A-SIT

Secure Information Technology Center – Austria

A survey on enhancing privacy awareness on Android

A-SIT

A survey on enhancing privacy awareness on Android

Autor:
Emina Ahmetovic
emina.ahmetovic@iaik.tugraz.at
Datum: 14.10.2022

Abstract/Zusammenfassung:
The prevalence of mobile devices in everyday life has led to the expansion of mobile apps on the market. While the apps evolve, providing a limitless number of possibilities and competing in design, there are rising concerns about privacy incidents on mobile devices via permission models. Users often do not understand the privacy implications that granting permissions carries, mainly because it is unclear how the app is processing privacy-sensitive resources. As a result, there are numerous cases of data leakage by the apps that users are not aware of. Researchers tackle this issue by introducing a field of meta-data analysis of the apps. By releasing the apps, developers provide meta information such as descriptions, privacy policies, categories, screenshots, etc. This data can further be used to predict the actual behavior of the apps and to measure the discrepancies between the actual behavior of the app and the described one, known as the description-to-permission fidelity in the literature.

In this project, we provide a literature overview of the ongoing work that focuses on enhancing the fidelity of the description in Android apps. We systematically review the latest progress in the field of application description analysis and elaborate on the techniques in the field of Natural Language Processing and Deep Learning that have shown potential to tackle the issue. Furthermore, we aim to address and discuss challenges and open research issues.

Contents

1. Introduction

The prevalence of mobile devices in everyday life led to a large number of mobile applications on the market. Both public and private sectors recognized the trend and turned to a *mobile-first* approach to provide services to a large population of mobile users. While the apps evolve over time, providing a limitless number of possibilities and competing in design, researchers point out the rising privacy invasions. In particular, one of the major concerns that the researchers have tackled is privacy incidents on mobile devices via permission models. As trivial as it seems, granting or denying permission the app requests often hides a challenging decision that can have a serious impact on the resources users would like to protect. As previous studies warn, granting certain permission requests can lead to privacy leaks that users are not aware of (Li, 2021) (Felt A. P., 2012) (Felt A. P., 2011) (Almuhimedi, 2015). While the security and privacy threat is a posing problem for all smartphone platforms, in this project, we will focus on Android as the dominant and most commonly used operating system, with over 2.8 billion active users and a global market share of 75% [1].

1.1. Users' understanding of permissions

One of the factors behind the rising privacy concerns is that users generally have misconceptions about how permissions work. Moreover, with increased application functionality, the number of permission requests also increases (Liu B. a., 2016). To assess permission requests, users rely on common sense, but they wonder why a beauty app[2] would want to access their contacts, location, and much other information. Especially alarming is the rising number of apps that lack correct mapping between the functionality and the permissions they claim. A study conducted by Shen et al. (Shen, 2021) revealed that one of the reasons behind the users' misunderstanding of the privacy implications of the permission model is insufficient information provided by the system. Smartphone systems act neutral and allow developers to provide explanations when requesting permissions. Developers often fail to provide an accurate and complete description that could justify dangerous permissions, making it harder for users to make an informative decision. In addition to the short descriptions, the change from install time to runtime model also introduced challenges for end users. In particular, users often fail to comprehend that the runtime model manages permission in groups, meaning that users grant or deny the entire permission group. For example, the READ CONTACTS and WRITE CONTACTS are both included in the CONTACTS group, and Android will ask users to grant the entire CONTACTS group if any individual permission is needed. If users are not aware of the privacy potential of each group and their corresponding permissions, it is harder for them to evaluate the security risks associated with that group. Lastly, the change from install time to runtime permission model, introduced in 2015 from Android 6, still poses a risk for users who are unaware that apps with a lower version of SDK cannot change to runtime permission model and that they grant permissions at installation time.

