Towards Using Unstructured User Input Request for Malware Detection

Oluwafemi Olukoya[a,*], Lewis Mackenzie[a] and Inah Omoronyia[a]

[a]School of Computing Science, University of Glasgow, Scotland, United Kingdom

ARTICLE INFO

Keywords:
User Input Identification
Mobile Privacy
Android Malware
Android Security
Android Privacy
Mobile Security

ABSTRACT

Privacy analysis techniques for mobile apps are mostly based on system-centric data originating from well-defined system API calls. But these apps may also collect sensitive information via their unstructured input sources that elude privacy analysis. The consequence is that users are unable to determine the extent to which apps may impact on their privacy when downloaded and installed on mobile devices. To this end, we present a privacy analysis framework for unstructured input. Our approach leverages app meta-data descriptions and taxonomy of sensitive information, to identify sensitive unstructured user input. The outcome is an understanding of the level of user privacy risk posed by an app based on its unstructured user input request. Subsequently, we evaluate the usefulness of the unstructured sensitive user input for malware detection. We evaluate our methods using 175K benign apps and 175K malware APKs. The outcome highlights that malicious app detector built on unstructured sensitive user achieve an average balance accuracy of 0.996 demonstrated with Trojan-Banker and Trojan-SMS when the malware family and target applications are known and balanced accuracy of 0.70 with generic malware.

1. Introduction

The analysis of privacy-sensitive disclosure in Android apps has been studied extensively (Enck et al., 2014; Sun et al., 2016; Arzt et al., 2014; Huang et al., 2014; Avdiienko et al., 2015). This is especially useful when identifying privacy violations, such as insecure information transmission, sensitive data leakage and malware identification (Fahl et al., 2012; Lu et al., 2012; Sounthiraraj et al., 2014; Gibler et al., 2012). Existing research has approached this analysis by identifying privacy violations in data originating from well-defined API calls made by mobile applications (Demetriou et al., 2016; HAN et al., 2013; Enck et al., 2014; Sun et al., 2016; Arzt et al., 2014; Huang et al., 2014). These are the so-called system-centric data sources, which only addresses a source of sensitive user data in a system-centric manner. In this approach, API calls where sensitive data is involved are well structured and explicitly represented in the semantics of the data returned by the API invocation. However, mobile apps frequently request user data input within their Graphical User Interfaces (GUIs). Such data may be highly unstructured and contain sensitive information that eludes privacy analysis. Furthermore, by their unstructured nature, API invocations are unable to represent the semantics of the data returned. This makes it impossible to resolve the degree of sensitivity of disclosed data and to what extent this impacts on user privacy. This alternative source of data input has so far received little research consideration during privacy analysis. This work is motivated by the view that, as mobile apps increasingly request a wide range of sensitive data from users, analysing sensitive disclosure using data originating from unstructured input sources becomes essential to preserve privacy.

When dealing with structured data, well-defined API calls for system-centric data such as location parameters can be retrieved using precise API calls such as `getLongitude()`. However, it is impossible to get such a level of precision for unstructured data. This is because API calls for user input through GUIs are generic. For example, if a user's credit card and social security number are entered into the `EditText` widget (the standard text entry widget in Android), both user information items are retrieved by the same `getText()` API calls. The consequence is that it becomes impossible to identify the type of data collected (here, whether the data is a credit card or social security number of the user), its value and the consequence of disclosure on privacy. One way to associate semantics with unstructured input is to utilise a user-centric approach. This involves asking users to assign sensitivity values to each piece of information disclosed. This can be based on preference or context of use (Zhou et al., 2014). Alternatively, a developer may manually specify content within apps that need to be protected, based on investigations of prior experience (Zhou et al., 2014; Xu et al., 2012; Yang et al., 2013). The problem with these approaches is that they usually entail intensive human intervention, hence increasing the burden on the user or the developer and can quickly become error-prone. Generally, it has been demonstrated that placing the responsibility of such decision-making on users is a key weakness to adequate protection of privacy (Acquisti et al., 2015).

In this study, we propose a technique that uses meta-data associated with unstructured input sources to understand privacy-sensitive information disclosure in mobile apps. First, we generate a taxonomy of the types of sensitive information that could be disclosed in mobile settings. The taxonomy is then combined with a word embedding model to annotate unstructured user input with a measure of their sensitivity. This measure is an indication of the privacy risk that the app poses to users. Finally, malware detectors built

*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)

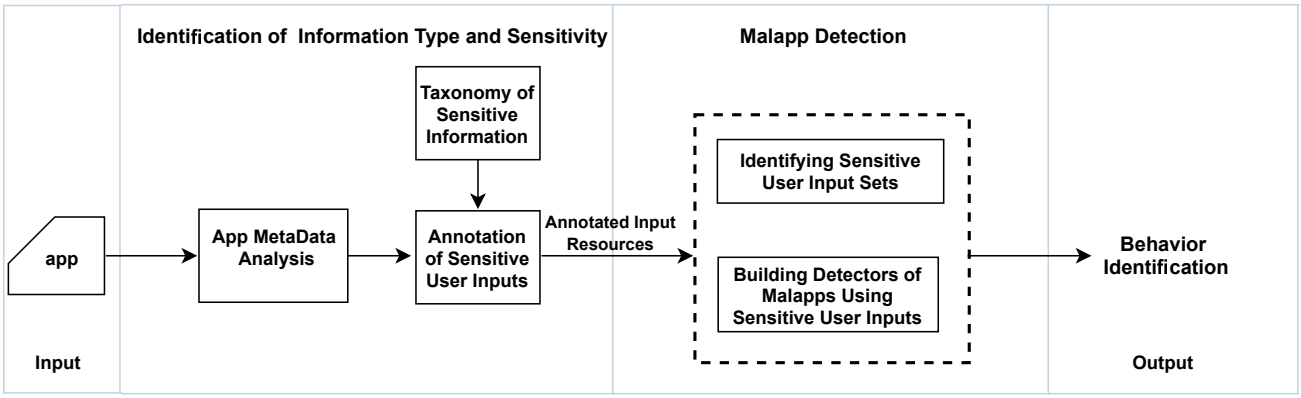

Figure 1: A framework for Exploring Unstructured Input Sources in Android Apps for Malicious Application Detection

on unstructured sensitive user input request are used to determine if the app is benign and privacy-preserving. One key advantage of our approach is that using app meta-data to ascribe semantics to unstructured data inputs is a common practice. For example, it is common to declare unstructured UI elements in XML (static layouts) for Android apps. This allows developers to better separate the presentation of the application from the code that controls its behaviour. These UI descriptions are external to the application code, which means developers can modify or adapt them without having to modify the source code and recompile. Another advantage is that major mobile operating systems, such as Android, iOS and Windows provides UI frameworks as part of their app development APIs. Finally, our approach provides for automatic determination of the sensitivity of unstructured user input request, without requiring users or developers continually to tag sensitivity values.

2. System Overview

Assume a scenario where a user is to disclose personal information through an unstructured input resource such as a text input widget on a mobile app. Our objective is to investigate whether the disclosure behaviour of the app, with respect to the information being disclosed, would be benign or malicious. Three steps are required to achieve this objective. These are: identifying the type and sensitivity of the information that is being disclosed from the unstructured input resource; determining how benign or malicious the app is given the derived information type and its sensitivity; and, finally, establishing the extent to which the user is vulnerable to privacy violation after disclosure through behaviour identification. The proposed framework is described in Figure 1.

To identify the type of information and its sensitivity from an unstructured input resource, the app's meta-data that describes the resource is first analysed, then benchmarked against a taxonomy of sensitive information. Analysing the meta-data of User Interface (UI) elements is important as it provides vital insights into the semantic information that would otherwise be lost because of the unstructured nature of the input. Furthermore, the taxonomy of sensitive information enables us to determine if the semantics derived from the analysis of resource meta-data relates to any sensitive personal information. This benchmarking is achieved by using a word embedding model to annotate the input resource. The outcome is a sensitivity index that indicates the amount of sensitive personal information that is being used by the app. A detailed description of this first step is given in Section 5.

During the second step, we investigate the risk introduced by the collaboration of several sensitive user input type sets. Thirdly, the detection of malware based on sensitive user input types is formulated as a classification problem and executed by building classifiers. In the final step, to determine the risk caused by sensitive user input type request and use it to report malware, detection rules are extracted from malware detectors. We then employ the detection rules to detect unknown malware. A detailed description of this second step is given in Section 6.

In summary, the proposed framework has two major components - the identification of information type and sensitivity phase, which annotates input resources; the malware detection phase, which builds detectors for malware using sensitive user inputs. The main contributions of this work are summarized below:

1. **Validated Taxonomy of Sensitive User Information**

   - A taxonomy model of unstructured input sources that can be used to generate a sensitivity spectrum that indicates the level of privacy risk that mobile app users are exposed to. The relevance of the fine-grained stratification of user inputs was validated for identifying mobile threat for data inputs.
   - A technique for the semantic resolution of unstructured user inputs based on the type of information request. The proposed approach infers

sensitive UI items in Android apps. Other privacy analysis of mobile apps based on GUI data can incorporate our approach for robust analysis.

- The proposed technique of annotating privacy-sensitive unstructured inputs in apps could be developed into a tool that helps developers understand the significance of the data they request from users. The proposed approach is built on the idea of measuring the distance to an existing taxonomy, and the approach can generalize to other, including customized taxonomies.

2. **Deep Insights about Unstructured User Inputs**

   - An approach, towards a vision of detecting generic and known malware family based on using sensitive user input requests. Informative features were reported from unstructured user input requests that can detect unknown and known malware. The proposed technique achieved a balanced accuracy of 99.6% distinguishing between Trojan-Banker and legitimate mobile banking applications and a 99.8% accuracy detection with Trojan-SMS and benign applications with similar functionalities based on unstructured user data request alone. The decision rules from unstructured user input request provide a promising vision with regards to a reliable measure of distinguishing between a legitimate mobile banking application and a Trojan-Banker based on an unstructured data request. The approach can detect generic malware with a detection accuracy of 70%.
   - The proposed technique demonstrates a strong association between malicious behaviour and the use of unstructured inputs, in certain categories of malware. The technique can complement state-of-the-art systems. Permission-based malware detection approach can incorporate our technique for "zero-permission" apps who do not require user-granted permissions for their functionalities. The analysis is based on a large dataset of malware for the evaluation to demonstrate the effectiveness of the approach. User input vectors extracted from the dataset and associated experimental results are published as a potential benchmark data for the research community[1]. To the best of our knowledge, ours is the largest dataset for malware analysis in published studies.

The rest of the manuscript is organised as follows. Section 3 details an analysis of relevant existing work. This is followed by Section 4 where the motivation for the study, research problem and the importance of the research is explained. Section 5 discusses the first step in the proposed privacy analysis framework which is concerned with identifying the nature and the sensitivity of the information being requested through unstructured user input sources. Section 6 presents the methodology of using the proposed privacy analysis framework for malware detection. The evaluation is presented in Section 7, where we investigate research questions associated with the accuracy of the semantic resolution of unstructured user inputs, the relevance of the taxonomy of sensitive user information for identifying mobile threats, the effectiveness of unstructured input for malware identification and detection in cases of known malware family with target applications and generic malware. Section 8 highlights the major advantages and contributions of the proposed approach by providing a comparative analysis with the state-of-the-art techniques. In section 9, the feasibility of the proposed technique is discussed with its strength and limitations. Section 10 concludes the study by presenting an overview of the study, main contributions, key findings and avenues for future work.

