Extracting Information Types from Android Layout Code Using Sequence to Sequence Learning

Mitra Bokaei Hosseini, Xue Qin, Xiaoyin Wang, and Jianwei Niu
University of Texas at San Antonio, San Antonio, TX, USA
{mitra.bokaeihosseini, xue.qin, xiaoyin.wang, jianwei.niu}@utsa.edu

Abstract

Mobile apps offer users with functionalities and services by collecting information in various ways. Android app manifest file and privacy policy are documents that provide users with guidelines about what information type is being collected. However, the information types mentioned in these files are often abstract and do not include fine-grained details about information collected through user input fields in apps. Existing approaches only focus on Android API method calls which can reveal collected information types from a general category of well-defined names. However, these approaches are unable to identify the information types based on direct user input as a major source of private information. These information types contain more sensitive data compared to API retrieved information types. Moreover, developers can design user input fields that refer to any kind of information which can also vary among different apps. To address these problems, we propose to apply natural language processing techniques to Android layout code to extract information types associated with user input fields.

Introduction

Mobile apps are used widely in domains where users' sensitive information is involved. According to a latest report in May 2017, 58.23% of mobile app users had downloaded a health-related mobile app (Krebs and Duncan 2015), which can collect information on body measurements, diet, exercise, and medical treatment, among others. Similarly, in personal finance domain, 73% of Mint app users pay their financial balances using this app[1]. With increased access to personal information and the scale of mobile app deployment, the need for tools to help developers to protect user privacy is increasingly important. Google encourages app developers to provide users with privacy policies that describe how personal information is collected from users (Slavin et al. 2016). These policies are written in natural language and describe the data practices related to information types, such as "location," "friends' contact information," "financial account information," and "photos." Such policies are also meant to fulfill legal requirements to protect privacy, such as the General Data Protection Regulation (GDPR) in Europe, or Federal Trade Commission (FTC) Act in the US. However, innovation and competition among mobile app developers challenges identifying the trace links between privacy polices and app code.

Therefore, there is a need for automatic extraction of information types being collected through app code that can be used to check the consistency between the code and the data practices in privacy policies.

Prior work by Slavin et al. (Slavin et al. 2016) and Zimmeck et al. (Zimmeck et al. 2017) attempt to identify information types collected through API method calls with static analysis. These API method calls concern personal information that are automatically collected from the device, such as sensor data. Wang et. al extract sensitive permissions related to only location and contact information types from apps(Wang et al. 2017). Li et al. Provide a mapping between Android permissions and user interface (UI) components by analyzing the permission-related API method calls and UI events(Li, Guo, and Chen 2016). These works are not focused on addressing personal information that *users provide directly through UI.* Figure 1 shows an example where sensitive information is provided to the app via the interface and is thus disconnected from any API method call. These user-based input fields are difficult to identify as they are both context-sensitive and can vary in implementation from developer to developer, so that they bring a new technical challenge as follows.

TC: Vague and Unbounded Information Types for User Input Data. The information types automatically collected through platform API methods are constrained to Android APIs which are described by comprehensive documents and information collected is well defined. These constraints limit the terminological space to only a few general category names (e.g., location, voice, etc.) In contrast, developers can design novel UIs that ask users to provide potentially *any kind of information*, which includes unstructured and semi-structured personal information in different formats and language types.

To address this challenge, we propose an approach that identifies the information types associated with user input fields automatically. Our approach is based on the assumption on the naturalness of Android XML layout code, so that it is possible to directly apply natural language process-

[1]https://blog.mint.com/credit/mint-by-the-numbers-which-user-are-you-040616/

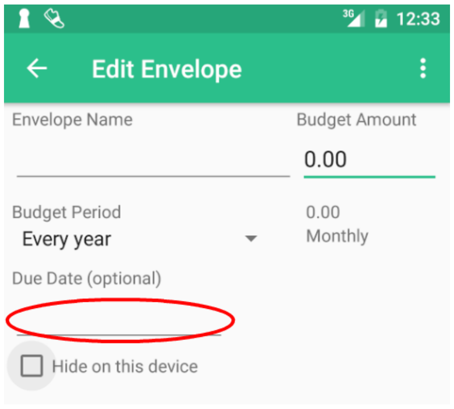

Figure 1: User Interface Screenshots from Good Budget

ing techniques to the layout code and extract the information types. Specifically, given a decompiled Android app, first we extract the static layout files and construct a context sequence for EditText Views by analyzing all the Views preceding each EditText View in the graphic user interface (GUI). Second, we establish a ground truth by asking subjects to identify input fields' information types in the GUI. The context sequence generated for each EditText View is then paired with human interpretations of the user input field. These data are used to train a sequence to sequence Long Short-Term Memory (LSTM) model. Finally, given a context sequence from UI static layout, our trained model is used to identify the information type.