1.2. Application metadata analysis

Android developers provide different kinds of application metadata, such as description, category, ranking, privacy policy, review, and similar, when distributing apps. Leveraging the advances in Natural Language Processing and applying it to the application metadata has emerged as a novel technique that already showed promising results in the security and privacy domain, particularly for Android malware detection. In addition, analyzing and processing application metadata has also shown its

[1] Android Statistics (2022). https://www.businessofapps.com/data/android-statistics/
[2] https://cybernews.com/privacy/android-apps-are-asking-for-too-many-dangerous-permissions-heres-how-we-know/

potential to find inconsistencies between the app's described and actual behavior. The term in the literature, *description-to-behavior fidelity*, is widely used to address the gap between application metadata and application behavior. The new approach that complements traditional approaches, such as static and dynamic analysis, gives users more transparency in terms of their personal data processed by the app but also helps developers write better descriptions (Sen, 2021). However, during this work, we will focus mostly on the studies that measure *description-to-permission fidelity*. The motivation behind description-to-permission fidelity is the use of dangerous permissions that provide access to private resources should be well explained to users. Otherwise, if users cannot grasp the privacy requirements of the apps from descriptions, it creates mistrust, and those apps should be further analyzed. Users are often unaware of how mobile applications handle their private data. Even though privacy policy provides more information, researchers came to the conclusion that users might not pay enough attention to them. On the other hand, application descriptions written informally are a user-friendly way of informing users what functionality to expect from the application. This makes them an important and effective communication channel where developers convey application functionalities. Nevertheless, many researchers agree that relying only on application descriptions could cause many false positives since descriptions have limited capability to explain the use of all dangerous permissions. Therefore, combining descriptions with other metadata sources should enhance the analysis.

In the following sections, we will provide an overview of the most recent and popular studies that have achieved state-of-the-art results in this field. We will also outline some of the limitations of the work and propose future research objectives.

2. Background

In this section, we provide a short overview of the permission mechanism in Android, outline different analyses of applications, and discuss the use of Natural Language Processing in the security and privacy domain.

2.1. Permissions in Android

Android users are offered a vast number of mobile apps that provide a variety of functionalities and assistance in everyday life. Since the use of the applications can invade the privacy of users, the permission model is introduced as a mechanism where users have control over their data and can protect their assets. This means that the app needs to obtain explicit permission from a user to access or use a certain resource. Based on the scope of the restricted actions and access to the restricted resources, Android defines different permission levels that we introduce in the following:

2.1.1. Install time permissions

Install-time permissions[3] such as INTERNET, and similar are characterized by limited access to restricted resources. Since they execute actions with a minimal impact on the other app or system, they are automatically granted by the system at the install time (a time when a user installs the app). The install time permissions are not displayed to the users by default but rather can be found in the Play Store details. Unlike runtime permissions, they cannot be revoked, and there are no runtime checks. They are also granted to all users on the device. The normal[4] and signature[5] permission belong to the install-time permissions. The system will assign the *normal* protection level to the normal permissions and the

[3] Install time permissions. https://developer.android.com/guide/topics/permissions/overview#install-time
[4] Normal permissions. https://developer.android.com/guide/topics/permissions/overview#normal
[5] Signature permissions. https://developer.android.com/guide/topics/permissions/overview#signature

signature protection level to the signature permissions. However, the signature permissions are used only by the apps signed with the same certificate as the app that defined them. In that case, the system will grant them at the install time.

2.1.2. Runtime permissions

Runtime permissions[6], such as SMS, CONTACTS, CALENDAR, LOCATION, and similar, are assigned to the dangerous protection level since they allow access to the restricted data, and allow app to perform restricted action. They are requested at runtime (right before performing the restricted action) and need explicit approval from the user. Users should be aware that runtime permissions can access private data, which potentially contains sensitive information. In contrast to the install time permissions, users can revoke runtime permissions at any time.

