Neural-Augmented Static Analysis of Android Communication

Jinman Zhao
University of Wisconsin-Madison
jz@cs.wisc.edu

Aws Albarghouthi
University of Wisconsin-Madison
aws@cs.wisc.edu

Vaibhav Rastogi
University of Wisconsin-Madison
vrastogi@cs.wisc.edu

Somesh Jha
University of Wisconsin-Madison
jha@cs.wisc.edu

Damien Octeau
Google
docteau@google.com

Presented by Joshua Learn
Applications on the mobile Android platform have the ability to communicate:
  ○ Ex: use external messaging app to send SMS message from within your app
These communication links can cause huge security vulnerabilities through taking advantage of the user privileges granted to an application.
Problem: detect if communication is possible between two application via static analysis.
Static analysis of large, complex applications is difficult and leads to many reported false positives.

Android App Communication Link Discovery
Inter-Component Communication (ICC)

- Android Apps communicate with a message system called Inter-Component Communication
- ICC Abuse causes many security vulnerabilities
  - Ex: Bus application broadcasting GPS location to all other applications
  - Ex: SMS spying app disguised as tip calculator
- We want to answer the question: *Can component c communicate with component d?*
- Process is called *link inference*
ICC Overview: Intents and Filters

- Intent used to initiate messages
  - Explicit
    - Target component specified
  - Implicit
    - Functionality specified
      - Action string: action to be performed
      - Set of category strings: additional info about what to do with the intent (ex: “BROWSABLE” - app handling action can open request in a web browser)
      - Set of data fields: data to be acted upon

- Filter used to convey willingness to receive intents
  - Actions: set of strings of accepted intent actions
  - Categories: set of strings of accepted intent categories
  - Data descriptors: descriptions of accepted data fields
Link Inference

- IC3 is a tool created for Android ICC analysis
- Uses static analysis to infer values of intents and filters
- Inferred values can be used to detect potential links (PRIMO)
- Three possible results:
  - Definite yes: confirmed link between two apps
  - Definite no: confirmed NO link between two apps
  - Maybe: possibility of link exists
- Complex applications yield a high rate of “maybe”s
- Disambiguating “maybe”s is the goal
Relevant Research: PRIMO

- Octeau et al. published probabilistic models for analysing false positives
- Models are handcrafted
- Model creation is months long
- Required deep domain knowledge
- Specific to current Android programming framework
- Includes matching procedure for detecting links between abstract intents/filters
Example

(a) ICC example with three applications

(b) Intent for sending an sms and associated filter

Figure 2: ICC Example
Vulnerability Example

```java
public void onClick(View v) {
    Location loc = LocationManager.getLastKnownLocation("gps");
    Uri query = Uri.parse("geo:" + loc.getLatitude() + "," + loc.getLongitude() + "?q=restaurants");
    Intent intent = new Intent("VIEW", query);
    startActivity(intent);
}
```

(a) Click handler sending Intent (1) from Figure 3.

```java
public class MapActivity extends Activity {
    public void onCreate(Bundle b) {
        Uri location = getIntent().getData();
        SmsManager.getDefault().sendTextMessage("12345", null, location.toString(), null, null); }
}
```

(b) Code leaking location data in the spy application from Figure 3.
Formalized Intents and Filters

- **Intents**
  - Pair \((act, cats)\) where
    - \(act \in \sum^* \cup \{\text{NULL}\}\)
    - \(cats \in 2^{\sum^*}\)
    - \(act\) is a string or null representing the action
    - \(cats\) is the set of strings representing the categories
      - Given no category, \(cats\) is just the singleton set \{“DEFAULT”\}

- **Filters**
  - Pair \((acts, cats)\) where
    - \(acts \in 2^{\sum^*}\)
    - \(cats \in 2^{\sum^*}\)
    - \(acts\) is the set of strings representing the actions
    - \(cats\) is the set of strings representing the categories
Abstract Intents and Filters

- Static analysis techniques used yield *abstract intents* and *abstract filters*
  - Programmatic creation of intents and filters can lead to many different possibilities at runtime
  - Represent a potentially infinite set of intents/filters through regular expressions

- Abstract versions have same representation structure
  - All strings are regular expressions
  - *Exact:* \(\text{act: } (\text{(.*)SEND}, \{\text{DEFAULT}\})\) is intent where action has suffix “SEND”

- For every intent/filter in an application, there will be an abstract intent that matches it
Abstract Matching Function

