



Olukoya, O., Mackenzie, L. and Omoronyia, I. (2020) Security-oriented view of app behaviour using textual descriptions and user-granted permission requests. *Computers and Security*, 89, 101685. (doi: [10.1016/j.cose.2019.101685](https://doi.org/10.1016/j.cose.2019.101685))

There may be differences between this version and the published version. You are advised to consult the publisher's version if you wish to cite from it.

<http://eprints.gla.ac.uk/205065/>

Deposited on 5 December 2019

Enlighten – Research publications by members of the University of Glasgow
<http://eprints.gla.ac.uk>

Security-Oriented View of App Behaviour using Textual Descriptions and User-Granted Permission Requests

Oluwafemi Olukoya^{a,*}, Lewis Mackenzie^a and Inah Omoronyia^a

^a*School of Computing Science, University of Glasgow, Scotland, United Kingdom*

ARTICLE INFO

Keywords:

Mobile Security
Malware
Android Malware
Android Security
Android Privacy

ABSTRACT

One of the major Android security mechanisms for enforcing restrictions on the core facilities of a device that an app can access is permission control. However, there is an enormous amount of risk with regards to granting permissions since 97% of malicious mobile malware targets Android. As malware is becoming more complicated, recent research proposed a promising approach that checks implemented app behaviour against advertised app behaviour for inconsistencies. In this paper, we investigate such inconsistencies by matching the permission an app requests with the natural language descriptions of the app which gives an intuitive idea of user expected behaviour of the app. Then, we propose exploiting an enhanced app description to improve malware detection based on app descriptions and permissions. To evaluate the performance, we carried out various experiments with 56K apks. Our proposed enhancement reduces the false positives of the state-of-the-art approaches, Whyper, AutoCog, CHABADA by at least 87%, and TAPVerifier by at least 57%. We proposed a novel approach for evaluating the robustness of textual descriptions for permission-based malware detection. Our experimental results demonstrate a high detection recall rate of 98.72% on 71 up-to-date malware families and a precision of 90% on obfuscated samples of benign and malware apks. Our results also show that analysing sensitive permissions requested and UI textual descriptions provides a promising avenue for sustainable Android malware detection.

1. Introduction


Permission control is one of the major Android security mechanisms (Barrera et al., 2010; Felt et al., 2011c). Permissions are requested by applications to access specific operations or core functionalities of a device (Enck et al., 2011). Some of the sensitive information they provide access to includes phone numbers, address book, precise locations, and SMS messages, thereby making privacy an important challenge in the Android permission model. The popularity of the Android platform has also been studied in parallel with a continual increase in the number of malicious applications (Felt et al., 2011b; Zhou et al., 2012). For example, Symantec's¹ latest threat intelligence proposes that one of the ways to stay protected from mobile malware is by paying close attention to the permissions requested by apps. Whenever a new Android app is installed, users run the risk of the app being malware, since 97% of malicious mobile malware targets Android (Kelly (2014)). However, users do not fully understand the risk of granting permissions and consequent implications (Felt et al. (2012); Kelley et al. (2012); King et al. (2011)).

Many detection techniques based on static analysis (Arp et al., 2014; Chen et al., 2013; Zhou et al., 2012; Grace et al., 2012; Feng et al., 2014; Aafer et al., 2013; Mariconti et al., 2017) and/or dynamic analysis (Yan and Yin, 2012; Tam et al., 2015; Rastogi et al., 2013a; Enck et al., 2014; Xue et al., 2017) have been proposed to detect mobile malware.

To detect malware, a signature-based technique extracts malicious behaviours as signatures from analyzing the semantics of known malware. However, the signature-based technique is challenging, as malicious behaviour often appears to be indistinguishable from that of benign apps. For example, an app that tracks the user's current location may be suspicious, but that is the normal behaviour of benign navigation apps or map applications. The question is then - whether the program behaves as advertised and not necessarily, whether it matches a specific pattern or not. Alternative approaches use machine-learning based classifiers for detecting malicious Android apps (Arp et al., 2014; Mariconti et al., 2017; Zhang et al., 2014, 2015a). Recent studies (Laskov et al., 2014; Xu et al., 2016; Rastogi et al., 2013b) show that the performance of the classifiers can be degraded and evaded by malware variants.

To increase the robustness of malware detection, recent research has suggested a promising approach that models stealthy behavior by checking the *implemented* program behavior against the *advertised* program behaviour. These approaches (e.g. AsDroid (Huang et al., 2014), Whyper (Pandita et al., 2013), AutoCog (Qu et al., 2014), CHABADA (Gorla et al., 2014)) profile an app's expected behaviour by extracting the semantic patterns from its description and characterizing the app's real behavior by examining the permission required by the app. TAPVerifier (Yu et al., 2016) extracted the app's semantic meaning from its privacy policies. DroidSIFT (Zhang et al., 2014) builds UI-dependent behavioural graphs to understand whether a sensitive action is likely benign or malicious. DescribeME (Zhang et al., 2015a) generate security-centric app descriptions, based on program analysis. To enhance user perception of security awareness, Android markets directly

*Corresponding author

 o.olukoya.1@research.gla.ac.uk (O. Olukoya);

Lewis.Mackenzie@glasgow.ac.uk (L. Mackenzie);

Inah.Omoronyia@glasgow.ac.uk (I. Omoronyia)

ORCID(s): 0000-0002-9510-5141 (O. Olukoya); 0000-0002-0868-0456 (L. Mackenzie); 0000-0001-5357-0945 (I. Omoronyia)

present two classes of literal app behaviour elements: i) permission requests and ii) textual descriptions. The textual description represents the advertised behaviour and permission request is a proxy for the implemented behaviour. Our work is a substantial improvement over the state-of-the-art for the following reasons:

1. **Textual Semantics from App Description:** We propose that an app's description cannot detail all its privacy-related behaviour, as developers could decide to describe the app in ways that do not specify sensitive behaviour. Also, since an app's descriptions on app distribution platforms, like Google Play, has a character limit², it cannot detail all behaviours, which can lead to false negatives. To this end, we aim to answer the question of gathering enough semantics for behavioural element mismatch. As opposed to the current state-of-the-art semantic gathering from description (Pandita et al., 2013; Qu et al., 2014; Gorla et al., 2014) or the combination with the privacy policy (Yu et al., 2016), we extract information from ten different sources to determine the semantic meaning of app descriptions. We also argue that different privacy-sensitive behaviour can be described in different aspects of an app's metadata. One behaviour could be described in the screenshots and featured graphics while another behaviour in the *What's new in this version* listing. Focusing on the description and privacy policy alone ignores other behaviours, potentially leading to an increased incidence of false positives. In the Android malware community, the behavioural element is synonymous to dynamic analysis where the elements are behavioural characteristics which have been extracted while the sample has been executed. In this work, we use the term for a different purpose, where we define behavioural elements are static features that describe the advertised behaviour of an app in terms of the permission it requests and the description of the functionality it provides.
2. **Permission State Space:** These approaches (Pandita et al., 2013; Qu et al., 2014; Yu et al., 2016; Gorla et al., 2014) have considered a maximum of 11 permissions, which is less than 20% of the permission space in Google Play, making a generalization of behavioural element mismatch challenging. For example, Google Play identifies 26 dangerous permissions that directly impact on a user's privacy, while the current state-of-the-art only considers six. Therefore, instead of developing permission semantics for 11 permissions, we integrate permission semantics for the 66 permissions as of API level 26 (Android 8.0).
3. **Semantic Meaning:** WHYPER (Pandita et al., 2013) and AutoCog (Qu et al., 2014) both leveraged a compositional vector grammar parser by Stanford to obtain typed dependencies. It was stated in Pandita et al. (2013) and Qu et al. (2014) that the major cause of the false positives and false negatives are the incorrect parsing of sentences by underlying NLP infrastruc-

ture. We have leveraged a state-of-the-art neural network dependency parser (Chen and Manning, 2014) that outperforms other parsers in speed and accuracy. The more accurate the parsing of a sentence by underlying NLP infrastructure, the more it reduces the common erroneous detection of the system, thereby reducing false negatives and false positives.

4. **Semantic Correlation:** The key component for semantic extraction in our design is the use of state-of-the-art distributed word embedding models that provide a dense vector representation of words that surround the target word. We use the Stanford word-embedding algorithm *GloVe*-(*Global Vectors for Word Representation*) (Pennington et al., 2014) trained on Wikipedia data comprising 6 billion tokens and a 40,000-word vocabulary, to find the semantic correlation between semantic patterns observed from the app description and a semantic model of requested permissions. Such a superior analysis further mitigates the problem of limited semantic information.

The main contributions of this work are summarized below:

- **Enhancing State-of-the-art approaches:** The proposed techniques of gathering app descriptions from different sources, enhancing the permission-state space, construction of semantic models and extraction of semantic meaning and semantic correlation provides a significant improvement upon the state-of-the-art approaches in investigating description-to-permission fidelity in Android apps. The proposed enhancement outperforms the current approaches in the extraction of semantic meaning from app descriptions.
- **Characterizing App Behaviour:** An approach that demonstrates that app descriptions and behavioural elements (e.g., a fine-grained stratification of Android permissions) are a promising way of enhancing a user perception of the actual behaviour of Android apps and the detection of malicious apps. This approach provides a technique to determine the semantic meaning of app descriptions, construct semantic models to match natural language text to permissions, and characterizes the app behaviour as malicious or benign based on the discrepancies in the semantic correlation between semantic patterns observed from the app description and the semantic model of requested permissions. The behaviour element mismatch characterization proposed in this study can be a feature for enhancing permission-based malware detection, where informative features are required to describe the behaviour of an app.
- **Robustness of Technique for Malware Detection:** We also evaluated the robustness of the proposed app behaviour characterization technique for permission-based malware detection. Our experimental re-

sults demonstrate a high detection sensitivity rate of 98.72% on 71 up-to-date malware families and precision of 90% on 6066 obfuscated samples of benign and malware apks. This shows that the proposed approach can enhance permission-based malware detection approach. We also demonstrate that analysing sensitive permissions requested with UI textual descriptions provides a promising avenue for long-span malware detection. This was achieved by investigating the performance deterioration of our approach with six state-of-the-art malware detectors for Android. For the reproducibility of research, the enhanced malware descriptions gathered and data results for the malware families investigated are publicly available. We also demonstrate that analysing sensitive permissions requested and UI textual descriptions provides a promising avenue for sustainable Android malware detection. The public repository also contains the package identifiers and results of all the samples investigated in the experimental evaluation. <https://www.dropbox.com/sh/dl4g2fqivc7xijo/AAAWTGM\G6NnXXYBoHdQQXIpta?dl=0>

2. Related Works

2.1. Permission Analysis

The effectiveness of the Android permission system in helping users make informed security decisions against malware was investigated by Felt et al. (2012). The complexity of Android permission requests is studied by Frank et al. (2012). Barrera et al. (2010) investigated 1100 most popular Android apps using the Self-Organizing Map (SOM) algorithm to perform visualization of permission-based systems. Felt et al. (2011a) developed a tool, *Stowaway*, to detect over-privilege in Android apps, a situation in which an Android app requests more permission than necessary. In Wei et al. (2012), a study on the permission evolution in the Android ecosystem and its usage was presented. Au et al. (2012) developed *PScout* to extract permission specification APIs from Android OS source code with static analysis. Backes et al. (2016) revisited the use-case of mapping Android permission to framework/SDK/API methods and presented novel mappings that could not be discovered due to insufficient knowledge about Android framework internals in *PScout* and *Stowaway*. In further addressing the complexity of demystifying the Android permission API specification, Aafer et al. (2018) proposed *ARCADE*, to derive a precise protection specification for Android APIs, using path-sensitive analysis and a novel graph abstraction technique. Zhang et al. (2013) proposed a dynamic analysis framework, *VetDroid*, to capture anomalies in permission use behaviour by examining the internal sensitive behaviours of the app. In contrast to all these approaches, our work is mainly based on investigating stealthy behaviours of Android applications using a permission sensitivity index and the behavioural elements mismatch.

2.2. Behavioural Elements Mismatch in Android Apps

Studies have been proposed to determine the mismatch between an app's description and its real behaviour to identify anomalies in Android apps (Pandita et al., 2013; Qu et al., 2014; Gorla et al., 2014; Yu et al., 2016; Huang et al., 2014). The common denominator among these methods is that they use app descriptions to infer expected behaviours and employ the permissions/APIs used by the app to represent its behaviours. Whyper (Pandita et al., 2013) is the pioneer detection system based on the behavioural element mismatch. It maps an app's description to three different permissions. AutoCog (Qu et al., 2014) handles more permissions (11) with better performance than Whyper (Pandita et al., 2013). CHABADA (Gorla et al., 2014) combines descriptions and invoked APIs to find suspicious apps with abnormal API usages. Asdroid (Huang et al., 2014) identifies stealthy behaviour as a semantic mismatch between the program behaviour with the user interface. TAPVerifier (Yu et al., 2016) proposed combining an app's privacy policy, bytecode, description and permission to obtain more information about its expected behaviour. These approaches focus on measuring description-to-permission fidelity in Android apps.

