Pope (2006). Trust network analysis with subjective logic, In The 29th Australasian Computer Science Conference.

[28] F. Zhuang, Jing, and X. Zhu, Movie review mining and summarization, In The 15th ACM international conference on Information and knowledge management, 2006.

[29] H. Tang, S. Tan, and X. Cheng, A survey on sentiment detection of reviews, Expert Systems with Applications, 2009.

[30] B. Pang and L. Lee, Opinion Mining and Sentiment Analysis, Foundations and Trends in Information Retrieval, 2008.

[31] S. Panichella, A. Sorboy, E. Guzmanz, C. Visaggioy, G. Canforay, and H. Gall, How Can I Improve My App? Classifying User Reviews for Software Maintenance and Evolution, In The International Conference on Software Maintenance and Evolution, 2015.

[32] B. Pang, L. Lee, and S. Vaithyanathan, ThumbsUp? Sentiment Classification using Machine Learning Techniques, In the Empirical Methods in Natural Language Processing, 2002.

[33] C. Sangani, & S. Ananthanarayanan (2013). Sentiment Analysis of App Store Reviews.

[34] D. Pagano and W. Maalej, User Feedback in the AppStore: An Empirical Study, In The 21st IEEE International Requirements Engineering Conference, 2013.

[35] F. Palomba, M. Linarcs-Vasqucz, G. Bavota, R. Oliveto, M. Penta, D. Poshyvanyk, and A. Lucia, User Reviews Matter! Tracking Crowdsourced Reviews to Support Evolution of Successful Apps, In The International Conference on Software Maintenance and Evolution, 2015.

[36] L. Gallege, and R. R. Raje, Parallel Methods for Evidence and Trust-based Selection and Recommendation of Software Apps from Online Marketplaces, In The 12th Annual Cyber and Information Security Research Conference, 2017.

[37] L. Gallege, Trust-based Service Selection and Recommendation for Online Software Marketplaces (TruSStReMark), PhD Thesis Report. Purdue University, 2016.

[38] FindBugs™ - find bugs in Java programs. (n.d.). Retrieved March 4, 2021, from http://findbugs.sourceforge.net/.

[39] Jlint. (n.d.). Retrieved March 04, 2021, from http://jlint.sourceforge.net/.

[40] H. Khalid, M. Nagappan, and A. E. Hassan (2016). Examining the relationship between FINDBUGS warnings and app ratings. IEEE Software, 33(4), 34-39. doi:10.1109/ms.2015.29.

[41] Espresso: Android developers. (n.d.). Retrieved March 04, 2021, from https://developer.android.com/training/testing/espresso.

[42] UI/Application exerciser Monkey: Android developers. (n.d.). Retrieved March 04, 2021, from https://developer.android.com/studio/test/monkey.

[43] X. Ma, N. Wang, P. Xie, J. Zhou, X. Zhang, and C. Fang (2016). An automated testing platform for mobile applications. 2016 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C). doi:10.1109/qrs-c.2016.25

[44] W. Choi, G. Necula, and K. Sen (2013). Guided GUI testing of Android apps with Minimal restart and approximate learning. Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages and Applications. doi:10.1145/2509136.2509552.

[45] R. Mahmood, N. Mirzaei, and S. Malek (2014). EvoDroid: Segmented evolutionary testing of Android apps. Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering - FSE 2014. doi:10.1145/2635868.2635896.

[46] A. Machiry, R. Tahiliani and M. Naik (2013). Dynodroid: An input generation system for Android apps. Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2013. doi:10.1145/2491411.2491450.

[47] B. P. Sarma, N. Li, C. Gates, R. Potharaju, C. Nita-Rotaru, and I. Molloy (2012). Android permissions. Proceedings of the 17th ACM Symposium on Access Control Models and Technologies - SACMAT '12. doi:10.1145/2295136.2295141.

[48] Y. Zhou, Z. Wang, W. Zhou, and X. Jiang (2012). Hey, you, get off of my market: Detecting malicious Apps in official and alternative android markets. In NDSS.

[49] Y. Wang, J. Zheng, C. Sun, and S. Mukkamala (2013). Quantitative security risk assessment of android permissions and applications. Lecture Notes in Computer Science, 226-241. doi:10.1007/978-3-642-39256-6_15.

[50] C. S. Gates, N. Li, H. Peng, B. Sarma, Y. Qi, R. Potharaju, C. Nita-Rotaru, I. Molloy (2014). Generating summary risk scores for mobile applications. IEEE Transactions on Dependable and Secure Computing, 11(3), 238-251. doi:10.1109/tdsc.2014.2302293.

[51] Y. Acar, M. Backes, S. Bugiel, S. Fahl, P. McDaniel, and M. Smith (2016). SoK: Lessons learned from Android security research for APPIFIED software platforms. 2016 IEEE Symposium on Security and Privacy (SP). doi:10.1109/sp.2016.33.

[52] O. Mirzaei, G. Suarez-Tangil, J. Tapiador, and J. M. De Fuentes (2017). TriFlow. Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security. doi:10.1145/3052973.3053001.

[53] Y. Acar, M. Backes, S. Bugiel, S. Fahl, P. McDaniel, and M. Smith (2016). SoK: Lessons learned from Android security research for APPIFIED software platforms. 2016 IEEE Symposium on Security and Privacy (SP). doi:10.1109/sp.2016.33.

[54] W. Enck, P. Gilbert, B. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth (2014). TaintDroid. Communications of the ACM, 57(3), 99-106. doi:10.1145/2494522.

[55] C. Gibler, J. Crussell, J. Erickson, and H. Chen (2012). AndroidLeaks: Automatically detecting potential privacy leaks in Android applications on a large scale. Trust and Trustworthy Computing, 291-307. doi:10.1007/978-3-642-30921-2_17.

[56] M. I. Gordon, D. Kim, J. Perkins, L. Gilham, N. Nguyen, and M. Rinard (2015). Information-Flow analysis of Android applications in DroidSafe. Proceedings 2015 Network and Distributed System Security Symposium. doi:10.14722/ndss.2015.23089.

[57] AndroBugs. (n.d.). AndroBugs/AndroBugs_Framework. Retrieved March 04, 2021, from https://github.com/AndroBugs/AndroBugs_Framework.

[58] Qark. Retrieved March 04, 2021, from https://github.com/linkedin/qark.