calendar	READ_CALENDAR WRITE_CALENDAR
camera	CAMERA
contacts	READ_CONTACTS WRITE_CONTACTS GET_ACCOUNTS
location	ACCESS_FINE_LOCATION ACCESS_COARSE_LOCATION ACCESS_BACKGROUND_LOCATION ACCESS_MEDIA_LOCATION
microphone	RECORD_AUDIO
phone	READ_PHONE_NUMBERS READ_PHONE_STATE CALL_PHONE READ_CALL_LOG WRITE_CALL_LOG ADD_VOICEMAIL USE_SIP PROCESS_OUTGOING_CALLS ANSWER_PHONE_CALLS ACCEPT_HANDOVER
sensors	BODY_SENSORS ACTIVITY_RECOGNITION
sms	SEND_SMS RECEIVE_SMS READ_SMS RECEIVE_WAP_PUSH RECEIVE_MMS
storage	READ_EXTERNAL_STORAGE WRITE_EXTERNAL_STORAGE

Table 1: The table outlines the permission groups assigned to a dangerous protection level and their individual permissions. The overview is available at https://developer.android.com/guide/topics/permissions/overview.

[6] Runtime permissions. https://developer.android.com/guide/topics/permissions/overview#runtime

2.1.3. System and custom permissions

Permissions can also be categorized as system and custom[7]. In contrast to the system permission defined by the system, custom permissions are defined by the third-party apps that define how the resources and capabilities of the app are shared with other apps. When creating the custom permissions, a name and its corresponding protection level need to be defined by the app that creates the custom permission, as well as the label and the description. Additionally, a permission group can also be defined, which can be either a system group or the custom group.

2.2. Analysis of Android Applications

There are several types of Android app analysis that have been proposed in previous works with the aim of detecting Android malware and permission misuse. We divide them into static analysis, dynamic, and taint analysis.

2.2.1. Dynamic analysis

Dynamic analysis is a process of detecting and analyzing the malicious behavior of the application. Dynamic analysis works by tracking and capturing the data flow while the application is running. One of the important features of dynamic analysis is that, unlike static analysis, monitoring and tracking real-time private data is not affected by obfuscation, encryption, or similar factors. However, since dynamic analysis works with real-time data, it is not capable of detecting privacy leaks that have not been triggered at the runtime. An additional limitation is high resource consumption due to the real-time operation, making these operations not affordable on some mobile devices (Kang, 2021).

2.2.2. Static analysis

In contrast to dynamic analysis, static analysis is done before running an app. Static analysis is a process of extracting application-specific features such as permissions and API calls by analyzing the source code or its binary representation. Usually, it requires the reverse engineering technique of the source code to detect malicious behavior from an app. Since the static analysis works on the source code level, the detection of the malware is affected by some evasion techniques, such as obfuscation of the code or dynamic code lading. Additionally, static analysis can contribute to a larger number of false positives. A popular open-source tool for static analysis on Android is FlowDroid (Arzt, 2014).

2.2.3. Taint analysis

Taint analysis is an information flow analysis technique used to track the flow of sensitive information from defined sources to defined sinks. Sources are defined as resources users want to protect, such as location or phone number, whereas sinks are defined as the points where the resources could leave the device, such as methods related to the transmission of the data. The taint analysis aims to identify the data leakage by observing the information flow between sources and sinks, and if the data from sources will reach a sink (Zhang J. a., 2021). It can be either static and dynamic, and it is mostly used to detect privacy leaks. A widely used tool for static taint analysis in Android apps is TaintDroid (Enck, 2014).

2.3. Natural Language Processing

Natural Language Processing, as one of the AI branches, focuses on human language understanding. It is a powerful technology that has many uses cases in daily life. One of the most common uses of the NLP are email filters, virtual assistance such as Apple's Siri and Amazon's Alexa, online search engines, chatbots, sentiment analysis of a brand on social media channels, machine translation, natural language

[7] https://developer.android.com/guide/topics/permissions/defining

generation, and much more[8]. NLP combines computational linguistics with another model, such as machine learning or deep learning to enable a computer to understand human language by providing textual or voice input. NLP analyses human language from different perspectives: syntax, semantics, pragmatics, and morphology[9]. Many NLP tasks, including application description analysis, are related to syntactic and semantic analysis of the language. While syntax analysis identifies the word order and dependency relationships between words in the sentence, semantic analysis focuses on the meaning of the words. Semantic analysis is also considered one of the most demanding tasks in NLP due to the nature of a language. Human language is sparse and filled with ambiguities, homophones, sarcasm, idioms, metaphors, and similar variations, which poses a great challenge to extract the meaning accurately. Commonly used techniques for semantic and syntax analysis are keyword extraction, topic modeling, tokenization, part-of-speech tagging, dependency parsing, lemmatization & stemming, stopword removal, and similar.