In contrast to these approaches, our work proposes a technique for characterizing app behaviour as benign or malware using a fine-grained stratification of Android permissions and app descriptions. The app behaviour is a function of the threshold between the semantics patterns observed from the description and a semantic model of requested sensitive permissions. Firstly, our technique outperforms current approaches of extracting semantic meaning from app descriptions. Secondly, the proposed app behaviour characterization approach is promising and reliable for permission-based malware detection approach. Thirdly, the proposed approach is robust and provides a promising avenue for sustainable Android malware detection. Droid-SIFT (Zhang et al., 2014) builds UI-dependent behavioural graphs to understand whether a sensitive action is likely benign or malicious. Describe-ME (Zhang et al., 2015a) generate security-centric app descriptions, based on program analysis. Our approach does not focus on generating app description from bytecode analysis, we focus on descriptions (app descriptions on the distribution platforms and textual descriptions in the app itself) and behavioural elements (permission requests) that users interact during adoption and utilization of applications.

3. Sensitive Android Permission

Android provides an attribute called "Protection Level" that characterizes potential risks implied in the requested permissions. However, the available levels are very coarse and do not provide a whole picture of the behaviour of the Android Ecosystem (Zhauniarovich and Gadyatskaya, 2016). The current protection level in the Android permissions system is presented in Figure 1. In this section, we present a proposed ranking of the Android permissions as shown in

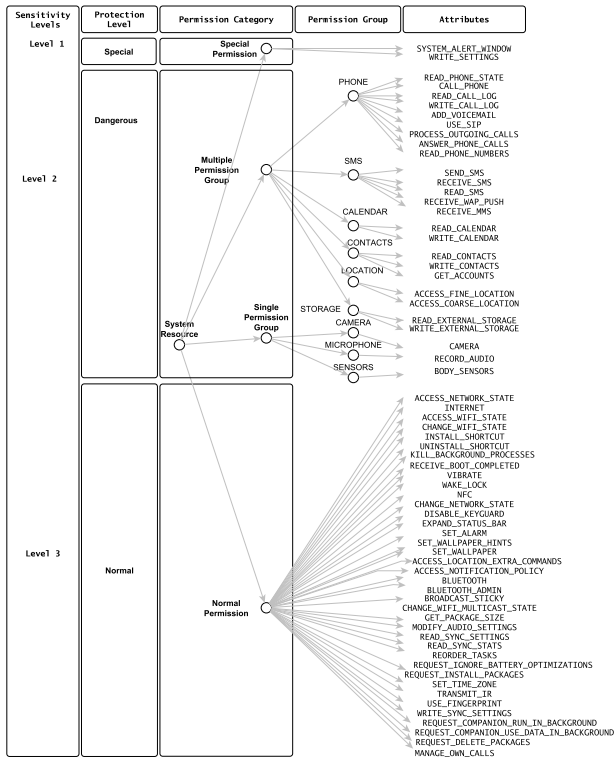


Figure 1: The Current Protection Level of Permissions as of Android 8.0 API Level 26

Figure 3, based on the following risk indicators that capture the sensitivity of the permission (Olukoya et al., 2019).

1. **Android Protection Level:** Android intuitively attributes protection levels to permissions by considering whether it can access directly the resources of the system. The permission attributes that are user-granted are categorized as *normal*, *dangerous*, and *special* permissions, in order of increasing risk. The protection level is a proxy for the sensitivity level i.e. *Special* > *Dangerous* > *Normal*.
2. **Permission Group:** Permissions are organized into groups related to a device’s capabilities or features. In order not to overwhelm the user with complex and technical permission requests, if the app has already been granted another dangerous permission in the same permission group, the system immediately grants the permission without any interaction with the user. By examining this behaviour, the question is - how many permissions does an app get access to, based on a single granted permission? For example, a permission group of 11 permissions, for example, is ranked higher than a permission group of one.
3. **Demoted Permissions:** These are permissions whose protection level changed from *dangerous* to *normal*, and from a security perspective, the implications are well documented in Zhauniarovich and Gadyatskaya (2016).

The proposed impact level of Android permissions

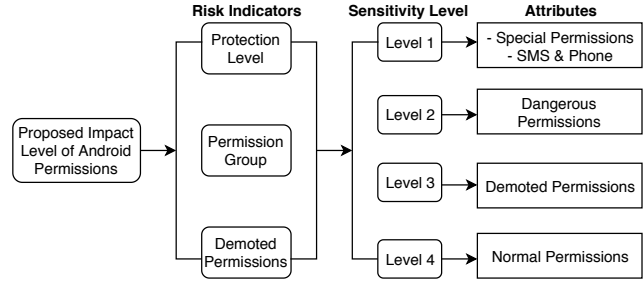


Figure 2: Taxonomy of Sensitive Permissions

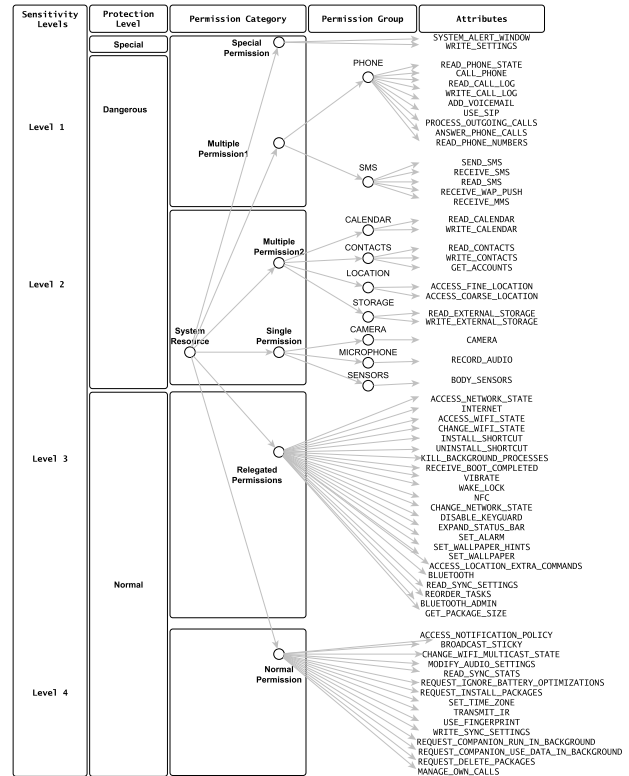


Figure 3: A Proposed Impact Level of Android Permissions as of Android 8.0 API Level 26

closely mirrors the Android rankings in Figure 1, but with the additional consideration of the permission category and permissions with a demoted security level. The sensitivity levels are a function of the impact of its default protection level, permission group and demoted permissions as described in Figure 2. Figure 3 shows the proposed impact level Android permission used for defining sensitive Android permissions used in this study.

4. Permission Sensitivity Index (PSI)

The revised permission model results in four levels of sensitive permissions based on the risk indicators. The enhanced ranking of sensitive permissions is used to extract an app’s permission sensitivity index (PSI) which is a function of the permissions requested and their sensitivity level. We

Table 1
Characterization of PSI

PSI	Characterization	Interpretation
VERY LOW	$P_1=P_2=P_3=P_4=0$	NONE
LOW	$P_1=P_2=P_3=0; P_4>0$	Level 4
MODERATE	$P_1=P_2=P_4=0; P_3>0$	Level 3
	OR $P_1=P_2=0; P_3, P_4 >0$	AND/OR Level 4
HIGH	$P_1 \leq \frac{P_{max}[1]}{2}$	Level 1
	$P_2 \leq \frac{P_{max}[2]}{2}$	AND/OR
	$(P_1 \text{ OR } P_2 \neq 0)$	Level 2
EXTREME	$P_1 > \frac{P_{max}[1]}{2}$	LARGE Level 1
	$P_2 > \frac{P_{max}[2]}{2}$	AND/OR
	$(P_1 \text{ OR } P_2 \neq 0)$	Level 2

infer 66 Android platform permissions from the risks with highest user concerns as of API level 26³ available to third-party applications, which serves as the ground truth.

The characterization of the sensitivity index is a spectrum from where an app does not require any permission in each level of sensitivity to when it requires all permissions in the sensitivity levels. The sensitivity index should be able to: (i) distinguish between an app that requests sensitive permissions and an app that does not; (ii) distinguish between apps that request sensitive permissions in varying quantity, while capturing the sensitivity levels of the permission request.

Finally, the proposed characterization of the PSI is along five levels of sensitivity - Moderate, Low and Very Low (distinguishing apps that do not require sensitive permissions); High and Extreme (distinguishing between apps that require sensitive information). Table 1 shows the characterization of the PSI based on the sensitivity of the permissions requested. P_i is the number of permissions requested in each sensitivity level of the taxonomy of permissions (P_1 for Level 1, P_2 for Level 2, P_3 for Level 3 and P_4 for Level 4). $P_{max}[1]$ and $P_{max}[2]$ are the maximum number of permissions in Level 1 and Level 2 of the taxonomy of sensitive permissions. The interpretation matches the PSI categorisation with the permission request pattern of the app. For example, an app with a LOW PSI means the app does not request any permission to be function, while an app with an EXTREME PSI requests a large number of sensitive permissions in Level 1 AND/OR Level 2 in the taxonomy of sensitive permissions. As shown in Table 1, an app is said to request a large number of permissions in Level 1 and Level 2 if P_1 or P_2 is greater than half of the maximum number of permissions in each level.

5. System Overview

To enhance user perception of security awareness, Android markets directly present two classes of literal app behaviour elements: 1) permission requests and 2) textual descriptions. The textual description represents the advertised behaviour and the permission requests implemented behaviour. Figure 4 provides an overview of the proposed

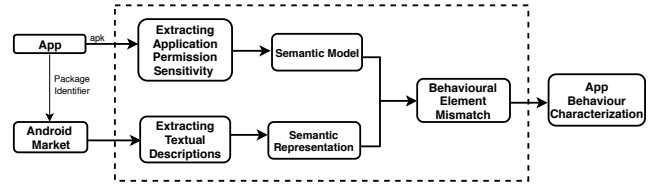


Figure 4: A Proposed framework for Exploring Behavioral Elements Mismatch

system, where an app's description is checked against its behaviour elements (permission request). For each Android app, the apk file is downloaded. Android Package (APK) is the package file format used by the Android operating system for distribution and installation of mobile apps and middleware. Each apk also has a package identifier that is a unique name to identify the app, which can be used to query the Android market's API for product information of an app. The type of information queried and extracted are the natural language descriptions of the app. The app's sensitivity is a function of the permission request pattern of the app, based on the risks presented by the permission set, determined by the sensitivity level. The semantic model aims to elicit the textual patterns that describe these permissions. The semantic representation is for generating textual dependencies in the app description provided by developers of the app distribution platforms. Finally, the Behavioural Element Mismatch (BE_m) investigates if the permissions requested by an app meets the user's expectation. This relies on finding relationships and correlations between textual patterns in the app descriptions and requested permissions. The proxy for measuring a user's expectation of an app's behaviour is through the natural language description of the app which gives an intuitive idea of its functionality. The app behaviour characterization involves classifying an app as benign or malicious. For our purposes, a malicious app is identified as one whose permission mismatch set P_m primarily occurs in Level 1 or Level 2 permissions i.e if $PSI(P_m) = \text{High or Extreme}$. Such an app requests critical permissions that are not motivated by the app's description, while a benign app convincingly motivates the need for the critical permissions it requests, but not necessarily the less-critical ones (Level 3 or Level 4 i.e. if $PSI(P_m) = \text{Very Low, Low OR Moderate}$).

6. Behavioral Elements Mismatch

The study by Lin et al. (2014) showed that users are uncomfortable with permission requests from a mobile app not strongly influenced by the purpose associated with such permission. Finding a mismatch between the implemented and advertised behaviour involves checking the gap between user expectations (what the user expects an application to do) and application functionality (what the application actually does). The question is - does the application description provide any motivation for its request for permission? The inputs to the system here are the natural language descriptions provided on app distribution platforms. This involves observing features of the app's metadata such as app descrip-

tion, privacy policy, screenshots etc. that may align with its permissions. The key challenge is to gather enough semantics from descriptions in natural language to reason about the permissions declared. This is an advancement towards practical solutions that can aid end-user privacy awareness, such that the rationale for permissions requested by Android apps can be automatically discovered by extracting them from the available sources of app's textual descriptions. Providing users with rationales of requested permissions can play a significant role in bridging the user expectation gap between the permission an app requests and the app's functionality.