[59] JAADAS. Retrieved March 04, 2021, from https://github.com/flankerhqd/JAADAS.

[60] H. Zhu, H. Xiong, Y. Ge, and E. Chen (2014). Mobile App Recommendations with Security and Privacy Awareness. In The 20th ACM SIGKDD Intl. conference on Knowledge discovery and data mining.

[61] L. Cen, D. Kong, H. Jin and L. Si (2015). Mobile app security risk assessment: A crowdsourcing ranking approach from user comments. Proceedings of the 2015 SIAM International Conference on Data Mining. doi:10.1137/1.9781611974010.74.

[62] B. Baskaran, and A. Ralescu (2016). A Study of Android Malware Detection Techniques and Machine Learning. MAICS.

[63] S. Y. Yerima, S. Sezer, and I. Muttik (2014). Android malware detection using parallel machine learning classifiers. 2014 Eighth International Conference on Next Generation Mobile Apps, Services and Technologies. doi:10.1109/ngmast.2014.23.

[64] M. A. Ali, D. Svetinovic, Z. Aung, and S. Lukman (2017). Malware detection in android mobile platform using machine learning algorithms. 2017 International Conference on Infocom Technologies and Unmanned Systems (Trends and Future Directions) (ICTUS). doi:10.1109/ictus.2017.8286109.

[65] M. Kakavand, M. Dabbagh, and A. Dehghantanha (2018). Application of machine learning algorithms for android malware detection. Proceedings of the 2018 International Conference on Computational Intelligence and Intelligent Systems - CIIS 2018. doi:10.1145/3293475.3293489.

[66] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck (2014). Drebin: Effective and Explainable detection of Android malware in your pocket. Proceedings 2014 Network and Distributed System Security Symposium. doi:10.14722/ndss.2014.23247.

[67] D. Wu, C. Mao, T. Wei, H. Lee, and K. Wu (2012). Droidmat: Android malware detection through manifest and api calls tracing. 2012 Seventh Asia Joint Conference on Information Security. doi:10.1109/asiajcis.2012.18.

[68] S. Seo, A. Gupta, A. Mohamed Sallam, E. Bertino, and K. Yim (2014). Detecting mobile malware threats to homeland security through static analysis. Journal of Network and Computer Applications, 38, 43-53. doi:10.1016/j.jnca.2013.05.008.

[69] W. Jiang, S. Hu, and Z. Liu (2014). Top K query FOR QoS-Aware automatic service composition. IEEE Transactions on Services Computing, 7(4), 681-695. doi:10.1109/tsc.2013.41.

[70] W. Jiang, C. Zhang, Z. Huang, M. Chen, S. Hu, and Z. Liu (2010). QSynth: A tool for Qos-aware automatic service composition. 2010 IEEE International Conference on Web Services. doi:10.1109/icws.2010.38.

[71] P. Bartalos, and M. Bielikova (2009). Semantic web service composition framework based on parallel processing. 2009 IEEE Conference on Commerce and Enterprise Computing. doi:10.1109/cec.2009.27.

[72] Y. Yan, B. Xu, Z. Gu, and S. Luo. (2009). A QoS-Driven approach for SEMANTIC service composition. 2009 IEEE Conference on Commerce and Enterprise Computing. doi:10.1109/cec.2009.44.

[73] L. Li, Y. Wang, and E. Lim (2009). Trust-Oriented composite Service selection and discovery. Service-Oriented Computing – ICSOC 2007, 50-67. doi:10.1007/978-3-642-10383-4_4.

[74] C. Hu, X. Wu and B. Li, "A Framework for Trustworthy Web Service Composition and Optimization," in IEEE Access, vol. 8, pp. 73508-73522, 2020, doi: 10.1109/ACCESS.2020.2984648.

[75] D. Hovemeyer, and W. Pugh (2004). Finding bugs is easy. ACM SIGPLAN Notices, 39(12), 92-106. doi:10.1145/1052883.1052895.

[76] Evaluation of FindBugs. (n.d.). Retrieved March 4, 2021, from https://www.cs.cmu.edu/~aldrich/courses/654/tools/square-root-FindBugs-2009.pdf.

[77] FindBugs™ - find bugs in Java programs. (n.d.). Retrieved March 4, 2021, from http://findbugs.sourceforge.net/.

[78] S. Arzt, S. Rasthofer, and E. Bodden (2013). SuSi: A Tool for the Fully Automated Classification and Categorization of Android Sources and Sinks.

[79] FlowDroid – taint analysis. (n.d.). Retrieved March 04, 2021, from https://blogs.uni-paderborn.de/sse/tools/flowdroid/.

[80] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. L. Traon, D. Octeau, P. McDaniel (2014). FlowDroid: Precise Context, Flow, Field, Object-Sensitive and Lifecycle-Aware Taint Analysis for Android Apps. Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation. doi:10.1145/2594291.2594299.

[81] Simplified text processing. (n.d.). Retrieved March 04, 2021, from https://textblob.readthedocs.io/en/dev/.

[82] The IBM Watson Natural Language Understanding [online] Available. https://cloud.ibm.com/docs/services/natural-language-understanding?topic=natural-language-understanding-getting-started.

[83] G. Shafer (1976). A mathematical theory of evidence. Princeton: Princeton University Press.

[84] A. Jøsang, R. Hayward, and S. Pope (2001). A Logic for Uncertain Probabilities, International Journal of Uncertainty, In Fuzziness and Knowledge Based Systems.

[85] A. Jøsang, S. Pope, and M. Daniel (2005). Conditional deduction under uncertainty. Lecture Notes in Computer Science, 824-835. doi:10.1007/11518655_69.

[86] B. Skoric and N. Zannone (2015). Flow-based reputation with uncertainty: Evidence-Based Subjective Logic, International Journal of Information Security.

[87] H. Zhou, W. Shi, Z. Liang, and B. Liang (2011). Using new fusion operations to improve trust expressiveness of subjective logic. Wuhan University Journal of Natural Sciences, 16(5), 376-382. doi:10.1007/s11859-011-0766-3.

[88] M. Kendall (1938). A New Measure of Rank Correlation. Biometrika.

[89] #1 tech media company in the world. (2021, March 22). Retrieved March 29, 2021, from http://www.idg.com/.

[90] Permissions on Android: Android developers. (n.d.). Retrieved March 06, 2021, from https://developer.android.com/guide/topics/permissions/overview.

