MANIS: Evading Malware Detection System on Graph Structure

Peng Xu¹, Bojan Kolosnjaji¹, Claudia Eckert¹, Apostolis Zarras²
{peng, kolosnjaji, eckert}@sec.in.tum.de,
apostolis.zarras@maastrichtuniversity.nl

¹ Technical University of Munich
² Maastricht University
Motivation

- **Smartphones** are now a basic life necessity
- **Android** is the world’s dominant mobile operating system
- According to McAfee, the number of discovered **Android malware** has touched **2.5 millions** in 2017, which led the overall mobile malware’s tally to reach **25 millions**
Motivation

- **Smartphones** are now a basic life necessity
- **Android** is the world’s dominant mobile operating system
- According to **McAfee**, the number of discovered **Android malware** has touched **2.5 millions** in 2017, which led the overall mobile malware’s tally to reach **25 millions**
Motivation

- **Smartphones** are now a basic life necessity
- **Android** is the world’s dominant mobile operating system
- According to **McAfee**, the number of discovered **Android malware** has touched **2.5 millions** in 2017, which led the overall mobile malware’s tally to reach **25 millions**
- Existing Android Malware Detection techniques:
  - Signature-based and code matching techniques are obsolete
  - Context-based machine learning approaches are not adequate
Motivation

- **Adversarial Machine learning**
  - Attack the machine learning system

- **Poisoning attack**
  - To the training step

- **Evasion attack**
  - To the testing step: Fast gradient sign method (FGSM), Jacobian Saliency Map Approach (JSMA)

Motivation

- **Adversarial Machine learning**
  - Attack the machine learning system

- **Poisoning attack**
  - To the training step

- **Evasion attack**
  - To the testing step

Motivation

- Adversarial Machine learning
  - Attack the machine learning system

- Poisoning attack
  - To the training step

- Evasion attack
  - To the testing step

Motivation

- Adversarial Machine learning
  - Attack the machine learning system
- Poisoning attack to the training step
- Evasion attack to the testing step

Positive review:
(1) Deep Learning Model

Original sample:
This film has a special place in my heart

Adversarial sample:
This film has a special place in my heart

Negative review:
(2) Deep Learning Model

https://qdata.github.io/secureml-web/4VisualizeBench/
Motivation
Motivation

- Robustness of ML-based malware detection under adversarial noise
Motivation

- Robustness of ML-based malware detection under adversarial noise
- Crafting the adversarial noise
Motivation

- Robustness of ML-based malware detection under adversarial noise
- Crafting the adversarial noise
Android Malware Detection on Graph Structure

- Manifest
  - Permission

- Structural Information
  - Control Flow Graph (CFG)
  - Function Call Graph (FCG)
  - Program Dependence Graph (PDG)
Android Malware Detection on Graph Structure

Figure 1: Adagio malware detection. Top: Detection system includes six steps. Bottom: 15-Dalvik instruction categories.
Android Malware Detection on Graph Structure

Figure 1: Adagio malware detection. Top: Detection system includes six steps. Bottom: 15-Dalvik instruction categories.
Android Malware Detection on Graph Structure

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

\[ h(v) = r(\ell(v)) \oplus \bigoplus_{z \in V_v} \ell(z) \]

http://www.prosec-project.org/docs/2013b-aisec.pdf
Android Malware Detection on Graph Structure

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

5404 -> [001-0101-0001-1100]
Android Malware Detection on Graph Structure

Figure 1: Adagio malware detection. Top: Detection system includes six steps. Botton: 15-Dalvik instruction categories.
Android Malware Detection on Graph Structure

Figure 1: Adagio malware detection. Top: Detection system includes six steps. Botton: 15-Dalvik instruction categories.
Android Malware Detection on Graph Structure

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

\[ h(v) = r(\ell(v)) \bigoplus \left( \bigoplus_{z \in V_v} \ell(z) \right) \]

\[ 5404 \rightarrow [001-0101-0001-1100] \]

http://www.prosec-project.org/docs/2013b-aisec.pdf
Android Malware Detection on Graph Structure

Receiver operating characteristic of Graph-based detector

- Mean ROC (AUC = 0.93 ± 0.02) with histogram extension
- Mean ROC (AUC = 0.95 ± 0.01) with non-histogram extension
Evading Malware Detection on Graph Structure

