Deep Program Reidentification: A Graph Neural Network Solution

Shen Wang et al.

University of Illinois at Chicago, NEC Labs America

To appear in SIAM International Conference on Data Mining (SDM’19)

Presenter: Weilin Xu

https://qdata.github.io/deep2Read
Outline

1 Introduction
   • Problem
   • Proposed Solution

2 Method
   • Program $\Rightarrow$ Graph
   • Node Feature Extraction
   • Graph Embedding
   • Channel-Aware Attention
   • Binary Classification

3 Experiments

4 Conclusion
Outline

1 Introduction
   • Problem
   • Proposed Solution

2 Method
   • Program $\Rightarrow$ Graph
   • Node Feature Extraction
   • Graph Embedding
   • Channel-Aware Attention
   • Binary Classification

3 Experiments

4 Conclusion
Program Reidentification

- Determine if an unknown program is variant of a known program.
- Used to detect disguised malware or ransomware.
Digital Code Signing is Useful

**Figure:** Program Properties

**Figure:** Digital Signature

Presenter: Weilin Xu

https://qdata.github.io/deep2Read
Digital Code Signing is Useful, but

- Not always used, especially by open source software. (False Positives)
- Malware can hijack a signed program. (False Negatives)
Weakness of previous techniques

- Digital code signing
  - Not always used.
- Anti-virus
  - Malware-free attack, evasive malware, etc.
- Sophisticated program watermarking techniques
  - Prohibitive computational costs.
Outline

1. Introduction
   - Problem
   - Proposed Solution

2. Method
   - Program $\Rightarrow$ Graph
   - Node Feature Extraction
   - Graph Embedding
   - Channel-Aware Attention
   - Binary Classification

3. Experiments

4. Conclusion
Proposed Solution

- Program $\Rightarrow$ Graph
- Graph $\Rightarrow$ Embedding.
- Embedding $\Rightarrow$ Identity Classification.
Outline

1 Introduction
   - Problem
   - Proposed Solution

2 Method
   - Program $\Rightarrow$ Graph
   - Node Feature Extraction
   - Graph Embedding
   - Channel-Aware Attention
   - Binary Classification

3 Experiments

4 Conclusion
Extract Graph from a Program

Possible choices:

- Static analysis
  - E.g. Call graph of code blocks.
- Dynamic analysis
  - E.g. System interaction graph.
Extract Graph from a Program

Possible choices:

- **Static analysis**
  E.g. Call graph of code blocks. *Complicated, local.*

- **Dynamic analysis**
  E.g. System interaction graph. *Simpler, global (this paper)*
Extract Graphs from Dynamic Behavior

Surveillance Data Collection → Behavior Graph Modeling → Multi-Channel Transformation

- $G_{Behavior}$
- $M_{P-P}$
- $M_{P-F}$
- $M_{P-I}$
- $G_{P-P}$
- $G_{P-F}$
- $G_{P-I}$
- $\hat{G}$

Shen Wang et al. (UIC, NEC Labs)

Presenter: Weilin Xu

https://qdata.github.io/deep2Read
Heterogeneous Graph

Three types of nodes:
- Fork another **program**.
- Read/Write a **file**.
- Access to a network **socket** $< \text{IPAddr} : \text{Port} >$.

**Solution**: separate into three homogeneous graphs (meta-path).
- Program - Program.
- Program - File.
- Program - Socket.
**Figure:** Attentional Multi-Channel Graph Neural Network.
Outline

1 Introduction
   • Problem
   • Proposed Solution

2 Method
   • Program $\Rightarrow$ Graph
   • Node Feature Extraction
   • Graph Embedding
   • Channel-Aware Attention
   • Binary Classification

3 Experiments

4 Conclusion
Feature Extraction

For each node $v$ in graph $G$, we extract a feature vector from

- **Connectivity features**
  
  \[
  X_{v}^{con} = \{ e_{v,1}, \ldots, e_{v,|V|} \}
  \]

- **Graph statistical features**

  \[
  X_{v}^{stat} = \{ X_{v}^{s1}, X_{v}^{s2}, X_{v}^{s3}, X_{v}^{s4} \}
  \]
  - Degree centrality
  - Closeness centrality
  - Betweenness centrality
  - Clustering coefficient
Feature Extraction

For each node $v$ in graph $G$, we extract a feature vector from

- Connectivity features
  \[ X^\text{con}_v = \{ e_{v,1}, \ldots, e_{v,|V|} \} \]