[91] K. W. Au, Y. F. Zhou, Z. Huang, and D. Lie (2012). PScout: Analyzing the Android Permission Specification. Proceedings of the 2012 ACM Conference on Computer and Communications Security - CCS '12. doi:10.1145/2382196.2382222.

[92] G. Stoneburner, A. Goguen, and A. Feringa (2002). Risk management guide for information technology systems: doi:10.6028/nist.sp.800-30.

[93] L. Gallon (2011). Vulnerability discrimination using cvss framework. 2011 4th IFIP International Conference on New Technologies, Mobility and Security. doi:10.1109/ntms.2011.5720656.

[94] Download APK free Online downloader. (n.d.). Retrieved March 06, 2021, from https://apkpure.com/.

[95] V. Avdiienko, K. Kuznetsov, A. Gorla, A. Zeller, S. Arzt, S. Rasthofer, and E. Bodden (2015). Mining apps for abnormal usage of sensitive data. 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering. doi:10.1109/icse.2015.61.

[96] Virusshare.com. (n.d.). Retrieved March 06, 2021, from https://virusshare.com/.

[97] The Drebin Dataset. Retrieved March 06, 2021, from https://www.sec.cs.tu-bs.de/~danarp/drebin/.

[98] A. Spathoulas (2014). Assessing Tools for Finding Bugs in Concurrent Java. Technical Report Master's Thesis Report. University of Edinburgh.

[99] DroidBench – benchmarks. (n.d.). Retrieved March 06, 2021, from https://blogs.uni-paderborn.de/sse/tools/droidbench/.

[100] K. Shung (2020, April 10). Accuracy, precision, recall or f1? Retrieved March 06, 2021, from https://towardsdatascience.com/accuracy-precision-recall-or-f1-331fb37c5cb9.

[101] Unicodedata - Unicode Database. (n.d.). Retrieved March 06, 2021, from https://docs.python.org/2/library/unicodedata.html.

[102] Stephanie. (2020, December 28). Absolute error and mean absolute Error (MAE). Retrieved March 06, 2021, from https://www.statisticshowto.datasciencecentral.com/absolute-error/.

[103] N. Jindal, and B. Liu (2007). Analyzing and detecting review spam. Seventh IEEE International Conference on Data Mining (ICDM 2007). doi:10.1109/icdm.2007.68.

[104] Y., C. Wu, S. Zhu, and H. Wang (2019). A machine learning based approach for mobile app rating manipulation detection. ICST Transactions on Security and Safety, 5(18), 157415. doi:10.4108/eai.8-4-2019.157415.

[105] MacKay and David. (2003). An Example Inference Task: Clustering. Information Theory, Inference and Learning Algorithms. Cambridge University Press.

[106] A. G. Hawkes (1971). Spectra of some self-exciting and mutually exciting point processes.

[107] A. G. Hawkes and D. Oakes (1974). A Cluster Process Representation of a Self-Exciting Process. Journal of Applied Probability.

[108] D. R. Johnson. (2008). Using Weights in the Analysis of Survey Data. Population Research Institute, The Pennsylvania State University.

[109] Intellica.AI. (2019, September 19). VADER, IBM Watson OR TextBlob: Which is better for Unsupervised sentiment analysis? Retrieved March 06, 2021, from https://medium.com/@Intellica.AI/vader-ibm-watson-or-textblob-which-is-better-for-unsupervised-sentiment-analysis-db4143a39445.

[110] Newnes, P. (2019, December 19). Sentiment analysis in the cloud with Google Cloud natural Language, AWS COMPREHEND, and IBM Watson. Retrieved March 06, 2021, from https://www.deducive.com/blog/2018/6/02/using-r-for-sentiment-analysis-with-aws-comprehend-google-cloud-natural-language-ibm-watson.

[111] Reed, B. (n.d.). Nowsecure. Retrieved March 07, 2021, from https://www.nowsecure.com/blog/2019/06/06/test-of-250-popular-android-mobile-apps-reveal-that-70-leak-sensitive-personal-data/?utm_source=pressandutm_medium=referral.

[112] Nowsecure. (n.d.). Retrieved March 07, 2021, from https://www.nowsecure.com/.

[113] Safety center - Mobile Safety. (n.d.). Retrieved March 19, 2021, from https://www.android.com/play-protect.

[114] AVG 2021: Free antivirus and Tuneup for PC, Mac, Android. (n.d.). Retrieved March 07, 2021, from https://www.avg.com/.

[115] Norton. (n.d.). Retrieved March 07, 2021, from https://my.norton.com/mobile/home.

[116] Maalej, W., and Nabil, H. (2015). Bug report, feature request, or simply praise? On automatically classifying app reviews. 2015 IEEE 23rd International Requirements Engineering Conference (RE). doi:10.1109/re.2015.7320414.

[117] Google play Ranking: Top free overall in the United States. (2021, March 18). Retrieved March 19, 2021, from https://www.appbrain.com/stats/google-play-rankings/top/_free/application/us#types.

[118] App store Ranking Factors: App store vs. Google Play. (n.d.). Retrieved March 19, 2021, from https://appradar.com/academy/bonus-chapters/app-store-ranking-factors/.

[119] DataTechNotes. (2019, June 26). Regression example with xgbregressor in python. Retrieved March 19, 2021, from https://www.datatechnotes.com/2019/06/regression-example-with-xgbregressor-in.html.

[120] Welcome to flask. (n.d.). Retrieved March 12, 2021, from https://flask.palletsprojects.com/en/1.1.x/.

[121] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck (2014). Drebin: Effective and Explainable detection of Android malware in your pocket. Proceedings 2014 Network and Distributed System Security Symposium. doi:10.14722/ndss.2014.23247.

[122] M. Spreitzenbarth, F. Freiling, F. Echtler, T. T/Schreck, and J. Hoffmann (2013). MobileSandbox: Looking Deeper into Android Applications. Proceedings of the 28th Annual ACM Symposium on Applied Computing - SAC '13. doi:10.1145/2480362.2480701.

[123] 16.6. multiprocessing - process-based "threading" interface. (n.d.). Retrieved March 12, 2021, from https://docs.python.org/2/library/multiprocessing.html.

[124] R. Gandhi (2018, July 05). Support vector machine - introduction to machine learning algorithms. Retrieved March 08, 2021, from https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47.