3. State-of-the-art studies

In this section, we provide a review of the studies that have different NLP and DL techniques to facilitate the application metadata analysis. The metadata analysis of the Android applications typically includes the following steps: collecting and processing APKs, extracting application-relevant features, designing and implementing a model for text processing, and, lastly, evaluating the model on real-world apps. We start this section by introducing APK collection and feature extraction. Furthermore, we introduce four pioneering frameworks that shaped the research direction. Lastly, we give an overview of the latest research done with deep learning.

3.1. APK collection and processing

The Android applications are packaged as Android Package Kit (APK) during the build process. Each APK file is a zip archive that contains the following folders:

- The META-INF directory, used to store signature information to verify whether the APK is complete.
- The res/ directory, stores the resource files, such as GUI XML layout, colors, graphic, images, icons and similar.
- AndroidManifest.xml, used to describe the metadata of the application, such as required permissions, app components, and similar.
- Classes.dex, All classes in Dalvik bytecode
- Resources.arsc, a compiled binary resource file

Android distribution has various sources; apps can be installed from the official Google Play store, and third-party app stores such as Amazon, F-Droid, and Samsung. To create a dataset for processing, researchers mostly implement their own crawlers that download apps from defined stores and with defined parameters, such as language, country, popularity, and similar. Optionally, there are some open datasets for Android applications that can be used to fetch the APKs, and the one with the largest dataset is currently Androzoo (Allix, 2016) (Li L. G., 2017). After collecting a significant amount of APKs, the following steps include extracting the relevant data from the APKs for further study. Usually, for feature extraction purposes, numerous tools are available. They receive the raw APKs as input and

[8] https://www.ibm.com/cloud/learn/natural-language-processing
[9] https://monkeylearn.com/natural-language-processing/

provide output that contains the disassembled codes, APIs, permissions, and intents (Qiu, 2020). Some of the most popular open-source tools for reverse engineering tools are Apktool[10], Java decompiler JADX[11], and Java/Android Bytecode Viewer[12].

3.2. Studies

One of the first research done in the domain of application description analysis was done by Pandita et al. (Pandita, 2013). The authors have proposed a framework called Whyper, which checks if the need for dangerous permissions used by the app is introduced in the description. It is a pioneering work that uses Natural Language Processing (NLP) to extract the semantic meaning from descriptions with the goal of detecting text that refers to permissions. By addressing the limitations of the keyword-based approach, such as confounding meaning and semantic inference, they build the semantic model by manually analyzing Android API documents. Their approach consists of five components. The *preprocessor* receives descriptions as input and performs some of the processing tasks on descriptions, such as period handling, sentence boundaries, and named entity and abbreviation handling. Furthermore, the *NLP parser* receives preprocessed documents as input and does the annotation of every sentence within each document using Stanford Parser. The *intermediate-representation generator* takes the output of the NLP parser and generates the First-Order Logic (FOL) representation of a sentence. The *Semantic Engine (SE)* checks if the semantic graphs of permissions, derived from application programming interface (API) documents by *Semantic-Graph Generator*, match description sentences or not. They evaluate Whyper against three permissions: READ CALENDAR, READ CONTACTS, and RECORD AUDIO, as frequently-used permissions that protect the privacy and security-sensitive resources, and they have evaluated the framework on the dataset of 581 apps.