This paper is organized as follows: First, we provide a motivating example on user provided information through input fields; second, we discuss our proposed approach for automatic extraction of information types from static layouts; and finally, we provide our proposed experiment setup.

Motivating Example

In this section, we provide a real example showing how GUI views, especially user input views can be constructed from layout file. In the Android framework, a *layout* defines the visual structure of the GUI, such as locations for views, buttons, windows, and widgets.

```
1  <LinearLayout android:id=
2    "@id/trans_message_header">
3   <TextView android:id=
4     "@id/trans_message_text"/>
5   ...
6   <TextView android:id="@id/name_label"
7     android:text=
8     "@string/edit_envelope_envelope_name"/>
9   <TextView android:id="@id/amount_label"
10    android:text="@string/edit_envelope_budget"/>
11  ...
12  <EEBAAutoCompleteTextView
13    android:id="@id/name"/>
14  <EditText android:id="@id/amount"
15    android:hint="@string/amount_hint"/>
16  ...
17  <TextView android:id="@id/period_label"
18    android:text="@string/envelope_period_label"/>
19  <TextView android:id="@id/helper_text_amount"
20    android:text="0.00" />
21  ...
22  <Spinner android:id="@id/period"
23    android:prompt="@string/period_prompt"/>
24  <TextView android:id="@id/helper_text_period"
25    android:text="@string/period_text_monthly"/>
26 </LinearLayout>
27 <LinearLayout android:id="@id/extra_fields">
28  <TextView android:id="@id/due_date_label"
29     android:text="@string/due_date_label"/>
30  <LinearLayout>
31   <EditText android:id="@id/due_date" android:text=""/>
32  ...
33  <CheckBox android:id="@id/local_hidden"/>
34  <TextView android:text="@string/edit_envelope_hide"/>
35  ...
```

Listing 1: Partial Code from edit_envelope.xml

Layouts. Layouts allow developers to pre-draw the GUIs and reduce the overhead at runtime which can be extracted by decompiling the app's APK files.

Static layout files contain the structure of pre-drawn GUIs, View IDs, and all the text labels we can see from the GUIs. Listing 1 shows the partial code of `edit_envelope.xml`, which is the static layout file of Figure 1.

In listing 1, lines 6-8 refer to a `TextView` element that corresponds to "Envelope Name" field label in Figure 1. This element has two attributes: `android:id` for identification purposes; and `android:text` which contains a reference to `string.xml` file including the actual text user observes on the GUIs. Similarly, line 31 refers to a `EditText` View corresponding to "Due Date" field label in Figure 1. This View also has two attributes: `android:id` and `android:text`.

GUI Context. Just like natural language text, input views can only be well understood with neighboring/ancestor views. For the circled input field in figure 1 which relates to line 31 in listing 1, without considering the context "envelope" only "due date" can be inferred as the information type. If the privacy policy contains the collection of "bill information" or "envelope information", the automatic consistency checkers fail to trace "due date" to "envelope information" without further context information. Therefore, GUI context is essential in understanding user input information types. We propose a learning model on GUI context to infer the proper information type for user input fields.

Proposed Approach

We present the overview of our approach in figure 2 which consists of two main steps: (1) given a mobile app decompiled code, the graphical user interface (GUI) analysis extracts the layout XML code and constructs a *context sequence* for each input field (`EditText`) which includes the id, text, and hint attributes of the `EditText`, and the id, text, and hint attributes of all views preceding the `EditText` in the layout XML file; (2) The sequence to sequence learning component takes an input field context sequence and maps it to a target sequence of words repre-

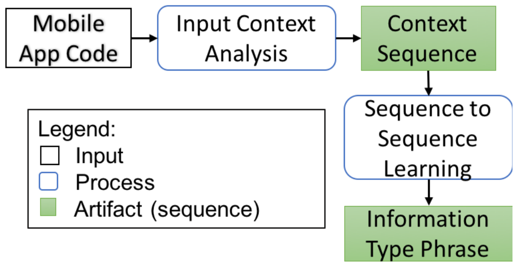

Figure 2: Identifying information type phrases from UI context analysis

senting an information type phrase. The results from these two steps are shown as artifacts in figure 2. The details for each step are presented in the following sub-sections.

Input Context Analysis

In the GUI context analysis phase, we first decompile the app's APK files and extract all XML layout files[2] associated with pre-drawn GUIs in the app. A layout file declares the ViewGroups in the GUI. A View[3] may have multiple attributes, such as id and text. In our study, we only focus on id, text, and hint attributes, since they are typically related to the semantics of a view. Android provides seven types of input controls[4] to help interact with app GUIs, including button, checkbox, text fields, etc. We only focus on *EditText* for user input analysis to identify the related information types. To construct the context sequence for a target *EditText* View, we analyze the XML file and gradually add IDs, text, and hints related to all the Views preceding the target *EditText* View resulting in context sequence. Moreover, strings will be transformed to their corresponding phrases in `string.xml` when adding to context sequence.

