Detecting and Categorizing Android Malware with Graph Neural Networks

Peng Xu¹, Claudia Eckert¹, Apostolis Zarras²

{Peng,eckert}@sec.in.tum.de

a.zarras@tudelft.nl

¹ Technical University of Munich
² Delft University of Technology
Motivation

The year in figures

In 2020, Kaspersky mobile products and technologies detected:

- 5,683,694 malicious installation packages,
- 156,710 new mobile banking Trojans,
- 20,708 new mobile ransomware Trojans.

Motivation

G DATA Mobile Malware Report 2019: New high for malicious Android apps

Over 10,000 new samples per day
A new malicious app is released every 7.5 seconds

18,792,234 Total number of all malware for Android devices

Peng Xu, Claudia Eckert, Apsotlis Zarras | IT Security, Technical University of Munich
Motivation

1. permission-based Android Malware Detection systems (DREBIN, FM)
2. API-call-based Android Malware Detection systems (DroidNative)

https://developer.android.com/reference/android/Manifest.permission

String, Opcode(word)
Motivation

String(permission), API Call(word)

Obfuscation

String Obfuscation

- Class Encryption
- String Encryption
- Reflection
  - replace each invoke instruction with specific bytecode

Trivial Obfuscation
- Only affects string, not bytecode

Trivial + String Encryption

Trivial + StringEnc + Reflection + ClassEnc
Motivation

https://www.microsoft.com/security/
Motivation

DroidOL: Android malware detection based on online machine learning
Motivation

Android Malware Detection on Graph Structure

Figure 1: Adagio malware detection. Top: Detection system includes six steps. Bottom: 15-Dalvik instruction categories.

Adagio: Structural Detection of Android Malware using Embedded Call-Graph
MANIS: evading malware detection system on graph structure

Peng Xu, Claudia Eckert, Apsotlis Zarras | IT Security, Technical University of Munich
Motivation

Neural Network–based Graph Embedding for Cross–Platform Binary Code Similarity Detection
Motivation

Table 1: Comparison with previous works

<table>
<thead>
<tr>
<th>Approach</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Permission</strong></td>
<td></td>
</tr>
<tr>
<td>Drebin [4]</td>
<td>Permission</td>
</tr>
<tr>
<td>FM [14]</td>
<td>Permission + Factorization Machine</td>
</tr>
<tr>
<td><strong>CFG</strong></td>
<td></td>
</tr>
<tr>
<td>Gemini [8]</td>
<td>CFG + Manually indicated features (frequency of 8 instructions(Bytecode))</td>
</tr>
<tr>
<td>Adagio [9]</td>
<td>CFG + Manually indicated one-hot embedding features (15 categories instructions(Bytecode))</td>
</tr>
<tr>
<td><strong>Byte Sequences</strong></td>
<td></td>
</tr>
<tr>
<td>McLaughlin et. al. [16]</td>
<td>Convolutional + trainable embedding + Bytecode</td>
</tr>
<tr>
<td>Droidnative [1]</td>
<td>N-gram + CFG + native code</td>
</tr>
<tr>
<td><strong>Our Solution</strong></td>
<td></td>
</tr>
<tr>
<td>GE</td>
<td>CFG + N-gram trainable instructions + Bytecode</td>
</tr>
</tbody>
</table>
Overview

APK file → Function call graph → Function call graph with opcode embedding → Function call graph with function embedding → Graph Embedding

Benign → 2-layer MLP Malware detection → MLP → 2-layer MLP Malware classification → Malware

Malware detection: Plankton, FakelInstaller, DroidKungFu, ...

Benign: Plankton

Malware: FakelInstaller, DroidKungFu, ...

Graph Embedding
Function Call Graph

Androguard to get Function call graph(e.g, Adagio, MANIS)
Opcode Embedding

- **Instruction**: Opcode + Operands
- **Why only consider Opcode?**
  - Other works: Address, Register are replaced by specific symbols
  - **Move Instruction**: move-wide vA, vB[04 12x], move-wide/from16 vAA, vBBBB[05 22x]
  - **Invoke Instruction**: invoke-super, invoke-direct, invoke-static, and invoke-interface
- **Word Embedding**
Function Embedding

- **Weighted Mean Function Embedding**

\[
\tilde{f} = \frac{\sum_{i=1}^{n} w_i x_i}{\sum_{i=1}^{n} w_i}
\]

- **SIF-Invoked Function Embedding**
  - SIF: A simple but tough-to-beat baseline for sentence embeddings.