- PRIMO paper offers procedure that infers links:

\[ \text{match}^\#: I^\# \times F^\# \rightarrow \{0, 1, \top\} \]

- Takes an abstract intent and filter
- Yields yes, no, or maybe
- Goal: disambiguate the maybes
Link Inference as a Classification Problem

- Classifier function:
  \[ h: I^# \times F^# \rightarrow [0, 1] \]

- Indicates the probability that a link exists \( h(i^#, f^#) = p(y \mid i^#, f^#) \)

- Created using Link Inference Neural Network (LINN)
  - Training data: non-maybe labels gathered from static analysis
    - \( D = \{(i_1^#, f_1^#), y_1\}, \ldots, \{(i_n^#, f_n^#), y_n\}\)
Link-Inference Neural Network

Estimated probability
\( \hat{\rho} \in [0, 1] \)

Classifier

Intent encoding
\( \mathbb{R}^n \)

Intent TDE
Abstract intent

Filter encoding
\( \mathbb{R}^m \)

Filter TDE
Abstract filter
Type-Directed Encoders

- Need some sort of input representation for abstract intents/filters
- Intents/Filters can be seen as compound data types (sets of strings, unions of strings and null, etc.)
- Type-Directed Encoders recursively encode compound data types
- Encoder of type $\tau$ to an $n$ dimensional vector:

$$g: \tau \rightarrow \mathbb{R}^n$$

- Encoding functions are Neural Networks jointly trained with the classifier
Encoding Base Types

- **Real Numbers**
  - already a real number, no encoding needed

- **Categories**
  - Finite number of possible values (characters, booleans, etc.)
  - Encode $k$ categories into $n$-dim vector by lookup table $w \in \mathbb{R}^{n \times k}$
  - Encoding for $j$th category is the $j$th column of $w$
  - Achieved using an embedding layer in the neural net
  - Allows us to choose dimensionality of output vector and capture meaning between categories
Encoding Compound Types

- Lists
  - flat function
    - trained as CNN or LSTM
- Sets
  - aggr function
    - Sum of vectors or Child-sum tree-LSTM
    - No ordering so treated differently than lists
- Products
  - comb function
    - MLP or Tree-LSTM unit
- Sums
  - Chooses which encoder to use based on type
Encoding functions

\[
\lambda x. \text{flat} (\text{map } g \; x) \Rightarrow L(\tau) \quad \lambda x. \text{aggr} (\text{map } g \; x) \Rightarrow S(\tau)
\]

\[
\lambda(x, y). \text{comb}(g_1(x), g_2(y)) \Rightarrow \tau_1 \times \tau_2
\]

\[
\lambda x. \text{if } x \in \tau_1 \text{ then } g_1(x) \text{ else } g_2(x) \Rightarrow \tau_1 + \tau_2
\]
## Encoding functions

<table>
<thead>
<tr>
<th>Encoder</th>
<th>Type</th>
<th>Possible differentiable implementations</th>
</tr>
</thead>
<tbody>
<tr>
<td>enumEnc</td>
<td>$\Sigma \rightarrow \mathbb{R}^l$</td>
<td>Trainable lookup table (embedding layer)</td>
</tr>
<tr>
<td>flat</td>
<td>$L(\mathbb{R}^n) \rightarrow \mathbb{R}^m$</td>
<td>CNN / LSTM</td>
</tr>
<tr>
<td>aggr</td>
<td>$S(\mathbb{R}^n) \rightarrow \mathbb{R}^m$</td>
<td>sum / Child-sum Tree-LSTM unit</td>
</tr>
<tr>
<td>comb</td>
<td>$\mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}^l$</td>
<td>Single-layer MLP / binary Tree-LSTM unit</td>
</tr>
</tbody>
</table>
Tree-LSTM
Intents

\[(L(\Sigma) + \Omega) \times S(L(\Sigma))\]
Intents

\((\mathbb{L}(\Sigma) + \Omega) \times S(\mathbb{L}(\Sigma))\)

Action: “view”
Intents

\((L(\Sigma) + \Omega) \times S(L(\Sigma))\)

Action: "view"
Intents

\((L(\Sigma) + \Omega) \times S(L(\Sigma))\)

Action: \textbf{“view”}
Intents

\[(L(\Sigma) + \Omega) \times S(L(\Sigma))\]

```
Action: "view"
```
Intents

\[(L(\sum) + \Omega) \times S(L(\sum))\]

Categories: \{"BROWSABLE", "OTHER"\}
Intents

\[(L(\Sigma) + \Omega) \times S(L(\Sigma))\]