3. Related Work

3.1. Privacy Analysis of Mobile Apps

There is considerable work on the privacy analysis of mobile apps that are focused on the Android platform (Au et al., 2012; Enck et al., 2014; Sun et al., 2016; Gibler et al., 2012; Grace et al., 2012b; HAN et al., 2013; Lu et al., 2015). In the identification of information sources, these works have predominantly focused on system-centric input points that usually have clearly defined permissions. Indeed, Rasthofer et al.(Rasthofer et al., 2014) proposed a machine learning approach that automatically identified and categorised the system-centric Android sources and sinks frequently used in program code. However, our work is focused on sensitive information from unstructured input sources which do not have well-defined API points on the Android platform.

Taint analysis has been used in software engineering to improve the design and implementation of mobile apps (Li et al., 2016; Arzt et al., 2014; Gordon et al., 2015; Sun et al., 2016; Wei et al., 2014; Huang et al., 2014; Lu et al., 2012; Yang and Yang, 2012; Li et al., 2015). Practically this involves labelling (tainting) sensitive data from certain sources and monitoring propagation through the program code. *FlowDroid*(Arzt et al., 2014), *IccTA*(Li et al., 2015), *AmanDroid*(Wei et al., 2014) and *DroidSafe*(Gordon et al., 2015) are the most prominent static taint analysis for Android applications. *FlowDroid*(Arzt et al., 2014) is a flow-, context-, field, and object-sensitive static analysis for Android applications. It precisely models the Android lifecycle and handles data propagation via callbacks of UI objects. *IccTA*(Li et al., 2015) extends FlowDroid with the analysis of inter-component communications (ICC) to detect privacy leaks in Android apps. *AmanDroid*(Wei et al., 2014) implements a flow- and context-sensitive intra-component data flow analysis. On top of an inter-procedural control flow graph and data flow graph, *Amandroid* builds a data-

[1] https://www.dropbox.com/sh/itvs2itwukboqlk/AAAO0BDJ_OS1H3YBRLeeut2Ga?dl=0

dependency graph for each component and then generates a summary table documenting possible component communication connections. *DroidSafe*(Gordon et al., 2015) implements an object-sensitive and flow-insensitive analysis. It builds a comprehensive Android execution model that contains analysis stubs for most of the Android framework and native methods. TaintDroid(Enck et al., 2014) is a notable dynamic taint analysis system for identifying Android applications that leak private information, such as device identifiers. This generic labelling technique has also been used for malware detection (Yan and Yin, 2012; Tam et al., 2015; Fuchs et al., 2009). Other approaches(Wang et al., 2014; **?**, 2018; **?**) have extracted a large number of features from apps to detect their malicious behaviours. While the traditional taint approach focuses on labels that originate from program code, our work is centred on information type that originates from user input widgets, identified by analysing the mobile app metadata.

3.2. User Input Identification in Mobile Apps

The most similar studies to our own in the area of semantic resolution of user input are *BIDTEXT*(Huang et al., 2016), *SUPOR*(Huang et al., 2015), *AppsPlayground*(Rastogi et al., 2013), *UiRef* (Andow et al., 2017) and *UIPicker*(Nan et al., 2015). *SUPOR*, *AppsPlayground* and *UIPicker* aims to identify and detect sensitive user input in Android apps by analysing semantic information within the app resources. *BIDTEXT* detects sensitive data disclosure via bi-directional text correlation analysis that propagates the variable sets through forward and backward data flow. While *SUPOR* focuses mainly on a specific type of UI elements (EditText), UIPicker considers limited categories of UI data. *BIDTEXT* focuses on identifying sensitive data that is not generated by specific API calls and reports potential disclosures when data is recognized only after the sink operations. *UiRef* improved upon *SUPOR* by addressing ambiguity in the semantic resolution of user inputs. The common denominator among these works is the application of semantic resolution of user inputs in Android applications to understand what information Android apps are requesting through their GUIs. Specifically, these papers focus more on detecting user inputs than on the nature and type of information (sensitivity of information). Hence, it is difficult to examine the privacy-preserving ability of an app without knowing the type of information the app collects and the degree of its sensitivity.

3.3. Privacy Taxonomy

The concept of privacy taxonomy has been studied in software systems. Solove(Solove, 2005, 2008) proposed a taxonomy of privacy violations that are broadly applicable to software systems. Thomas et al.(Thomas et al., 2014) as a follow up from previous empirical studies of identifying privacy violations related to the use of mobile applications(Mancini et al., 2009), proposed a taxonomy of privacy threats and associated harms that can arise due to inappropriate information flow to the end-user. Our approach involves the use of a taxonomy of predefined degrees of sensitivity. This is based on the observation that some data are more sensitive than others. This builds on the philosophy, that not all data is created equal. Hence, separating information into distinct categories or levels by which different requirements or protection applies is an important step in adequately protecting privacy in mobile applications. It is difficult to determine if the privacy requirements guiding the information flow is satisfied if the question of the value of the information has not been answered. Likewise, it is difficult to answer the question of the value of information, without the type and nature being known. By having a taxonomy, we can analyse the sensitive information flow for mobile privacy. In this work, we propose a privacy taxonomy for mobile settings to help inform privacy risks, rather than ask end-users to categorise data.

In conclusion, from our review of past literature, we observe some interesting patterns and gaps: Firstly, existing research has approached this analysis by identifying privacy violations in data originating from permissions and well-defined API calls made by mobile applications. These are the so-called system-centric data sources. In this approach, API calls where sensitive data is involved are well structured and explicitly represented in the semantics of the data returned by the API invocation. However, mobile apps frequently request user data input within their Graphical User Interfaces (GUIs). Such data may be highly unstructured and contain sensitive information that eludes privacy analysis. For example, zero-permission apps do not request any permission to be functional, hence, such apps may evade privacy analysis. Also, state-of-the-art approaches have been focused on malware analysis using the system-centric sources, this study is motivated with the potential of providing a complementary solution to the malware detection problem.

4. Motivation for Analysing Unstructured User Input

Android permission is the major security mechanism for the operating system. Security analysis of mobile apps should, therefore, include analysis of the privileges a smartphone has access to. Furthermore, at the core of every basic security concept is access control models. In providing a complementary solution to the detection problem, we adopt the access control model for the context of mobile software systems as shown in Figure 2. The core elements of a typical access control model are *identification, authentication* and *authorization*. *Identification* and *Authentication* often require graphical user inputs, while the *Authorization* aspect is an example of smartphone privileges e.g. Android permissions. While using permissions for analysis recognises the security problems of mobile software systems as an access control problem, we propose a complementary approach that analyses the privacy and security of mobile apps based on data inputs that relate to *Identification* and *Authentication*. In the context of mobile software systems, user inputs associated with Identification may involve username, e-mail address, user id, passport number etc, while those associated

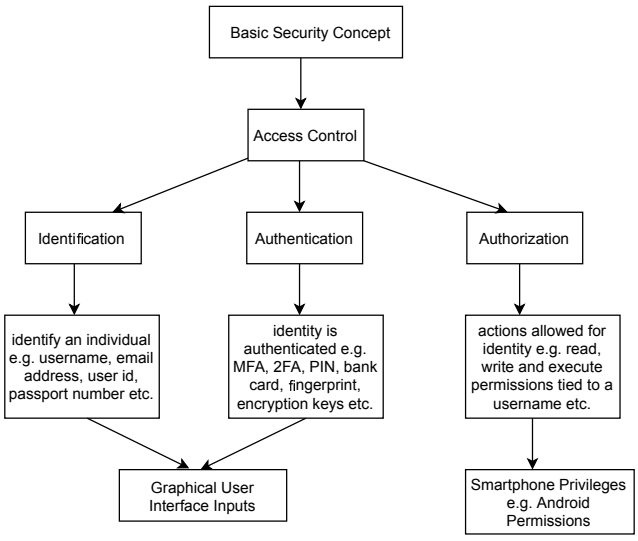

Figure 2: Access Control in Mobile Software Systems

with authentication may include PIN, password, fingerprint etc.

This study is focused on exploring more informative features like those related to identification and authentication, 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 malapps. We argue that if the goal is to improve the security of Android, its identification, authentication and authorization features needs to be explored and investigated. Majority of state-of-the-art systems have approached privacy problems in mobile apps from *authorization*, i.e. the smartphone privileges aspect of it, and that is why user-granted permissions are often used for analysing mobile systems security. The proposed technique is focused on the unstructured user input mainly from the GUI as the basis for analysis because, at its core, smartphone privileges and GUI inputs form access control elements for mobile software systems.

By privacy analysis of unstructured user input, this study is not concerned with the end-users' input at runtime, but more about using the app resources extracted from the reverse-engineering of the APKs to semantically resolve the user input request from the graphical user interface. Some of the metadata for inferring the unstructured user input request includes the GUI widgets, string resources, layout files etc. This work is concerned with the user input request from the GUI at design-time. While there are other approaches for analysing privacy and security in the Android ecosystem, the proposed approach is motivated by the possibility of complementing state-of-the-art structured static techniques with unstructured static features. Since unstructured static features often elude privacy analysis, this paper demonstrates the feasibility of incorporating structured and unstructured static features for robust privacy analysis in the Android ecosystem.

5. Information Types and their Sensitivity

The first step in our privacy analysis framework is to identify the nature of the information that is being requested via unstructured input sources and its sensitivity. There are three key processes involved. These are the analysis of app metadata to extract semantic information about unstructured inputs; the construction of a taxonomy of sensitive information; and finally the use of machine learning to annotate unstructured inputs that are benchmarked against the taxonomy. This section describes each process.

5.1. App MetaData Analysis

One way to analyse unstructured input data in a mobile app is to understand the semantics of the layout metadata description. These are typically expressed in an extensible markup language (XML). Specifically, the objective of metadata analysis is to determine whether a layout file contains widgets that accept user inputs. This is achieved by a two-stage app extraction process.

First, the app is parsed to identify its constituent folders. For Android, extraction of an apk file will generate a number of folders including `assets`, `res`, `lib` and `certificates`. Relevant to metadata analysis is the `res` folder since it is the parent of `Layout` which contains the front-end XML files, media files, etc., and `value` which contains string resources and other resource identifiers. App resources such as images, strings etc. are externalised so that they can be maintained independently and accessed using generated resource identifiers. Resources in the `Layout` folder defines the architecture for the UI; whereas, a string resource which is contained in `value` provides text strings for the application.

The next stage is to determine if an app resource requests unstructured sensitive user input. One approach could be to identify resource widgets that explicitly defines user inputs in Android, we parse the XML describing the resource and automatically inspect all the child tags against the predefined common user controls. Some of the common user widgets include EditText (standard text entry widget in Android Apps), TextView and its subclasses (AutoComplete, MultiAutoComplete, CheckedTextView), Pickers (Date Picker, Time Picker), Spinner, Switch, CheckBox, RadioButton, ToggleButton, ListView, GridView, ImageView, and Custom widgets (customised controls created by extending an existing widget). Such techniques have been investigated in SUPOR (Huang et al., 2015), UIPicker (Nan et al., 2015), and AppsPlayground (Rastogi et al., 2013).

Identifying resource widgets that explicitly define user input is straightforward since they can easily map to known Android-defined native widgets. But this is not the case for implicit custom widgets which extends native widgets, and developers can freely name their customised widgets without any conventional approach. Such customised widgets potentially contain sensitive user information. This might be the reason why these approaches do not consider customised user widgets. Also, mobile platforms like Android offers options to efficiently reuse completed layout which could potentially contain user input widgets. Embed-

ding a reusable layout is achieved by using the <include/> and <merge/> tags. Furthermore, not all sensitive information can be characterised by text entry widgets - sensitive information through voice, facial recognition, fingerprint, voice note; unstructured user input not provided by user e.g sensitive information obtained by interacting with interconnected apps through linked accounts, sensitive information displayed on non-editable widget resource etc. Another limitation of this approach is that not all app layouts can be analysed for input widgets. Specifically, hybrid layouts use both native and HTML-based user interfaces; such layout interacts with each other using WebView. Furthermore, there are also dynamically generated layouts that are instantiated at runtime depending on the usage context. It becomes unclear the sensitive user inputs that this alternative layout may contain. To identify all UI pages, a purely static approach may miss parts of UIs that are dynamically rendered, whereas a pure dynamic approach(Chen et al., 2018; Su, 2016; Su et al., 2017) may not be able to reach all pages in the app, especially those requiring authentication.