- **Mathematic Model**
  - Detection: \( Y = X^*W + B = f(G)^*W + B \)
  - **Loss Function:** \( \text{Loss} = 1/N \sum_{i=1}^{N} \text{Loss}(f(G_i), y_i) \)
  - **Adversarial noise:** \( |G_i^* - G_i| < \mathcal{B} \)

\[
\begin{align*}
\max_{G^*} \text{Loss}^* \\
\text{Loss}^* &= 1/N \sum_{i=1}^{N} \text{Loss}(f(G_i^*), y_i) \\
G_i^* &= G_i + \alpha \cdot \xi_{\text{adv}}(G_i),
\end{align*}
\]

- **N-strongest Nodes**
  - Finding the nodes which has the largest influence
  - Injecting those nodes multiple times

- **Gradient-Based Approach**
  - Gradient computation
  - Direction vector
Evading Malware Detection on Graph Structure

- **N-strongest Nodes**
  - Finding the nodes which has the largest influence
  - Injecting those nodes multiple times

- **Gradient-Based Approach**
  - Using gradient’s direction
Evading Malware Detection on Graph Structure

- **N-strongest Nodes**
  - **Initialization**: Prepare the weights and find the node(s) which have the minimum weight value
  - **Injection operation**
    - Boolean Representation of the n-strongest node(s)
    - Injecting the bool representation of n-strongest node(s) at neighborhood hashing step
    - Feature embedding histogram with the injected n-strongest node(s)
    - Histogram and non-histogram extension with the injected n-strongest node(s)
  - **Classifier**
    - $|G_i - G^*| < \beta$ and $Y = -1$
Evading Malware Detection on Graph Structure

- Gradient-based Approach

  - Requirements:
    - R1: The occurrence of graph’s node cannot be expressed less than zero
    - R2: In histogram extension mode, all “1” should align at the beginning of P-dimensional vector
    - R3: Cannot reduce the original occurrence of graph’s nodes in order to keep the functionalities
Evading Malware Detection on Graph Structure

- Gradient-based Approach
  - Requirements
  - Crafting methods
    - Gradient Computation : direction vector
Evading Malware Detection on Graph Structure

- Gradient-based Approach
  - Requirements
  - Crafting methods
    - Gradient Computation: direction vector

\[
\min_{\mathbf{G}_h} f(\mathbf{G}_h) \\
\text{s.t. } d(\mathbf{G}_h, \mathbf{G}'_h) \leq m, \\
f(X) = (X \ast W + B - Y)^2, 
\]
Evading Malware Detection on Graph Structure

- Gradient-based Approach
  - Requirements
  - Crafting methods
    - Gradient Computation: direction vector

\[
\min_{n} f(G_n) \\
\text{s.t. } d(G_n, G'_n) \leq m, \\
f(X) = (X \ast W + B - Y)^2, \\
\nabla f(X)/X
\]
Evading Malware Detection on Graph Structure

- Gradient-based Approach
  - Requirements
  - Crafting methods
    - Gradient Computation: direction vector
    - Gradient vector adjusting

\[ \nabla f(X)/X \]
Evading Malware Detection on Graph Structure

- **Gradient-based Approach**
  - Requirements
  - Crafting methods
    - Gradient Computation: direction vector
    - Gradient vector adjusting
    - Node Projection

\[ \nabla f(X)/X \]

\( f \)

(a) \hspace{1cm} (b) \hspace{1cm} (c) \hspace{1cm} (d)

- Remove unrelated components

- Direction vector

- Direction vector
Evading Malware Detection on Graph Structure

- Gradient-based Approach

  - Requirements

  - Crafting methods
    - Gradient Computation: direction vector \( \nabla f(X)/X \)
    - Gradient vector adjusting
    - Node Projection

\[ f \]

\begin{align*}
& \text{(a)} \quad \text{(b)} \quad \text{(c)} \quad \text{(d)} \\
& \text{Direction vector} \quad \text{Direction vector} \\
& \text{Remove negative components}
\end{align*}
Evading Malware Detection on Graph Structure

