Semantics-Aware Android Malware Classification Using Weighted Contextual API Dependency Graphs

Mu Zhang
Yue Duan
Heng Yin
Zhiruo Zhao

Department of Electrical Engineering and Computer Science, Syracuse University
McAfee Threat Report:
Totaled **3.73 million** samples at the end of 2013, a **197% increase** over 2012
McAfee Threat Report:
2.47 million new mobile malware samples were collected in 2013
Motivation: Existing Techniques have Limitations

• Code Pattern-based
  – Riskranker [MobiSys’12], DroidRanger [NDSS’12], Antivirus Software, etc.
  – Rely on code patterns
  – Evaded by transformation attacks (DroidChameleon [TIFS’14, ASIACCS’13])

• Machine Learning-based
  – DroidMiner [ESORICS’14], Drebin [NDSS’14],
    DroidAPI Miner [SecureComm’13], Peng et al. [CCS’12], etc.
  – Rely on application syntax rather than program semantics
  – Susceptible to evasion
DroidSIFT: Semantics-Aware Malware Classification

• **Deployment**
  – Complement to Bouncer
  – Signature detection: new variants
  – Anomaly detection: zero-day

• **Design Goals**
  – Semantic-based Detection
  – High Scalability
  – Variant Resiliency
Related Work: Semantic-based Malware Detection

• **Semantic-based Approaches**
  – Control-flow Graph: M. Christodorescu et al. [Oakland’05]
  – Data Dependency Graph: M. Fredrikson et al. [Oakland’10], C. Kolbitsch et al. [Usenix Security’09]
  – Permission Event Graph: K. Z. Chen et al. [NDSS’13]

• **Limitations**
  – Manually crafted specifications
  – Specifications are produced from known malware
  – To pursue exact matches
Approach Overview

- **DroidSIFT**
  - *Contextual API Dependency Graphs*, automatically and statically extracted “specifications”
  - *Weighted Graph Similarity*, to address malware variants & zero-day malware

![Diagram of Android Apps and Graphs]

**Behavior Graph Generation**  **Matching-based Graph Query**  **Similarity-based Feature Vector Extraction**  **Classification-based Anomaly & Signature Detection**
Weights are assigned to API nodes, giving greater weights to the nodes containing critical calls.
• Context (Entry Point) Discovery

Entry point discovery is to reveal whether the user is aware that a certain API call has been made.
Graph Similarity-based Classification

• **Graph Similarity-based Feature Extraction**
  – Generate behavior graphs for dataset
  – Each unique graph → A feature
  – Example:

<table>
<thead>
<tr>
<th>Index of Graph in DB</th>
<th>G1</th>
<th>G2</th>
<th>G3</th>
<th>G4</th>
<th>G5</th>
<th>G6</th>
<th>G7</th>
<th>G8</th>
<th>...</th>
<th>G861</th>
<th>G862</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.9</td>
<td>0</td>
<td>0</td>
<td>0.9</td>
<td>0.7</td>
<td>...</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Similarity to the Graphs of a given APP
Graph Similarity Score

• **Weighted Graph Similarity (WGS)**

\[
\text{wgs}(G, G', \beta) = 1 - \frac{\text{wged}(G, G', \beta)}{\text{wged}(G, \phi, \beta) + \text{wged}(\phi, G', \beta)}
\]

• **Weighted Graph Edit Distance (WGED)**

\[
\text{wged}(G, G', \beta) = \min \left( \sum_{v_I \in \{V' - V\}} \beta(v_I) + \sum_{v_D \in \{V - V'\}} \beta(v_D) + |E_I| + |E_D| \right)
\]

  – Weight only on vertices
  – Need to enhance Bipartitie algorithm
Weight Assignment

• Selection of Critical API Labels
  – Sensitive to Malware
  – Concept Learning
    • Rarely occur in benign apps
    • Happen more frequently in malware
  – 108 Critical APIs, automatically assigned weights > > 1
  – The rest, assigned a weight of 1
Weight Assignment

Optimization Problem:

Homogeneous Pairs: Malware vs. Malware

Heterogeneous Pairs: Malware vs. Benign

\[
\max f(\{<G,G'>\}, \beta)
\]

Output: Optimal Weight Vector
Graph Database Query

• **Bucket-based Indexing**
  – Bitvector of Critical API Package Names as Index
  – **Exact** match on index
Malware Classification

• Anomaly Detection
  – Binary detector: compare against benign graphs
  – Empirically: all similarity scores <70% = Anomaly

• Signature Detection
  – Multi-label detector: compare against malware graphs
  – Generate feature vectors to train a Naive-Bayes classifier

<table>
<thead>
<tr>
<th></th>
<th>G1</th>
<th>G2</th>
<th>G3</th>
<th>G4</th>
<th>G5</th>
<th>G6</th>
<th>G7</th>
<th>G8</th>
<th>…</th>
<th>G861</th>
<th>G862</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADRD</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.8</td>
<td>0.9</td>
<td>0</td>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DroidDream</td>
<td>0.9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.8</td>
<td>0.7</td>
<td>0</td>
<td>0</td>
<td>…</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DroidKungFu</td>
<td>0</td>
<td>0.7</td>
<td>0</td>
<td>0</td>
<td>0.6</td>
<td>0</td>
<td>0.6</td>
<td>0</td>
<td></td>
<td>0</td>
<td>0.9</td>
</tr>
</tbody>
</table>
• **Dataset**
  
  – 2200 malware instances
    • Android Malware Genome Project, McAfee Labs
  
  – 13500 benign samples
    • Google Play
Evaluation: Runtime Performance

- Most apps (96%) can be processed within 10 minutes.
Evaluation: Classification Results

• **Signature Detection**
  – Database: **862** unique graphs from Android Malware Genome Project
  – **1050** malware samples to train classifier
  – **193** testing samples
  – Correctly label the families of **93%** malware
  – Mislabeled cases:
    • DroidKungFu ←→ DroidDream
    • Zitmo, Zsone, YZHC
Evaluation: Classification Results

- Anomaly Detection
  - Convergence of unique behavioral graphs for benign apps
Evaluation: Classification Results

• **Anomaly Detection**
  – Database: **10420** unique graphs from **11400** benign apps
  – **2200** malware testing sample
    • False negative rate: 2% (Exploits and Downloaders)
  – **2100** benign testing sample
    • False positive rate: 5.15%
  – Detection of new malware (Android.HeHe)
Evaluation: Obfuscated Samples

• Detection of Transformation Attacks (TIFS’14)
  – 21 Malware, 2 Benign
Evaluation: Effectiveness of Weight Generation

Bipartite algorithm produces 73% true positive rate in signature detection and 10% false negative rate in anomaly detection.

Weighted graph similarity metric is more sensitive to program semantics.


Conclusion

• We propose novel *semantic-based* approach that classifies Android malware via dependency *graphs*.

• To fight against malware variants and zero-day malware, we introduce *graph similarity metrics* to uncover homogeneous application behaviors while tolerating minor implementation differences.
Questions?
Evaluation: Measurements of Graphs

- The amount of graphs/nodes is manageable.

(a) Graphs per Benign App.

(b) Graphs per Malware.

(c) Nodes per Benign Graph.

(d) Nodes per Malware Graph.