Discussing the limitations of Whpyer, in particular using a fixed vocabulary derived from Android API documents and synonyms of keywords there, and addressing issues in Whypers methodology such as limited semantic information, lack of associated API, and lack of automation, Qu et al. (Qu, 2014) propose fully automated framework AutoCog. As they are the first to introduce the term description-to-permission fidelity, the authors state that users should gain an intuitive idea about the security and privacy functionality of the application by reading descriptions and that descriptions should give an idea about the requested permissions. Similar to Whyper, they use NLP techniques to infer permission use from app descriptions; however, their key component for semantic extraction is Explicit Semantic Analysis (ESA) which leverages an extensive knowledge base (i.e., Wikipedia), in contrast to using a dictionary-based corpus like WordNet used in Whyper. The authors point out that AutoCog can be used by both developers and end users; developers can utilize AutoCog to write better descriptions, and final users can utilize it to evaluate the risk of running the applications. Lastly, AutoCog can be used by the application market to improve its trustworthiness.

Checking app behavior against app descriptions is also done by Gorla et al. (Gorla, 2014). They propose CHABADA framework, which uses description and the called APIs to detect anomalies. Chabada uses the Latent Dirichlet Allocation (LDA), a tool and technique for topic modeling, on descriptions to identify the main topic for each application and to further cluster applications by related topics. In each cluster, they extract sensitive APIs and use an unsupervised clustering algorithm to find outliers with respect to API usage. Their dataset[13] of 22500+ applications from the Google Play Store is available for reproducibility purposes.

10 Apktool https://ibotpeaches.github.io/Apktool/
11 JADX https://github.com/skylot/jadx
12 Bytecode https://github.com/Konloch/bytecode-viewer
13 CHABADA dataset. http://www.st.cs.uni-saarland.de/chabada/

Recognizing the importance of the description as the most effective communication channel between end users and app developers, the work done by Watanabe et al. (Watanabe, 2015) asks why the app descriptions fail to refer to the use of privacy-sensitive resources. To answer this question, their introduced ACODE framework that leverages a keyword-based approach. ACODE uses two-stage filter combining static code and keyword-based text analysis. The static code analysis checks whether the code includes APIs/URIs that require permission to access private resources and checks if APIs/URIs are actually callable by tracing function calls. The description analysis uses information retrieval (IR) and NLP techniques to distinguish between apps with text descriptions that refer to privacy-sensitive resources and apps without such descriptions. In the keyword extraction approach, ACODE extracts the keywords that have the largest relevance weights, which is a technique that measures a relation between the relevant and non-relevant document distributions for a term modulated by its frequency. They evaluate ACODE on a dataset of 200 000 apps that are from the official as well as from third-party markets. Authors report comparable results; however, unlike other studies, they include both English and Chinese language.

Name	Year	Number of apps	Number of permissions	Analysis technique
Whyper: Towards automating risk assessment of mobile applications	2013	581	3	Semantic analysis
AutoCog: Measuring the Description-to-permission Fidelity in Android Applications	2014	83 656	11	Semantic analysis
CHABADA: Checking app behavior against app descriptions	2014	22500+	-	Topic modeling
ACODE: Understanding the Inconsistencies between Text Descriptions and the Use of Privacy-sensitive Resources of Mobile Apps	2015	200 000	11	Keyword-based approach

Table 2: The table provides an overview of the four pioneering frameworks in the field of application description analysis; Whyper, AutoCog, Chabada, and Acode.

3.3. Enhancing Description-to-Permission Fidelity with Deep Learning

Machine learning (ML) has played an important role in many mobile security fields, such as Android malware detection and classification. Nevertheless, deep learning (DL) has shown the potential to alleviate the obstacles of the traditional machine learning problem that involves human intelligence. Traditional machine learning approaches unconditionally apply feature engineering, where hand-crafted features are expensive and prone to error. For instance, applying machine learning in Android

malware detection often requires security experts to extract features manually for malware detection. On the other hand, deep learning provides a certain level of generalization as it allows features to be learned automatically and alleviates the tasks of feature engineering. In particular, when enough data is available, DL is more efficient in capturing the semantic meaning from the Android application (Qiu, 2020). In this section, we provide an overview of the studies that have facilitated deep learning techniques to tackle description analysis.