- Gradient-based Approach
  - Requirements
  - Crafting methods
    - Gradient Computation: direction vector \( \nabla f(X)/X \)
    - Gradient vector adjusting
    - Node Projection

\[ f \]

(a) \hspace{2cm} (b) \hspace{2cm} (c) \hspace{2cm} (d)
Evading Malware Detection on Graph Structure

. Gradient-based Approach
  . Requirements
  . Crafting methods
    . Gradient Computation : direction vector $\nabla f(X)/X$
    . Gradient vector adjusting
    . Node Projection

\[
\begin{align*}
\text{(a)} & & \text{(b)} & & \text{(c)} & & \text{(d)} \\
\text{Direction vector} & & \text{Direction vector} & & \text{Direction vector} & & \text{Direction vector} \\
\text{Remove unrelated components} & & \text{Remove negative components} & & \text{Components occurrence adjustment} & &
\end{align*}
\]
Evading Malware Detection on Graph Structure

Gradient-based Approach

- Requirements
- Crafting methods
  - Gradient Computation: direction vector
  - Gradient vector adjusting
  - Node Projection

\[
\begin{align*}
\text{R1} & : [1, 2, 1, 1, -1, 0, 0, -1, 2, -4, \ldots, 1, -4, 2, \ldots]_{250} \\
\text{R2} & : [1, 1, 1, 1, 0, 0, 1, 1, 1, \ldots, 1, 1, 1, \ldots]_{250} \\
\end{align*}
\]
Evaluation

- Dataset
  - Benign: 49,947 (AndroZoo + VirusTotal)
  - Malware: 5,560 (Drebin)
  - Five folds

- White-box and gray-box
  - While-box: access all of information
  - Gray-box: can access limited information for the targeting classifier, but can get other classifier’s information, which trained by other folds

- Histogram extension and Non-histogram extension
  - Including the histogram extension or not
Evaluation

- Strongest nodes Number: 1, 2, 3, 4, 5
- White-box and gray-box

### Table 1: N-strongest nodes (non-histogram extension)

<table>
<thead>
<tr>
<th>Strongest nodes</th>
<th>Non-histogram extension (white-box)</th>
<th>Non-histogram extension (gray-box)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Injected nodes ratio ((\bar{x}))</td>
<td>22.7%</td>
<td>23.7%</td>
</tr>
<tr>
<td>Injected nodes ratio ((\sigma))</td>
<td>5.8%</td>
<td>22.3%</td>
</tr>
<tr>
<td>Misclassified Rate ((\bar{x}))</td>
<td>72.2%</td>
<td>26.7%</td>
</tr>
<tr>
<td>Misclassified Rate ((\sigma))</td>
<td>15.4%</td>
<td>28.4%</td>
</tr>
</tbody>
</table>

- Results:
  - Injecting 22.7%(20.69%) one strongest node will get 72.2%(80.79%) misclassification with white-box(gray-box) setting
  - Injecting around 23% and 17% two- and three-strongest nodes will get 26% misclassification with white box, and 21.25%,32.26% injection two- and three nodes with get 49.43% and 35.62% misclassification
  - Misclassification rates depend on the number of n-strongest nodes
Evaluation

- Strongest nodes Number: 1, 2, 3, 4, 5
- White-box and gray-box

**Table 2: N-strongest nodes (histogram extension)**

<table>
<thead>
<tr>
<th>Strongest nodes</th>
<th>histogram extension (white-box)</th>
<th>histogram extension(gray-box)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Injected nodes ratio ($\bar{x}$)</td>
<td>24.30%</td>
<td>37.27%</td>
</tr>
<tr>
<td>Injected nodes ratio ($\sigma$)</td>
<td>8.6%</td>
<td>8.2%</td>
</tr>
<tr>
<td>Misclassified Rate ($\bar{x}$)</td>
<td>6.01%</td>
<td>29.77%</td>
</tr>
<tr>
<td>Misclassified Rate ($\sigma$)</td>
<td>1.89%</td>
<td>22%</td>
</tr>
</tbody>
</table>

- Results:
  - Misclassification rates with histogram extension are significantly lower than non-histogram extension
  - Injecting two strongest nodes with 37.37%(21.33%) will cause around 30%(21.22%) misclassification with white-box(gray-box) setting
  - White-box attacking gets better misclassification than gray-box setting
Evaluation