[125] O. Harrison (2019, July 14). Machine learning basics with the k-nearest Neighbors ALGORITHM. Retrieved March 08, 2021, from https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761.

[126] A. Pant (2019, January 22). Introduction to logistic regression. Retrieved March 08, 2021, from https://towardsdatascience.com/introduction-to-logistic-regression-66248243c148.

[127] J. Brownlee (2020, August 14). Naive Bayes for machine learning. Retrieved March 08, 2021, from https://machinelearningmastery.com/naive-bayes-for-machine-learning/.

[128] Decision trees for classification: A machine learning algorithm. (n.d.). Retrieved March 08, 2021, from https://www.xoriant.com/blog/product-engineering/decision-trees-machine-learning-algorithm.html.

[129] T. You (2019, August 14). Understanding random forest. Retrieved March 08, 2021, from https://towardsdatascience.com/understanding-random-forest-58381e0602d2.

[130] KuafuDet download. (n.d.). Retrieved March 08, 2021, from https://nsec.sjtu.edu.cn/kuafuDet/download.html.

[131] Learn. (n.d.). Retrieved March 08, 2021, from https://scikit-learn.org/stable/.

[132] Classification: ROC curve and Auc | machine Learning crash course. (n.d.). Retrieved March 10, 2021.

[133] S. K. Bansal, A. Bansal, and M. B. Blake (2010). Trust-based dynamic web service composition using social network analysis. 2010 IEEE International Workshop On: Business Applications of Social Network Analysis (BASNA). doi:10.1109/basna.2010.5730308.

[134] Dataset - Google Drive. (n.d.). Retrieved March 22, 2021, from https://tinyurl.com/E-SERS-dataset.

APPENDIX A. SURVEY I RESPONSES

We carried out an informal survey – our survey audience contained Computing students and professionals. We sent our survey to a sample of the general population in the Computing domain. The audiences of this survey are anonymous users as we did not request the user to provide any personally identifiable data. We asked the following question:

"*In general, what is the most important factor that users considered to assess an App before downloading*?" – we received 130 responses. The response is given below.

Timestamp	Responses
2020/01/14 9:07:49 PM EST	Average Rating Score
2020/01/21 1:36:54 PM EST	Number of Installs
2020/01/21 1:38:00 PM EST	Number of Installs
2020/01/21 1:38:14 PM EST	Average Rating Score
2020/01/21 1:40:24 PM EST	User Reviews
2020/01/21 1:43:50 PM EST	Average Rating Score
2020/01/21 1:46:21 PM EST	Average Rating Score
2020/01/21 1:52:22 PM EST	User Reviews
2020/01/21 1:58:21 PM EST	User Reviews
2020/01/21 2:12:34 PM EST	User Reviews
2020/01/21 2:16:16 PM EST	User Reviews
2020/01/21 2:22:15 PM EST	User Reviews
2020/01/21 2:31:29 PM EST	User Reviews
2020/01/21 2:38:22 PM EST	Average Rating Score
2020/01/21 4:12:49 PM EST	User Reviews
2020/01/21 4:19:22 PM EST	User Reviews
2020/01/21 8:22:29 PM EST	Developer info
2020/01/21 9:12:19 PM EST	User Reviews
2020/01/21 11:13:06 PM EST	Average Rating Score
2020/01/21 11:22:26 PM EST	Average Rating Score
2020/01/22 1:21:43 AM EST	App Size
2020/01/22 5:59:47 AM EST	Average Rating Score
2020/01/22 6:58:47 AM EST	User Reviews
2020/01/22 8:13:26 AM EST	Average Rating Score
2020/01/22 8:26:49 AM EST	User Reviews
2020/01/22 12:43:55 PM EST	User Reviews
2020/01/23 9:34:25 PM EST	Average Rating Score
2020/01/24 2:08:27 AM EST	Average Rating Score
2020/01/24 10:05:13 AM EST	User Reviews

2020/01/24 10:07:20 AM EST	Average Rating Score
2020/01/24 10:08:06 AM EST	User Reviews
2020/01/24 10:08:11 AM EST	Average Rating Score
2020/01/24 10:08:48 AM EST	User Reviews
2020/01/24 10:09:23 AM EST	User Reviews
2020/01/24 10:12:04 AM EST	Average Rating Score
2020/01/24 10:13:07 AM EST	Number of Installs
2020/01/24 10:13:33 AM EST	User Reviews
2020/01/24 10:38:56 AM EST	Number of Installs
2020/01/24 10:53:40 AM EST	User Reviews
2020/01/24 11:05:24 AM EST	Number of Installs
2020/01/24 11:20:19 AM EST	User Reviews
2020/01/24 11:24:23 AM EST	Average Rating Score
2020/01/24 11:25:37 AM EST	User Reviews
2020/01/24 11:34:22 AM EST	Average Rating Score
2020/01/24 12:11:19 PM EST	Average Rating Score
2020/01/24 12:29:08 PM EST	Average Rating Score
2020/01/24 1:51:24 PM EST	Number of Installs
2020/01/25 12:15:59 PM EST	User Reviews
2020/01/26 12:37:44 AM EST	Average Rating Score
2020/02/04 1:01:54 AM EST	Average Rating Score
2020/02/04 8:01:00 AM EST	App Size
2020/02/04 8:44:54 AM EST	Average Rating Score
2020/02/04 8:46:15 AM EST	Average Rating Score
2020/02/04 8:55:19 AM EST	Number of Installs
2020/02/04 11:12:08 AM EST	Average Rating Score
2020/02/04 11:58:06 AM EST	Average Rating Score
2020/02/05 6:16:58 AM EST	Average Rating Score
2020/02/05 8:29:34 AM EST	User Reviews
2020/02/05 9:50:21 AM EST	User Reviews
2020/02/05 10:00:19 AM EST	Number of Installs
2020/02/05 10:33:30 AM EST	User Reviews
2020/02/05 11:24:00 AM EST	Average Rating Score
2020/02/05 12:39:24 PM EST	Number of Installs
2020/02/05 10:09:54 PM EST	User Reviews
2020/02/05 10:10:01 PM EST	Developer info
2020/02/05 10:10:07 PM EST	App Size
2020/02/05 10:53:32 PM EST	Average Rating Score
2020/02/06 12:33:32 AM EST	User Reviews
2020/02/06 12:33:45 AM EST	User Reviews
2020/02/06 9:29:36 AM EST	User Reviews
2020/02/06 9:40:46 AM EST	Average Rating Score
2020/02/06 3:29:36 PM EST	Average Rating Score