The following example shows the context sequence extracted from listing 1 for *EditText* View in line 31 which should be mapped to "envelope due date" through the learning process:

Context Sequence: {trans message header, trans message text, name label, Envelope Name, amount label, Budget Amount, name, amount, 0.00, period label, Budget Period, helper text amount, period, Select a Budget Period, helper text period, Monthly, extra fields, due date label, Due Date, due date}

Next, we describe our model to infer information types using the extracted sequences.

Sequence to Sequence Modeling

Recurrent Neural Networks (RNNs) (Rumelhart et al. 1988) are natural generalization of feedforward neural networks used for processing long sequential data(Rumelhart et al. 1988; Werbos 1990). RNNs connect computational units of the network in a directed cycle such that at each time step *i*, a unit in the RNN takes both the input of the current step (i.e., the $word_i$ in the sequence), and the hidden state of the same unit from the previous time step *i-1* (Guo, Cheng, and Cleland-Huang 2017). However, a standard RNN model can map a source sequence to a target sequence whenever the dimensionality of the source and target is known ahead of time (Sutskever, Vinyals, and Le 2014). To solve this problem, Cho et al. proposed a model that uses two RNNs as encoder and decoder which maps the source sequence to a fixed size vector which is then mapped to a target sequence (Cho et al. 2014). However, RNNs are known for losing long term dependencies in a sequence (Bengio, Simard, and Frasconi 1994) and therefore, LSTM networks were introduced to preserve long term dependencies through a memory cell vector (Hochreiter and Schmidhuber 1997).

Information type phrases are comprised of sequence of words with various lengths that are not known at the time. To identify the information types from the GUI context sequences, we plan to use two LSTMs (Sutskever, Vinyals, and Le 2014) to map a source to a target sequence. First, we encode the input sequence to a vector of fixed dimension that includes the semantics of the input sequence using a multilayered LSTM. Next, we feed the input vector to another LSTM which decodes the target sequence from the vector. The target sequence cannot be identified using a classification model since information types related to GUI input fields are not bound to a finite set of well-defined phrases.

Figure 3 depicts the sequence to sequence learning model. First, we present the source sequence as a vector of word tokens $(x_1, x_2, ..., x_s)$, where x_i corresponds to the *ith* word in the source sequence. Next, each word is mapped to its vector representation through *Word Embedding* layer (Mikolov et al. 2013). We plan to learn the embedding vectors from Wikipedia text. The goal of our model is to estimate the conditional probability $p(y_1, ..., y_t|x_1, ..., x_s)$, where $(y_1, ..., y_t)$ is the target sequence with length t which differs from the source sequence length s. For this reason, the embedded source vectors are sequentially fed into the LSTM units which result in a single vector X representing the semantics of the source sequence. Next, the model computes the probability of $(y_1, ..., y_t)$ with another LSTM network whose initial hidden state is set to X which represents the source sequence semantics:

$$p(y_1, ..., y_t|x_1, ..., x_s) = \prod_{i=1}^{t} p(y_i|X, y_1, ..., y_{i-1})$$

In this equation, $p(y_i|X, y_1, ..., y_{i-1})$ predicts each word in the target sequence using the previous predicted words and the source sequence semantics. This prediction is modeled using a softmax classifier. It is also necessary that both source and target sequences end with a special vector representation <EOS>, which enables the model to define a distribution over sequences of various length (Sutskever, Vinyals, and Le 2014). In the final stage, the predicted $(y_1, ..., y_t)$ is transformed to related word tokens $(w_1, ..., w_t)$ using the *Word Embedding* layer.

Proposed Experiment Setup

The ground truth (GT) for the proposed model contains pairs of GUI context sequences and related information type