6.1. Sources of App Textual Descriptions

In this section, we describe the sources of the textual descriptions analyzed. The rationale for using the sources is that they give developers the opportunity to explain the features of their app: they are the key Google-Play listing elements for app store optimization (ASO) that improve keyword rankings in search, increase conversion rates to install and drive more downloads. Some of these semantic features allow for elements such as app title, short description, full description, 'what's new in this version', interactive elements etc.; that potentially contain keywords where permission requirements could be inferred. For example, "Shares Location" and "Unrestricted Internet" are interactive elements that suggest the use of `Location` and `Internet` permissions respectively. Others are image data (graphics, screenshots, logo/icon) containing textual data for permission inference. Some of the scenarios as to how these features can potentially motivate permission requests are described below:

1. **Privacy Statement:** The privacy statement indicates the privacy policy of an app such as kinds of data collected, sources of data collected etc., any of which could be a pointer to the permissions required. A sentence from Tinder's (com.tinder) privacy statement reads "We may collect your geolocation information with your consent". This statement suggests that the app uses `Location` permissions. This is captured by the "geolocation information" governor dependent pair. Furthermore, the study in [Baalous and Poet \(2018\)](#) showed how dangerous permissions are described in Android App's privacy policies. For example, their study showed that privacy policies often use the terms: "address book" and "device's phone-book" to describe that they are using the "Contacts" permissions.
2. **Featured Graphics and Screenshots:** We also argue that permissions could be inferred from the featured graphics on app distribution platforms which could be in form of screenshots. For screenshots or graphics to be useful, the image has to be binarized first. The reasons for this are: -i) to handle documents with multicoloured texts and different background shades; and ii) because documents can have texts of widely varying sizes. Consequently, there has to be a way of extracting the output for the image data. We extract text with Optical Character Recognition (OCR) for all image types in python using `pytesseract`, an OCR tool for python. `pytesseract` is widely used as it can read all image types - png, jpeg, gif, tiff, bmp etc. The image data on Google Play are first converted from their `WebP` format (an image format for the Web) to the image types (PNG or JPEG) readable by `pytesseract`. One of the screenshots of the app, `DuoLingo` (com.duolingo) is used to demonstrate this. The text, "Speak, listen, read and write Speak this sentence. I can't use a microphone right now" suggests the use of the `Microphone` permission. The rationale for using screenshots is that they are not often bare, as they provide explanations to each slide which may give inference to permissions needed.
3. **App Title:** The app, `Facebook Messenger` has "Messenger – Text and Video Chat for Free" as its title. This suggests the use of the `camera` permission, as a result of the presence of "video chat". Another example is `Skype`, a video-calling app with the title "Skype - free IM & video calls"
4. **Short and Long Description** - The productivity app, `com.joshy21.vera.free.calendarplus` (`Calendar+ Schedule Planner App`) has its short description as the "Event calendar + schedule app. Planner for business, personal, office & event". This suggests the use of the `CALENDAR` permission, `READ_CALENDAR` and `WRITE_CALENDAR`. The app `GO SMS Pro - Messenger, Free Themes, Emoji` (com.jb.gosms), has a sentence in its long description: "GO SMS Pro comes with beautiful themes, lovely stickers, private box, pop up windows, GO chat (send free SMS & MMS), dual sim support, and much more". This suggests the use of `SMS` and `MMS` permission.
5. **Ads:** An app that contains ads has this statement as part of its metadata "Contains Ads", which suggests the use of the `Internet` permission to load the ads.
6. **What's New** - This is the area where developers specify any updates or improvements made to their software. These updates could also give a clue as to permissions required by the app to function. Google app (com.google.android.googlequicksearchbox&hl=en_GB) has one of the bullet points under its "What's New" section to be: "Use voice commands while navigating – even when your device has no connection. Try saying "cancel my navigation" "what's my ETA?" or "what's my next turn?". This suggests the use of `Microphone` and `Audio Settings` permission.
7. **Interactive Elements** - Interactive elements focus on what information the app has access to, what it can do with it, and whether there's user-generated content inside it that may be outside of the control of the publisher. The interactive elements from which permission request can be inferred include:
 - **Shares Location** - if the app can show your location to other users. This suggests the use of `LOCATION` permission.

- **Unrestricted Internet** - if the app has complete access to the Internet. This clearly suggests the use of the Internet, Network and Wifi state permission.

8. **App Logo/Icon:** An app logo may also give a hint to permissions requested. For example, caller.id.phone.mobile.number.location.tracker and com.fivestar.mobilelocationtracker have "Live Mobile Location Dialer, Location & Call Blocker" and "Mobile Location Tracker" respectively as part of their icon. Both sentences suggest the use of the location permission.

The framework for extracting textual patterns is described in Figure 5. We propose exploiting the textual description sources for checking an app's real behaviour against its advertised behaviour.

Using one of the textual description sources alone will lead to many false positives because each of them individually often fails to declare all the permissions. For example, on app distribution platforms, like Google Play, app descriptions have a character limit, which means that they cannot detail all privacy-related behaviours. Also, developers may provide descriptions with no relationship to permissions needed. This is because an app's description is like an advertisement for promoting the app and attracting more users, hence, developers tend to present them in ways that are appealing to users, which may not necessarily be permission-related. For example, Duolingo (com.duolingo), one of the all-time most downloaded apps in the Education category and also an Editor's Choice with 100,000,000+ downloads, has its description mostly occupied with comments from the public domain (Google, The Wall Street Journal, TIME Magazine, PC Magazine and Slate) and social media channels about the app. We propose exploiting all the textual description to find semantic correlations between an app's behaviour and its advertisement. Based on the 10 sources of the app description considered, we believe that we have gathered enough semantics from app descriptions in natural language to reason about the permissions declared.

After gathering the semantics, the textual patterns are extracted from the natural language descriptions as shown in Figure 5. Given the statement, *We may collect your geolocation information with your consent*, suitable logically dependent word pairs extracted for our semantic patterns are - "your information", "geolocation information", "collect information", "your consent", and "your geolocation information". The typed dependencies are generated using neural networks, via the Stanford CoreNLP(Chen and Manning, 2014). We leveraged the python wrapper(Guo, 2018) for Stanford CoreNLP that provides a simple API for text processing, such as named entity recognition, constituency parsing, dependency parsing, and more.

6.2. Semantic Model of Permissions

The framework for the semantic graph generator for Android permission is described in Figure 6. The key ideas

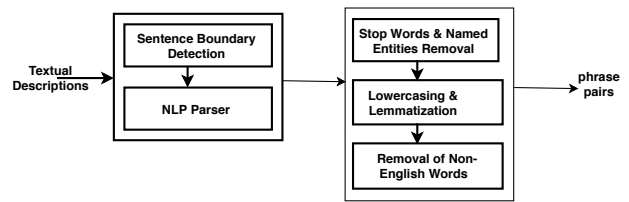


Figure 5: App Description Processing Module

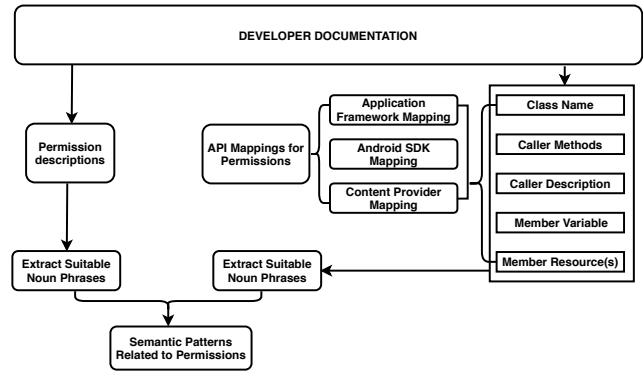


Figure 6: Framework for Permission Patterns

are - finding semantic patterns related to permissions; inferring semantic patterns from Android permission mappings; and extracting permission descriptions from the Google Play store, also finding their synonyms. The methodology is to leverage API documents and the description of each permission as provided by Google to extract suitable noun phrases.

We employ three different mappings: Application Framework, Android SDK and Content Provider.

1. Application Framework Mapping - The framework permission mapping includes any permission-protected, public method/API of the application framework that is accessible via inter-process communication (IPC).
2. Android SDK Mapping - The SDK mapping includes the documented API that requires at least one permission. The documented API comprises everything the app developer is supposed to use when implementing against the SDK.
3. Content Provider Mapping - The ContentProvider (CP) mappings include any system ContentProviders that protect read/write operations on the entire CP or paths thereof with permissions (e.g. contact content provider).

Specifically, for each permission, we get the relevant API/URI mappings using PScout(Au et al., 2012) and use mappings by Backes et al. (2016) for handling the latest version of the Android framework, to extract the corresponding resources(Figure 6). This step is similar to building a semantic graph in Whyper(Pandita et al., 2013), except that we also include building a semantic graph from the official descriptions of permissions by Google⁴ and combining the

more recent API-to-Permission mappings by Backes et al. (2016). The output of the semantic model of permissions is security-centric phrases describing sensitive operations pertinent to permissions. Based on this methodology, we generated 3000 textual patterns for the permissions. For example, Google's official long description for *BODY_SENSOR* permission states that "Allows an application to access data from sensors that the user uses to measure what is happening inside his/her body, such as heart rate.", while the short description states that "Wearable sensors/Activity data - body sensors (like heart rate monitors)". Based on these sources, some of the textual patterns generated for *BODY_SENSOR* permission include "body sensor", "heart rate", "wearable data", "blood pressure" etc.

6.3. Semantic Correlation

The goal of this section is to find the semantic correlation between the textual semantics from app textual descriptions and permission semantics, to determine if a mismatch occurs. The semantic correlation is an indication of the Behaviour Elements Mismatch (BE_m) of apps. The key idea is to compute the semantic similarity between two noun phrases using a lexical database and corpus statistics. The proposed methodology for finding phrase similarity considers the phrase as a sequence of words and deals with all the words in the phrase separately according to their semantic and syntactic structure based on a word embedding model.

The most commonly used word embedding models are word2vec (by Google), fastText (by Facebook) and GloVe (by Stanford) (Pennington et al., 2014) which are unsupervised approaches based on the distributional hypothesis (words that occur in the same contexts tend to have similar meanings). Based on the context and our domain, we investigated the word embedding models preferable for the context and domain of the work. Selecting three well-known pre-trained models and leveraging gensim to load those model. We use gensim, a well-known python NLP library (Rehurek and Sojka, 2010) that already implements an interface to deal with these three models.

We download pre-trained word vectors learned on different sources trained using these three models. The GloVe pre-trained vectors are trained on Wikipedia data, with various models from 25, 50, 100, 200 to 300 dimensions base on 2, 6, 42, 840 billion tokens. For word2vec we download Google's pre-trained model whose word vectors is trained on google news and provided by Google. Based on 100 billion words from Google News data, they trained the model with 300 dimensions. fastText is released by Facebook which provides five models with 300 dimensions. To justify the choice of the pre-trained word vectors suitable for our work, we compare these models (word2Vec(100B.300D), GloVe (6B.50D), and fastText 1M.300D.subword) on sample word pairs and evaluate the similarity for the given phrases. We also evaluated the suitability of each model using a standard dataset which has noun pairs originally measured by (Rubenstein and Goodenough, 1965). The result is shown in Table 2, where the 50-dimensional vector of the Stanford GloVe is

Table 2

Similarity Results using GloVe, word2vec(W2V) and fast-Text(fT).

First Word Pair	Second Word Pair	GloVe	W2V	fT
scan barcode	barcode scanner	0.88	0.78	0.85
device location	device location	1.00	1.00	1.0
photo gallery	picture gallery	0.87	0.80	0.9
save document	save file	0.85	0.71	0.81
contain ads	display ads	0.76	0.57	0.77
find place	search location	0.72	0.37	0.62
plan vacation	plan holiday	0.87	0.78	0.88
your voice	your speaking	0.82	0.62	0.76
flight mode	airplane mode	0.89	0.81	0.86
schedule appointment	plan meeting	0.65	0.32	0.55
geographical location	device battery	0.37	0.17	0.42
logic gate	passport number	0.30	0.57	0.38
automobile	car	0.70	0.58	0.78
midday	noon	0.83	0.55	0.78
coast	shore	0.80	0.51	0.74
journey	voyage	0.83	0.68	0.77
grin	smile	0.86	0.86	0.79
magician	wizard	0.73	0.49	0.67
forest	woodland	0.76	0.64	0.74
IP address	home address	0.71	0.13	0.74

preferable. Working with the 50-dimensional vectors (glove 6B.50d) trained on Wikipedia data, we can find the similarity of the phrases.

Given two phrases P and Q originating from the textual semantics of the app description and permission request respectively, where $P = [P_1 \ P_2]$ and $Q = [Q_1 \ Q_2]$. P_1 , P_2 , Q_1 , and Q_2 are the sequence of words (tokens) from P and Q respectively. The final similarity which involves finding the aggregated similarity, $Sim(P, Q)$, of the phrases, is calculated as shown in Equation 1. Sample similarity results between phrase pairs that suggest permission usage using Equation 1 are presented in Table 2.

$$Sim(P, Q) = \frac{Sim(P_1, [Q_1, Q_2]) + Sim(P_2, [Q_1, Q_2])}{2}$$

$$Sim(P_1, [Q_1, Q_2]) = \max[Sim(P_1, Q_1), Sim(P_1, Q_2)] \quad (1)$$

$$Sim(P_2, [Q_1, Q_2]) = \max[Sim(P_2, Q_1), Sim(P_2, Q_2)]$$

According to Rubenstein and Goodenough (1965), the benchmark synonymy value of two words is 0.8025, while AutoCog (Qu et al., 2014) proposed 0.67 as the threshold, which was also implemented in TAPVerifier (Yu et al., 2016). The standard originally measured by Rubenstein and Goodenough (1965) has been used in several investigations over the years and has been established as a benchmark source of the semantic similarity measure. It has been a stable source of comparative analysis for word and sentence similarity in the state-of-the-art approaches (Pawar and Mago, 2018; Li et al., 2006; Islam and Inkpen, 2008; Lee et al., 2014) for calculating similarities between words and sentences. We have set the threshold to 0.80 by following the benchmark proposed in Rubenstein and Goodenough (1965), such that if the semantic similarity equals or exceeds this threshold, the collected information can be mapped to the associated permission. For example, using our similarity metric, the

similarity value for the noun phrase pairs, "save document; save file", "your voice; your speaking" was found to be 0.85 and 0.82 respectively. Hence, these are mapped to storage and microphone permissions respectively. The proposed similarity approach is built on the idea of measuring the distance to an existing taxonomy.