2020/02/06 3:29:44 PM EST	Average Rating Score
2020/02/07 1:07:24 AM EST	User Reviews
2020/02/07 8:08:29 AM EST	Average Rating Score
2020/02/07 9:25:06 AM EST	Average Rating Score
2020/02/07 10:24:03 AM EST	Average Rating Score
2020/02/07 11:00:05 AM EST	Average Rating Score
2020/02/07 11:17:16 AM EST	Average Rating Score
2020/02/07 11:40:13 AM EST	Average Rating Score
2020/02/07 1:47:51 PM EST	User Reviews
2020/02/08 2:53:49 AM EST	App Size
2020/02/08 7:16:55 AM EST	Average Rating Score
2020/02/08 9:26:10 AM EST	Average Rating Score
2020/02/08 2:06:43 PM EST	Average Rating Score
2020/02/09 12:06:58 AM EST	Number of Installs
2020/02/09 7:02:16 AM EST	Number of Installs
2020/02/10 12:23:39 PM EST	Average Rating Score
2020/02/11 12:45:00 AM EST	User Reviews
2020/02/12 3:23:42 AM EST	Number of Installs
2020/02/12 3:47:28 AM EST	Average Rating Score
2020/02/12 11:52:43 AM EST	Number of Installs
2020/02/13 2:18:59 PM EST	Number of Installs
2020/02/13 2:19:33 PM EST	Number of Installs
2020/02/14 9:20:02 AM EST	User Reviews
2020/02/14 10:24:00 AM EST	User Reviews
2020/02/15 8:46:19 AM EST	User Reviews
2020/02/15 12:38:09 PM EST	User Reviews
2020/02/15 1:08:31 PM EST	Average Rating Score
2020/02/16 8:18:25 AM EST	User Reviews
2020/02/16 11:27:06 AM EST	App Size
2020/02/16 10:15:27 PM EST	Average Rating Score
2020/02/17 8:54:53 AM EST	User Reviews
2020/02/17 11:35:30 AM EST	User Reviews
2020/02/17 11:38:03 AM EST	Average Rating Score
2020/02/17 12:09:20 PM EST	Number of Installs
2020/02/17 12:36:01 PM EST	User Reviews
2020/02/17 12:55:43 PM EST	Average Rating Score
2020/02/18 11:16:34 PM EST	Average Rating Score
2020/02/19 11:13:42 AM EST	Average Rating Score
2020/02/19 2:36:16 PM EST	User Reviews
2020/02/19 8:36:02 PM EST	Number of Installs
2020/02/20 11:21:10 AM EST	Developer info
2020/02/20 11:28:18 AM EST	User Reviews
2020/02/21 1:07:03 AM EST	Average Rating Score

2020/02/21 8:37:52 AM EST	App Size
2020/02/21 3:29:40 PM EST	Average Rating Score
2020/02/26 4:58:09 AM EST	User Reviews
2020/02/28 9:57:28 PM EST	User Reviews
2020/03/09 9:25:16 AM EST	User Reviews
2020/03/10 11:13:01 AM EST	User Reviews
2020/03/11 11:27:16 PM EST	Average Rating Score
2020/03/13 8:32:49 AM EST	User Reviews
2020/03/16 1:47:35 PM EST	User Reviews
2020/03/17 2:00:39 AM EST	Average Rating Score
2020/03/17 10:23:41 AM EST	App Size
2020/03/17 10:45:19 AM EST	User Reviews
2020/03/20 1:38:40 PM EST	Number of Installs
2020/03/24 7:20:07 AM EST	User Reviews
2020/03/27 5:27:45 AM EST	User Reviews

APPENDIX B. SURVEY II RESPONSES

We carried out an informal survey – our survey audience contained Computing students and professionals. We sent our survey to a sample of the general population in the Computing domain. The audiences of this survey are anonymous users as we did not request the user to provide any personally identifiable data. We asked the following question:

"Which one of the following ranking schemes could be the right fit to evaluate an App?" – we received 130 responses. The response is given below.

Timestamp	Responses
2020/01/14 9:07:49 PM EST	Combined Ranking Scheme (Internal & External factors)
2020/01/21 1:36:54 PM EST	Ranking based on Internal factors (Data leaks, Security Vulnerabilities, Bugs, and others))
2020/01/21 1:38:00 PM EST	Ranking based on Average User Rating
2020/01/21 1:38:14 PM EST	Ranking based on External factors (Google PlayStore Rank)
2020/01/21 1:40:24 PM EST	Combined Ranking Scheme (Internal & External factors)
2020/01/21 1:43:50 PM EST	Combined Ranking Scheme (Internal & External factors)
2020/01/21 1:46:21 PM EST	Ranking based on Average User Rating
2020/01/21 1:52:22 PM EST	Combined Ranking Scheme (Internal & External factors)
2020/01/21 1:58:21 PM EST	Combined Ranking Scheme (Internal & External factors)
2020/01/21 2:12:34 PM EST	Ranking based on Internal factors (Data leaks, Security Vulnerabilities, Bugs, and others))
2020/01/21 2:16:16 PM EST	Combined Ranking Scheme (Internal & External factors)
2020/01/21 2:22:15 PM EST	Ranking based on Internal factors (Data leaks, Security Vulnerabilities, Bugs, and others))
2020/01/21 2:31:29 PM EST	Ranking based on Average User Rating
2020/01/21 2:38:22 PM EST	Ranking based on User's Review Sentiment
2020/01/21 4:12:49 PM EST	Ranking based on User's Review Sentiment
2020/01/21 4:19:22 PM EST	Ranking based on Average User Rating
2020/01/21 8:22:29 PM EST	Ranking based on Internal factors (Data leaks, Security Vulnerabilities, Bugs, and others))
2020/01/21 9:12:19 PM EST	Combined Ranking Scheme (Internal & External factors)
2020/01/21 11:13:06 PM EST	Combined Ranking Scheme (Internal & External factors)
2020/01/21 11:22:26 PM EST	Ranking based on External factors (Google PlayStore Rank)
2020/01/22 1:21:43 AM EST	Ranking based on User's Review Sentiment
2020/01/22 5:59:47 AM EST	Ranking based on External factors (Google PlayStore Rank)