AC-Net (Feng, 2019) is one of the first works that combines Natural Language Processing techniques and deep learning to detect the inconsistency between mobile app descriptions and permissions. AC-Net consists of two phases: the training phase and the assessing phase. The training phase consists of description preprocessing and model construction. In the processing, techniques such as *sentences split*, *stopword removal and stemming*, and *word vectors initialization* are applied to raw descriptions to convert them into a form applicable for further model training. For model construction, they use a deep neural network model for text classification called *TextGRU*, which is a variant of the Recurrent Neural Network (RNN) architecture (Graves, 2012). To tackle the known issue of vanishing gradient in the RNN, they leverage the Gated recurrent unit (GRU) (Cho, 2014) to learn long-term dependencies. In the assessing phase, authors provide an unknown description and do the preprocessing as in the training phase and obtain output as binary probability distributions for each declared permission. They included the 16 most common permissions classified into three protection levels: dangerous, normal, and signature. The used dataset[14] is available for validation and reproducibility purposes.

Similar to the AC-Net, Alecakir et al. (Alecakir, 2021) used RNN with GRU to detect if the use of dangerous permissions is explained in the description. They introduce two models: *sentence-based* and *document-based*. The sentence-based model is used to identify permission sentences in the description like it is done in the previous studies. Unlike the other studies, a document-based model is an approach based on a hierarchical attention network that represents descriptions as a whole. Their study is one of the first attempts to apply a novel *attention mechanism* recently used in various NLP tasks, such as language translation. They propose a two-level attention mechanism to extract the meaning of each description hierarchically in the proposed document-based model, pointing out that, unlike the previous work, they do not process each word in the description equally as they don't have the same effect on the meaning. They share with the community their manually labeled dataset DesRe[15]. DesRe contains descriptions of applications for three permissions, namely READ CONTACTS, RECORD AUDIO, and STORAGE, and in addition, it also contains the five most helpful reviews that can be used in review-to-behavior studies. They evaluate both *sentence-based* and *document-based* models on AC-Net and DesRe datasets and compare them with models in previous work. The evaluation points out document-based model performs significantly better than the sentence-based model.

The work done by Feichtner et al. (Feichtner, 2020) combines NLP with Convolutional Neural Networks (CNN) to predict the likelihood that an app will require certain permission. They leverage the LIME model explanation algorithm (Ribeiro, 2016) to estimate the impact of the works on obtained permission results. The authors have evaluated their solution on more than 77,000 real-world app descriptions taking into consideration only dangerous permissions split into nine permission groups: CALENDAR, CALL LOG, CAMERA, CONTACTS, LOCATION, MICROPHONE, PHONE, SMS, and STORAGE. To collect the applications, they used the PlayDrone[16] dataset.

Furthermore, a framework named FCDP (Wu Z. a.-J., 2020) contributes to calculating description-to-permissions fidelity in Android Apps. Their approach leverages the attention mechanism of Bidirectional

14 AC-Net dataset. https://github.com/LinkyVon/AC-Net
15 DesRe dataset. https://wise.cs.hacettepe.edu.tr/projects/security-risks/dataset/
16 PlayDrone dataset. https://systems.cs.columbia.edu/projects/playdrone/

Long Short-Term Memory (Bi-LSTM). LSTM (Liu P. a., 2017) (Graves, 2012) is a variant of the RNN that provide a solution for the known problem of the vanishing and exploding gradients in RNN. As in the previous work, the attention mechanism was used to find words with a bigger semantical meaning to the sentence. FCDP was evaluated on 64,265 apps.

Name	Year	Number of apps	Number of permissions	Analysis technique
AC-Net: Assessing the Consistency of Description and Permission in Android Apps	2019	AC-Net dataset	16	RNN GRU
FCDP: Fidelity Calculation for Description-to-Permissions in Android Apps	2020	64 265	16	RNNLSTM
Understanding Privacy Awareness in Android App Descriptions Using Deep Learning	2020	77 758	9	CNN LIME
Attention: there is an inconsistency between android permissions and application metadata!	2021	DesRe dataset	3	RNN GRU, attention mechanism

Table 2: The table provides an overview of the four frameworks that use deep learning in the description-to-permission fidelity problem.