The behavioural element mismatch is a function of the inferred permissions (P_i) from the app description against the actually requested permissions (P_r), as shown in Equation 2. The mismatch is measured by the permission sensitivity index (PSI) (see Table 1), inferred from the semantic analysis, against the actual permissions requested.

$$BE_m = PSI(P_r - P_i) \quad (2)$$

An app whose BE_m is High or Extreme shows that the app's sensitive behaviour is not properly advertised in its textual semantics, while an app whose BE_m is Very Low, Low Or Moderate means that all sensitive behaviour is properly described in the app's textual description. An app passes if the difference between the requested and inferred permission is dominated in the Level 3 and Level 4 permissions.

6.4. App Behaviour Characterization

An app is classified based on the characterization of its behavioural element mismatch (BE_m). We describe an app's behaviour as benign or malicious based on the BE_m . We characterize apps with Very Low, Low Or Moderate BE_m as potentially benign apps because this is characterized by apps that follow best practices of transparency. These apps are clear about the core functionalities of the device they are accessing in their descriptions. Apps with High or Extreme BE_m do not follow the transparent policies because of the need to access sensitive capabilities of the phone are not motivated in the app descriptions. Furthermore, this approach of characterizing an app behaviour was also recently corroborated in MAPS (Sebastian et al., 2019) - a privacy compliance analysis technique for spotting a potential compliance issue in an app. An *issue* is spotted when an app is performing a privacy practice (e.g. a first party is accessing GPS location data) while its associated privacy policies do not disclose it either generally (e.g. "our app accesses your location data.") or specifically (e.g., "our app access your GPS data.")

Generally, if the permission requested is in Level 1 or Level 2 of the taxonomy of sensitive permissions and it's not motivated in the app's textual descriptions, the app is likely malicious while an app that provides security-centric descriptions for sensitive permissions is likely benign. We chose to characterize the behaviour of an app based on permissions in Level 1 and Level 2 only in the taxonomy of sensitive permissions for the following reasons: (i) our study is motivated by user's privacy risks, hence we only considered the most sensitive permissions (*Special* and *Dangerous*) that may adversely affect the user. (ii) these are the permissions that the user has control of in the Android device setting, by allowing or revoking them at any time. (iii) Permissions in Level 3 and Level 4, are granted automatically by the system, hence, pose a little privacy risk to the user.

Most malware detection systems are usually divided into two major parts - misuse and anomaly. The technique in this study is anomalous. The proposed technique for categorizing an anomaly in this study is motivated by the semantics-aware approach in DroidSIFT (Zhang et al., 2014), where applications are characterized based on UI-dependent behavioural graphs. The anomaly detector involves understanding whether a sensitive action (e.g., send text) depends on user interactions (likely benign) or is automatically performed in the background (likely malicious). Similar to semantics-aware Android malware classification in their study, an app behaviour is predicted as *benign* if the advertised behaviour of the app and its implemented behaviour are in sync. An anomalous behaviour which characterizes and app as likely *malicious* occurs when there is a mismatch between the app's advertised behaviour and its sensitive functional behaviour. The anomaly detector provides a security-oriented overview of the expected (predicted) app behaviour, while the evaluation involves validating the predicted behaviour with its actual behaviour by investigating the correctness of the anomaly detector it in Android malware detection settings.

7. Evaluation

To evaluate the effectiveness of our technique, we investigate the following main research questions:

- R1: Does an app's textual description provide useful information for measuring its behaviour as advertised?
- R2: How does our approach compare to state-of-the-art techniques.?
- R3: Does an enhanced app textual description provide useful insights into malware analysis and detection?
- R4: What is the robustness of the textual description in determining the expected behaviour of an application?

7.1. Behavioural Elements Mismatch

7.1.1. RQ1: Does an app's textual description provide useful information for measuring its behaviour as advertised?

For this purpose, we evaluate our proposed enhancement with 30 random benign apps from Google Play Store against sources of the state-of-the-art textual semantics as shown in Table 4. We selected top downloaded apps of all time in which 80% are Editors' Choice apps on Google Play because we believe the description of these apps have been carefully constructed by the developers and provides a benchmark for comparisons. The description of the 30 apps investigated with assigned reference labels are shown in Table 3 with their package identifiers as displayed on the Google Play Store. The goal is to investigate whether security-centric descriptions that describe an app's behaviour element is motivated in its description on Android Market. 90% of the apps

Table 3

Description of the 30 apps investigated

Label	Package Identifier
B01	com.airbnb.android
B02	com.amazon.avod.thirdpartyclient
B03	com.autoscout24
B04	com.CultureAlley.japanese.english
B05	com.dropbox.android
B06	com.duolingo
B07	com.google.android.apps.maps
B08	com.google.android.apps.walletnfcrel
B09	com.google.android.apps.youtube.kids
B10	com.google.android.play.games
B11	com.handmark.tweetcaster
B12	com.imdb.mobile
B13	com.jumia.android
B14	com.lumoslabs.lumosity
B15	com.microblink.photomath
B16	com.netflix.mediaclient
B17	com.northpark.beautycamera
B18	com.pinterest
B19	com.popularapp.periodcalendar
B20	com.sec.android.app.shealth
B21	com.sonymobile.sketch
B22	com.spotify.music
B23	com.szyk.myheart
B24	com.ted.android
B25	com.tinder
B26	com.trivago
B27	com.ubercab.eats
B28	com.whatsapp
B29	net.skyscanner.android.main
B30	net.zedge.android

provided textual descriptions that were useful in matching the permissions requested and were correctly tagged as 'Benign' (cf Table 4), while 10% of the apps requested sensitive permissions that were not motivated in the app description.

The Accuracy of Permission Semantics

The reasons for the (few) false positives are explained below following individual investigation:

1. Occasionally, permission patterns are discovered in the app textual description but the associated permissions are not requested by the app. An example is an app whose textual description includes "device id", "mobile id" etc, which are textual patterns associated with *device identity*, but the app does not go on to request the permission.
2. Some permission patterns belong to more than one permission. For example, "change wallpaper" and "HD wallpaper" are semantics associated with RECEIVE_BOOT_COMPLETED, besides the fact that Android has two wallpaper permissions which also share similar semantics.

The Accuracy of Behavioural Element Mismatch

98% of the false positives were a result of inadequate token phrases for Level 4 permissions, while 2% were due to (Level

Table 4Comparison with State-of-the-art Approaches: T_P - Privacy Policy, T_D - Description, T_N - Proposed Enhancement

App	BE_m			App	BE_m		
	T_P	T_D	T_N		T_P	T_D	T_N
B01	Y	N	Y	B16	N	N	N
B02	Y	N	Y	B17	N	Y	Y
B03	Y	N	Y	B18	Y	N	Y
B04	N	N	N	B19	N	Y	Y
B05	Y	N	Y	B20	N	N	Y
B06	Y	N	Y	B21	Y	N	Y
B07	Y	N	Y	B22	Y	N	Y
B08	Y	N	Y	B23	N	Y	Y
B09	N	N	N	B24	Y	Y	Y
B10	Y	N	Y	B25	Y	N	Y
B11	Y	N	Y	B26	Y	N	Y
B12	Y	Y	Y	B27	Y	N	Y
B13	Y	N	Y	B28	Y	N	Y
B14	Y	Y	Y	B29	Y	Y	Y
B15	Y	N	Y	B30	Y	N	Y

1, Level 2, Level 3) permissions not motivated by the textual descriptions

- Token Phrases for Permissions: There were fewer token phrases generated for Level 4 permissions compared to more critical permissions. This is due to fewer API mappings, an indication of the limited device resources they control.
- Some permission patterns are not motivated in the app textual description, but requested (B04,B09,B16 in Table 4). An example is the SYSTEM_ALERT_WINDOW permission, which is a permission that allows a developer to use the screen overlay.

7.1.2. RQ2: How does our approach compare to the state-of-the-art techniques?

In comparing our approach, we evaluate the semantic sources of the state-of-the-art with our enhancement, to evaluate the behavioural element mismatch. The goal is to investigate whether the enhanced sources of textual description improve the evaluation results. We evaluated the proposed app behaviour characterization for the sources of semantic information gathered in this study against the sources of semantic information in other state-of-the-art approaches. We aim to investigate whether the 10 sources of semantic information proposed in this study provides any significant enhancement compared to the semantic sources used in other techniques in measuring description-to-permission fidelity. Our proposed enhancement performs better than the modern approaches as shown in Table 4. These benign apps are labelled B01 - B30 for comparison, and we manually verify the results to validate the outcomes. The column T_P represents the behavioural element mismatch obtained by using the privacy policy of the app. T_D shows the classification result obtained by using the app description alone, while T_N

represents the proposed approach that considers 10 different sources of app textual information for semantic meaning. γ represents a benign app correctly identified as benign and N for benign apps wrongly classified as malicious based on the technique. As shown in Table 4, there are scenarios where using the app description proves more useful than using the privacy policy (B17, B19, B37) and vice versa (B01, B02,..., B30). Using the app's description and privacy policy alone could also prove inadequate and limited (B20). The proposed enhancement, T_N , can reduce the false negatives in the state-of-the-art-approaches that uses description alone (T_D) e.g. *Whyper*(Pandita et al., 2013), *AutoCog*(Qu et al., 2014), *CHABADA*(Gorla et al., 2014)) by 87%, while using privacy policy (T_P), as in *TAPVerifier*(Yu et al., 2016) by at least 57%.

Comparison with Existing Approaches

- Privacy policies often motivate the permission needed under the heading "Information you provide or Information We Obtain" which oftentimes, are an indication of the information that users provide directly or generate through the use of an app. However, some permissions (e.g. SMS permissions) do not fall under this category. Such permissions are an indication of the functionality of the phone being used and not the information being provided. The limitation of the privacy policy also involves broken links in the play store or redirection to non-policy documents. Also, the app may lack privacy links on their play store pages altogether(Story et al., 2018).
- Descriptions are also limited because they are an avenue for developers to market their products in a way that attracts users. The descriptions may be devised in a way that does not motivate the permissions required. Because descriptions are limited by app distribution platforms, they are often designed for app search optimization, as opposed to being privacy-related or security-centric.
- When apps introduce an update that requires new permissions, it is often motivated in the "What's New" sections or by adding screenshots. More often than not, a description does not get revised to accommodate the latest updates. Furthermore, privacy policy does not get revised often unless there is a new regulatory policy, such as EU's General Data Protection Regulation (GDPR), that mandates all institutions to revise and review the way user information is handled.
- The more permissions an app requires, the more sources for semantic information are needed. This was observed in Table 4, where the result of our proposed enhancement is the same as just using the privacy policy alone. Upon investigation, it was discovered that for every app, where our proposed enhancement outperforms using privacy policy alone, the number of permissions is at least two times the permissions required when our results are the same. It might be pos-

sible for all permissions requested to be motivated in the privacy policy or even in the descriptions, but with more permissions, an enhanced source of semantics is required.

- There are some permissions that are difficult for developers to motivate in the app's textual descriptions regardless of the proposed enhancement unless they are expressly stated in the app descriptions. Such permissions include NFC, BROADCAST STICKY, EXPAND STATUS BAR. These do not affect our approach because they are Level 3 and Level 4 (least-sensitive permissions). Our app behaviour characterization is benchmarked by anomalies in Level 1 and Level 2 permissions (most-sensitive ones).

Some of the textual descriptions may be repetitive as a result of considering several sources of app textual information in the Android market. This happens when some textual description in one source is also present in another source on the app distribution platforms. For example, a description present in the app's *featured graphics* may also be present in the app's *long description*. This does not in any way affect the results because unique semantic patterns are considered, rather it provides significant enhancement to the state-of-the-art approaches in description-to-permission fidelity. The enhancement is shown in Table Table 4. Considering many other sources of textual descriptions on app distribution platform gives room for robustness in evaluating the fidelity of descriptions on the app market and permissions requested. The proposed enhancement has shown that gathering enough textual semantics can yield better results for determining stealthy behaviours in Android applications. By considering other sources of natural language description made available to users on app distribution platforms, we can get useful insights as to the permissions the app might require based on its advertisement.

7.1.3. Malware Sources

To ensure that the malware samples investigated in this study are up-to-date, we gathered malware from the following sources:

- **Mal_AMD** - contains 24,553 samples, categorized in 135 varieties among 71 malware families ranging from 2010 to 2016. The dataset provides an up-to-date picture of the current landscape of Android malware and is publicly shared with the community (<https://amd.arguslab.org/>).
- **AndroZoo** - a growing collection of Android applications collected from several sources with benign and malicious apps(<https://androzoo.uni.lu/>).
- **Contagio** - a public repository of mobile malware mini dump - (<http://contagiomobile.deependresearch.org/index.html>)
- **Other Sources** - We also analysed sources of malware verified in literature such as:

Table 5

Identification of Malware Using App Descriptions from {App Market} and {UI texts} with Permission Requests

Source	Size	TP	FN	TPR
App Market	1241	1074	167	86.54%
UI Description	1241	1174	67	94.60%

1. DREBIN (<https://www.sec.cs.tu-bs.de/~danarp/drebin/>)
2. VirusShare (<https://virusshare.com/>)
3. AndroidPRAGuardDataset(<http://pralab.die.unica.it/en/AndroidPRAGuardDataset>) for ground-truth of obfuscated malware apps.
4. Understanding android obfuscation techniques: A large-scale investigation in the wild - for obfuscated benign apps.(https://drive.google.com/file/d/1pJEW9vNxGb_D5wk0XGqNGCQJZiRzLfwP/view) (<https://drive.google.com/file/d/1bILWq8iQU0G9lqVDY6thAntkJwMV3mT7/view>)
5. MamaDroid and DroidSpan dataset (https://bitbucket.org/haipeng_cai/droidspan/src/master/)

7.1.4. RQ3: Does an enhanced app textual description provide useful insights into Malware Analysis and Detection?

We gathered malware from the following sources verified in literature - Mal_AMD (Wei et al., 2017), Mal_DREBIN (Arp et al., 2014), Virus Share(Roberts, 2011) and PRAGuard Dataset (Maiorca et al., 2015). Extracting the malware textual semantics from Google Play was challenging because the majority of such apps have been pulled off the store and the difficulty of getting malware descriptions from app distribution platforms are well documented in Gorla et al. (2014). As at the time of measurement, 98.3% of the malware dataset used in the study were no longer present in the Android market.