2020/01/22 6:58:47 AM EST	Ranking based on Internal factors (Data leaks, Security Vulnerabilities, Bugs, and others))
2020/01/22 8:13:26 AM EST	Combined Ranking Scheme (Internal & External factors)
2020/01/22 8:26:49 AM EST	Combined Ranking Scheme (Internal & External factors)
2020/01/22 12:43:55 PM EST	Ranking based on Internal factors (Data leaks, Security Vulnerabilities, Bugs, and others))
2020/01/23 9:34:25 PM EST	Ranking based on External factors (Google PlayStore Rank)
2020/01/24 2:08:27 AM EST	Ranking based on User's Review Sentiment
2020/01/24 10:05:13 AM EST	Ranking based on User's Review Sentiment
2020/01/24 10:07:20 AM EST	Ranking based on External factors (Google PlayStore Rank)
2020/01/24 10:08:06 AM EST	Combined Ranking Scheme (Internal & External factors)
2020/01/24 10:08:11 AM EST	Ranking based on Internal factors (Data leaks, Security Vulnerabilities, Bugs, and others))
2020/01/24 10:08:48 AM EST	Combined Ranking Scheme (Internal & External factors)
2020/01/24 10:09:23 AM EST	Ranking based on User's Review Sentiment
2020/01/24 10:12:04 AM EST	Ranking based on Average User Rating
2020/01/24 10:13:07 AM EST	Ranking based on User's Review Sentiment
2020/01/24 10:13:33 AM EST	Combined Ranking Scheme (Internal & External factors)
2020/01/24 10:38:56 AM EST	Ranking based on Average User Rating
2020/01/24 10:53:40 AM EST	Combined Ranking Scheme (Internal & External factors)
2020/01/24 11:05:24 AM EST	Ranking based on External factors (Google PlayStore Rank)
2020/01/24 11:20:19 AM EST	Ranking based on Average User Rating
2020/01/24 11:24:23 AM EST	Ranking based on Internal factors (Data leaks, Security Vulnerabilities, Bugs, and others))
2020/01/24 11:25:37 AM EST	Ranking based on User's Review Sentiment
2020/01/24 11:34:22 AM EST	Combined Ranking Scheme (Internal & External factors)
2020/01/24 12:11:19 PM EST	Ranking based on Average User Rating
2020/01/24 12:29:08 PM EST	Combined Ranking Scheme (Internal & External factors)
2020/01/24 1:51:24 PM EST	Combined Ranking Scheme (Internal & External factors)
2020/01/25 12:15:59 PM EST	Combined Ranking Scheme (Internal & External factors)
2020/01/26 12:37:44 AM EST	Combined Ranking Scheme (Internal & External factors)
2020/02/04 1:01:54 AM EST	Ranking based on Average User Rating
2020/02/04 8:01:00 AM EST	Ranking based on Average User Rating
2020/02/04 8:44:54 AM EST	Ranking based on Average User Rating
2020/02/04 8:46:15 AM EST	Combined Ranking Scheme (Internal & External factors)
2020/02/04 8:55:19 AM EST	Ranking based on Internal factors (Data leaks, Security Vulnerabilities, Bugs, and others))
2020/02/04 11:12:08 AM EST	Ranking based on User's Review Sentiment
2020/02/04 11:58:06 AM EST	Ranking based on External factors (Google PlayStore Rank)
2020/02/05 6:16:58 AM EST	Combined Ranking Scheme (Internal & External factors)

2020/02/05 8:29:34 AM EST	Combined Ranking Scheme (Internal & External factors)
2020/02/05 9:50:21 AM EST	Combined Ranking Scheme (Internal & External factors)
2020/02/05 10:00:19 AM EST	Combined Ranking Scheme (Internal & External factors)
2020/02/05 10:33:30 AM EST	Ranking based on User's Review Sentiment
2020/02/05 11:24:00 AM EST	Ranking based on User's Review Sentiment
2020/02/05 12:39:24 PM EST	Combined Ranking Scheme (Internal & External factors)
2020/02/05 10:09:54 PM EST	Ranking based on Average User Rating
2020/02/05 10:10:01 PM EST	Ranking based on Internal factors (Data leaks, Security Vulnerabilities, Bugs, and others))
2020/02/05 10:10:07 PM EST	Combined Ranking Scheme (Internal & External factors)
2020/02/05 10:53:32 PM EST	Ranking based on Internal factors (Data leaks, Security Vulnerabilities, Bugs, and others))
2020/02/06 12:33:32 AM EST	Combined Ranking Scheme (Internal & External factors)
2020/02/06 12:33:45 AM EST	Combined Ranking Scheme (Internal & External factors)
2020/02/06 9:29:36 AM EST	Ranking based on Average User Rating
2020/02/06 9:40:46 AM EST	Ranking based on User's Review Sentiment
2020/02/06 3:29:36 PM EST	Ranking based on Internal factors (Data leaks, Security Vulnerabilities, Bugs, and others))
2020/02/06 3:29:44 PM EST	Combined Ranking Scheme (Internal & External factors)
2020/02/07 1:07:24 AM EST	Combined Ranking Scheme (Internal & External factors)
2020/02/07 8:08:29 AM EST	Ranking based on Internal factors (Data leaks, Security Vulnerabilities, Bugs, and others))
2020/02/07 9:25:06 AM EST	Ranking based on Average User Rating
2020/02/07 10:24:03 AM EST	Ranking based on Average User Rating
2020/02/07 11:00:05 AM EST	Ranking based on User's Review Sentiment
2020/02/07 11:17:16 AM EST	Ranking based on Average User Rating
2020/02/07 11:40:13 AM EST	Ranking based on Internal factors (Data leaks, Security Vulnerabilities, Bugs, and others))
2020/02/07 1:47:51 PM EST	Ranking based on User's Review Sentiment
2020/02/08 2:53:49 AM EST	Combined Ranking Scheme (Internal & External factors)
2020/02/08 7:16:55 AM EST	Combined Ranking Scheme (Internal & External factors)
2020/02/08 9:26:10 AM EST	Combined Ranking Scheme (Internal & External factors)
2020/02/08 2:06:43 PM EST	Combined Ranking Scheme (Internal & External factors)
2020/02/09 12:06:58 AM EST	Combined Ranking Scheme (Internal & External factors)
2020/02/09 7:02:16 AM EST	Combined Ranking Scheme (Internal & External factors)
2020/02/10 12:23:39 PM EST	Combined Ranking Scheme (Internal & External factors)
2020/02/11 12:45:00 AM EST	Ranking based on Average User Rating
2020/02/12 3:23:42 AM EST	Combined Ranking Scheme (Internal & External factors)
2020/02/12 3:47:28 AM EST	Combined Ranking Scheme (Internal & External factors)
2020/02/12 11:52:43 AM EST	Combined Ranking Scheme (Internal & External factors)
2020/02/13 2:18:59 PM EST	Ranking based on External factors (Google PlayStore Rank)
2020/02/13 2:19:33 PM EST	Ranking based on External factors (Google PlayStore Rank)