4. Related work

Many studies focus on enhancing the application description analysis by considering application description together with other types of metadata. FideDroid (Wu Z. a.-J., 2021) is a framework that considers the application category and permission common for that category. The authors point out that permissions common for one specific category are rarely elaborated in the description, which produces false positives in predicting permissions from app descriptions.

Another reason that could affect the results of the fidelity calculation is the impact of the third-party libraries (TPL). In work done by Zhang et al. (Zhang C. a., 2018) authors state that apps that use TPL will be identified as outliers which causes false positives, as application description typically would not describe the functionality of the TPL. The authors concluded that separating whether the malicious behavior is caused by the custom code or in the TPL would affect the studies, and a significant number of apps would no longer be identified as an outlier.

A study done by Yu et al. (Yu, 2017) revisited the previous approaches and concluded that analyzing only descriptions and permissions may lead to many false positives. They propose a TAPVerifier, an approach that exploits the privacy policy of applications and their bytecode to enhance inconsistency detection.

Few studies have elaborated on how users could learn from the experiences of others by measuring review-to-behavior fidelity. In a study done by Tain et al. (Tian, 2015), it was found that reviews,

especially negative ones, can be useful in making better security and privacy decisions on mobile apps. Other studies took a similar approach and performed review mining to predict mobile app behavior (Wu J. a., 2017) (Kong, 2015).

In addition, measuring the impact of user reviews on security and privacy updates was done by Nguyen et al. (Nguyen, 2019). The authors have automatically classified reviews into security-privacy-related and non-security-privacy-related ones. Furthermore, they mapped security-and-privacy-related reviews to app versions and corresponding updates and concluded that security-privacy-related reviews had triggered a security and privacy-related update.

5. Conclusion

The popularity of mobile apps is rising every day. According to statistics, around 70 % of the population owns a mobile phone, whereas Android belongs to the most used platform. There are currently 3.45 million Android applications on the Play Store, and 3,739 apps are added every day[17]. However, the success of mobile applications also attracts attackers who profit from access to private users' data. The situation is aggravated by the fact that users are unaware of the privacy and security risks that applications on the phone carry. NLP, in combination with DL, has shown the potential to tackle this issue by extracting the semantic meaning from application description and calculating description-to-permission fidelity. The novel research direction of application metadata analysis has proven successful and can be combined with traditional static and dynamic analysis approaches. In this project, we aim to provide a quick literature overview of the topic. Our goal was to summarize the NPL and DL techniques that have been applied in concrete studies and to shed light on work that tries to bridge the semantic gap between application description and application behavior. We have concluded that the evolving trend of deep learning will continue to enhance this research direction. Future work could include different deep neural network architectures that will be able to perform the analysis faithfully. One of the limitations of the previous work was taking only app descriptions into account. Future work in the area should combine different metadata sources, such as categories, reviews, etc., to enhance fidelity calculation and reduce false positives.

[17] https://appinventiv.com/blog/google-play-store-statistics/

Bibliography

Alecakir, H. C. (2021). Attention: there is an inconsistency between android permissions and application metadata! *International Journal of Information Security*, 20(6):797–815.

Allix, K. B. (2016). Androzoo: Collecting millions of android apps for the research community. *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)* (S. 468--471). IEEE.

Almuhimedi, H. a. (2015). Your location has been shared 5,398 times! A field study on mobile app privacy nudging. *Proceedings of the 33rd annual ACM conference on human factors in computing systems*, (S. 787--796).

Arzt, S. a. (2014). Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. *Acm Sigplan Notices*, 49(6):259–269.

Cho, K. V. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint*.

Enck, W. a.-G. (2014). Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. *ACM Transactions on Computer Systems (TOCS)*, 32(2):1–29.

Feichtner, J. a. (2020). Understanding privacy awareness in android app descriptions using deep learning. *Proceedings of the tenth ACM conference on data and application security and privacy*, 203--214.