However, for 1241 cases in the malware data set gathered, we were able to find the same package identifier on the store. In these cases, we used the package identifier to query Google Play API and extract the available textual descriptions (see Section 6.1) for each app, to investigate whether these textual semantics correlate advertised apps with the permissions requested. To measure the correctness of the proposed metric for characterizing app behaviour, "apps without security-centric descriptions for sensitive permissions requested are likely to be malicious", we evaluate the precision of the metric on the malware data set. The precision can be seen as a measure of exactness or fidelity of the classifier that shows what proportion of apps classified as malicious are malware. Table 5 shows the result of the evaluation.

The high value of the detection sensitivity (True Positive Rate - TPR) shows an indication that requesting sensitive permissions without providing a functional motivation or need for such behaviour is a strong indication of malware

behaviour. 86.54% of the malware investigated exhibit such behaviour. This shows that an app requesting sensitive permissions without a clear motivation in its metadata is a reliable warning signal.

While our results show a good performance for the detection of malware, we carefully investigated in detail the false positives it produces. We are aware that this approach, where only permission requests as features are considered, may have difficulties in improving the current detection accuracy. This is because while malicious apps generally do not motivate sensitive behaviours in their descriptions, there are still some request patterns that evade the analysis for a mismatch. Such requests patterns include 13.46% of the malware in the sample size: i) that do not request permissions or, ii) explained the need for sensitive permissions in their descriptions and transparent in their descriptions of the core functionalities of the device they are accessing. In this case, relying on only permission request against app description is not feasible for the detection of malware. This gets complex if a large number of benign apps request no permissions too or not clear about the sensitive capabilities of the device they are accessing. By using only permission request information and enhanced app description for the detection, this may lead to false positives. Therefore, additional information is required as features to reduce false positives. However, our analysis results strongly show evidence that the use of permission requests and enhanced sources of app textual description can be used as a valid warning sign to flag apps for further investigation.

Furthermore, to investigate the contribution of our technique to community ratings on app distribution platforms, we queried Google Play API using the package identifier for the malware apps in our data set for the app information. Specifically, the app information we are interested in is the user ratings of the apps. From the results gathered, 96.42% of the identified malware requests sensitive permissions, yet 91.86% were rated high (a rating in the range of 3.50 to 5.0) by users. One reason for this could be that the average user does not have the technical knowledge to fully understand the privacy and security aspects of an application's metadata (Felt et al., 2012). Thus, the user rating is designed for users to rate an app based on functionality, content, features and benefit. For instance, if users are motivated by user ratings of apps to adopt an app, then with a benchmark rating of 3.5, 88.33% of the malicious apps in our data set would be potentially adopted. Whereas, 87% of such apps would have been flagged using our transparency metric of behavioural element mismatch. Since community ratings reflect how users perceive an app in terms of functions, features, and performance; the privacy and the security perspective can be augmented with our proposed technique of behavioural element mismatch assessment.

7.2. R4: Robustness of Textual Description

To evade this approach, malware developers may provide inaccurate app descriptions. A malicious developer may manipulate other app's textual description information

to introduce subtle malicious indicators that would then be matched against the actual permission semantics; this would not determine any mismatch and the app would, therefore, be flagged as benign. There are constraints, of course, but one approach might be to compare the description against actual code patterns (Zhang et al., 2015a). The underlying technique of their approach is to associate APIs to permissions to generate a behavioural graph that is used to automatically generate security-sensitive descriptions. This work suffers three known limitations - i) Runtime Permission Checks - Maps for SDK/Frame-work >23 are incomplete due to the fact that parts of the permission checking have been moved to the AppOpsManager (runtime permission checks). ii) Maps are currently missing APIs with permissions checked in native code (CAMERA - a dangerous permission etc.). Thus, any critical functionality implemented using these technologies can evade behaviour analysis. These limitations are well documented in the state-of-the-art Android Permission mappings (Backes et al., 2016). iii) Benign and Malicious apps can intentionally obfuscate their programs via Android Packers, such that the program code cannot be extracted for behaviour analysis (Zhang et al., 2015b). Furthermore, features extracted by Android reverse engineering tools are prone to errors because they do not work in all cases, which leads to imperfect control flow graph (Mirzaei et al., 2019). With the changes in Android API such as deprecation of API calls with new API releases, monitoring app behaviour through program code becomes a challenging endeavour. The conclusion is that program code cannot be extracted sometimes for behavioural analysis. If the goal is to improve the security awareness of end-users, the two classes of literal app information that the Android market provides - permission requests and textual descriptions needs to be carefully investigated. The natural language processing of an app's permission request with its description provides an intuitive security-oriented view of the expected behaviour of the app.

There are two sources of textual description available to end-users - text provided by publishers on an app distribution platform and textual descriptions within the application itself. The aim is to analyze the string resources which are a representation of the UI textual description of the apps. We leverage the string resources because these provide text strings for the application that users interact with. These texts often come as labels in the app that describes the functions of each XML page the user is interacting with, a textual description in form of hints for required user inputs, text, labels associated with user input widgets, etc. Also, these texts describe the action, operation or function a user can interact with, which oftentimes indicates the use of potential permission or accessing the core functionalities of a device to respond to a user action or input. For example, a UI button could have the following text descriptions add a contact, delete contact, send SMS, make a phone call, upload a file, save document etc., which all require sensitive APIs protected by permissions. It would, therefore, be suspicious of an app requiring such permissions without correspond-

ing motivation or text descriptions of user required actions. These string resources provide textual semantics that is combined with the permission semantics.

The advantage of our approach of using the string resources for semantic resolution of permission is that it provides an accurate textual description of the sensitive device information or core functionality the app is trying to access that is not subject to developer inconsistencies. Since developers of both benign and malicious apps have the same goal of obtaining accurate inputs from users, this necessitates clear UI textual descriptions. For example, GoogleMap has edittext resource with a hint "Your location" that allows a user to specify the starting point of navigation and a text label "location sharing", which clearly suggests the use of location permission. Also, another Editor's choice App, Instagram, has the following text labels on a page in the app "Instagram's Better with Friends. See which of your friends are on Instagram and choose whom to follow", with an Android widget button, with the text description Connect Contacts. The textual description on the button suggests that Instagram needs to find accounts on the device and read user's contacts, which motivates the contact permission request. Hence, assuming text descriptions on the app distribution platforms are inappropriately specified, permission request patterns can still be semantically resolved based on the UI textual descriptions.

We compare the results of identifying known malware using the app description on Android markets against the string resources provided within the app itself. For each apk, we disassemble using Androguard (Desnos, 2017) to extract the UI textual descriptions from its resources. As shown in Table 5, the sensitivity of the detection increased from 86.54% using description on app markets to 94.60% using the string description within the app itself, thereby reducing false negatives by 60%. This shows that 100 malware samples that evaded detection using description on Android markets were detected by using the UI textual descriptions. Evading previous analysis means that these malware apps provided security-centric descriptions on the Android market. The key finding in this regard is that the textual description on app distribution platforms can be enhanced by UI textual descriptions.

Furthermore, we investigated the robustness of textual description on the different malware families provided by Ma1_AMD (Wei et al., 2017), which provides an up-to-date picture of the current landscape of Android malware, categorized in 135 varieties among 71 malware families, with 47 out of the 71 malware families obfuscated by anti-analysis techniques such as dynamic loading, native payload, string encryption etc. The detection result achieved a true positive rate score of 0.9872 as shown in Table 6. In addition, we downloaded 5000 benign apps from Google Play (benignA) and we evaluate the performance of the model using F-Score and Balanced Accuracy.

The F-score $2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$ summarizes the precision $\frac{TP}{TP + FP}$ and recall $\frac{TP}{TP + FN}$ of a classifier and a description of the balance between the two. F-Score is usually used to eval-

Table 6

Classification Performance on {Mal_AMD, benignA}, {PRaGuard, benignB}

Datset	TPR	Precision	F-Score	bACC
Mal_AMD, benignA	0.9872	0.96	0.94	0.92
PRaGuard, benignB	0.960	0.90	0.93	0.82

uate the performance of an unbalanced binary classification problem where the class distribution is highly skewed and when it is more important to correctly label an app as malicious, as opposed to labelling the benign one. An F-Score close to 1 indicates the good performance on correctly classifying the classes because it conveys the balance between the exactness (precision) and recall (completeness) of the model.

The Balanced Accuracy (bACC): Balanced Accuracy is used to measure how accurate is the overall performance of a model is, considering both positive and negative classes without worrying about the imbalance of a data set (Mower, 2005). Consider a dataset of 90 negative and 10 positive samples, classifying all as negative gives 90% accuracy score. This is because the accuracy value was highly dependent on the specificity value because the positive samples were proportionately lower than the negative samples. Balanced Accuracy $\frac{TPR+TNR}{2}$ overcomes this problem, by normalizing true positive and true negative predictions by the number of positive and negative samples, respectively, and divides their sum into two. Regarding the previous example of 90 negative and 10 positive samples, classifying all as negative gives 50% balanced accuracy score out of the maximum bACC of 100%, which is equivalent to the expected value of a random guess of a balanced data.

With Mal_AMD (17273 apks) and benignA (5000 apks), we achieved an F-Score of 0.94 as shown in Table 6. Our analysis shows that descriptions and behavioural elements (e.g. permission requests) are a promising way of enhancing a user perception of the expected behaviour of applications. To the best of our knowledge, the proposed approach for evaluating the robustness of textual descriptions for enhancing permission-based malware detection is distinctive within the research community.

7.3. Augmenting Privacy Analysis with Behavioural Element Mismatch

We also investigated the benefit of our approach in enhancing privacy analysis of mobile apps. Since the majority of current analysis framework relies on program code analysis, recent studies have shown that obfuscation has turned out to be a new barrier to protect Android users (Duan et al., 2018; Dong et al., 2018; Wang and Rountev, 2017) because of the obstacles it poses to privacy analysis of Android apps, we aim to investigate the performance of our detection approach on obfuscated apps. To achieve this, we form a dataset of malware and benign applications obfuscated using different techniques.

Table 7

Detailed Detection Results On the AMD Dataset (Wei et al., 2017)

Family	#	TP	FN	Family	#	TP	FN
Airpush	7658	7487	171	Mecor	1799	1799	0
AndroRAT	7	7	0	Minimob	183	183	0
Andup	26	26	0	Mmarketpay	12	12	0
Aples	2	2	0	MobileTX	12	12	0
BankBot	93	92	1	Mseg	211	210	1
Bankun	20	20	0	Mtk	60	60	0
Boqx	199	199	0	Nandrobox	76	76	0
Boxer	7	7	0	Obad	4	4	0
Cova	14	14	0	Ogel	4	4	0
Dowgin	3038	3038	0	Opfake	7	7	0
DroidKungFu	190	189	1	Penetho	6	5	1
Erop	1	1	0	Ramnit	5	4	1
FakeAngry	9	9	0	Roop	30	30	0
FakeAV	1	1	0	RuMMS	8	8	0
FakeDoc	0	0	0	SimpleLocker	21	20	1
FakeInst	57	57	0	SlemBunk	2	2	0
FakePlayer	2	2	0	SmsKey	90	90	0
FakeTimer	1	1	0	SmsZombie	9	0	9
FakeUpdates	5	5	0	Spambot	13	13	0
Finspy	2	2	0	SpyBubble	2	2	0
Fjcon	4	4	0	Stealer	13	13	0
Fobus	4	2	2	Steek	12	0	12
Fusob	130	129	1	Sypeng	4	4	0
GingerMaster	105	105	0	Tesbo	5	5	0
GoldDream	46	46	0	Triada	148	146	2
Gorpo	21	21	0	Univert	3	3	0
Gumen	128	128	0	UpdtKiller	2	2	0
Jisut	24	16	8	Utchi	12	12	0
Kemoge	15	15	0	Vidro	1	1	0
Koler	41	41	0	VikingHorde	4	4	0
Ksapp	32	32	0	Vmvol	5	5	0
Kuguo	1096	1096	0	Winge	13	13	0
Kyview	136	135	1	Youmi	1239	1238	1
Leech	5	4	1	Zitmo	4	4	0
Lnk	3	3	0	Ztor	19	19	0
Lotoor	113	106	7				
SUM = 17273		TP = 17052				FN = 221	
True Positive Rate (TPR) = 98.72%							

We have investigated AMD Dataset as ground truth for investigating the benefit of our approach. The dataset is labelled based on several behavioural criteria, including the presence of different anti-analysis techniques in the samples of each family and variant. Finally, we have used an additionally related dataset, known as PRAGuard (Maiorca et al., 2015) to evaluate the performance of our approach over obfuscated malware apks. The dataset is composed of 10,479 samples, obtained by obfuscating the MalGenome (Jiang and Zhou, 2012) and the Contagio Mobile Malware Minidump⁵ datasets with seven different obfuscation techniques. The final number of apps is 4982, having discarded apps that cannot be disassembled properly with Androguard (Desnos, 2017) from our datasets. PRaGuard classifies Obfuscation strategies into two class: Trivial (TR) - an ensemble of obfuscation strategies that only affects strings, without changing the bytecode instructions; Non-Trivial - techniques that affect both the strings and the bytecode of the executable, such as Reflection (RE), String Encryption (SE), Class Encryption (CE) etc. The PRaGuard dataset contains a uniform distribution of a combination of major obfuscation techniques, which makes it ideal for validation. For exam-

Table 8
Detailed Detection Results in PRAGuard Dataset

Obfuscation Technique(s)	Size	TP	FN
CE	830	788	42
RE	843	807	36
SE	843	779	39
TR	818	807	36
TR + SE	605	586	19
TR + SE + RE	539	525	14
TR + SE + RE + CE	504	485	19
Total	4982	4777	205
True Positive Rate (TPR) = 0.96			

ple, 4 out of the sample set contains a joint application of at least two obfuscation technique, while one of the samples is obfuscated by four different obfuscation technique as shown in Table 8. Table 8 also shows the detection results of PRAGuard with the true positive rate achieved for the seven obfuscation techniques applied to the dataset. The PRAGuard dataset contains a uniform distribution of a combination of major obfuscation techniques, which makes it ideal for validation. Table 6 shows the detection recall score of 0.96 achieved for the different obfuscation technique combination in PRAGuard.