2020/02/14 9:20:02 AM EST	Ranking based on User's Review Sentiment
2020/02/14 10:24:00 AM EST	Combined Ranking Scheme (Internal & External factors)
2020/02/15 8:46:19 AM EST	Ranking based on User's Review Sentiment
2020/02/15 12:38:09 PM EST	Combined Ranking Scheme (Internal & External factors)
2020/02/15 1:08:31 PM EST	Combined Ranking Scheme (Internal & External factors)
2020/02/16 8:18:25 AM EST	Ranking based on Internal factors (Data leaks, Security Vulnerabilities, Bugs, and others))
2020/02/16 11:27:06 AM EST	Combined Ranking Scheme (Internal & External factors)
2020/02/16 10:15:27 PM EST	Combined Ranking Scheme (Internal & External factors)
2020/02/17 8:54:53 AM EST	Ranking based on User's Review Sentiment
2020/02/17 11:35:30 AM EST	Combined Ranking Scheme (Internal & External factors)
2020/02/17 11:38:03 AM EST	Ranking based on Average User Rating
2020/02/17 12:09:20 PM EST	Ranking based on User's Review Sentiment
2020/02/17 12:36:01 PM EST	Ranking based on Internal factors (Data leaks, Security Vulnerabilities, Bugs, and others))
2020/02/17 12:55:43 PM EST	Combined Ranking Scheme (Internal & External factors)
2020/02/18 11:16:34 PM EST	Ranking based on Average User Rating
2020/02/19 11:13:42 AM EST	Combined Ranking Scheme (Internal & External factors)
2020/02/19 2:36:16 PM EST	Combined Ranking Scheme (Internal & External factors)
2020/02/19 8:36:02 PM EST	Combined Ranking Scheme (Internal & External factors)
2020/02/20 11:21:10 AM EST	Ranking based on Average User Rating
2020/02/20 11:28:18 AM EST	Ranking based on Internal factors (Data leaks, Security Vulnerabilities, Bugs, and others))
2020/02/21 1:07:03 AM EST	Ranking based on Average User Rating
2020/02/21 8:37:52 AM EST	Combined Ranking Scheme (Internal & External factors)
2020/02/21 3:29:40 PM EST	Ranking based on Average User Rating
2020/02/26 4:58:09 AM EST	Ranking based on Internal factors (Data leaks, Security Vulnerabilities, Bugs, and others))
2020/02/28 9:57:28 PM EST	Ranking based on Internal factors (Data leaks, Security Vulnerabilities, Bugs, and others))
2020/03/09 9:25:16 AM EST	Ranking based on User's Review Sentiment
2020/03/10 11:13:01 AM EST	Combined Ranking Scheme (Internal & External factors)
2020/03/11 11:27:16 PM EST	Combined Ranking Scheme (Internal & External factors)
2020/03/13 8:32:49 AM EST	Ranking based on User's Review Sentiment
2020/03/16 1:47:35 PM EST	Combined Ranking Scheme (Internal & External factors)
2020/03/17 2:00:39 AM EST	Ranking based on Internal factors (Data leaks, Security Vulnerabilities, Bugs, and others))
2020/03/17 10:23:41 AM EST	Ranking based on Average User Rating
2020/03/17 10:45:19 AM EST	Combined Ranking Scheme (Internal & External factors)
2020/03/20 1:38:40 PM EST	Ranking based on Internal factors (Data leaks, Security Vulnerabilities, Bugs, and others))
2020/03/24 7:20:07 AM EST	Combined Ranking Scheme (Internal & External factors)
2020/03/27 5:27:45 AM EST	Combined Ranking Scheme (Internal & External factors)

VITA

NAHIDA S CHOWDHURY

nahida.knu@gmail.com | (317) 737-8453
https://linkedin.com/in/nahida-chowdhury

Software Development Engineering | Distributed Computing | Big Data | Machine Learning

Enthusiastic Computer Science PhD student with strong understanding in Data structures and Algorithms. Passionate about creating and implementing high quality innovative solutions to complex problems.

EDUCATION

PURDUE SCHOOL OF SCIENCE, IUPUI, IN, UNITED STATES Expected Spring 2021
DOCTOR OF PHILOSOPHY IN COMPUTER AND INFORMATION SCIENCE (GPA 3.67/4.0)

KYUNGPOOK NATIONAL UNIVERSITY, DAEGU, SOUTH KOREA July 2012
MASTER'S IN COMPUTER SCIENCE AND ENGINEERING (GPA 4.01/4.5)

UNIVERSITY OF ASIA PACIFIC, DHAKA, BANGLADESH April 2008
BACHELOR'S IN COMPUTER SCIENCE AND ENGINEERING (GPA 3.91/4.0) **(Class Valedictorian)**

CORE COMPETENCIES

EXPERTISE
Distributed Systems, Scalability, Algorithms, Software Engineering, Big Data, Object Oriented Design & Programming, Parallel Programming, Database Systems, Machine Learning, Artificial Intelligence, Deep Learning, NLP, Sentiment Analysis, Topic Modeling, OWASP, CWE

PROGRAMMING LANGUAGES
Python, Java, C#, C, C++, CUDA, MATLAB, JavaScript, PHP, XML, HTML5, CSS, SQL, MySQL

TOOLS AND TECHNOLOGIES
AWS (EC2, EMR, S3), IBM Cloud, Map Reduce, Hadoop, Pig, Hive, Cloudera, Junit, PyUnit, Selenium, Flask, CBMC, JBMC, SPIN, TLA+, Bootstrap, Kafka, Google Map API, Oracle, SQL Server, Android SDK, Tableau, Anaconda, TensorFlow2, Scikit-Learn, Pandas, SciPy, Matplotlib, NumPy, NLTK, Gensim, TextBlob, Turicreate, GitHub, LaTeX, IntelliJ, Google Colab, Linux