- Gradient sign method

- Parameter: \( \alpha \quad G_i^* = G_i + \alpha \ast \xi_{adv}(G_i) \)

  *threshold*  limit the number of injected nodes

- White-box and gray-box

**Table 3: Gradient sign method (non-histogram extension)**

<table>
<thead>
<tr>
<th>( \alpha )</th>
<th>0.1</th>
<th>0.2</th>
<th>0.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>threshold</td>
<td>&lt; 0.1</td>
<td>&lt;= 0.1</td>
<td>&lt;= 0.2</td>
</tr>
<tr>
<td></td>
<td>&lt; 0.2</td>
<td>&lt;= 0.2</td>
<td>&lt;= 0.4</td>
</tr>
<tr>
<td></td>
<td>&lt; 0.3</td>
<td>&lt;= 0.3</td>
<td>&lt;= 0.6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>56.5x</th>
<th>17.48%</th>
<th>17.06%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Injected nodes ratio (( \bar{x} ))</td>
<td>51.8x</td>
<td>15.46%</td>
<td>14.82%</td>
</tr>
<tr>
<td>Injected nodes ratio (( \sigma ))</td>
<td>2.74</td>
<td>1%</td>
<td>0.6%</td>
</tr>
<tr>
<td>Misclassified rate (( \bar{x} ))</td>
<td>94.21%</td>
<td>47.46%</td>
<td>43.69%</td>
</tr>
<tr>
<td>Misclassified rate (( \sigma ))</td>
<td>0.6%</td>
<td>3.7%</td>
<td>6.7%</td>
</tr>
<tr>
<td>non-histogram extension (white-box)</td>
<td>46.9x</td>
<td>14.22%</td>
<td>13.89%</td>
</tr>
<tr>
<td>Injected nodes ratio (( \bar{x} ))</td>
<td>59.62x</td>
<td>15.61%</td>
<td>15.35%</td>
</tr>
<tr>
<td>Injected nodes ratio (( \sigma ))</td>
<td>4.71</td>
<td>1.18%</td>
<td>0.78%</td>
</tr>
<tr>
<td>Misclassified rate (( \bar{x} ))</td>
<td>79.76%</td>
<td>37.59%</td>
<td>38.95%</td>
</tr>
<tr>
<td>Misclassified rate (( \sigma ))</td>
<td>3.78%</td>
<td>7.95%</td>
<td>8.29%</td>
</tr>
<tr>
<td>non-histogram extension (gray-box)</td>
<td>48.57x</td>
<td>14.72%</td>
<td>14.7%</td>
</tr>
<tr>
<td>Injected nodes ratio (( \bar{x} ))</td>
<td>3.48</td>
<td>0.99%</td>
<td>0.98%</td>
</tr>
<tr>
<td>Injected nodes ratio (( \sigma ))</td>
<td>96.06%</td>
<td>43.69%</td>
<td>44.18%</td>
</tr>
<tr>
<td>Misclassified rate (( \bar{x} ))</td>
<td>2.08%</td>
<td>8.11%</td>
<td>8.23%</td>
</tr>
<tr>
<td>Misclassified rate (( \sigma ))</td>
<td>1.33%</td>
<td>8.43%</td>
<td>8.07%</td>
</tr>
</tbody>
</table>
**Evaluation**

**Table 3: Gradient sign method (non-histogram extension)**

<table>
<thead>
<tr>
<th>(\alpha) threshold</th>
<th>0.1</th>
<th>0.2</th>
<th>0.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\bar{\alpha})</td>
<td>(&lt; 0.1)</td>
<td>(\leq 0.1)</td>
<td>(\leq 0.2)</td>
</tr>
<tr>
<td>Injected nodes ratio ((\bar{\alpha}))</td>
<td>56.5x</td>
<td>17.48%</td>
<td>17.06%</td>
</tr>
<tr>
<td>Injected nodes ratio ((\sigma))</td>
<td>1.98</td>
<td>1.1%</td>
<td>1.1 %</td>
</tr>
<tr>
<td>Misclassified rate ((\bar{\alpha}))</td>
<td>79.6%</td>
<td>45.3%</td>
<td>39.68%</td>
</tr>
<tr>
<td>Misclassified rate ((\sigma))</td>
<td>5.2%</td>
<td>5.6%</td>
<td>1.9%</td>
</tr>
</tbody>
</table>