To create a dataset of Android applications with the obfuscated techniques, we gathered obfuscated benign apps that were used as ground truth for the testing set in Dong et al. (2018). The total number of obfuscated apps was 1084. We present our results using the classification performance measures as shown in Table 6.

Asides obfuscated apps being a challenge to Android code analysis, dynamic tracking approaches may also miss some data leaks and yield an under-approximation because of specifically designed methods to circumvent security tracking (Babil et al., 2013). On the other hand, static analysis approaches may yield an over-approximation because all the application's code is analyzed, even the code that will never be executed at runtime. Our work can augment these efforts by providing a highly-precise detection technique without code analysis to apps that are out of reach due to limitations provided by obfuscation or code analysis, to enable them to perform code analysis on apps without these limitations to get better results. Thus having a mechanism to detect potential malware in obfuscated apps, using its string resources can contribute to saving resources for analysis. To contribute in this direction, our proposed technique of analysing UI textual description shows promising accuracy ratios for permission-based malware detection in obfuscated apps.

7.4. Towards Sustainable Android Malware Detection

While we have proposed an approach that relies on the sensitivity of permission requested and textual descriptions, we are also aware that the approach might deteriorate over time due to the dynamic nature of the Android ecosystem and evolution of permissions. For example, if malware stop

requesting sensitive permission or provides a textual description that necessitates permission request, then our approach may suffer low detection sensitivity. The same argument can be used for low detection precision when benign apps request sensitive permissions that are not motivated in the textual descriptions. This performance deterioration is because of app evolution and has also been the bane of long-span malware detection of state-of-the-art malware detectors for Android. To understand the over-time classification performance of existing malware detection solutions, we chose six state-of-the-art malware detectors for Android. The baseline malware detectors for Android investigated are the two state-of-the-art dynamic app classifiers *DroidSpan* (Fu and Cai, 2019), *Afonso* (Afonso et al., 2015) and four state-of-the-art static approaches *MamaDroid* (Mariconti et al., 2017), *DroidSieve* (Suarez-Tangil et al., 2017), *DroidAPIMiner* (Aafer et al., 2013), *RevealDroid* (Garcia et al., 2018). *MamaDroid* and *DroidAPIMiner* uses features based on API calls in an app, extracted through static analysis. *DroidSpan* uses longitudinal characterization study of Android app with a focus on their dynamic behaviours as features for app classification. *DroidSieve* uses features computed from app resources such as the API calls, code structure, permissions and the set of invoked components; *Afonso* classifies apps based on calls to predefined lists of APIs and system calls, and *RevealDroid* approaches app classification based on apps' usage of APIs, native code, and reflection.

We conducted two studies: (i) to investigate the performance of the proposed security-oriented view in over-time detection settings and, (ii) to comparatively evaluate the capabilities of the proposed approach against the results presented in *MamaDroid* and *DroidSpan* in five state-of-the-art malware detectors for Android as baselines for the sustainability of detector. Firstly, we compare the performance of our approach to the results presented in *MamaDroid* (Mariconti et al., 2017), where the evaluation of robustness to evolution in malware development and changes in Android API was compared with *DroidAPIMiner* (Aafer et al., 2013). The comparison of our approach to the results is useful because *MamaDroid* was the first work to investigate the sustainability of Android Malware detectors to the best of our knowledge. We evaluate our approach on the datasets that yielded poor results due to the unsustainability of the detectors over three-year spans. The results are F-Score values obtained when the testing dataset is one to three-year apart from the training set. The datasets are malicious (2014, 2015, 2016) and benign (oldbenign). The 2016(M16) contains 2974 malware APKs and oldbenign(OB) contains 5879 benign apks. The 2014(M14) contains 24317 malware APKs and 2015(M15) contains 5314 malware apks. The results are presented in Table 10, where our approach outperforms *MamaDroid* and *DroidAPIMiner* in terms of sustainability of detection accuracy.

In a similar fashion as the previous experiment, we repeated the same procedure for the results presented in *DroidSpan*. The entire study dataset includes 13,627 be-

Table 9

Features Used in the Investigated State-of-the-art Approaches

Detector	Features
Ours	Sensitive permission requests and app textual description
DroidSpan	Longitudinal characterization study of Android app and their dynamic behaviours
MamaDroid	API Calls
DroidAPIMiner	API-Level Behaviour
Afonso	Predefined list of API and system calls
RevealDroid	APIs, native code and reflection
DroidSieve	API Calls, code structure, permissions and the set of invoked components

Table 10Sustainability of Malware Detection in Over-Time Detection Settings in *MamaDroid*

(a) Classification Performance (F1) of Proposed Approach for Sustainable Malware Detection.

Dataset	F-Score
OB+M14	93%
OB+M15	77%
OB+M16	64%
Avg. F-Score	78%

(b) Comparison of Average Performance (F1) Among the Two Malware Detectors in Over-Time Detection Settings (three-year span).

Detector	F-Score
<i>Our Work</i>	78%
<i>DroidAPIMiner</i>	33%
<i>MamaDroid</i>	49%

nign and 12,705 malicious samples, for a total of 26,332 benchmarks. These benign and malicious apps were developed across the past eight years from 2010 to 2017, according to which the entire datasets were divided into 16 yearly datasets, depicted as M10 to M17 for malware and B10 to B17 for benign apps. All the 16 datasets are mutually disjoint (there were no apps shared by any two datasets). The datasets are the same dataset used by *DroidSpan* (Fu and Cai, 2019) in investigating the deterioration of learning-based malware detectors for Android.

Table 11 lists the F1 accuracy of each of the eight independent tests (noted in the first column) achieved by the proposed four security rules. As shown, the proposed approach achieved an average F1 of 70.11% for the datasets of any of the past eight years. Also, the classification performance had very small variations. In comparison, the baselines all had relatively large variations, showing the dependence of their performance on particular datasets. The implication of this is that the proposed approach of using permission request and UI textual descriptions can be useful in identifying old and emerging malware with at least 70% accuracy while giving equal importance to the true positive rates (recall) and precision. For over-time detection, Table 11a lists the mean F1 accuracy of each technique with one-through seven-year spans between training and testing data. The numbers indicate our classifier performed better than the four baselines at any of the seven spans. While *DroidSpan* performed better than our approach, the gap was not very large with the seven-

Table 11Sustainability of Malware Detection in Over-Time Detection Settings in *DroidSpan*

(a) Classification Performance (F1) of Proposed Approach for Sustainable Malware Detection.

Dataset	F-Score
B10+M10	83.84%
B11+M11	71.30%
B12+M12	75.78%
B13+M13	68.26%
B14+M14	55.33%
B15+M15	78.16%
B16+M16	76.84%
B17+M17	51.37%
Avg. F-Score	70.11%

(b) Comparison of Average Performance (F1) Among the Five Malware Detectors in Over-Time Detection Settings (seven-year span).

Detector	F-Score
<i>Our Work</i>	70.11%
<i>DroidSpan</i>	71.43%
<i>MamaDroid</i>	63.67%
<i>Afonso</i>	49.91%
<i>RevealDroid</i>	45.94%
<i>DroidSieve</i>	29.87%

year (71.43% versus 70.11%). However, with all other techniques, the advantages of our approach over the baselines were much more substantial. Over the seven spans, the average F1 of our approach was the highest among the other techniques, followed by *MamaDroid*, *Afonso*, *RevealDroid*, and *DroidSieve* in order. These findings reveal the promise of the proposed approach to long-span malware detection, such that old, new and emerging malware can be detected.

The result in Table 10 and Table 11 shows that the proposed techniques have a better chance of succeeding where other approaches have not. Furthermore, leveraging apps' textual descriptions and sensitive permissions provide a promising vision towards predicting the expected behaviour of an app.

7.5. Discussion

This section provides a discussion of an overview of the work under two main broad headings - i) the rationale for permission and textual descriptions as a feature for Malware Detection ii) major advantages of the proposed approach.

7.5.1. Permissions and App Descriptions as a Feature for Malware Detection

Android permission is the major security mechanism for the Android operating system. Security analysis of mobile apps should, therefore, include analysis of the sensitive privileges a smartphone has access to. Furthermore, at the core of every basic security concept is access control. Adopting the core elements of an access model for mobile software system will be similar to Figure 7. The core elements of an access control model are *Identification*, *Authentication* and *Authorization*. From Figure 7, it can be observed that user information associated with identification and authentication in mobile settings would require graphical user inputs, while the context for Authorization is a typical example of smartphone privileges e.g. Android permissions. Therefore, using Android permissions, which are user-granted privileges, for analysis recognises the security problems of mo-

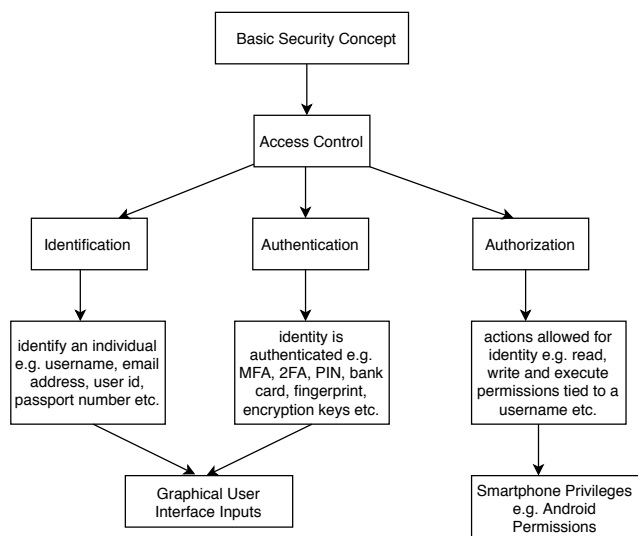


Figure 7: Access Control in Mobile Software Systems

mobile software systems as an authorization problem which deals with access control.

Furthermore, the proposed approach does not just use Android permissions alone, but in conjunction with the app's textual description. This is in line with the fundamental behaviour classification proposed in this study that detects malware based on the mismatch between the app's advertised behaviour and its functional behaviour. The proxy for the app's expected behaviour in this study is the app's textual description, while the proxy for its functional behaviour is the permissions it requests. This is built on the established problem with Android permissions where there is often a mismatch between *required permissions* and *requested permissions* or inconsistencies between what an app is requesting and the functionality (or purpose of the app) (Sarma et al., 2012; Felt et al., 2011a; Lin et al., 2014; Olukoya et al., 2019). If the goal is to improve the security awareness of end-users, the two classes of literal app information that the Android market provides - permission requests and textual descriptions needs to be carefully investigated. The natural language processing of an app's permission request with its description provides an intuitive security-oriented view of the expected behaviour of the app.

While we agree that using this feature leads to false negatives and positives, which is the common attribute of every detection system. We are exploring more informative features like those related to identification and authentication in mobile apps, out of which we have recognised graphical user input as the main source. We believe that the combination of features associated with identification, authentication and authorization in mobile software systems can improve the performance of the detection of malware. We argue that if the goal is to improve the security of Android, one of the major ways to view it is as an access control problem, needing exploration and investigation of user inputs and resources associated with identification, authentication and authorization. The current approach explores the authoriza-

tion aspect of it, and that was the motivation for adopting Android permissions as a feature in this study. A possible extension to the study in which we are actively investigating is evaluating permissions and GUI inputs as the basis for mobile app analysis because, at their core, they form user inputs and resources associated with access control elements in mobile software systems. Another area of a potential extension to this study is by incorporating the Android API permission specification using path-sensitive analysis and graph abstraction proposed in Aafer et al. (2018) for generating the semantic graph for Android permissions. The precise Android API protection specification map can be leveraged for inferring semantic patterns to detect behavioural element mismatch.