- Graph statistical features
  \[ X^\text{stat}_v = \{ X^{s1}_v, X^{s2}_v, X^{s3}_v, X^{s4}_v \} \]
  - Degree centrality
  - Closeness centrality
  - Betweenness centrality
  - Clustering coefficient

How to combine as $X_v$? Concatenation?
Outline

1 Introduction
   • Problem
   • Proposed Solution

2 Method
   • Program \(\Rightarrow\) Graph
   • Node Feature Extraction
   • Graph Embedding
   • Channel-Aware Attention
   • Binary Classification

3 Experiments

4 Conclusion
Graph Embedding Function

Given homogeneous graph (single channel) $G = (V, E, A)$, each $V$ associated with feature $X (|V| \times (|V| + 4))$

**Goal:** to construct and learn a graph embedding function $f_G : G \rightarrow h_G$
Graph Embedding Function

Given homogeneous graph (single channel) 
\[ G = (V, E, A), \] 
each \( V \) associated with feature \( X (|V| \times (|V| + 4)) \)

**Goal:** to construct and learn a graph embedding function \( f_G : G \rightarrow h_G \)

**Proposed form:** a three-layer Contextual Graph Encoder

\[
\begin{align*}
  h^1 &= ReLU((PX)W^0) \\
  h^2 &= ReLU((Ph^1)W^1) \\
  h^3 &= ReLU((Ph^2)W^2) \\
  h_G &= h_{vt} = h^3
\end{align*}
\]

\( W^l \): shared trainable weight matrix for all entities at layer \( l \).
Graph Embedding Function

Given homogeneous graph (single channel) 
\[ G = (V, E, A), \text{ each } V \text{ associated with feature } X (|V| \times (|V| + 4)) \]

**Goal:** to construct and learn a graph embedding function \( f_G : G \rightarrow h_G \)

**Proposed form:** a three-layer Contextual Graph Encoder

\[
\begin{align*}
    h^1 &= ReLU((PX)W^0) \\
    h^2 &= ReLU((Ph^1)W^1) \\
    h^3 &= ReLU((Ph^2)W^2) \\
    h_G &= h_{v_t} = h^3
\end{align*}
\]

Each layer: 
\[
\hat{h}^l = PROP(h^l) = Ph^l \quad (h^0 = X) \\
\hat{h}^{l+1} = PERCE(\hat{h}^l) = \sigma(\hat{h}^l W^l) = ReLU(\hat{h}^l W^l)
\]

\( W^l \): shared trainable weight matrix for all entities at layer \( l \).
Propagation Function based on Random Walk

\[ \hat{h}^l = PROP(h^l) \]
\[ = Ph^l \]
\[ = D^{-1}Ah^l \]
\[ = diag(A1)^{-1}Ah^l \]

(1)

A: Adjacency matrix;  \( \mathbf{1} \): all one vector.
\( D = diag(A\mathbf{1}) \): degree matrix of A.
\( P = D^{-1}A \): propagation matrix shared in each layer.
Propagation Function based on Random Walk

\[ \hat{h}^l = PROP(h^l) \]
\[ = Ph^l \]
\[ = D^{-1}Ah^l \]
\[ = diag(A1)^{-1}Ah^l \]  \hspace{1cm} (1)

A: Adjacency matrix; \textbf{1}: all one vector.

\( D = diag(A1) \): degree matrix of A.

\( P = D^{-1}A \): propagation matrix shared in each layer.

**Implication**: weighted sum of the contexts’ current representation.

\[ \hat{h}^l = \sum_{u \in N(v_t)} P_{uv_t} h^l, \]
\[ \mathcal{F} = \{ N(v_t) \} : \text{receptive field} \]
\[ P \in \mathcal{R}^{N \times N} : \text{converged stationary distribution of the Markov process.} \]
\[ i^{th} \text{ row: likelihood of diffusion from entity.} \]
\[ A = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} \quad D = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad D^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{2} & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \]

\[ P = D^{-1}A = \begin{bmatrix} \frac{1}{2} & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ 1 & 0 & 0 \end{bmatrix} \]

**Figure:** Propagation matrix example.
Outline

1. Introduction
   - Problem
   - Proposed Solution

2. Method
   - Program ⇒ Graph
   - Node Feature Extraction
   - Graph Embedding
   - Channel-Aware Attention
   - Binary Classification

3. Experiments

4. Conclusion
Motivation

Treat three channels differently

- Programs;
- Files;
- Sockets.