\[
Pr[i \notin f|\tilde{c_f}] = \alpha p(i) + (1 - \alpha) \frac{\exp(<c_f, v_i>)}{Z_{\tilde{c}_f}},
\]

where \(c_f = \beta c_0 + (1 - \beta) c_f\), \(c_0 \perp c_f\), \(\alpha\) and \(\beta\) are scalar hyperparameters, and \(Z_{\tilde{c}_f} = \sum_{i \in f} \exp(<\tilde{c}_f, v_i>)\) the normalizing constant.
MLP Classifier

• Malware Classification:

\[ f(G_h) = \langle \langle g_i, w_{i1} > + b_{i1} \rangle, w_{i2} > + b_{i2} \rangle \]

• Malware Categorization:

\[ f(G_h) = \text{softmax}(\langle \langle g_i, w_{i1} > + b_{i1} \rangle, w_{i2} > + b_{i2} \rangle) \]
Evaluation

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Accuracy (%)</th>
<th>Precision (%)</th>
<th>Recall (%)</th>
<th>F1 (%)</th>
<th>FPR (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ge-SIF</td>
<td>99.86</td>
<td>99.75</td>
<td>99.75</td>
<td>99.42</td>
<td>0.7</td>
</tr>
<tr>
<td>Ge-Mean</td>
<td>99.74</td>
<td>99.92</td>
<td>99.63</td>
<td>99.78</td>
<td>0.4</td>
</tr>
<tr>
<td>Drebin</td>
<td>96.58</td>
<td>95.37</td>
<td>97.85</td>
<td>96.59</td>
<td>2.35</td>
</tr>
<tr>
<td>Droidmat</td>
<td>89.87</td>
<td>90.89</td>
<td>88.28</td>
<td>89.56</td>
<td>4.36</td>
</tr>
<tr>
<td>Adagio</td>
<td>95.0</td>
<td>91.07</td>
<td>100</td>
<td>95.32</td>
<td>5.0</td>
</tr>
</tbody>
</table>

![Graph showing Trc vs False Positive Rate]
Evaluation – Various learning rate

ROC of malware detection

- Learning rate 0.001: 0.998457
- Learning rate 0.01: 0.996290
- Learning rate 0.05: 0.943722
- Learning rate 0.1: 0.734021
Evaluation – Various training Epoch

ROC of malware detection

- Epoch - 5 : 0.997758
- Epoch - 10 : 0.998457
- Epoch - 15 : 0.998383
- Epoch - 20 : 0.998444
Evaluation – Various training Epoch

ROC of malware detection

False Positive Rate

True Positive Rate

- Epoch - 5 : 0.997758
- Epoch - 10 : 0.998457
- Epoch - 15 : 0.998383
- Epoch - 20 : 0.998444
Evaluation – Various n-hop neighbors

ROC of malware detection

True Positive Rate

False Positive Rate

- Epoch : 5, T_iteration : 2 - 0.982967
- Epoch : 5, T_iteration : 3 - 0.975828
- Epoch : 5, T_iteration : 4 - 0.996345
- Epoch : 10, T_iteration : 2 - 0.996290
- Epoch : 10, T_iteration : 3 - 0.996123
- Epoch : 10, T_iteration : 4 - 0.998485
Evaluation – Obfuscated Application

Table 2: Detection rate of obfuscated APK

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PRAGuard</td>
<td>38.0</td>
<td>64.0</td>
<td>96</td>
<td>90.0</td>
<td>50.0</td>
<td>44.0</td>
<td>32.0</td>
</tr>
<tr>
<td>Drebin</td>
<td>99.12</td>
<td>98.99</td>
<td>86.58</td>
<td>98.32</td>
<td>98.99</td>
<td>99.32</td>
<td>96.98</td>
</tr>
<tr>
<td>Our framework</td>
<td>99.33</td>
<td>98.99</td>
<td>86.58</td>
<td>98.32</td>
<td>98.99</td>
<td>99.32</td>
<td>96.98</td>
</tr>
</tbody>
</table>
Evaluation – Obfuscated Application

ROC of malware detection

- classEnc AUC : 0.997738
- striEnc AUC : 0.995267
- trivial AUC : 0.994858
- Ref. AUC : 0.994586
- Trivial + StrEnc : 0.996273
- Trivial + StrEnc + Ref. : 0.995932
- Trivial + StrEnc + Ref. + classEnc : 0.990707
### Table 3: Family classification results