7.5.2. Major Advantages of the Approach

- Alternative technique for monitoring app behaviour** - Most detection technology detects malware by monitoring its behaviour, and they often rely on program code analysis to monitor such behaviour. We have argued that program code analysis cannot be extracted sometimes for behaviour analysis due to its limitations (cf Section 7.2). Therefore, our approach is contributing to this area by proposing an app behavioural analysis devoid of program code inspection. This can augment state-of-the-art detection systems for robust behavioural monitoring of apps.
- Sustainability of Android Malware Detectors** - Recent study has shown that existing solutions to Android malware detection deteriorate largely and rapidly over time, such that new and emerging malware often evade detection (Fu and Cai, 2019). We proposed a new classification approach based on permission and app description features with a focus on sensitive behaviours. We evaluated this new approach against the six existing detectors and our approach outperformed five out of the six modern detectors, while being comparable to the sixth (70.11% vs 71.43%) over the 7 year span (cf Table 9, Table 10 and Table 11). The main lesson learned is that studying app behaviour in terms of its sensitive permission request and textual description provides a promising avenue for long-span malware detection.
- Informative Feature for Malware Detection:** Modern detection systems e.g. DREBIN (Arp et al., 2014) or approaches using an ensemble of classifiers e.g. Wang et al. (2018) are exploring more informative features to better characterize the behaviour of apps. The proposed approach can complement this endeavours by demonstrating strong evidence that app descriptions and sensitive permission requests are an informative feature for malware detection. In conclusion, the proposed technique has shown that analysing app advertised behaviour against its functional behaviour is an important feature for characterizing stealthy behaviour of Android apps.

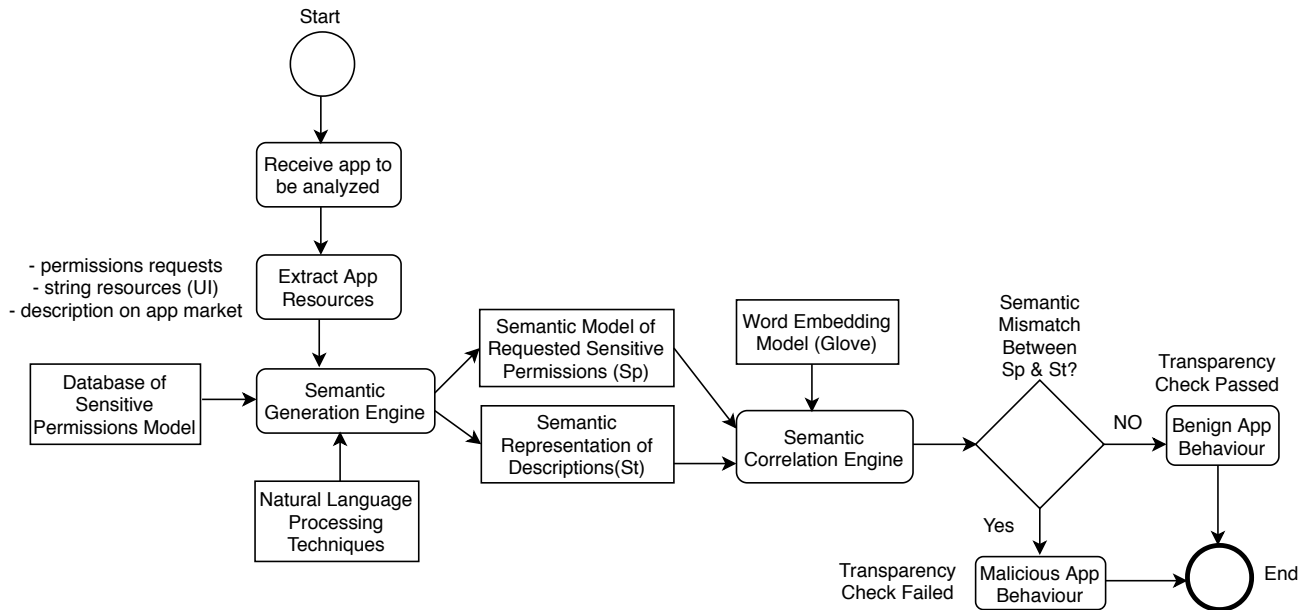


Figure 8: Process Diagram for the Proposed Technique

7.6. Deployment of the Proposed Technique

Figure 8 shows the process diagram for deploying the proposed technique as a tool. It takes as input an app to be analysed and extract the app resources. The resources of concern are the permissions requested, string resources (UI textual description) and the description on the app market. These inputs are fed into the Semantic Generation engine, that has in its system a database of a sensitive permissions model. Based on this resource in its engine, it maps the sensitive permissions requested by the app to the semantic model of permissions and generated the specific semantic model of the permissions requested by the app. Also, based on the app descriptions supplied, it converts them into a semantic representation using natural language processing techniques. The outputs of the semantic generation engine are the semantic representation of the app's textual descriptions and semantic model of its sensitive permissions. These are fed as input into the next stage.

The next stage is the semantic correlation engine, which already has in its memory an unsupervised learning algorithm, a word embedding model, whose function is to measure the mismatch as a function of the similarity between the app's permission semantic representation and textual descriptions. The semantic correlation engine presents as output a decision metric that characterizes an app behaviour based on the transparency metric. An app passes a transparency metric and is categorized as benign, if there is no mismatch between its advertised behaviour (app's textual descriptions) and implemented behaviour (sensitive permission request). Alternatively, the app is characterized as malicious.

For app distribution platform, the approach can be deployed as a tool to measure the compliance and commitment of applications to transparency. This can be used as

part of the app certification process. It could also be used as a rating of the app to augment current forms of community ratings by providing a security perspective to ratings. Since the low rating is bad for the adoption of an app, the approach would allow for more transparency on app distribution platforms. For app developers, this technique can be deployed as a compliance tool for app designers to be aware of the transparency of their design decisions. For app users, the tool can be deployed as a reliable privacy awareness tool that scans apps installed on the phone and users can be informed of potentially malicious apps. It is a reliable privacy awareness tool because the proposed technique has validated that an app with inconsistencies in its advertised and functional behaviour is likely malicious. This can aid user perception of actual app behaviour before adoption. Hence, the behavioural element mismatch proposed in this study nudges the end-user towards privacy awareness, the app distribution platform towards enhanced app ratings, and the developer alike towards enhanced transparency.

7.7. Limitations

Most malware request sensitive permissions that allow them to access sensitive API, code or data in the device, while most benign apps do not (Wang et al., 2014). Intuitively, malware can be distinguished from benign ones by analyzing its description and behavioural elements. We highlight the limitations of our approach regardless of the source of descriptions used - Android Markets or App resources, which is responsible for the 5.4% false positives in Table 5, 1.28% false positives in the 71 malware families.

First, some malware request sensitive permissions that are motivated in their app descriptions. This gives a negative impact on false positives. Secondly, some malware does not request any permissions, while some do not request any

sensitive permission to be malicious, often referred to as Zero permission apps. In this case, relying on only the static behavioural element is not feasible for the description of the app behaviour, as some benign apps, exhibit the same behaviour. As a case study, two out of the 71 malware families in the AMD dataset exhibit such behaviour - SmsZombie and Steek, where our current approach could not identify any malware belonging to this category. Also, malware with descriptions written in other languages, besides English can evade our analysis. This shows that the description and behavioural elements may not be informative enough to comprehensively describe the behaviours of an app.

Another limitation of this approach occurs with benign applications whose developer did not invest time in providing an accurate application description either on the app market or actions that trigger the motivation for its sensitive behaviour in the UI textual description. In this case, the application is marked as malicious even though benign. In this case, the behavioural element mismatch proposed in this study can aid developer awareness as a measure for checking compliance. Therefore, developers can be aware of the gap between the advertised behaviour and functional behaviour during design time. While we have discussed how the proposed behavioural element mismatch nudges user towards privacy awareness, app developers can also benefit from it as an enhancement tool during design for transparency.

However, we argue that the capability of the analysis technique is orthogonal to our main focus. In future work, we are exploring more behavioural elements to reduce false positives. One of the ways of handling such complexities may be investigating the system-call runtime behaviour of such apps (Tam et al., 2015; Yan and Yin, 2012; Yang et al., 2013).

8. Conclusion

This study provides a security-oriented overview of the expected (predicted) app behaviour, while the evaluation involves validating the predicted behaviour with its actual behaviour by investigating the correctness of the anomaly detector it in Android malware detection settings. We propose gathering enough textual semantics from app description and permission requests to enhance malware detection systems that rely on checking the behavioural element mismatch in apps. Specifically, we propose a revision of the Android permission system and investigating mismatches between the *advertised* and *implemented* app behaviour. Our approach improves the accuracy of state-of-the-art approaches by reducing the false negatives with Whyper, AutoCog, CHABADA by at least 87%, and TAPVerifier by at least 57%. We also evaluated the robustness of our technique on 71 malware families and achieved a precision of 98.72%. We also demonstrate that our approach is obfuscation resilient, and could be applied for advancing the security of mobile applications. Finally, we demonstrate that analysing sensitive permissions requested and UI textual descriptions provides

a promising avenue for long-span malware detection. The key finding in this study is that app descriptions and a fine-grained stratification of user-granted privileges (e.g. permissions) are a promising way of enhancing a user perception of the actual behaviour of Android apps and the detection of malicious apps.

In general, the proposed approach augments existing work and presents a complementary mechanism for enhancing the privacy of mobile device users. For example, the behavioural element mismatch characterization as a measure of trustworthiness of an app can be used as a privacy risk indicator to support current forms of community ratings, to aid user comprehension of transparency. Previous studies have shown that community ratings on distribution platforms are not sufficient to enable users to become aware of the level of privacy risk they are exposed to when they download an app (Chia et al., 2012). Our study further showed that user and community ratings on app distribution platforms are poor indicators of security. Finally, our analysis shows that exploring behavioural elements (app descriptions, UI textual descriptions and permission requests) is a promising way of enhancing a user perception of the actual behaviour of Android applications and the detection of malicious apps.

References

- Aafer, Y., Du, W., Yin, H., 2013. Droidapiminer: Mining api-level features for robust malware detection in android, in: International conference on security and privacy in communication systems, Springer. pp. 86–103.
- Aafer, Y., Tao, G., Huang, J., Zhang, X., Li, N., 2018. Precise android api protection mapping derivation and reasoning, in: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, ACM. pp. 1151–1164.
- Afonso, V.M., de Amorim, M.F., Grégio, A.R.A., Junquera, G.B., de Geus, P.L., 2015. Identifying android malware using dynamically obtained features. *Journal of Computer Virology and Hacking Techniques* 11, 9–17.
- Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., Rieck, K., Siemens, C., 2014. Drebin: Effective and explainable detection of android malware in your pocket., in: *Ndss*, pp. 23–26.
- Au, K.W.Y., Zhou, Y.F., Huang, Z., Lie, D., 2012. Pscout: analyzing the android permission specification, in: Proceedings of the 2012 ACM conference on Computer and communications security, ACM. pp. 217–228.
- Baalous, R., Poet, R., 2018. How dangerous permissions are described in android apps' privacy policies?, in: Proceedings of the 11th International Conference on Security of Information and Networks, ACM. p. 26.
- Babil, G.S., Mehani, O., Boreli, R., Kaafar, M.A., 2013. On the effectiveness of dynamic taint analysis for protecting against private information leaks on android-based devices, in: 2013 International Conference on Security and Cryptography (SECRYPT), IEEE. pp. 1–8.
- Backes, M., Bugiel, S., Derr, E., McDaniel, P.D., Octeau, D., Weisberger, S., 2016. On demystifying the android application framework: Revisiting android permission specification analysis., in: USENIX Security Symposium, pp. 1101–1118.
- Barrera, D., Kayacik, H.G., van Oorschot, P.C., Somayaji, A., 2010. A methodology for empirical analysis of permission-based security models and its application to android, in: Proceedings of the 17th ACM conference on Computer and communications security, ACM. pp. 73–84.
- Chen, D., Manning, C., 2014. A fast and accurate dependency parser using neural networks, in: Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP), pp. 740–750.
- Chen, K.Z., Johnson, N.M., D'Silva, V., Dai, S., MacNamara, K., Magrino, T.R., Wu, E.X., Rinard, M., Song, D.X., 2013. Contextual policy