Existing techniques are insufficient in detecting sensitive user input. Our approach is to analyse unstructured user input at the category level as opposed to the widget level. The aim is to analyze the string resources which are a representation of the UI textual description in the apps. We leverage the string resources because these provides text strings for the application. The string resources provide textual semantics for input resources.

The output of the app metadata analysis is an aggregation of the string resources used for the semantic resolution of user inputs. Another rationale for using the string resources is that they already provide explicit property descriptions for input fields. These include the `android.id` that uniquely identifies the resource; `android:hint` to give information about the input required from the user; `android.inputType` to specify the type of characters allowed for the input widget; and `android.text` to give the text description or content of user interface control.

The advantage of our approach of using the string resources for semantic resolution of user inputs is that it provides an accurate textual description of the sensitive user input requests in the app that is not subject to developer inconsistencies. For example, a username edittext resource is likely to have a username label resource. Hence, assuming the properties of the edittext username resource are inappropriately specified, it can still be semantically resolved based on the textual description of the username label. The output of the app metadata analysis are sentences describing sensitive operations pertinent to unstructured user input. As an illustration, we present a sample UI of a ticket booking application as shown in Figure 3. Extracted UI strings include `text labels`: "Billing Address", "Payment Details", "Contact Information", "Card Expiration", "CVC"; `Android widget text hint resource`: "Name on Card", "Street Address 1", "Card Verification Code", "Card Number", "Phone Number", "Street Address 2", "Post Code" etc. Furthermore, the app metadata analysis does not rely on program code analysis, hence, it is not inhibited by anti-analysis techniques such as obfuscated (packed apps).

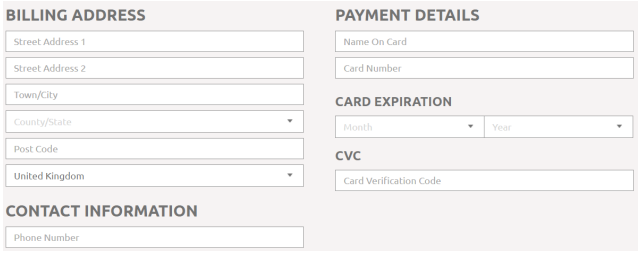

Figure 3: Sample UI screen of a Ticket Booking App

5.2. Taxonomy of Sensitive Information

We utilise a taxonomy of sensitive information to identify the sensitivity of personal data entered into user input widgets. The advantage of a taxonomy is to eliminate the need to depend on users and/or developers to tag personal data with their degrees of sensitivity before disclosure. Generally, a taxonomy organises and classifies information in a hierarchical format. Often this corresponds to categories (aka concepts), attributes and relations between categories (Nickerson et al., 2013; Gregor, 2006). Building a taxonomy for sensitive information, therefore, involves first identifying information types that could be disclosed as categories and their attributes; then organising the categories based on their sensitivity.

We leveraged two main data sources to identify the categories in our taxonomy. These are repositories of vocabularies that define information exchange in socio-technical setting and privacy laws. We first used the core vocabularies from ISA - Interoperability Solutions for European Public Administrations (European-Commission, 2015). The main objective of ISA is to promote the interoperability of software specifications. They achieve this by sharing reusable linked data models that capture the fundamental characteristics of the data entity in a context-neutral fashion. Using ISA, we extracted a subset of categories and their attributes related to person and location core vocabularies. These two vocabularies have been validated in Cwalina et.al.(Cwalina et al., 2013), and further referenced in practice guides for mobile app developers provided by the UK and Australian Information Commissioners (UK ico; OAIC). To validate the effectiveness of resulting categories in a mobile context, we matched identified categories with COPPA - Children's Online Privacy Protection Act (Robins, 2000), PCI DSS - Payment Card Industry Data Security Standard (Morse and Raval, 2008), GDPR - General Data Protection Regulation (Carey, 2018) and HIPAA - Health Insurance Portability and Accountability Act (Annas, 2003) privacy laws. This also enabled us to identify additional categories that were not previously stated in ISA. The resulting categories and their attributes based on this process are as shown in the middle and last column of Figure 4.

The next step is to specify the sensitivity of captured categories in the taxonomy. We first classified categories into

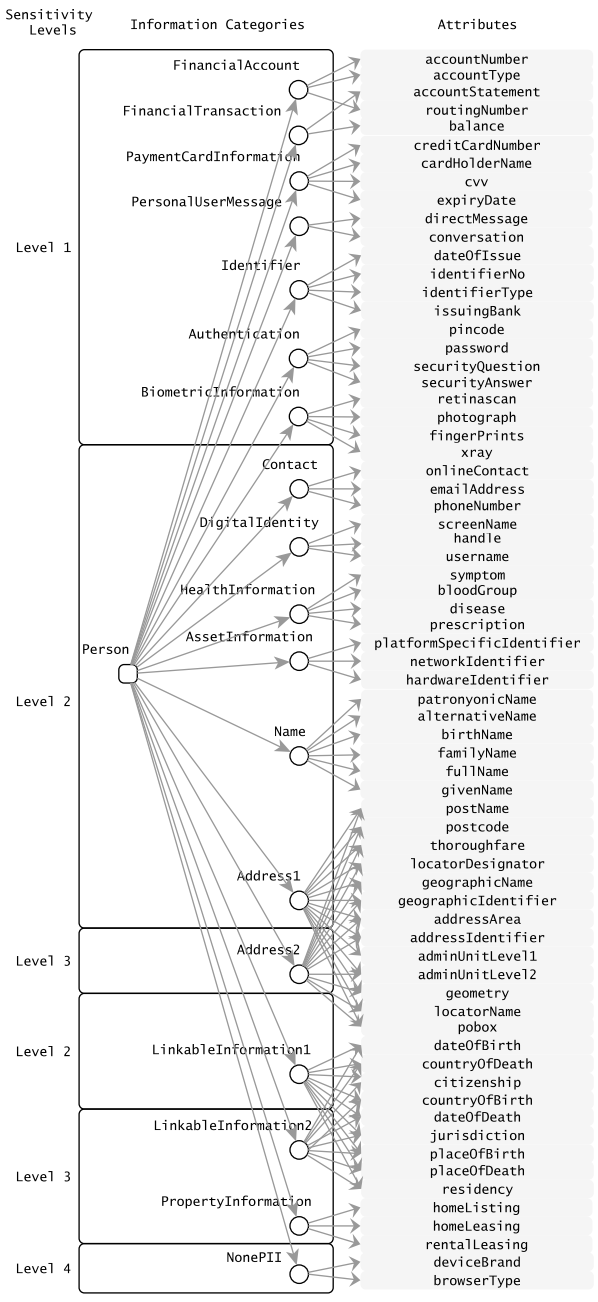

Figure 4: A taxonomy of Sensitive Information

four levels to achieve a balance in the sensitivity spectrum. Having more levels would further reduce the spectrum such that the distance between the most sensitive and the least-sensitive would be too small, making it impossible to distinguish disclosure behaviour with more critical privacy consequences. Having fewer than four levels is also not optimal since it leads to a lesser degree of freedom during sensitivity assignment. For example, a binary spectrum suggests that information is either sensitive or is not, whereas, protected attributes in privacy laws highlight scenarios where two attributes may be identified as sensitive, but with one being

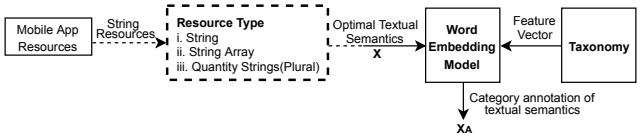

Figure 5: Category annotation of an unstructured string resource

more sensitive than the other. However, using just three categories will inhibit the distinction of user input widgets that collect the whole set of attributes associated with a category from widgets that only collect a subset.

To assign categories to sensitivity levels, we combined information provided by privacy laws about protected categories with their worth in underground economies (Holt and Lampke, 2010; Motoyama et al., 2011; Ablon et al., 2014). By combining these two sources, our sensitivity assignments can reflect the impact on user privacy loss when the information is disclosed inappropriately. The outcome of sensitivity assignments is as shown in Table 1. Overall, the likelihood of identifying an individual if the information is disclosed inappropriately reduces with decreasing levels. More sensitive categories denote that the law requires stricter guidelines for its handling. This may also be an indication of a higher impact of privacy loss, a higher value of associated information, and potentially an increased risk to the individual when inappropriately disclosed.

Finally, we define the App Sensitivity Index (ASI) as a measure of privacy sensitive information type that an app collects via its user input resources. Hence, a characterisation of the sensitivity index should, therefore, be a spectrum from where an app does not require any input in each level of sensitivity to when it requires all input type in the sensitivity levels. The characterisation of the ASI is along 4 levels of sensitivity - `Very Low`, `Moderate` (distinguishing apps that require less-sensitive input); `High and Extreme` (distinguishing between apps that require sensitive information) as shown in Table 2. N_i represents the number of information type requested in Level i, while $G_{max[i]}$ is the maximum number of information type that can be requested in each Level i.

5.3. Annotation of Sensitive User Inputs

Our taxonomy of sensitive information provides a convenient classification system for automated tagging of unstructured data. Specifically, we utilise this taxonomy to identify whether sentences describing sensitive information are contained in the string resources. Our approach is highlighted in Figure 5. For each string resource, we aggregate the value provided in the `name` attribute and combine the string values to derive its optimal textual semantics. The aggregation are a set of sentences for the input resources and the UI textual descriptions. These are then used as input to an unsupervised machine learning algorithm for obtaining a distributed vector representation for words(Bengio et al., 2003), which combined with feature vectors from the taxonomy, annotates the textual semantics with category attributes and sensitivity levels.

Table 1
The Rationale for Categorisation of Sensitivity Levels

Level	Description
1	The category is protected by privacy laws and its attributes can uniquely identify an individual without any additional attribute or category
2	The category is protected by privacy laws but associated individual attributes are unable to uniquely identify an individual except it is combined with one or more attributes in Level 2 or 1.
3	This category is not protected by privacy law. Its attributes can only be used to uniquely identify if combined with one or more attributes in Levels 2 or 1.
4	Non-Personal Identifiable Information (Non-PII)

Table 2
ASI spectrum and Characterization

ASI	Characterization	Interpretation
VERY LOW	$N_1=N_2=N_3=N_4=0$ OR $N_1=N_2=N_3=0;\ N_4=1$	No sensitive inputs
MODERATE	$N_1=N_2=N_4=0; N_3 \geq 1$ OR $N_1=N_2=0,\ N_3,N_4 \geq 1$	Requests Level 3 inputs ONLY.
HIGH	$N_1 <= \frac{G_{max}[1]}{2}$ $N_2 <= \frac{G_{max}[2]}{2}$ $(N_1 \geq 1$ OR $N_2 \geq 1)$	Requests inputs in Level 1 or Level 2 or both
EXTREME	$N_1 > \frac{G_{max}[1]}{2}$ $N_2 > \frac{G_{max}[2]}{2}$ $(N_1 \geq 1$ OR $N_2 \geq 1)$	Requests large sensitive inputs in Level 1 AND/OR Level 2

5.3.1. *Deriving Textual Semantic from String Resources*

The next stage is to determine if an app's resources request unstructured sensitive user input. To identify information types and sensitivity levels the app requests, we parse the XML resource describing the string resources and extract the values provided for each `name` attribute. We propose Natural Language Processing (NLP) techniques for semantic resolution of app string resources to alleviate the shortcomings of keyword-based searching.