<table>
<thead>
<tr>
<th>Family</th>
<th>Samples</th>
<th>5-epoch</th>
<th>10-epoch</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Accuracy</td>
<td>Precision</td>
<td>Recall</td>
<td>F1</td>
<td>FPR</td>
<td>Accuracy</td>
<td>Precision</td>
<td>Recall</td>
</tr>
<tr>
<td>FakeInstaller</td>
<td>925</td>
<td>99.61</td>
<td>98.78</td>
<td>98.90</td>
<td>99.39</td>
<td>0.57</td>
<td>99.21</td>
<td>98.78</td>
<td>98.78</td>
</tr>
<tr>
<td>DroidKungFu</td>
<td>667</td>
<td>99.60</td>
<td>98.10</td>
<td>98.10</td>
<td>99.04</td>
<td>0.50</td>
<td>99.20</td>
<td>98.10</td>
<td>98.06</td>
</tr>
<tr>
<td>Plankton</td>
<td>624</td>
<td>99.65</td>
<td>92.31</td>
<td>92.31</td>
<td>96.00</td>
<td>0.37</td>
<td>99.29</td>
<td>92.31</td>
<td>92.31</td>
</tr>
<tr>
<td>Opfake</td>
<td>613</td>
<td>99.35</td>
<td>97.21</td>
<td>97.05</td>
<td>98.50</td>
<td>0.82</td>
<td>99.08</td>
<td>97.87</td>
<td>97.86</td>
</tr>
<tr>
<td>GinMaster</td>
<td>339</td>
<td>99.64</td>
<td>95.92</td>
<td>95.92</td>
<td>97.91</td>
<td>0.38</td>
<td>99.29</td>
<td>92.31</td>
<td>95.92</td>
</tr>
<tr>
<td>BseBridge</td>
<td>330</td>
<td>99.61</td>
<td>96.62</td>
<td>96.62</td>
<td>98.28</td>
<td>0.44</td>
<td>99.14</td>
<td>96.31</td>
<td>96.31</td>
</tr>
</tbody>
</table>

**Mean**

<table>
<thead>
<tr>
<th>Family</th>
<th>Samples</th>
<th>5-epoch</th>
<th>10-epoch</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Accuracy</td>
<td>Precision</td>
<td>Recall</td>
<td>F1</td>
<td>FPR</td>
<td>Accuracy</td>
<td>Precision</td>
<td>Recall</td>
</tr>
<tr>
<td>FakeInstaller</td>
<td>925</td>
<td>99.5</td>
<td>98.45</td>
<td>98.45</td>
<td>99.22</td>
<td>0.74</td>
<td>99.0</td>
<td>98.45</td>
<td>97.59</td>
</tr>
<tr>
<td>DroidKungFu</td>
<td>667</td>
<td>99.53</td>
<td>97.76</td>
<td>97.76</td>
<td>98.87</td>
<td>0.59</td>
<td>99.06</td>
<td>97.76</td>
<td>98.21</td>
</tr>
<tr>
<td>Plankton</td>
<td>624</td>
<td>99.64</td>
<td>92.59</td>
<td>92.59</td>
<td>96.15</td>
<td>0.37</td>
<td>99.29</td>
<td>92.59</td>
<td>97.18</td>
</tr>
<tr>
<td>Opfake</td>
<td>613</td>
<td>99.44</td>
<td>97.38</td>
<td>97.38</td>
<td>98.67</td>
<td>0.72</td>
<td>99.22</td>
<td>98.20</td>
<td>97.16</td>
</tr>
<tr>
<td>GinMaster</td>
<td>339</td>
<td>99.50</td>
<td>94.8</td>
<td>94.4</td>
<td>97.32</td>
<td>0.55</td>
<td>98.76</td>
<td>92.8</td>
<td>97.87</td>
</tr>
<tr>
<td>BseBridge</td>
<td>330</td>
<td>99.47</td>
<td>95.07</td>
<td>95.38</td>
<td>97.63</td>
<td>0.6</td>
<td>99.01</td>
<td>95.69</td>
<td>96.85</td>
</tr>
</tbody>
</table>

**SIF**

### Evaluation – Categorization/Family Classification
Question?
Thanks!
Backup
Backup – Structure2vec

\[ \mu_v^{(t+1)} = F(x_v, \sum_{u \in N_v} \mu_u^{(t)}), \forall v \in V. \]

\[ F(x_v, \sum_{u \in N_v} \mu_u^{(t)}) = \tanh(W_1 x_v + \sigma(\sum_{u \in N(v)} \mu_u)) \]

\[ \sigma(l) = P_1 \ast \text{ReLU}(P_2 \ast \ldots \text{ReLU}(P_n l)) \]