[2]https://developer.android.com/guide/topics/ui/declaring-layout.html

[3]https://developer.android.com/guide/topics/ui/overview.html#Layout

[4]https://developer.android.com/guide/topics/ui/controls.html

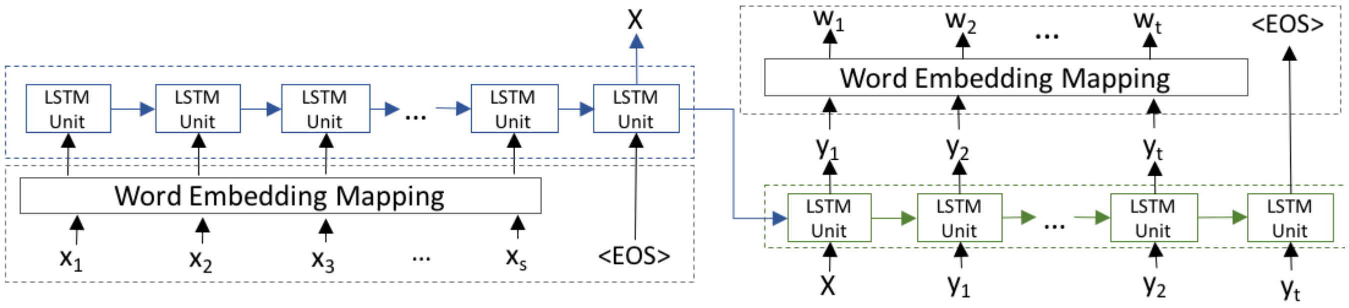

Figure 3: Mapping source sequence to target sequence using two LSTM networks

phrases which can be used to train the learning model. To elicit the information types for GUI input fields, we designed a free listing survey (Bernard 2011), in which crowd workers were asked to identify the information type that describes the input field shown in a red circle in the GUI screenshot (see Figure 1). We surveyed 53 input fields from 19 apps. We recruited 30 participants using Amazon Mechanical Turk. Participants were located in the United States with an overall HIT approval rating greater than 95%. We plan to improve the GT by expanding this study and base our experiment on the expanded GT.

Through the study, we obtained 30 information types per input field. Since there are multiple ways to describe the same concept, we pre-processed the results to more easily comparable elicited types(Porter 1980; 2001). After preprocessing, we combined similar type names for each field and calculate the type name frequency, which is the number of workers who provided each syntactically unique type name per field. Finally, for each field, we select the most frequent type name, which remains linked to a set containing the less frequent type names for that field.

We understand that our current GT does not contain sufficient amount of sequence pairs for training the learning model. We are planning to publish a new study using the method explained earlier in this section to acquire additional training samples.

We also analyzed the 53 input fields and inferred information types by concatenating the file name and input field labels. The results was compared with the most frequent input types provided by crowd workers showing 33.9% match. This suggest that a naive approach with local context is not effective.

References

Bengio, Y.; Simard, P.; and Frasconi, P. 1994. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks* 5(2):157–166.

Bernard, H. R. 2011. *Research methods in anthropology: Qualitative and quantitative approaches*. Rowman Altamira.

Cho, K.; Van Merriënboer, B.; Gulcehre, C.; Bahdanau, D.; Bougares, F.; Schwenk, H.; and Bengio, Y. 2014. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

Guo, J.; Cheng, J.; and Cleland-Huang, J. 2017. Semantically enhanced software traceability using deep learning techniques. In *ICSE*, 3–14. IEEE Press.

Hochreiter, S., and Schmidhuber, J. 1997. Long short-term memory. *Neural computation* 9(8):1735–1780.

Krebs, P., and Duncan, T. D. 2015. Health app use among us mobile phone owners: A national survey. *JMIR mHealth uHealth* 3(4):e101.

Li, Y.; Guo, Y.; and Chen, X. 2016. Peruim: Understanding mobile application privacy with permission-ui mapping. In *ubicomp*, 682–693. ACM.

Mikolov, T.; Chen, K.; Corrado, G.; and Dean, J. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Porter, M. F. 1980. An algorithm for suffix stripping. *Program* 14(3):130–137.

Porter, M. F. 2001. Snowball: A language for stemming algorithms.

Rumelhart, D. E.; Hinton, G. E.; Williams, R. J.; et al. 1988. Learning representations by back-propagating errors. *Cognitive modeling* 5(3):1.

Slavin, R.; Wang, X.; Hosseini, M. B.; Hester, J.; Krishnan, R.; Bhatia, J.; Breaux, T. D.; and Niu, J. 2016. Toward a framework for detecting privacy policy violations in android application code. In *ICSE*, ICSE '16, 25–36. New York, NY, USA: ACM.

Sutskever, I.; Vinyals, O.; and Le, Q. V. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, 3104–3112.

Wang, H.; Li, Y.; Guo, Y.; Agarwal, Y.; and Hong, J. I. 2017. Understanding the purpose of permission use in mobile apps. *ACM Transactions on Information Systems (TOIS)* 35(4):43.

Werbos, P. J. 1990. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE* 78(10):1550–1560.

Zimmeck, S.; Wang, Z.; Zou, L.; Iyengar, R.; Liu, B.; Schaub, F.; Wilson, S.; Sadeh, N.; Bellovin, S. M.; and Reidenberg, J. 2017. Automated analysis of privacy requirements for mobile apps. In *NDSS*.