Deriving textual semantics involves accepting as input natural language UI textual description and processing the sentences for optimal textual semantics which comprises a set of logically-dependent phrases. The NLP module is made up of two components. The sentence boundary detection is a way of segmenting the sentences of the string resources to find appropriate boundaries specified by sentence breaks. The NLP parser accepts each sentence and annotates every sentence using standard NLP techniques. From an implementation perspective, we chose the Stanford CoreNLP(Manning et al., 2014) to annotate the sentences with enhanced typed dependencies using neural networks(Chen and Manning, 2014). The typed dependencies are a simple description of the grammatical relationships in a sentence and are targeted towards the extraction of textual relationships. We leveraged the python wrapper (Lynten, 2018) for Stanford CoreNLP that provides a simple API for text processing such as Tokenization, Part of Speech Tagging, Dependency Parsing, etc. The pruning module takes as input the logically-dependent pairs and processes them to generate the optimal textual semantics. Next, we use an example to illustrate the annotations added by the NLP parser. Consider the example string value, - **"The username or password you entered is not correct."**, that indirectly refers to unstructured data resource type *Authentication*. Examples of logically-dependent pairs that make up the optimal textual semantics are - "username password, "username entered", "username correct", "password correct".

5.3.2. *Semantic Correlation*

The attributes of categories in our taxonomy can be used in different forms of textual semantics from a string resource. The goal of the semantic correlation is to use the word embedding model to find the similarity between the resulting textual semantics (set of phrase pairs) from the string resources and sensitive attributes in the taxonomy (set of phrase pairs). Word embedding is state-of-the-art in NLP and its model uses pre-defined vector space to present every word. 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. For every token in Phrase 1, it is compared to the tokens in Phrase 2 to find the word similarity by using a corresponding and transverse word pair similarity. The aggregate word pair similarity is the mean of the word pair similarity measures that give the maximum similarity value. The final phrase similarity is the mean of the maximum similarity word pairs.

To find the similarity between two phrases, we use distributional and word embedding models - a state-of-the-art approach for representing text in natural language processing. The most commonly used 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 models. We use *gensim*(Rehurek and Sojka, 2010), a well-known NLP python library, that already implements an interface to deal with these three models. We downloaded pre-trained word vectors learned on different sources trained using these three models. Since there are

Table 3
Similarity Results for Phrase Pairs using Glove, word2ec (W2V) and fastText(fT)

First Word Pair	Second Word Pair	Glove	W2V	fT
credit card	debit card	0.82	0.65	0.81
thumb print	finger print	0.88	0.82	0.84
email address	contact email	0.80	0.66	0.78
buy necklace	purchase jewellery	0.74	0.68	0.72
memory card	phone number	0.62	0.20	0.42
scan barcode	barcode scanner	0.88	0.78	0.85
device location	device location	1.00	1.00	1.00
photo gallery	picture gallery	0.87	0.80	0.90
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
ip address	home address	0.71	0.13	0.74

several pre-trained vectors trained using fastText and GloVe, we evaluated the suitability of each model using a standard dataset which has 65 noun pairs originally measured by Rubinstein and Goodenough(R&G)(Rubenstein and Goodenough, 1965). In the comparisons, the 50-dimensional vector of the Stanford `Glove` is preferable. Furthermore, we compare these models on sample word pairs and evaluate the similarity for the given phrases (cf Table 3).

Working with the 50-dimensional vectors (`glove 6B.50d`) trained on Wikipedia data with 6 billion tokens and a 400,000-word vocabulary, we can find the similarity of the phrases. Given two phrases X and Y originating from the textual semantics of the string resources and taxonomy respectively, where $X = [X_1 \quad X_2]$ and $Y = [Y_1 \quad Y_2]$. X_1 and X_2 are tokens from `X`, while Y_1 and Y_2 are the sequences of words from `Y`. The final similarity which involves finding the aggregated similarity, $Sim(X, Y)$, of the phrases is calculated as shown in Equation 1.

$$\begin{aligned} Sim(X,Y) &= \frac{Sim(X_1,[Y_1,Y_2]) + Sim(X_2,[Y_1,Y_2]}{2} \\ Sim(X_1,[Y_1,Y_2]) &= \max[Sim(X_1,Y_1), Sim(X_1,Y_2)] \\ Sim(X_2,[Y_1,Y_2]) &= \max[Sim(X_2,Y_1), Sim(X_2,Y_2)] \end{aligned} \tag{1}$$

According to Rubinstein 1965(Rubenstein and Goodenough, 1965), the benchmark synonymy value of two words is 0.8025. We set the threshold to 0.80, such that if the semantic similarity equals or exceeds the threshold, the collected information can be mapped to the corresponding information type and sensitivity level in the taxonomy. If $Sim(X, Y) < 0.80$, the textual semantic is mapped to Non-Personal Identifiable Information (Non-PII data).

6. Malware Detection

Unstructured user input is requested within an app's Graphical User Interface (GUI). Malware are basically different from benign apps on interacting with unstructured user input. Malware may perform malicious activities once it gains access to the device to steal user credentials or it may disguise itself as a benign app to request sensitive PII from the users, often choosing popular apps to repackage or infect(Zhou et al., 2012), so that victims will download their rogue version. Malware may attack the system from a system level, like `Ransomware`, or `spyware` that monitors and records information about user's actions on their devices, without their knowledge or permission. Malware comes in different categories based on their mode of operation and may have different characteristics e.g. different distribution of unstructured user input requests. The aim is to detect features that are characteristic of the dataset in defining a class variable as benign or malicious. The focus is to characterize malware behaviour using unstructured user inputs. We are motivated to detect malware with varied characteristics with a set of detection rules extracted from four features with Decision Tree presented. The four features are highlighted below:

6.1. Identifying ASI Patterns

The relevance of the ASI feature is to establish a measure of user input request that is characteristic of the dataset. A strong correlation between the ASI characterization and the class variable establishes this relevance. This involves defining rules with respect to the ASI request patterns.

6.2. Individual Unstructured Input Categories

We also measure the relevance between individual unstructured input category and class label, where each category is a feature. Measuring the relevance of a feature and class variable is known as a feature ranking in machine learning, which has a goal of selecting the most informative features. The aim is to select the most relevant unstructured input category for distinguishing malware from benign apps. Mitigation rules are formed for single unstructured input categories.

6.3. Identifying Sensitive Unstructured Input Set

We identify the sensitive unstructured user input category subsets that are risky either because of their combinations or their interaction with each other. An interesting set should be useful for reporting malware. Therefore, we must define rules with respect to multiple unstructured user input. Multiple unstructured inputs are set of single unstructured inputs.

6.4. ASI and Unstructured Inputs

On their own, neither unstructured user input nor ASI may be enough to characterize malware behaviour. This feature considers the combination of ASI and unstructured input categories value as a single feature for reporting malware. The outcome is the combination of the characterisation of ASI request patterns with single and/or multiple unstructured user input category that is relevant to a class label.

Table 4
Feature Categories For Classification

Features of Behaviour Characterization
ASI
Single Unstructured Input Category
Multiple Unstructured Input Category
ASI and Sensitive Unstructured Category

6.5. Building Detectors of Malware Using Sensitive User Input

In building detectors, we: i) obtain the ASI characterisation that is most relevant to a class label of apps. ii) obtain the top n-categories that are most relevant to a class. iii) identify sets of unstructured inputs which might interact to make an app risky. We can then build a classifier on these features. We build mitigation rules based on the ASI characterisation, single unstructured user input security rules, and multiple sensitive user input detection rules.

6.6. Detection Rules for Explicitly Outlining Malware

Rules represent the knowledge in the form of IF-THEN structure, i.e. IF `condition`, THEN `conclusion`, which is easy for users to interpret. The rules make reasoning explicitly where the condition is a conjunction of feature-value pairs and the conclusion is the class label (1 for Benign, 0 for malicious). We are interested in rules because rules with sensitive user inputs as condition and malware as the class label explicitly define the profile of malware. Our extracted rules are from an empirically quantitative analysis of sensitive user inputs requested by a very large app sets and are thus more comprehensive on a larger scale. Due to its easiness of interpretation, the rules can be further used for reporting malware in a straightforward manner with no requirement on the user's classification expertise. Rules are extracted from a well-founded decision tree classification. The main features that help in malware identification in this study are presented in Table 4.

7. Evaluation

We have presented a framework that detects when sensitive information is disclosed by users via unstructured input resources of mobile apps. The framework takes an app and determines whether it is likely to be benign or malicious. This depends on the usage of unstructured input sources entered by the user. To achieve this, we proposed a technique that relies on reverse-engineering an app to identify its unstructured user input resources. We then presented a taxonomy that can be used to detect if an app requests sensitive information through its input resources. The tagging of the resource pertaining to sensitive information based on the taxonomy was achieved using a word embedding model. Determining whether an app is likely to be benign or malicious then depends on the privacy risk that a tagged resource may pose to the user. We computed this risk by identifying the sensitive information requests for malware detection.

This section reports the results of unstructured user input category ranking, unstructured user input category subsets identified, malware detector performance evaluation, and the extracted explicit decision rules. Our first aim is to ensure the validity of our design choices. Hence, we evaluate whether string resources can infer sensitive information from user inputs. We also validate the typed dependencies and word-embedding approach against key phrase matching on the input widget properties as proposed in SUPOR(Huang et al., 2015), AppsPlayground(Rastogi et al., 2013), and UIPicker(Nan et al., 2015). To evaluate the effectiveness of our approach and using unstructured input usage to detect malware, we have conducted four evaluations. We seek to answer the following research questions:
RQ1: How accurate is our semantic resolution of user inputs using string resources? How does our approach compare to the approach using input widgets?
RQ2: How relevant is the taxonomy for identifying mobile threat with respect to data inputs?
RQ3: How does unstructured sensitive user inputs request contribute to the effectiveness of malware identification?
RQ4: How effective is our approach of using unstructured user input requests in identifying malware? These research questions are important to be considered because they provide insights about unstructured user inputs usage and its significance.

7.1. RQ1: How accurate is our semantic resolution of user inputs using string resources? How does our approach compare to the approach using input widgets?

To validate the accuracy of our learning model, we first select 1400 apps from six app groups on GooglePlay: `Events`, `House&Homes`, `Social`, `Medical`, and `Shopping`. We then carried out a manual inspection of the results generated by the model based on selected apps. This was to determine the effectiveness of our approach in detecting unstructured sensitive input request from the semantic resolution of the app's string resources. Furthermore, we wanted to investigate if the knowledge of UI context may affect our results. For example, a simple warning such as *"Never disclose your password to untrusted third parties."* might be misunderstood as an input request for a password, without knowledge of the context. We also compared the accuracy of our semantic resolution of user inputs using string resources combined with a word embedding model against the combination of input widget property resolution and key phrase matching proposed in SUPOR(Huang et al., 2015), AppsPlayground(Rastogi et al., 2013), and UIPicker(Nan et al., 2015). The manual inspection is the ground standard for making the comparison of the two approaches. The files of the manual inspection are available as part of our efforts towards reproducibility research. The accuracy of each approach hinges on the number of times the vector of the unstructured user input using the manual inspection and that of the approach are equal. The outcome is shown in Table 5.