Categories: \{“BROWSABLE”, “OTHER”\}
(L(∑) + Ω) x S(L(∑))

Categories: {“BROWSABLE”, “OTHER”}
Intents

\[(\mathbb{L}(\Sigma) + \Omega) \times S(\mathbb{L}(\Sigma))\]

Categories: \{“BROWSABLE”, “OTHER”\}
Intents

\[(L(\Sigma) + \Omega) \times S(L(\Sigma))\]
Filters

\[(S(L(\sum)) \times S(L(\sum)))\]
Different Implementations

**Table 2: Instantiations of TDE parameters**

<table>
<thead>
<tr>
<th>Instantiation</th>
<th>Type</th>
<th>enumEnc</th>
<th>flat</th>
<th>TDE parameters</th>
<th>TDE parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>str-RNN</td>
<td>L(Σ)</td>
<td>lookup</td>
<td>RNN</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>str-CNN</td>
<td>L(Σ)</td>
<td>lookup</td>
<td>CNN</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>typed-simple</td>
<td>full</td>
<td>lookup</td>
<td>CNN</td>
<td>sum</td>
<td>1-layer perceptron</td>
</tr>
<tr>
<td>typed-tree</td>
<td>full</td>
<td>lookup</td>
<td>CNN</td>
<td>Tree-LSTM</td>
<td>Tree-LSTM</td>
</tr>
</tbody>
</table>
# Hyperparameters

<table>
<thead>
<tr>
<th>Hyperparameter</th>
<th>Choice</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lookup table</td>
<td>dimension: 16</td>
</tr>
<tr>
<td>CNN</td>
<td>kernel sizes: {1, 3, 5, 7}</td>
</tr>
<tr>
<td></td>
<td>kernel counts: {8, 16, 32, 64}</td>
</tr>
<tr>
<td></td>
<td>activation: relu</td>
</tr>
<tr>
<td></td>
<td>pooling: max</td>
</tr>
<tr>
<td>RNN (LSTM)</td>
<td>hidden size: 128</td>
</tr>
<tr>
<td>1-layer perceptron</td>
<td>dimensions: 64</td>
</tr>
<tr>
<td></td>
<td>activation: relu</td>
</tr>
<tr>
<td>Multilayer perceptron</td>
<td>dimensions: {16, 1}</td>
</tr>
<tr>
<td></td>
<td>activation: {relu, \sigma}</td>
</tr>
</tbody>
</table>
Implementation Details

- Python with Keras (TensorFlow backend)
- Cross Entropy loss function (model outputs a probability)
- RMSprop variation of stochastic gradient descent
- Relu used for all activation functions
- LINN trained on GPU
Experimental Setup

- **PRIMO corpus used for dataset**
  - 10,500 Android Apps from Google Play
- **IC3 + PRIMO abstract matching for static analysis**
  - Provides dataset with must/may link labels
- **Synthetic *may* links used for training and testing the model**
- **Model trained on a sampled subset of links**
  - Using all available data too costly
  - Number of links inferred quadratic to the number of intents/filters
  - Sampling balanced between positive and negative labels
- **Testing done only on *may* links**
Simulating Imprecision

- Ground truth of *may* labels is unknown
- Synthetic *may* labels created by introducing imprecision to *must* links
  - Ex: add “(.*)” to the beginning of a string
  - Technique used by Octeau et al. when creating PRIMO
- First study empirical distribution of imprecision from corpus
  - Add imprecisions guided by the distribution of imprecision observed
Evaluation Metrics Used

- **F1 Score**
  - Measure of predictor’s false-negative and false-positive rates
  - Perfect precision/recall has F1 score of 1

- **ROC Curve**
  - Plot of true positive against true negative rate
  - Perfect model has area under curve of 1

- **Kruskal’s $\gamma$**
  - Correlation between ranking computed by model and ground truth
  - Useful because we want to use model to present results in order of likelihood for programmers to observe
### Results