- enforcement in android applications with permission event graphs., in: NDSS, p. 234.
- Chia, P.H., Yamamoto, Y., Asokan, N., 2012. Is this app safe?: a large scale study on application permissions and risk signals, in: Proceedings of the 21st international conference on World Wide Web, ACM. pp. 311–320.
- Desnou, A., 2017. Androguard.(2017). Technical Report. Retrieved 2017-2-18 from <https://github.com/androguard/androguard>.
- Dong, S., Li, M., Diao, W., Liu, X., Liu, J., Li, Z., Xu, F., Chen, K., Wang, X., Zhang, K., 2018. Understanding android obfuscation techniques: A large-scale investigation in the wild, in: International Conference on Security and Privacy in Communication Systems, Springer. pp. 172–192.
- Duan, Y., Zhang, M., Bhaskar, A.V., Yin, H., Pan, X., Li, T., Wang, X., Wang, X., 2018. Things you may not know about android (un) packers: a systematic study based on whole-system emulation, in: 25th Annual Network and Distributed System Security Symposium, NDSS, pp. 18–21.
- Enck, W., Gilbert, P., Han, S., Tendulkar, V., Chun, B.G., Cox, L.P., Jung, J., McDaniel, P., Sheth, A.N., 2014. Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. ACM Transactions on Computer Systems (TOCS) 32, 5.
- Enck, W., Octeau, D., McDaniel, P.D., Chaudhuri, S., 2011. A study of android application security., in: USENIX security symposium, p. 2.
- Felt, A.P., Chin, E., Hanna, S., Song, D., Wagner, D., 2011a. Android permissions demystified, in: Proceedings of the 18th ACM conference on Computer and communications security, ACM. pp. 627–638.
- Felt, A.P., Finifter, M., Chin, E., Hanna, S., Wagner, D., 2011b. A survey of mobile malware in the wild, in: Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices, ACM. pp. 3–14.
- Felt, A.P., Greenwood, K., Wagner, D., 2011c. The effectiveness of application permissions, in: Proceedings of the 2nd USENIX conference on Web application development, pp. 7–7.
- Felt, A.P., Ha, E., Egelman, S., Haney, A., Chin, E., Wagner, D., 2012. Android permissions: User attention, comprehension, and behavior, in: Proceedings of the eighth symposium on usable privacy and security, ACM. p. 3.
- Feng, Y., Anand, S., Dillig, I., Aiken, A., 2014. Appscopy: Semantics-based detection of android malware through static analysis, in: Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, ACM. pp. 576–587.
- Frank, M., Dong, B., Felt, A.P., Song, D., 2012. Mining permission request patterns from android and facebook applications, in: Data Mining (ICDM), 2012 IEEE 12th International Conference on, IEEE. pp. 870–875.
- Fu, X., Cai, H., 2019. On the deterioration of learning-based malware detectors for android, in: Proceedings of the 41st International Conference on Software Engineering: Companion Proceedings, IEEE Press. pp. 272–273.
- Garcia, J., Hammad, M., Malek, S., 2018. Lightweight, obfuscation-resilient detection and family identification of android malware. ACM Transactions on Software Engineering and Methodology (TOSEM) 26, 11.
- Gorla, A., Tavecchia, I., Gross, F., Zeller, A., 2014. Checking app behavior against app descriptions, in: Proceedings of the 36th International Conference on Software Engineering, ACM. pp. 1025–1035.
- Grace, M., Zhou, Y., Zhang, Q., Zou, S., Jiang, X., 2012. Riskranker: scalable and accurate zero-day android malware detection, in: Proceedings of the 10th international conference on Mobile systems, applications, and services, ACM. pp. 281–294.
- Guo, L., 2018. stanfordcorenlp: A python wrapper to stanford corenlp server. <https://github.com/Lynten/stanford-corenlp>.
- Huang, J., Zhang, X., Tan, L., Wang, P., Liang, B., 2014. Asdroid: Detecting stealthy behaviors in android applications by user interface and program behavior contradiction, in: Proceedings of the 36th International Conference on Software Engineering, ACM. pp. 1036–1046.
- Islam, A., Inkpen, D., 2008. Semantic text similarity using corpus-based word similarity and string similarity. ACM Transactions on Knowledge Discovery from Data (TKDD) 2, 10.
- Jiang, X., Zhou, Y., 2012. Dissecting android malware: Characterization and evolution, in: 2012 IEEE Symposium on Security and Privacy, IEEE. pp. 95–109.
- Kelley, P.G., Consolvo, S., Cranor, L.F., Jung, J., Sadeh, N., Wetherall, D., 2012. A conundrum of permissions: installing applications on an android smartphone, in: International Conference on Financial Cryptography and Data Security, Springer. pp. 68–79.
- Kelly, G., 2014. Report: 97% of mobile malware is on android. this is the easy way you stay safe. Forbes Tech .
- King, J., Lampinen, A., Smolen, A., 2011. Privacy: Is there an app for that?, in: Proceedings of the Seventh Symposium on Usable Privacy and Security, ACM. p. 12.
- Laskov, P., et al., 2014. Practical evasion of a learning-based classifier: A case study, in: Security and Privacy (SP), 2014 IEEE Symposium on, IEEE. pp. 197–211.
- Lee, M.C., Chang, J.W., Hsieh, T.C., 2014. A grammar-based semantic similarity algorithm for natural language sentences. The Scientific World Journal 2014.
- Li, Y., McLean, D., Bandar, Z.A., O'shea, J.D., Crockett, K., 2006. Sentence similarity based on semantic nets and corpus statistics. IEEE transactions on knowledge and data engineering 18, 1138–1150.
- Lin, J., Liu, B., Sadeh, N., Hong, J.I., 2014. Modeling users' mobile app privacy preferences: Restoring usability in a sea of permission settings, in: Symposium on Usable Privacy and Security (SOUPS), USENIX Association.
- Maiorca, D., Ariu, D., Corona, I., Aresu, M., Giacinto, G., 2015. Stealth attacks: An extended insight into the obfuscation effects on android malware. Computers & Security 51, 16–31.
- Mariconti, E., Onwuzurike, L., Andriotis, P., Cristofaro, E., Ross, G., 2017. Mamadroid: Detecting android malware by building markov chains of behavioral models, in: Proceedings of the 24th Network and Distributed System Security Symposium, NDSS. Internet Society.
- Mirzaei, O., de Fuentes, J.M., Tapiador, J., Gonzalez-Manzano, L., 2019. Androdet: An adaptive android obfuscation detector. Future Generation Computer Systems 90, 240–261.
- Mower, J.P., 2005. Prep-mt: predictive rna editor for plant mitochondrial genes. BMC bioinformatics 6, 96.
- Olukoya, O., Mackenzie, L., Omoronyia, I., 2019. Permission-based risk signals for app behaviour characterization in android apps, in: Proceedings of the 5th International Conference on Information Systems Security and Privacy - Volume 1: ICISSP, INSTICC. SciTePress. pp. 183–192. doi:10.5220/0007248701830192.
- Pandita, R., Xiao, X., Yang, W., Enck, W., Xie, T., 2013. Whyper: Towards automating risk assessment of mobile applications., in: USENIX Security Symposium.
- Pawar, A., Mago, V., 2018. Calculating the similarity between words and sentences using a lexical database and corpus statistics. IEEE transactions on knowledge and data engineering .
- Pennington, J., Socher, R., Manning, C., 2014. Glove: Global vectors for word representation, in: Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP), pp. 1532–1543.
- Qu, Z., Rastogi, V., Zhang, X., Chen, Y., Zhu, T., Chen, Z., 2014. Autocog: Measuring the description-to-permission fidelity in android applications, in: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, ACM. pp. 1354–1365.
- Rastogi, V., Chen, Y., Enck, W., 2013a. Appsplyground: automatic security analysis of smartphone applications, in: Proceedings of the third ACM conference on Data and application security and privacy, ACM. pp. 209–220.
- Rastogi, V., Chen, Y., Jiang, X., Can, C.M.I.Y., 2013b. Evaluating android anti-malware against transformation attacks. Electrical Engineering and Computer Science Department, Northwestern University, North Carolina State University, Technical Report NUEECS-13-01 , 2.
- Rehurek, R., Sojka, P., 2010. Software framework for topic modelling with large corpora, in: In Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks, Citeseer.
- Roberts, J.M., 2011. Virus share.(2018). URL <https://virusshare.com> .

- Rubenstein, H., Goodenough, J.B., 1965. Contextual correlates of synonymy. *Communications of the ACM* 8, 627–633.
- Sarma, B.P., Li, N., Gates, C., Potharaju, R., Nita-Rotaru, C., Molloy, I., 2012. Android permissions: a perspective combining risks and benefits, in: *Proceedings of the 17th ACM symposium on Access Control Models and Technologies*, ACM. pp. 13–22.
- Sebastian, Z., Peter, S., Daniel, S., Abhilasha, R., Ziqi, W., Joel, R., N. Cameron, R., Norman, S., 2019. Maps: Scaling privacy compliance analysis to a million apps. *Proceedings on Privacy Enhancing Technologies* 3, 66–86.
- Story, P., Zimmeck, S., Sadeh, N., 2018. Which apps have privacy policies?, in: *Privacy Technologies and Policy*, Springer International Publishing. pp. 3–23.
- Suarez-Tangil, G., Dash, S.K., Ahmadi, M., Kinder, J., Giacinto, G., Cavallaro, L., 2017. Droidsieve: Fast and accurate classification of obfuscated android malware, in: *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, ACM. pp. 309–320.
- Tam, K., Khan, S.J., Fattori, A., Cavallaro, L., 2015. Copperdroid: Automatic reconstruction of android malware behaviors., in: *NDSS*.
- Wang, W., Li, Y., Wang, X., Liu, J., Zhang, X., 2018. Detecting android malicious apps and categorizing benign apps with ensemble of classifiers. *Future Generation Computer Systems* 78, 987–994.
- Wang, W., Wang, X., Feng, D., Liu, J., Han, Z., Zhang, X., 2014. Exploring permission-induced risk in android applications for malicious application detection. *IEEE Transactions on Information Forensics and Security* 9, 1869–1882.
- Wang, Y., Rountev, A., 2017. Who changed you?: obfuscator identification for android, in: *Proceedings of the 4th International Conference on Mobile Software Engineering and Systems*, IEEE Press. pp. 154–164.
- Wei, F., Li, Y., Roy, S., Ou, X., Zhou, W., 2017. Deep ground truth analysis of current android malware, in: *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, Springer. pp. 252–276.
- Wei, X., Gomez, L., Neamtiu, I., Faloutsos, M., 2012. Permission evolution in the android ecosystem, in: *Proceedings of the 28th Annual Computer Security Applications Conference*, ACM. pp. 31–40.
- Xu, W., Qi, Y., Evans, D., 2016. Automatically evading classifiers, in: *Proceedings of the 2016 Network and Distributed Systems Symposium*.
- Xue, L., Zhou, Y., Chen, T., Luo, X., Gu, G., 2017. Malton: Towards on-device non-invasive mobile malware analysis for art, in: *26th USENIX Security Symposium (USENIX Security 17)*.
- Yan, L.K., Yin, H., 2012. Droidscape: Seamlessly reconstructing the os and dalvik semantic views for dynamic android malware analysis., in: *USENIX security symposium*, pp. 569–584.
- Yang, Z., Yang, M., Zhang, Y., Gu, G., Ning, P., Wang, X.S., 2013. Appintent: Analyzing sensitive data transmission in android for privacy leakage detection, in: *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, ACM. pp. 1043–1054.
- Yu, L., Luo, X., Qian, C., Wang, S., 2016. Revisiting the description-to-behavior fidelity in android applications, in: *Software Analysis, Evolution, and Reengineering (SANER), 2016 IEEE 23rd International Conference on*, IEEE. pp. 415–426.
- Zhang, M., Duan, Y., Feng, Q., Yin, H., 2015a. Towards automatic generation of security-centric descriptions for android apps, in: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, ACM. pp. 518–529.
- Zhang, M., Duan, Y., Yin, H., Zhao, Z., 2014. Semantics-aware android malware classification using weighted contextual api dependency graphs, in: *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, ACM. pp. 1105–1116.
- Zhang, Y., Luo, X., Yin, H., 2015b. Dexhunter: toward extracting hidden code from packed android applications, in: *European Symposium on Research in Computer Security*, Springer. pp. 293–311.
- Zhang, Y., Yang, M., Xu, B., Yang, Z., Gu, G., Ning, P., Wang, X.S., Zang, B., 2013. Vetting undesirable behaviors in android apps with permission use analysis, in: *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, ACM. pp. 611–622.
- Zhauniarovich, Y., Gadyatskaya, O., 2016. Small changes, big changes: an updated view on the android permission system, in: *International Symposium on Research in Attacks, Intrusions, and Defenses*, Springer. pp. 346–367.
- Zhou, Y., Wang, Z., Zhou, W., Jiang, X., 2012. Hey, you, get off of my market: detecting malicious apps in official and alternative android markets., in: *NDSS*, pp. 50–52.

Oluwafemi Olukoya completed his PhD at the School of Computing Science, the University of Glasgow with a specialization in malware analysis and mitigation techniques for mobile systems and security. He obtained his master's degree in Artificial Intelligence from the University of Edinburgh. His main research includes malware analysis, secured software engineering, decision & game-theoretic approaches to network security, reverse engineering and permission granting in modern operating systems.

Lewis Mackenzie is a Senior Lecturer in Computing Science at the School of Computing Science, University of Glasgow. His research interests include multicompiler architectures and simulation, mobile ad hoc, vehicular and wireless sensor networks; network security, usable security, and theory of computation.

Inah Omoronyia is a Lecturer in Software Engineering and Information Security at the School of Computing Science, University of Glasgow. He obtained his PhD from University of Strathclyde and was awarded a European Research Consortium for Informatics and Mathematics (ERCIM) fellowship in 2010. He has worked as a research fellow at the Norwegian University of Science and Technology (NTNU) and the Irish Software Research Centre (Lero). His main research interest is in designing secured and privacy-preserving software systems, privacy and security requirements. Most of his current research revolves around building frameworks, tools and techniques for engineering secured and privacy-preserving software. (<http://www.dcs.gla.ac.uk/inah/>)