From Table 5, it can be inferred that our approach gen-

Table 5
Comparison of Semantic Resolution Approaches against Manual Inspection of Apps

App Group	#	TP_1	FP_1	FN_1	TP_2	FP_2	FN_2
Events	59	56	3	0	30	6	23
Business	161	157	3	1	80	24	57
Finance	137	135	2	0	75	35	27
House & Homes	113	109	2	2	60	20	33
Medical	238	228	7	3	100	26	112
Shopping	228	225	2	1	150	33	45
Social	331	325	4	2	100	45	186
Travel & Local	133	133	0	0	50	20	63
$\sum$	1400	1368	23	9	645	209	546

TP = True Positive, FP = False Positive, FN = False Negative. 1 - Semantic Resolution using String Resources, 2 - Semantic Resolution using User Input Widgets

erated an average precision of `98.35%` and recall of `99.34%`. The alternative approach of identifying unstructured input request type using input widget properties resulted in an average precision of `75.53%` and recall of `54.16%`. Observed reasons for the comparative advantage our approach has over the resolution of input widget properties proposed by SUPOR, UIPicker and AppsPlayground are identified below:

1. **Key Phrase Matching** - SUPOR, AppsPlayground and UIPicker each used key-phrase matching for the semantic resolution of input widget properties. This approach suffers some limitations such as the inability to specify a threshold for the semantic resolution. For instance, if a user has an app installed on the phone, which has a text entry widget for entering input, there is no way to categorise the widget if it doesn't contain the expected key-phrase.The proposed approach is built on the idea of measuring the distance to an existing taxonomy, and the approach can generalize to other, including customized taxonomies. Also, the approach is limited by semantic inference. Input widget properties may often describe a sensitive user input without actually referring to the keyword. For example, "Enter your security question and answer". This sentence fragment describes the request for an "Authentication" information; however, the "password" keyword is not used. Using our approach, one of the typed dependencies extracted from the sentence is "security question" and this maps to the following attributes: "secret key", "login security", "security phrase", "security answer", "security code" of the Authentication class with similarity values of 0.71, 0.81, 0.81, 0.81, 0.96, 0.81 respectively. Furthermore, the approaches view UI strings individually. However, strings may mean completely different things, based on the context of the UI in which they are instantiated. For example, "address" may mean email address, physical address, or IP address, based on the screen in which the string is instantiated. With the proposed taxonomy and word embedding model, the strings are correctly classified as contact, address and asset information respectively.

2. **Scope and Type of Input Widgets** - The state-of-the-art approaches either only consider an EditText as the input widget to be analysed or focus solely on Android-defined widgets. Upon inspection, at least 30% of the input widgets containing sensitive resources are developer-defined widgets(custom-widgets). For instance, 94% of sensitive inputs from `com.avuscapital.trading212` are not via the EditText tag. Also, 95% of sensitive inputs request by `com.facebook.katana` are accounted for by custom widgets.

3. **The Unreliability of Input Widget Properties**: From our sampling of Android layouts from the App Store, we also observed that programmers do not always adhere to best practices in initialising these properties. For example, the `android.inputType` property does not necessarily reflect the type of information a widget collects and so a developer could inappropriately specify an input type of `"textVisiblePassword"` for a field that collects a user's full name. We also observed scenarios where `android.inputType` and `android:hint` properties conflicted with each other. For instance, we identified an app with a layout file with the attribute `android.inputType` specifying that the associated resource is an email address field, while the `android:hint` indicates that the resource accepts the full name of a recipient. This shows that developer-defined properties cannot be relied upon alone for the semantic resolution of user inputs.

4. **Non-unique widget identifiers** - Finally, we observed rare cases where an app used the same widget identifier for more than one sensitive input field. The consequence is that the same input resource could be assigned different degrees of sensitivity. A typical example is when an app uses the same text entry widget ID for user messages, via direct messaging service to a friend, and for receiving feedback about the app from the user.

While we have shown the comparative advantage of our approach, we also highlight some threats to the validity of the approach:

1. **Language** - Our dataset contains applications whose descriptions and text resources are not in English. The UI text of some apps in the string resource folder was written in languages other than English, so the classifier could not predict them accurately.

2. **Global concepts** - We observed the lack of regional interoperability in specific apps, such that not all typed dependencies extracted from the string resources are global concepts. For example, words like `USREOU code` (a bank account information used in Russia) could only be operationalised within their respective regions.

Table 6
Most Impersonated Brands for Phishing Attacks

Category	Brands	# Apps
Cloud	Microsoft, Google, Netflix, Docusign, DropBox, Adobe, Google, WeTransfer	26
Financial Services	Paypal, Chase, Wells Fargo, USAA, Bank of America, Credit Agricole, American Express, CIBC, RBC, UBS	11
E-commerce/ Logistics	DHL, Apple, AliBaba, Amazon,Ebay	3
Internet/Telecom	Orange, Comcast, Yahoo!, AT&T, NBC, UK Gov, Fox News	22
Social Media	Facebook, LinkedIn, WhatsApp, Facebook Messenger	4

3. **Hybrid Apps**: Another limitation to our resource metadata analysis is that not all app-layout can be analysed. Specifically, our technique does not consider hybrid apps that use both native and HTML user interfaces.

In conclusion, the high detection results of the proposed approach with manual verification indicate that UI textual descriptions can be counted upon for semantic resolution of unstructured user input request in mobile settings.

7.2. RQ2: How relevant is the taxonomy for identifying mobile threat with respect to data inputs?

To validate our taxonomy, we investigate the usefulness of using the unstructured user input request in detecting attacks based on data input. Wandera's 2019 Mobile threat(Wandera, 2019) landscape report revealed that phishing is still the number one mobile threat. Since the goal of phishing is to make unsuspecting users click a link or run files to launch malicious code to start an attack of stealing data. We investigate the unstructured user input request of the app of such reputable brands for many reasons. First, it provides some ground truth for what constitutes a sensitive unstructured input request. Secondly, this unstructured user request would also give insights into associated risks with respect to information disclosure.

To leverage the ground truth of apps, we download the services in forms of apps offered by these reputable brands that are most impersonated in (Wandera, 2019) and (Hadley, 2019), making a total of 66 apps for evaluation. The unstructured user input request of the top phishing brands is presented in Figure 6, with contact information (email address, phone number etc.) and digital identity (username, user id etc.), Personal User Message (user message e.g. text messages, SMS-related transactions, chats, messages inbox for emails) and Authentication (passwords, login credentials), Address Information (physical address information) being the top requested unstructured user input with 85%, 74%, 68%, 66% and 53% respectively. This result presents a form of validation for the proposed taxonomy of sensitive unstructured user input as the top 5 requested unstructured user input by popular brands often impersonated by phishing have sensitivity Level 1 and Level 2 in our taxonomy - most sensitive (cf Fig 4). Hence, unstructured data input is relevant in identifying a mobile threat. We argue that our findings have contributed to redefining what sensitive user information means. Finally, privacy risk and vulnerability has been measured in terms of permissions and API usage, we argue that our taxonomy of sensitive information presents an additional step of evaluating privacy risk, based on sensitive unstructured user input request and usage. Finally, the approach of annotating privacy-sensitive unstructured inputs in apps could be developed into a tool that helps developers understand the significance of the data they request from users.

7.3. RQ3: How do unstructured sensitive user inputs request contribute to the effectiveness of malware identification?

RQ3 evaluates the effectiveness of unstructured input type requests in identifying malicious behaviour. Investigating the usage of unstructured user input between malware and benign apps opens directions for malware detection research, which seek reliable app features to use in machine-learning processes. To answer RQ3 we investigate the feasibility of building classifiers from the ASI, single unstructured inputs and sets of unstructured input types. This involves selecting attributes that are most relevant to the class label of apps. Rules are constructed using the sensitivity index of the app and the 17 relevant unstructured inputs using the malicious and benign dataset with the decision tree. Each of such IF-THEN rules has a condition as a conjunction of unstructured input values and consequence as a class label, which is 1(Benign) and 0 (malicious) in our case. In a rule, the conjunction of unstructured input sets forms a sufficient condition for detecting malware. To extract the rule sets, we divided our study subjects into training and testing sets as shown in Table 7.

7.3.1. App Sensitivity Index Rule Sets

We extract rule sets from the ASI characterisation of the training set in order to describe some kinds of malicious behaviour. The technique of building decision rules from sensitive unstructured user input is motivated by the study in *Kirin*(Enck et al., 2009) and permission-induced risks(Wang et al., 2014), where decision rules were constructed from risky permission requests to detect malicious behaviour in Android apps. Based on the extracted rule sets, benign apps tend to request more sensitive data for their normal operation (indicated by `Rule 01`, `Rule 02` and `Rule 03`). The extracted rules and its likelihood show that the app sensitivity index has relevance with the class label. However, with a 70% and 60% probability (`Rule 02` and `Rule 03`), there is the need for further abstraction into what type of sensitivity user inputs

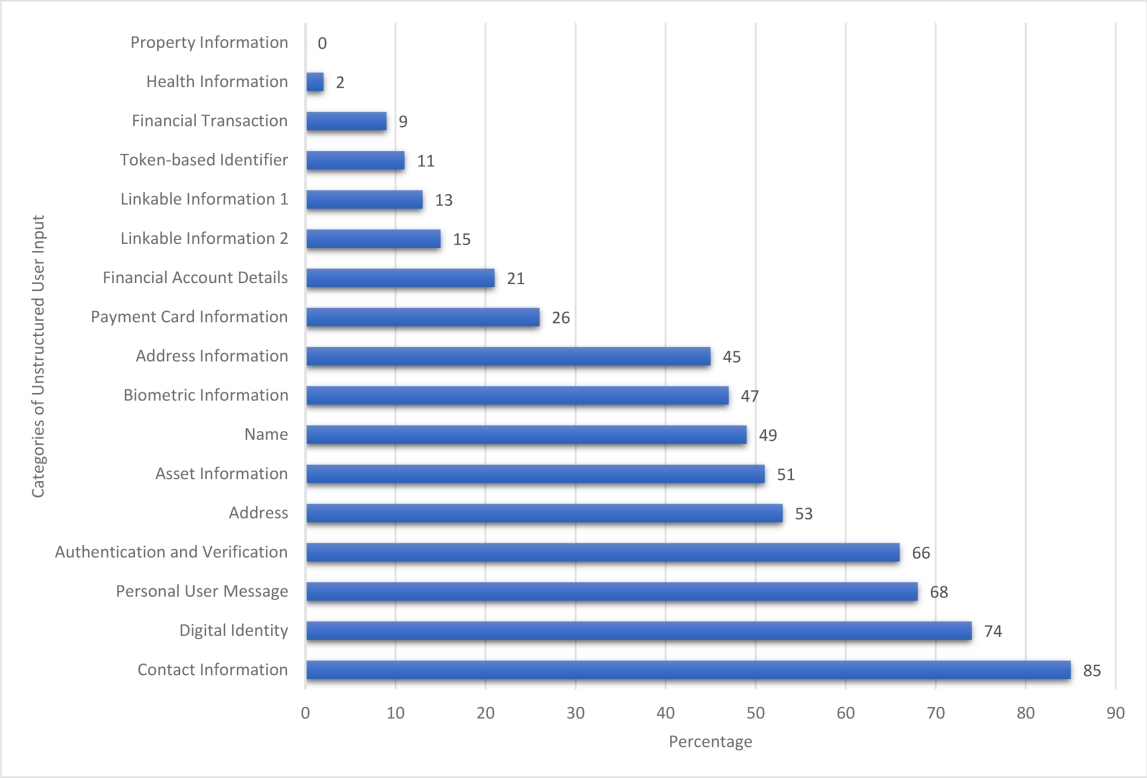

Figure 6: Unstructured User Input Request of Apps of Frequently Impersonated Brands

Table 7
Splitting Study Subjects into Training and Testing Sets

Label	Training Set	Size	%	Testing Set	Size	%
Malware	VirusShare(Roberts, 2011)	13.2K	45	Argus(Wei et al., 2017)	480	50
	Drebin(Arp et al., 2014)	2691		AMG(Jiang and Zhou, 2012)	449	
	Contagio(Parkour, 2018)	245		AndroZoo(Allix et al., 2016)	157K	
	Koodous(Project, 2018)	114				
Benign	GooglePlay	20K	55	AndroZoo(Allix et al., 2016)	156K	50
				GooglePlay	370	
Total		36.2K			315K	

the class label of apps actually request. This is done with a view to increasing the relevance of unstructured data request for distinguishing benign apps from malware based on the presence or absence of sensitive user information.