| \(\bar{\alpha}\)     | \(< 0.1\) | \(\leq 0.1\) | \(\leq 0.2\) | \(< 0.2\) | \(\leq 0.2\) | \(\leq 0.4\) | \(< 0.3\) | \(\leq 0.3\) | \(\leq 0.6\) |
| Injected nodes ratio (\(\bar{\alpha}\)) | 59.62x | 15.61\% | 15.35\% | 48.57x | 14.72\% | 14.7\% | 50.09x | 15.07\% | 15.26\% |
| Injected nodes ratio (\(\sigma\)) | 4.71 | 1.18\% | 0.78 \% | 3.48 | 0.99\% | 0.98\% | 1.89 | 1.55\% | 1.86\% |
| Misclassified rate (\(\bar{\alpha}\)) | 79.76\% | 37.59\% | 38.95\% | 96.06\% | 43.69\% | 44.18\% | 97.43\% | 44.65\% | 43.05\% |
| Misclassified rate (\(\sigma\)) | 3.78\% | 7.95\% | 8.29\% | 2.08\% | 8.11\% | 8.23\% | 1.33\% | 8.43\% | 8.07\% |

**Results:**

- More than 1x injection ratio should be removed
- With threshold \(\leq 0.1\) and \(\leq 0.2\), we get around 40% misclassification ratio with 17\%~14\% node injection with white-box and gray-box
Evaluation

- **N**: the different number of adjusted occurrence of a node

- **White-box and gray-box**

### Table 4: Gradient sign method (histogram extension)

<table>
<thead>
<tr>
<th></th>
<th>histogram extension (white-box)</th>
<th>histogram extension(gray-box)</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Injected nodes ratio ($\bar{x}$)</td>
<td>10.47%</td>
<td>22.05%</td>
</tr>
<tr>
<td>Injected nodes ratio ($\sigma$)</td>
<td>0.04%</td>
<td>2.23%</td>
</tr>
<tr>
<td>Misclassified rate ($\bar{x}$)</td>
<td>21.12%</td>
<td>20.56%</td>
</tr>
<tr>
<td>Misclassified rate ($\sigma$)</td>
<td>5.8%</td>
<td>6.52%</td>
</tr>
</tbody>
</table>

### Results:

- Misclassification rates with histogram extension are significantly lower than non-histogram extension

- Misclassification rates are around 20% with 10%~30% node injection with adjusted 1,2,3 nodes with white-box setting

- Misclassification rates are 18.8% with different injected number under gray-box setting

- White-box attacking gets better misclassification than gray-box setting
Evaluation

- Randomly selected nodes injections
- White-box and gray-box

**Table 5: Randomization method.**

<table>
<thead>
<tr>
<th>Injected nodes ratio (x)</th>
<th>non-histogram extension (white-box)</th>
<th>histogram extension(gray-box)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10%</td>
<td>20%</td>
</tr>
<tr>
<td>Misclassified rate (x)</td>
<td>2.74%</td>
<td>1.92%</td>
</tr>
<tr>
<td>Misclassified rate (σ)</td>
<td>0.93%</td>
<td>0.85%</td>
</tr>
</tbody>
</table>

**Results:**

- Misclassification rates randomly selected node injected are very low, from 1.9%~3.8% with white-box setting, and from 1.2%~1.87% with gray-box setting
- The injected nodes ratios, from 10%~50%, do not affect misclassification rate significantly
Evaluation

Adversarial samples crafting

- nsn_non_histogram
- nsn_histogram
- gsm_non_histogram
- gsm_histogram
- random_non_histogram
- random_histogram

Misclassified rate vs Injected node ratio
Conclusion

- **Android Malware detection on graph structure**
- **Adversarial example crafting for Android malware detection**
  - N-Strongest nodes
  - Gradient-sign method
- **Limitation**
  - Only evaluate for the call function graph
  - Graph kernel-hashing embedding