TECHNICAL PROJECTS

[Personal Project] Detecting Android malicious Apps Spring 2021
- Extract 28,170 features for each App; Use data sets containing 1,200 benign Apps and 2,500 malicious Apps; Use ensemble of multiple classifiers to improve the detection accuracy; Reach detection accuracy as 90.07% (Python, TensorFlow2)

[Personal Project] Analyze the Sentiment of Tweets from Twitter Data Fall 2020
- Compare two different topics and identify the popular one based on the most recent Tweets sentiment (Tweepy in Python, IBM Watson NLP)

[CSCI 59000 Big Data Management] App's user sentiment analysis in terms of reviews Fall 2018
- The dataset contains 2,357 Google PlayStore Apps with App's basic details and 10,680,22 number of reviews (*Python, MySQL, Selenium, Cloudera-CentOS-OS, Hadoop, Map Reduce, Sqoop, Hive).*

[CSCI 50700 Object Oriented Design & Programming] Online Marketplace Spring 2018
- Developed a distributed marketplace based on Java RMI using Object Oriented design patterns including architectural, structural, behavioral and infrastructure patterns.

[CSCI 53700 Distributed Computing] Distributed System in Java Fall 2017
- Developed a distributed system based on Java using a master object and four process objects that supports clock consistency with the concept of logical clock.

[CSCI 50400 Computer Organization] Prime Factorization with Parallel Execution in CUDA Spring 2017
- Implemented a factorized integer of size 10^16, an improved running time compared to the serial execution (*PyCUDA*).

WORK EXPERIENCE

Teaching Assistant, Indiana University - Purdue University Indianapolis, USA Jan 2017 – Present

- [CSCI 50900: Software Quality Assurance] | [ECE 49500: Principles of Software Design] | [CSCI 45000: Principles of Software Engineering] | [CSCI 36200: Data Structures]

Programmer Analyst Intern, Corteva Agriscience, USA May 2020 – Aug 2020

- Developed three front-end window applications and upgrade one module in environmental exposure assessment system (*C#, SQL server, Advanced Installer*)
- The applications greatly reduce the level of modeling effort, from weeks down to < 1 day

Assistant Professor, University of Asia Pacific, Bangladesh Oct 2013 – Dec 2016

- Courses taught: Compiler Design, Operating System, Software Development, Computer Programming
- Consistently ranked as the **best teacher** of the department achieving **96% rating** per student evaluation

Research Engineer, National ICT Research Center of Australia, Australia Sep 2012 – Aug 2013

- Designed and developed a secure embedded micro-kernel-based safety critical system (*C, AUTOSAR OS*)

RESEARCH EXPERIENCE

Security-related and Evidence-based Holistic Rating Framework to Identify the Right Apps in Distributed Platform [PhD research] Jan 2017 – Present

- Quantify the trust of software apps based on programmatic view and user perception (*Python, Selenium, MySQL, IBM Watson NLP API, Machine learning, Static code analysis, Linux*).

COVID CV: A System for Creating Holistic Academic CVs during a Global Pandemic [IU Office of the Vice President Funded Project] Sep 2020 – Present

- Color-coded CV from the user's data entries documenting the work and home life and categorizing corresponding events as good or bad. (*React JS, Spring BOOT, Java, MySQL, HTTPS*).

Community Data Analytics for Social Harm (CDASH) [NSF Funded Project] Aug 2017 – Present

- Building distributed software control systems for near real-time policing of heterogeneous social harm events (*AWS, C#, Java, Spring Boot, Kafka, MySQL, Google Map API*).

seL4-AUTOSAR: Safety Critical Systems with seL4 Sep 2012 – Aug 2013

- Design and implementation of a secure embedded microkernel-based Safety-Critical-System (*C, Linux*).

Automated Scenario Generation for Model Checker Trampoline OSEK/VDX OS Sep 2010 – Jul 2012

- Implement an automated environment model based on structural data dependency information of the source, able to reduce the memory space up to 38% and runtime up to 82% compared to other existing technique (*C, Linux*).

HONORS AND AWARDS

- Student Scholar, Grace Hopper Celebration 2020
- Best Presenter, IEEE 20th International Conference on Computer and Information Technology 2017
- Chair, Organizing Committee, The 2016 ACM-ICPC Asia Dhaka Regional Contest 2016
- Co-Contest Director, 2016 Bangladesh National Collegiate Programming Contest 2016
- Korean Government Scholarship 2009 - 2012
- Vice Chancellor's Gold Medal for securing first position in undergrad level 2008
- Finalist, The 2007 ACM-ICPC Asia Dhaka Regional Programming Contest 2007
- Champion, University of Asia Pacific application development contest 2007

PUBLICATIONS

1. Nahida Chowdhury, Rajeev R. Raje, “E-SERS: Enhanced Security-related and Evidence-based Holistic Ranking and Composition Framework for Distributed Services”, CRA-W Grad Cohort Workshop, 2021. (Tech Talk Presentation)
2. Nahida Chowdhury, Rajeev R. Raje, “SERS: A Security-related and Evidence-based Ranking Scheme for Mobile Apps”, Proceedings of the First IEEE International Conference on Trust, Privacy and Security in Intelligent Systems, and Applications, Los Angeles, CA, 2019. (published)
3. Nahida Chowdhury, Rajeev R. Raje, “A Holistic Ranking Scheme for Apps”, Proceedings of the 21st IEEE ICCIT, Dhaka, Bangladesh, 2018. (published)
4. Nahida Chowdhury, Rajeev R. Raje, “Disparity between the Programmatic Views and the User Perceptions of Mobile Apps”, Proceedings of the 20th IEEE ICCIT, Dhaka, Bangladesh, 2017. (published)
5. Nahida Chowdhury, Rajeev R. Raje, “SecureRank - Trust of mobile apps using subjective opinion”, The 2nd World Summit on Advances in Science, Engineering and Technology, 2019. (poster)
6. Nahida Chowdhury, Rajeev R. Raje, “A comprehensive ranking and selection of applications”, CRA-W Grad Cohort Workshop, 2019. (poster)
7. Nahida Chowdhury, Rajeev R. Raje, “TRR: Trust-Based Mobile Apps Selection and Ordering over Traditional Feedback Mechanism”, IUPUI Research Day, 2018. (poster).