7.3.2. Unstructured Input Rule Sets

Here, we extract single and multiple unstructured inputs that are relevant to the class label. First, we rank the unstructured input type request for each class label. The rank provides validation for our classification of sensitive inputs such that 5 out of the top 5 relevant input type relevant to benign apps and malware have sensitivity Level 1 and Level 2. These are relevant to extracting the rules because of the fact that the Authentication information (A&V) is higher than every other input type in benign apps suggests that apps collecting sensitive information should provide a form of authentication, however, in the malicious dataset, A&V is not the most requested unstructured input type despite sensitive information requests. As a result of investigating single unstructured user inputs and multiple unstructured inputs that are relevant to the class label, we discovered that Authentication information type is relevant to the benign apps described by `Rule 04`. `Rule 04` can be interpreted as follows - An app

Table 8
Unstructured User Input Ruleset

(a) App Sensitivity Index Rules

Rule 01
IF (ASI = MODERATE)
THEN Class = 1 [85.38%]

Rule 02
IF (ASI = HIGH)
THEN Class = 1 [70%]

Rule 03
IF (ASI = EXTREME)
THEN Class = 1 [60%]

(b) Hybrid Rule Sets.

Rule 04
IF (ASI = HIGH OR EXTREME)
AND AUTHENTICATION = 1
AND CONTACT INFORMATION = 1
OR DIGITAL IDENTITY = 1
THEN Class = 1 [98.55%]

Rule 05
IF (ASI = HIGH OR EXTREME)
AND AUTHENTICATION = 0
AND PERSONAL USER MESSAGE = 1
OR NAME = 1
OR LINKABLE INFORMATION = 1
OR ADDRESS INFORMATION = 1
THEN Class = 1 [99.33%]

with a High or EXTREME ASI that does not require any form of authentication is indicative of malicious behaviour. This means that an app is requesting sensitive unstructured input in Level 1 and Level 2 without any form of user authentication in terms of passwords, security questions, user account authentication etc., while the reverse when an app with High or EXTREME ASI that provides some form of authentication is a benign behaviour. The rule is also supported by the result of investigating the most popular type of unstructured data requested by apps of frequently impersonated brands (cf Fig. 6).

However, we also needed a flip side to it, as the results does show that some malicious apps request authentication information, hence, there needs to be a further abstraction to reduce false negatives. A malicious app with ASI value of HIGH or EXTREME may not request authentication yet requests its dependency alongside it because of their uniqueness such as contact (email addresses, phone numbers), and Digital Identity (for unique online identity). To achieve a further distinguishing factor, we investigate the combinations of information type that often go along with authentication request in benign apps. The information type that is distinguishing in this regard is other personal information such as Address Information, Linkable Information, Name, Asset Information, Financial Transaction and Personal User Message. For instance, the ratio of benign to malicious apps that request address information with authentication information is 54:1. Hence we say that an app that requires authentication, with the rarely requested information type in the malware testing set is likely to be malicious. In summary benign apps that request sensitive unstructured data often requires some form of user authentication and they request a wider range of other unstructured data, whereas a malicious data may request sensitive user data without providing a form of user authentication and often requires a focused set of sensitive data and not wide-range.

Overall, we have shown that characterisation of the sensitivity index and unstructured user input request can be used as static features for categorizing class labels of apps. The app distribution platform benefits from these findings as a measure of risk analysis that can be used to determine the type of checks that should be carried out and the degree of compliance that is required for an app exhibit such request pattern to gain approval that could prove useful during certification. The expectation is that unstructured user input request identifiable with a malware suggests the need for a more rigorous check before certification. App distribution platforms frequently provide users with information such as frequency of downloads, likes, references and other related apps. This does not provide insight into the privacy risk posed by an app. To improve user privacy awareness, the characterisation of the user input request can be provided as a piece of additional app information by the distribution platform, such that users are informed about unstructured user input request in the same way they are aware of the permissions an app request. This additional awareness information can be vital in establishing trust in privacy preservation when determining which app to download. Specifically, it is now common practice to design an app to inform users when it requires structured input sources such as location, address book, microphone, etc. to execute its functional requirements. But this does not include the sensitive information it collects from unstructured input sources. The result is that users are still unsure of the actual privacy risk posed by an app even after being informed of the structured input resources it depends on. Overall, users could then build their privacy expectations and decide to download an app based on not only the app's functionality but also on potential privacy risk as a result of its user input request.

7.4. RQ4: How effective is our approach of using unstructured user inputs request in identifying malware?

The identification of security-sensitive input is an essential building block for investigating the privacy of applications. However, its suitability alone for detecting malware needs to be investigated, as malware have different family and therefore exhibit different behaviour in relying on security-sensitive input to be malicious. To answer RQ4, we measure the performance of the extracted rules in identifying generic malware and known malware family by answering the following questions:

1. RQ4-1: How effective is the approach when the malware family and target application are known?

2. RQ4-2: How effective is the approach when the malware and target application are unknown and random i.e. generic?

7.4.1. RQ4-1: Known Malware Family and Target Applications Analysis

We focus on the benefit of assessing security-related input by demonstrating the capabilities of the proposed method in a narrower application setting. We investigate two malware types that dominated the AMD dataset(Wei et al., 2017) because of the accessibility of their negative counterparts. The malware family types are:

- **Trojan-Banker**: Trojan-Banker programs are generic detection name for trojans that are designed to steal user account data relating to online banking systems, e-payment systems and plastic card systems from customers of these services and send the data to the author or 'master' of the trojan.

- **Trojan-SMS**: These Trojans use the SMS (text) messaging services of a mobile device to send and intercept messages. Programs of this type are used to send text messages from infected mobile devices to premium rate numbers that are hard code into the Trojan's body.

The rationale for choosing the negative samples is to gather benign apps that actually exhibit similar behaviour as the associated family of each malware type. For instance, the most usual target of Android banking Trojans is, unsurprisingly, banking apps. All the associated families of the Trojan-Banker and Trojan-SMS malware dataset are gathered from the AMD Dataset(Wei et al., 2017). The negative samples that consist of legitimate mobile banking apps and messaging apps were downloaded from GooglePlay. As the official app store for Android, Google Play is the main Android app distribution channel. Thus, its sample set could reflect the unstructured input patterns used by mainstream developers.

The results in Table 9 show that our approach can distinguish between trojan banker and legitimate banking apps with a balanced accuracy of 0.995; also differentiate between Trojan-SMS and legitimate SMS text messaging apps with a balanced accuracy of 0.998. The implication of these results is that all the malware variants in the malware families correlate with the decision rules proposed. The behaviour is such that they request sensitive information without authentication information, and when they do, they do not request the other combination of information that often goes with it in the benign counterpart. For example, a Trojan-Banker app may just request payment card information with financial bank details alone without any form of authentication or other linkable information, while an SMS trojan might just request contact information and digital identity which is not the standard request pattern of legitimate SMS apps.

Table 9
Detection Results on Associated Family of Trojan-Banker and Trojan-SMS with Corresponding Benign (Negative) Samples

Malware Type	Negative Samples	Family	#	TP	FP	FN
Trojan-Banker	313 mobile banking apps for Android	Bankbot	93	93	3	0
		Bankun	20	20	3	0
		SlemBunk	5	5	3	0
		SvPeng	4	4	3	0
		Zitmo	7	7	3	0
Trojan-SMS	57 SMS/text messaging apps for Android on GooglePlay	Boxer	7	7	0	0
		Cova	14	14	0	0
		Erop	1	1	0	0
		FakeInst	54	54	0	0
		FakePlayer	2	2	0	0
		Gumen	130	129	0	1
		Leech	7	7	0	0
		Ogel	4	4	0	0
		Opfake	7	7	0	0
		RuMMS	8	8	0	0
		SmsKey	99	99	0	0
		SpyBubble	2	2	0	0
		Stealer	10	10	0	0
		Tesbo	4	4	0	0
		Vidro	2	2	0	0

The conclusion is that the decision rules are a reflection of the behaviour of malicious Trojan SMS and Trojan-Banker with their legitimate target applications. This suggests an interesting correlation between an app's maliciousness and its request of sensitive user input.

7.4.2. RQ4-2: Towards Detecting Generic Malware

So far, we have investigated known malware family and target applications. One could argue that the testing data is not representative of the reality of apps in the Android ecosystem which may lead to the model having a very high false alarm rate in reality. We wanted to investigate how the model would fare with a large subset of benign and malicious apps, which have a wide range of goals and behaviors with their corresponding malicious counterparts. To achieve this, we investigated the effectiveness of our approach on *Androzoo* - a repository of benign and malware samples from 2008 - 2019, to investigate if the presence or absence or sensitive user input fields is enough to perform malware detection. Samples from a fresh dataset containing 300000 apks with 156638 being malware apks and 155850 being benign apks malware were used to test the accuracy of the newly built hybrid classifier on an independent dataset.

By using strictly the rule set (cf Table 8b) for malware classification, our approach identifies malware with a balanced accuracy of 70% and an F-measure value of 80%, which shows that our approach returned substantially more correct identifications than incorrect ones. We could have also used Support Vector Machines (SVM)(Steinwart and Christmann, 2008) and Random Forest(Breiman, 2001) for feature classification as opposed to Decision Tree(Quinlan, 1986) but that would not improve the accuracy of detection. This is because the accuracy was inhibited by benign apps in our dataset that do not request a rich set of sensitive user information. This is a limitation on our features and not the type of classifier used. In this case, relying only on unstructured user input request alone is limited by benign apps that

Table 10
Identification of Generic Malware Using Unstructured input Rule Set

Performance Measures	Generic Malware
F-Score(%)	0.80
Accuracy(%)	0.71
Balanced Accuracy (bACC)(%)	0.70

do not request sensitive unstructured data for generic malware detection. This is similar to the problem faced with permission-based malware detection, a challenge often referred to as "zero-permission" apps - a scenario where an app does not declare any permission for its functionality.

8. Comparative Analysis

This section aims to provide in details the difference between the proposed approach and techniques found in the literature to highlight our contribution. This is to demonstrate the major advantages of the proposed approach and how it can complement state-of-the-art detection systems.

1. **Malware analysis technique for unstructured user input:** Existing research has approached this analysis by identifying privacy violations in data originating from permissions and well-defined API calls made by mobile applications. These are the so-called structured system-centric data sources. In this approach, API calls where sensitive data is involved are well structured and explicitly represented in the semantics of the data returned by the API invocation. However, mobile apps frequently request user data input within their Graphical User Interfaces (GUIs). Such data may be highly unstructured and contain sensitive information that eludes privacy analysis. Table 11 shows the features used in 23 state-of-the-art malware detectors with none of the approach built on unstructured user input features. The proposed technique has provided a complementary approach to analysing privacy and security issues in the Android ecosystem. While malicious application detection is not a new topic, this paper presents a new perspective to view the problem.

2. **Informative features for malware detection:** Modern detection systems e.g.DREBIN(Arp et al., 2014), approaches using an ensemble of classifiers e.g.Wang et al. (2018), or techniques built on the combination of string and structural static features e.g. DroidEnsemble(**?**) are exploring more informative features to better characterize the behaviour of apps. The proposed approach can complement these endeavours by demonstrating a strong association between malicious behaviour and the use of unstructured inputs, in certain categories of malware. In conclusion, the proposed technique has shown that the privacy analysis of unstructured user input is an important feature for characterizing stealthy behaviour of Android apps.

3. **A technique for analysing zero-permission apps:** Furthermore, as shown in Table 11, majority of the state-of-the-art malware analysis tools often require permissions as one of their key features. However, zero-permission apps do not request any permission to be functional, hence, such apps may evade privacy analysis. For example, Table 12 shows the number of zero permission apps in the malware families analysed in the AMD dataset - a dataset that provides an up-to-date picture of the current landscape of Android malware with malware families ranging from 2010 to 2016. In such cases, unstructured user input request can be used for such analysis (e.g. `BankBot` in Table 9) and vice-versa or the approaches can be used in stages depending on the elements in the app.