Example

- Ransomware: active in files.
- VPN: active in socket.
Attention Weight

Attention weight $ATT(h_{G_i})$ for channel $i$:

$$\alpha_i = \frac{\exp(\sigma(a[h_{G_i}||W_a h_{G_k}])))}{\sum_{k' \in |C|} \exp(\sigma(a[h_{G_i}||W_a h_{G_{k'}} ])))}$$

Each channel $i = 1, 2, \ldots, |C|$

$h_{G_i}$: graph embedding of a target channel

$h_{G_k}$: graph embedding of other channels.

$a$: trainable attention vector.

$W_a$: trainable weight mapping (input features $\Rightarrow$ hidden space)

$||$: concatenation

$\sigma$: nonlinear gating function.
Joint representation of all channels:

\[ h_{G_{Join}} = \sum_{i=1}^{\vert C \vert} ATT(h_{G_i}) h_{G_i} \]
Outline

1 Introduction
   • Problem
   • Proposed Solution

2 Method
   • Program ⇒ Graph
   • Node Feature Extraction
   • Graph Embedding
   • Channel-Aware Attention
   • Binary Classification

3 Experiments

4 Conclusion
Train a binary classifier for each known program.

**Input:** A claimed program event data.

**Prediction:** If the program behaves like the claimed one.

- Logistic regression classifier.
- Binary cross entropy loss.
- Adam optimizer.
- Early stopping with good accuracy.
Experimental Setup

- **Dataset**: Real-world system monitoring data of 3 Terabytes. 87 machines over 20 weeks. 300M events, 2K processes, 600K files, 18K sockets. Behavior graph per program per day.

- **Baselines**.
  - LR, SVM, XGB, MLP using raw features.
  - MLP: special case that $PROP()$ is identity matrix.

- **Metrics**: ACC, F-1 score, AUC, precision and recall.
## Result

<table>
<thead>
<tr>
<th>Method</th>
<th>Settings</th>
<th>Evaluation Criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>ACC</td>
</tr>
<tr>
<td><strong>LR</strong></td>
<td>fea-1</td>
<td>0.693</td>
</tr>
<tr>
<td></td>
<td>fea-2</td>
<td>0.705</td>
</tr>
<tr>
<td></td>
<td>fea-3</td>
<td>0.724</td>
</tr>
<tr>
<td><strong>SVM</strong></td>
<td>fea-1</td>
<td>0.502</td>
</tr>
<tr>
<td></td>
<td>fea-2</td>
<td>0.795</td>
</tr>
<tr>
<td></td>
<td>fea-3</td>
<td>0.504</td>
</tr>
<tr>
<td><strong>XGB</strong></td>
<td>fea-1</td>
<td>0.775</td>
</tr>
<tr>
<td></td>
<td>fea-2</td>
<td>0.833</td>
</tr>
<tr>
<td></td>
<td>fea-3</td>
<td>0.855</td>
</tr>
<tr>
<td><strong>MLP_{shallow}</strong></td>
<td>fea-1</td>
<td>0.633</td>
</tr>
<tr>
<td></td>
<td>fea-2</td>
<td>0.775</td>
</tr>
<tr>
<td></td>
<td>fea-3</td>
<td>0.778</td>
</tr>
<tr>
<td><strong>MLP_{deep}</strong></td>
<td>fea-1</td>
<td>0.633</td>
</tr>
<tr>
<td></td>
<td>fea-2</td>
<td>0.801</td>
</tr>
<tr>
<td></td>
<td>fea-3</td>
<td>0.815</td>
</tr>
<tr>
<td><strong>DeepRe-ID_{shallow}</strong></td>
<td>/</td>
<td>0.905</td>
</tr>
<tr>
<td><strong>DeepRe-ID_{deep}</strong></td>
<td>/</td>
<td><strong>0.929</strong></td>
</tr>
</tbody>
</table>

**Figure**: Comparison of other classification methods.
Conclusion

- **DeepRe-ID**, an attentional graph neural network method to verify the program identity based on behavior graph.
- Can encode heterogeneous complex dependency.
- Outperform all baseline methods.
Drawbacks:

- No open dataset or open source code.
- Require feature engineering: graph statistical features.
- Require adjacency matrix.
- Binary classification with many classes.
- No interpretation of trained models.