Felt, A. P. (2011). Android permissions demystified. *Proceedings of the 18th ACM conference on Computer and communications security*, (S. 627--638).

Felt, A. P. (2012). Android permissions: User attention, comprehension, and behavior. *Proceedings of the eighth symposium on usable privacy and security*, (S. 1--14).

Feng, Y. a. (2019). AC-Net: Assessing the consistency of description and permission in Android apps. *IEEE Access (7)*, 57829--57842.

Gorla, A. a. (2014). Checking app behavior against app descriptions. *Proceedings of the 36th international conference on software engineering*, 1025--1035.

Graves, A. (2012). Long short-term memory. *Supervised sequence labelling with recurrent neural networks*, 37--45.

Kang, H. a. (2021). {A modified flowdroid based on chi-square test of permissions. *Entropy*, 23(2):174.

Kong, D. a. (2015). Autoreb: Automatically understanding the review-to-behavior fidelity in android applications. *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 530--541.

Li, L. G. (2017). Androzoo++: Collecting millions of android apps and their metadata for the research community. *arXiv preprint arXiv:1709.05281*.

Li, R. a. (2021). Android Custom Permissions Demystified: From Privilege Escalation to Design Shortcomings. *2021 IEEE Symposium on Security and Privacy (SP)* (S. 70--86). IEEE.

Liu, B. a. (2016). Follow my recommendations: A personalized privacy assistant for mobile app permissions. *Twelfth symposium on usable privacy and security (SOUPS 2016)*, (S. 27--41).

Liu, P. a. (2017). Adversarial multi-task learning for text classification. *arXiv preprint arXiv:1704.05742*.

Nguyen, D. C. (2019). Short text, large effect: Measuring the impact of user reviews on android app security & privacy. *2019 IEEE symposium on Security and Privacy (SP)*, 555--569.

Pandita, R. X. (2013). {WHYPER}: Towards automating risk assessment of mobile applications. *USENIX Security Symposium (USENIX Security 13)*, 527–542.

Qiu, J. a. (2020). A survey of android malware detection with deep neural models. *ACM Computing Surveys (CSUR)*, 53(6):1–36.

Qu, Z. a. (2014). Autocog: Measuring the description-to-permission fidelity in android applications. *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, 1354--1365.

Ribeiro, M. T. (2016). " Why should i trust you?" Explaining the predictions of any classifier. *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 1135--1144.

Sen, S. a. (2021). Android Security using NLP Techniques: A Review. *arXiv preprint arXiv:2107.03072*.

Shen, B. a. (2021). Can Systems Explain Permissions Better? Understanding Users' Misperceptions under Smartphone Runtime Permission Model. *30th USENIX Security Symposium (USENIX Security 21)*, (S. 751--768).

Tian, Y. a. (2015). Supporting privacy-conscious app update decisions with user reviews. *Proceedings of the 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices*, 51--61.

Watanabe, T. a. (2015). Understanding the inconsistencies between text descriptions and the use of privacy-sensitive resources of mobile apps. *Eleventh Symposium On Usable Privacy and Security (SOUPS 2015)*, 241--255.

Wu, J. a. (2017). Pacs: Pemission abuse checking system for android applictions based on review mining. *2017 IEEE Conference on Dependable and Secure Computing*, 251--258.

Wu, Z. a.-J. (2020). FCDP: Fidelity calculation for description-to-permissions in Android apps. *IEEE Access (9)*, 1062--1075.

Wu, Z. a.-J. (2021). Enhancing fidelity of description in Android apps with category-based common permissions. *IEEE Access (9)*, 105493--105505.

Yu, L. a. (2017). Enhancing the description-to-behavior fidelity in android apps with privacy policy. *IEEE Transactions on Software Engineering*, 44(9):834–854.

Zhang, C. a. (2018). Re-checking App Behavior against App Description in the Context of Third-party Libraries. *SEKE*, 665--664.

Zhang, J. a. (2021). Analyzing android taint analysis tools: FlowDroid, Amandroid, and DroidSafe. *IEEE Transactions on Software Engineering*.