4. **An alternative approach for monitoring app behaviour:** Secondly, the state-of-the-art approach to privacy analysis of mobile apps majorly revolves around Android program code analysis which often depends on Android reverse engineering tools. Systems that make use of features extracted by these tools are prone to errors because the state-of-the-art Android reverse engineering tools have been shown not to work properly in all cases(Mirzaei et al., 2019). For instance, tools which extract control flow graphs are not perfect, especially when apps adopt advanced anti-analysis techniques. Such advanced code obfuscation techniques in Android may use a combination of transformations(Dalla Preda and Maggi, 2017) that makes malware analysis systems built on code analysis ineffective. Also, static analysis malware detection tools based on API calls can be evaded if malware authors learn what API calls are used as signatures for detection. For example, they could use a combination of different API calls that allow them to achieve the same function. Moreover, the Android framework is constantly changing with the addition and deprecation of API calls with new API releases, which has led to advanced evasion techniques by malware authors. The implication is that program code analysis cannot be extracted sometimes for behaviour analysis due to its limitations. 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.

   We investigated AMD Dataset as the ground truth for investigating the benefit of our approach in known malware family and target applications (cf Table 9). The dataset are labelled based on several behavioural criteria, including the presence of different anti-analysis techniques such as identifier renaming (IR), string encryption (SE), dynamic loading (DL), native payload (NP), evade dynamic analysis (EDA)(Check Device Info (CDI), Encrypt Communication (EC), Check Installed App (CIA)) in the apps of

Table 11
Features Used in the State-of-the-art Malware Analysis Techniques

S/N	Detector	Features
1	Riskranker(Grace et al., 2012a)	platform-level exploits, sensitive behaviours without user interaction, Encrypted native code execution, Unsafe Dalvik code loading
2	Droidmat(Wu et al., 2012)	Manifest (e.g requested permissions, Intent messages passing,etc.) & API calls
3	Droidapiminer(Aafer et al., 2013)	API-Level Behaviour
4	Peiravian and Zhu (2013)	Permission and API Calls
5	Yerima et al. (2013)	API calls, system commands and permissions
6	Drebin(Arp et al., 2014)	Hardware components, Requested permissions, App components, Filtered intents, Restricted API calls, Used permissions, Suspicious API calls, and Network addresses
7	Zhang et al. (2014)	weighted contextual API dependency graph as program semantics
8	Wang et al. (2014)	Requested Permissions
9	Droid-sec(?)	required permission, sensitive API and dynamic behavior.
10	Apposcopy(Feng et al., 2014)	Inter-component callgraph, control and dataflow properties.
11	Afonso(Afonso et al., 2015)	Predefined list of API and system calls
12	Chen et al. (2015)	App's UI structure and method's control flow graph
13	MamaDroid(Mariconti et al., 2016)	API Calls
14	Droiddetector(?)	Required permissions, sensitive APIs, and dynamic behaviors
15	DroidSieve(Suarez-Tangil et al., 2017)	API calls, code structure, permissions and the set of invoked components
16	McLaughlin et al. (2017)	Raw opcode sequence
17	?	Features in Arp et al. (2014), Certification Information, Payload information ,code patterns,Strings - URLs, IP Addresses, File Paths, Numbers
18	?	method opcode, method API, Shared library function opcode, String, permissions, components and environmental information
19	RevealDroid(?)	APIs, native code and reflection
20	DroidEnsemble(?)	Features in Arp et al. (2014) & Structural Features (function call graph)
21	Wang et al. (2018)	Features in Arp et al. (2014), Code-related Information
22	DroidSpan(?)	Longitudinal characterization study of Android app with a focus on their dynamic behaviours
23	Olukoya et al. (2019)	Permissions and UI textual descriptions

Table 12
Malware Families in the AMD Dataset(Wei et al., 2017) with zero permissions

S/N	Malware Family	Apps with Zero Permissions
1	Airpush	21
2	Steek	12
3	Jisut	3
4	DroidKungFu	1
5	BankBot	1

each family of one particular variety. Table 13 shows the selected app families with anti-analysing techniques investigated for RQ2 (cf Section 7.4.1) where the proposed technique achieved an average balanced accuracy of 0.995. To demonstrate that the proposed approach of using unstructured user input request is obfuscation-resilient, we show the obfuscation techniques used for the malware family and its variants used in Table 13, where 85% of the investigated malware used anti-analysis techniques. Furthermore, we demonstrated in our previous work(Olukoya et al., 2019) that extracting features from the UI textual descriptions is obfuscation resilient.

9. Discussion

In this section, we discuss two major questions for the detection and analysis of malware using user input requests. We discuss the feasibility and the main limitations behind the empirical results. While our results show a high balanced accuracy detection performance on the detection of known malware and corresponding target applications (Trojan-Banker vs legitimate mobile banking apps and Trojan-SMS vs benign messaging apps), we carefully investigate in detail the false negatives it produces for generic malware detection.

In particular, a balanced accuracy of 70% was reported in detecting generic malware. The limitation of this approach in detecting generic malware without target applications is benign apps that do not request a rich set of unstructured user input data. By thorough investigation, we are aware that only considering unstructured input requests as features may have difficulties to improve the current detection accuracy for generic malware detection. Therefore, additional information is required to reduce false positives. One way to reduce the false positive rate could be the analysis of known malware families with corresponding benign applications with similar behaviours and functionalities in a narrower application setting, as shown by the high accuracy

Table 13
Anti-analysis Malware Behaviours of Apps From AMD Dataset(Wei et al., 2017) investigated in Section 7.4.1

Malware Type	Family	Anti-Analysis Technique				
		IR	SE	DL	NP	EDA
Trojan-Banker	BankBot	✓	✓	✓		CDI
	Bankun					CDI
	SlemBunk		✓	✓	✓	
	Svpeng					CDI
	Zitmo	✓				
Trojan-SMS	Boxer	✓	✓			
	Cova	✓				
	Erop					
	FakeInst	✓	✓			
	FakePlayer	✓				
	Gumen				✓	DL
	Leech	✓	✓	✓		EC
	Ogel	✓			✓	
	Opfake		✓			
	RuMMS	✓	✓	✓		
	SmsKey	✓				
	SpyBubble					
	Stealer	✓				
	Tesbo	✓	✓			
	Vidro					

of detection of Trojan-SMS and Trojan-Banker in Table 9.

Focusing only on unstructured user inputs miss a lot of information, and it does not seem to be enough when the benign applications are of different behaviours and goals in a way that they are not target applications for malicious applications. An alternative approach could be investigating the system-call runtime behaviour of such apps(Tam et al., 2015; Yan and Yin, 2012; Yang et al., 2013). Another way of handling such complexities could be the use of an enhanced feature set. One of such could be combining unstructured input with permission request and API usage to form a large feature space vector. However, since not all applications request permission also, approaches built on permission detection can use the proposed unstructured user input request for robust analysis. Another area of future work that has a huge potential of improving the performance detection rate of the proposed approach with generic malware is incorporating the structured static features in (Wang et al., 2014; **?**, 2018; **?**) with the unstructured static features in an ensemble for a comprehensive robust privacy analysis in the Android ecosystem.

10. Conclusion

We have presented a framework for analysing unstructured input in mobile apps towards a vision of detecting malicious behaviour. Our framework first combines a taxonomy of sensitive information with a word embedding model. This is to analyse the app's meta-data and identify unstructured input request from users. The outcome is used to realise a sensitivity spectrum that indicates the level of unstructured input request pattern of apps. Meta-data analysis is then followed by a combination of identifying subsets of unstructured input types and rule sets with a decision tree to determine malware with different characteristics. We have evaluated our approach using apps downloaded from the Google Play Store. In a narrow application setting where malware share similar functionalities with benign applications, our approach is 99% accurate, while yielding a 70% with random benign and malware apps. In particular, we show that our approach is very feasible for distinguishing Trojan SMS and Banking Trojan from their legitimate benign targets. The approach of annotating privacy-sensitive unstructured inputs in apps could be developed into a tool that helps developers understand the significance of the data they request from users. The proposed approach demonstrates a strong association between malicious behaviour and the use of unstructured inputs, in certain categories of malware. The study has also shown that analysing unstructured user input may also be effective in scenarios other than malware detection e.g. in the design of anti-phishing mechanisms. The insight that there is a pattern of unstructured user input usage in frequently impersonated brands may be the first step in such endeavours.

We found utilities for our approach, the first is in distinguishing with high accuracy between malware families and target applications. The second is that with generic malware detection, the use of unstructured input request can be incorporated in other detection systems as an initial step, especially when processing a very large set of samples. Thirdly, applications that evade permission-based malware detection approaches because they are "zero-permission" apps can be analysed based on unstructured user input request. Overall, our analysis thus provides a vision regarding the use of user input data for the identification and detection of malware. In future work, we will explore combining user input data with permission request and API usage for enhanced feature set. Another promising area for future work is investigating the information flow of the most requested unstructured user input of apps of frequently impersonated brands. This area of research would be particularly useful in identifying a phishing app from a benign app.

As malicious apps are at an arms race with their benign counterparts, more features need to be explored in order to improve the capacity of detectors for the detection of generic malware. It is beyond doubt that mobile devices will continue to be an attractive target for cybercriminals, we argue that analysing unstructured user input request contributes to taking extra measures for mobile apps privacy and security.

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.

Ablon, L., Libicki, M.C., Golay, A.A., 2014. Markets for cybercrime tools and stolen data: Hackers' bazaar. Rand Corporation.

Acquisti, A., Brandimarte, L., Loewenstein, G., 2015. Privacy and human behavior in the age of information. Science 347, 509–514.

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.

Allix, K., Bissyandé, T.F., Klein, J., Le Traon, Y., 2016. Androzoo: Collecting millions of android apps for the research community, in: Proceedings of the 13th International Conference on Mining Software Repositories, ACM. pp. 468–471.

Andow, B., Acharya, A., Li, D., Enck, W., Singh, K., Xie, T., 2017. Uiref: analysis of sensitive user inputs in android applications, in: Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks, ACM. pp. 23–34.

Annas, G.J., 2003. Hipaa regulations-a new era of medical-record privacy? New England Journal of Medicine 348, 1486–1490.

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.

Arzt, S., Rasthofer, S., Fritz, C., Bodden, E., Bartel, A., Klein, J., Le Traon, Y., Octeau, D., McDaniel, P., 2014. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. Acm Sigplan Notices 49, 259–269.

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.

Avdiienko, V., Kuznetsov, K., Gorla, A., Zeller, A., Arzt, S., Rasthofer, S., Bodden, E., 2015. Mining apps for abnormal usage of sensitive data, in: Proceedings of the 37th International Conference on Software Engineering-Volume 1, IEEE Press. pp. 426–436.

Bengio, Y., Ducharme, R., Vincent, P., Jauvin, C., 2003. A neural probabilistic language model. Journal of machine learning research 3, 1137–1155.

Breiman, L., 2001. Random forests. Machine learning 45, 5–32.

Carey, P., 2018. Data protection: a practical guide to UK and EU law. Oxford University Press, Inc.

Chen, C., Su, T., Meng, G., Xing, Z., Liu, Y., 2018. From ui design image to gui skeleton: a neural machine translator to bootstrap mobile gui implementation, in: Proceedings of the 40th International Conference on Software Engineering, ACM. pp. 665–676.

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., Wang, P., Lee, Y., Wang, X., Zhang, N., Huang, H., Zou, W., Liu, P., 2015. Finding unknown malice in 10 seconds: Mass vetting for new threats at the google-play scale., in: USENIX Security Symposium.