#### Table 4: Summary of model evaluations

<table>
<thead>
<tr>
<th>Instantiation</th>
<th># Parameters</th>
<th>Inference time (μs/link)</th>
<th>Testing $\gamma$</th>
<th>Testing F1</th>
<th>AUC</th>
<th>Entropy of $\hat{y}$</th>
<th>$Pr(y = 1 \mid \hat{y} &gt; 0.95)$</th>
<th>$Pr(\hat{y} &gt; 0.95)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>str-RNN</td>
<td>154,657</td>
<td>2220</td>
<td>0.970</td>
<td>0.891</td>
<td>0.975</td>
<td>3.002</td>
<td>0.980</td>
<td>0.089</td>
</tr>
<tr>
<td>str-CNN</td>
<td>27,409</td>
<td>57</td>
<td>0.988</td>
<td>0.917</td>
<td>0.988</td>
<td>2.534</td>
<td><strong>0.998</strong></td>
<td>0.139</td>
</tr>
<tr>
<td>typed-simple</td>
<td>142,417</td>
<td>157</td>
<td>0.989</td>
<td>0.920</td>
<td>0.988</td>
<td>2.399</td>
<td>0.996</td>
<td>0.173</td>
</tr>
<tr>
<td>typed-tree</td>
<td>634,881</td>
<td>171</td>
<td><strong>0.992</strong></td>
<td><strong>0.931</strong></td>
<td><strong>0.991</strong></td>
<td><strong>2.220</strong></td>
<td>0.994</td>
<td><strong>0.200</strong></td>
</tr>
</tbody>
</table>

#### Figure 5: Detailed results for the typed-tree instantiation

(a) Receiver operating characteristic (ROC)

(b) Distribution of predicted link probabilities
Observations

- Typed-tree yields the best overall results
- Typed-simple is still slightly better than Str-CNN
- str-CNN has the fastest inference time and best probability of true-positive among highly ranked links
- str-CNN may be preferable but market scale analysis would benefit from slight increases in accuracy
- 10 epochs of training take <20 minutes for all except str-RNN
  - Average computer used
    - Intel i7-6700 (3.4 GHz)
    - 32GB RAM
    - 1TB SSD
    - Nvidia GeForce GTX 970 GPU
- Most complex model has only 5.6MB storage cost
Str-CNN Characteristics

Figure 6: Explaining individual instances
Str-CNN Characteristics

- Tested input strings to see what patterns kernels are picking up
- Important segments seem to be picked up
  - conv1d_size5:14 kernel activated on “.*”
  - conv1d_size5:3 kernel activated on “null”
  - conv1d_size7:0 kernel activated on “VIEW”

Table 5: Some CNN kernels and their top stimuli

<table>
<thead>
<tr>
<th>conv1d_size5:14 segment</th>
<th>activation</th>
<th>conv1d_size5:3 segment</th>
<th>activation</th>
<th>conv1d_size7:0 segment</th>
<th>activation</th>
</tr>
</thead>
<tbody>
<tr>
<td>(.*)R</td>
<td>1.951</td>
<td>null}</td>
<td>3.796</td>
<td>TAVIEWA</td>
<td>3.704</td>
</tr>
<tr>
<td>(.*)u</td>
<td>1.894</td>
<td>null,</td>
<td>2.822</td>
<td>n.VIEW&quot;</td>
<td>3.543</td>
</tr>
<tr>
<td>(.*)t</td>
<td>1.893</td>
<td>sulle</td>
<td>2.488</td>
<td>y.VIEW&quot;</td>
<td>3.384</td>
</tr>
</tbody>
</table>
Typed-Simple Visualization

- t-SNE non-linear dimensionality reduction
  - Similar objects mapped to nearby points
  - Dissimilar objects mapped to distant points
- Six imprecise versions of VIEW captured
  - (.*\*) occurs at different points in the string
  - Imprecision reflected spatially
- DEFAULT, (.*), null categories all in close proximity

Figure 7: Intent encodings visualized using t-SNE
Possible Concerns/Invalidities

● Tested on synthetic *may* links
  ○ Follows empirical distribution of imprecisions
  ○ Might not capture all meaning in real world data

● Neural network setup is complex
  ○ Difficult to know if relevant features are being captured or the NN is getting “lucky”
  ○ Best performing model has many parameters and may be overfitting

● Performance is not significantly better than plain str-CNN
  ○ More time invested may discover a simpler and better way to embed intents/filters
Future Work

- Main novelty of this paper was Type-Directed Encoders
  - Framework for composing neural networks
  - Applies nicely to the problem of link inference in the Android domain
- TDE could be applied to other contexts that exhibit a structure of data composed of subtypes
References

- http://delivery.acm.org/10.1145/2840000/2837661/p469-octeau.pdf?ip=160.39.169.169&id=2837661&acc=CHORUS&key=7777116298C9657D%2ECCCAF7F43E96773E%2E4D4702B0C3E38B35%2E6D218144511F3437&__acm__=1554088993_7d6f68b89b8c94e87c503e466f2cb7a