Cwalina, C.G., Raysman, R., Roosa, S.B., 2013. Mobile app privacy: The hidden risks.

Dalla Preda, M., Maggi, F., 2017. Testing android malware detectors against code obfuscation: a systematization of knowledge and unified methodology. Journal of Computer Virology and Hacking Techniques 13, 209–232.

Demetriou, S., Merrill, W., Yang, W., Zhang, A., Gunter, C.A., 2016. Free for all! assessing user data exposure to advertising libraries on android., in: NDSS.

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., Ongtang, M., McDaniel, P., 2009. On lightweight mobile phone application certification, in: Proceedings of the 16th ACM conference on Computer and communications security, ACM. pp. 235–245.

European-Commission, 2015. e-government core vocabularies handbook).

Fahl, S., Harbach, M., Muders, T., Baumgärtner, L., Freisleben, B., Smith, M., 2012. Why eve and mallory love android: An analysis of android ssl (in) security, in: Proceedings of the 2012 ACM conference on Computer and communications security, ACM. pp. 50–61.

Feng, Y., Anand, S., Dillig, I., Aiken, A., 2014. Apposcopy: 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.

Fuchs, A.P., Chaudhuri, A., Foster, J.S., 2009. Scandroid: Automated security certification of android. Technical Report. University of Maryland, Technical Reports of the Computer Science Department.

Gibler, C., Crussell, J., Erickson, J., Chen, H., 2012. Androidleaks: automatically detecting potential privacy leaks in android applications on a large scale, in: International Conference on Trust and Trustworthy Computing, Springer. pp. 291–307.

Gordon, M.I., Kim, D., Perkins, J.H., Gilham, L., Nguyen, N., Rinard, M.C., 2015. Information flow analysis of android applications in droidsafe., in: NDSS, p. 110.

Grace, M., Zhou, Y., Zhang, Q., Zou, S., Jiang, X., 2012a. 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.

Grace, M.C., Zhou, Y., Wang, Z., Jiang, X., 2012b. Systematic detection of capability leaks in stock android smartphones., in: NDSS, p. 19.

Gregor, S., 2006. The nature of theory in information systems. MIS quarterly , 611–642.

Hadley, E., 2019. Phishers' favorites: Microsoft remains #1 driven by new multi-phased attacks, while netflix claims the #2 spot with a big christmas. https://www.vadesecure.com/en/phishers-favorites-q4-2018/. Accessed: 2019-04-10.

HAN, J., YAN, Q., GAO, D., ZHOU, J., DENG, R.H., 2013. Comparing mobile privacy protection through cross-platform applications, in: Proceedings of NDSS 2013: Network and Distributed System Security Symposium, 24-27 February, San Diego, Internet Society.

Holt, T.J., Lampke, E., 2010. Exploring stolen data markets online: products and market forces. Criminal Justice Studies 23, 33–50.

Huang, J., Li, Z., Xiao, X., Wu, Z., Lu, K., Zhang, X., Jiang, G., 2015. {SUPOR}: Precise and scalable sensitive user input detection for android apps, in: 24th {USENIX} Security Symposium ({USENIX} Security 15), pp. 977–992.

Huang, J., Zhang, X., Tan, L., 2016. Detecting sensitive data disclosure via bi-directional text correlation analysis, in: Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, ACM. pp. 169–180.

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.

Jiang, X., Zhou, Y., 2012. Dissecting android malware: Characterization and evolution, in: 2012 IEEE symposium on security and privacy, IEEE. pp. 95–109.

Li, L., Bartel, A., Bissyandé, T.F., Klein, J., Le Traon, Y., Arzt, S., Rasthofer, S., Bodden, E., Octeau, D., McDaniel, P., 2015. Iccta: Detecting inter-component privacy leaks in android apps, in: Proceedings of the 37th International Conference on Software Engineering-Volume 1, IEEE Press. pp. 280–291.

Li, L., Bissyandé, T.F., Octeau, D., Klein, J., 2016. Reflection-aware static analysis of android apps, in: 2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE), IEEE. pp. 756–761.

Lu, K., Li, Z., Kemerlis, V.P., Wu, Z., Lu, L., Zheng, C., Qian, Z., Lee, W., Jiang, G., 2015. Checking more and alerting less: Detecting privacy leakages via enhanced data-flow analysis and peer voting., in: NDSS.

Lu, L., Li, Z., Wu, Z., Lee, W., Jiang, G., 2012. Chex: statically vetting android apps for component hijacking vulnerabilities, in: Proceedings of the 2012 ACM conference on Computer and communications security, ACM. pp. 229–240.

Lynten, G., 2018. Python wrapper for stanford corenlp. URL: https://github.com/Lynten/stanford-corenlp.

Mancini, C., Thomas, K., Rogers, Y., Price, B.A., Jedrzejczyk, L., Bandara, A.K., Joinson, A.N., Nuseibeh, B., 2009. From spaces to places: emerging contexts in mobile privacy, in: Proceedings of the 11th international conference on Ubiquitous computing, ACM. pp. 1–10.

Manning, C., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S., McClosky, D., 2014. The stanford corenlp natural language processing toolkit, in: Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations, pp. 55–60.

Mariconti, E., Onwuzurike, L., Andriotis, P., Cristofaro, E., Ross, G., 2016. Mamadroid: Detecting android malware by building markov chains of behavioral models, in: Em Proceedings of the 24th Network and Dis-

tributed System Security Symposium, NDSS. Internet Society.

McLaughlin, N., Martinez del Rincon, J., Kang, B., Yerima, S., Miller, P., Sezer, S., Safaei, Y., Trickel, E., Zhao, Z., Doupe, A., et al., 2017. Deep android malware detection, in: Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy, ACM. pp. 301–308.

Mirzaei, O., de Fuentes, J., Tapiador, J., Gonzalez-Manzano, L., 2019. Androdet: An adaptive android obfuscation detector. Future Generation Computer Systems 90, 240–261.

Morse, E.A., Raval, V., 2008. Pci dss: Payment card industry data security standards in context. Computer Law & Security Review 24, 540–554.

Motoyama, M., McCoy, D., Levchenko, K., Savage, S., Voelker, G.M., 2011. An analysis of underground forums, in: Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference, ACM. pp. 71–80.

Nan, Y., Yang, M., Yang, Z., Zhou, S., Gu, G., Wang, X., 2015. Uipicker: User-input privacy identification in mobile applications, in: 24th {USENIX} Security Symposium ({USENIX} Security 15), pp. 993–1008.

Nickerson, R.C., Varshney, U., Muntermann, J., 2013. A method for taxonomy development and its application in information systems. European Journal of Information Systems 22, 336–359.

OAIC, 2014. Mobile Privacy: A better practice guide for mobile app developers. Https://www.oaic.gov.au/agencies-and-organisations/guides/guide-for-mobile-app-developers.

Olukoya, O., Mackenzie, L., Omoronyia, I., 2019. Security-oriented view of app behaviour using textual descriptions and user-granted permission requests. Computers & Security , 101685.

Parkour, M., 2018. Contagio mobile. mobile malware mini dump. http://contagiomobile.deependresearch.org/index.html .

Peiravian, N., Zhu, X., 2013. Machine learning for android malware detection using permission and api calls, in: Tools with Artificial Intelligence (ICTAI), 2013 IEEE 25th International Conference on, IEEE. pp. 300–305.

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.

Project, K., 2018. Koodous. https://koodous.com/apks?search=detected:true .

Quinlan, J.R., 1986. Induction of decision trees. Machine learning 1, 81–106.

Rasthofer, S., Arzt, S., Bodden, E., 2014. A machine-learning approach for classifying and categorizing android sources and sinks., in: NDSS.

Rastogi, V., Chen, Y., Enck, W., 2013. Appsplayground: automatic security analysis of smartphone applications, in: Proceedings of the third ACM conference on Data and application security and privacy, ACM. pp. 209–220.

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.(2011). URL https://virusshare. com .

Robins, M.D., 2000. Coping with coppa: Privacy, children, and the internet. COMPUTER LAWYER 17, 17–30.

Rubenstein, H., Goodenough, J.B., 1965. Contextual correlates of synonymy. Communications of the ACM 8, 627–633.

Solove, D.J., 2005. A taxonomy of privacy. U. Pa. L. Rev. 154, 477.

Solove, D.J., 2008. Understanding privacy. volume 173. Harvard university press Cambridge, MA.

Sounthiraraj, D., Sahs, J., Greenwood, G., Lin, Z., Khan, L., 2014. Smvhunter: Large scale, automated detection of ssl/tls man-in-the-middle vulnerabilities in android apps, in: In Proceedings of the 21st Annual Network and Distributed System Security Symposium (NDSS'14, Citeseer.

Steinwart, I., Christmann, A., 2008. Support vector machines. Springer Science & Business Media.

Su, T., 2016. Fsmdroid: guided gui testing of android apps, in: 2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C), IEEE. pp. 689–691.

Su, T., Meng, G., Chen, Y., Wu, K., Yang, W., Yao, Y., Pu, G., Liu, Y., Su, Z., 2017. Guided, stochastic model-based gui testing of android apps, in: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ACM. pp. 245–256.

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.

Sun, M., Wei, T., Lui, J., 2016. Taintart: A practical multi-level information-flow tracking system for android runtime, in: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, ACM. pp. 331–342.

Tam, K., Khan, S.J., Fattori, A., Cavallaro, L., 2015. Copperdroid: Automatic reconstruction of android malware behaviors., in: Ndss.

Thomas, K., Bandara, A.K., Price, B.A., Nuseibeh, B., 2014. Distilling privacy requirements for mobile applications, in: Proceedings of the 36th International Conference on Software Engineering, ACM. pp. 871–882.

UK ico, 2013. Privacy in mobile apps: Guidance for app developers. Https://developer.nhs.uk/library/save-legal-secure/information-governance/uk-ico-privacy-in-mobile-apps-guidance-for-app-developers/.

Wandera, 2019. Understanding the mobile threat landscape in 2019. https://www.wandera.com/mobile-security/mobile-threat-landscape/. Accessed: 2019-04-10.

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.

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, F., Roy, S., Ou, X., et al., 2014. Amandroid: A precise and general inter-component data flow analysis framework for security vetting of android apps, in: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, ACM. pp. 1329–1341.

Wu, D.J., Mao, C.H., Wei, T.E., Lee, H.M., Wu, K.P., 2012. Droidmat: Android malware detection through manifest and api calls tracing, in: 2012 Seventh Asia Joint Conference on Information Security, IEEE. pp. 62–69.

Xu, R., Saïdi, H., Anderson, R.J., 2012. Aurasium: practical policy enforcement for android applications., in: USENIX Security Symposium.

Yan, L.K., Yin, H., 2012. Droidscope: Seamlessly reconstructing the {OS} and dalvik semantic views for dynamic android malware analysis, in: Presented as part of the 21st {USENIX} Security Symposium ({USENIX} Security 12), pp. 569–584.

Yang, Z., Yang, M., 2012. Leakminer: Detect information leakage on android with static taint analysis, in: 2012 Third World Congress on Software Engineering, IEEE. pp. 101–104.

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.

Yerima, S.Y., Sezer, S., McWilliams, G., Muttik, I., 2013. A new android malware detection approach using bayesian classification, in: 2013 IEEE 27th international conference on advanced information networking and applications (AINA), IEEE. pp. 121–128.

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.

Zhou, W., Zhou, Y., Jiang, X., Ning, P., 2012. Detecting repackaged smartphone applications in third-party android marketplaces, in: Proceedings of the second ACM conference on Data and Application Security and Privacy, ACM. pp. 317–326.

Zhou, Y., Singh, K., Jiang, X., 2014. Owner-centric protection of unstructured data on smartphones, in: International Conference on Trust and Trustworthy Computing, Springer. pp. 55–73.

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 multicomputer 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/)