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

Outline

Program Reidentification

- Determine if an unknown program is variant of a known program.
- Used to detect disguised malware or ramsomeware.

Digital Code Signing is Useful

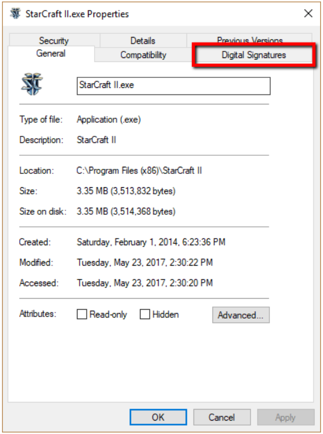


Figure: Program Properties

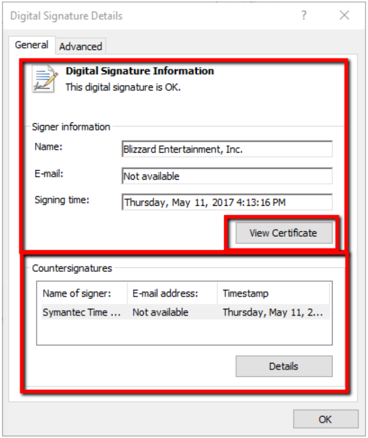


Figure: Digital Signature

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

Proposed Solution

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

Outline

Extract Graph from a Program

Possible choices:

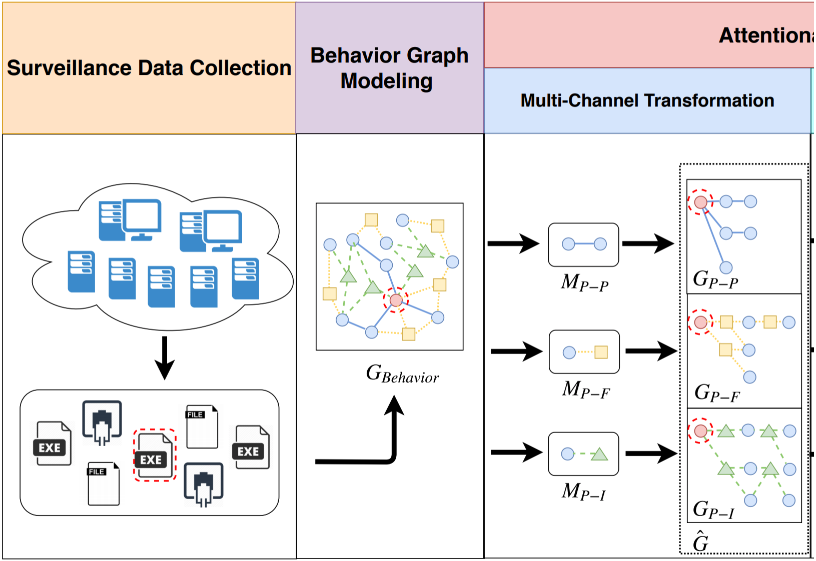
- Static analysis
  E.g. Call graph of code blocks.
- Dynamic analysis
  E.g. System interaction graph.

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



Heterogeneous Graph

Three types of nodes:

- Fork another **program**.
- Read/Write a **file**.
- Access to a network **socket** $< IPAddr : Port >$.

Solution: separate into three homogeneous graphs (meta-path).

- Program - Program.
- Program - File.
- Program - Socket.

Attentional Multi-Channel Graph Neural Network

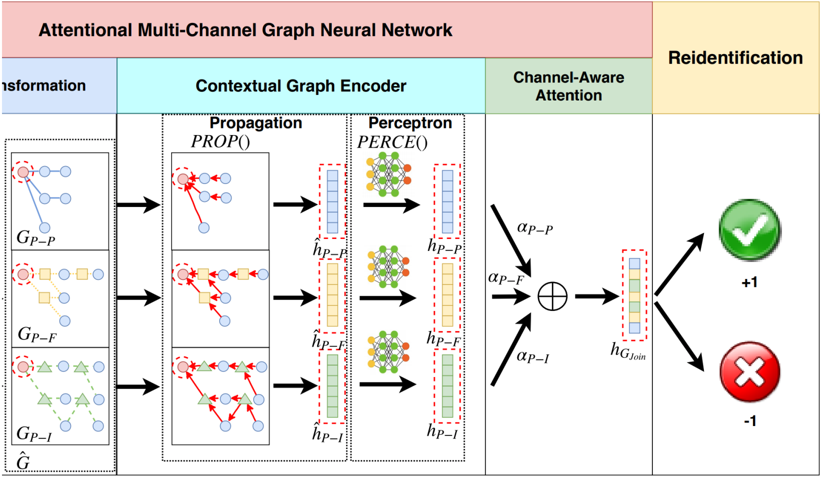


Figure: Attentional Multi-Channel Graph Neural Network.

Outline

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
  $X_v^{con} = \{e_{v,1} \ldots, e_{v,|V|}\}$
- 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

How to combine as X_v? Concatenation?

Outline

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 $f_G : G \to h_G$

Proposed form: a three-layer Contextual Graph Encoder

$$h^1 = ReLU((PX)W^0)$$
$$h^2 = ReLU((Ph^1)W^1)$$
$$h^3 = ReLU((Ph^2)W^2)$$
$$h_G = h_{v_t} = h^3$$

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 $f_G : G \to h_G$

Proposed form: a three-layer Contextual Graph Encoder

$$h^1 = ReLU((PX)W^0)$$
$$h^2 = ReLU((Ph^1)W^1)$$
$$h^3 = ReLU((Ph^2)W^2)$$
$$h_G = h_{v_t} = h^3$$

Each layer: $\hat{h}^l = PROP(h^l) = Ph^l$ $(h^0 = X)$
$h^{l+1} = PERCE(\hat{h}^l) = \sigma(\hat{h}^l W^l) = ReLU(\hat{h}^l W^l)$
W^l: shared trainable weight matrix for all entities at layer l.

Propagation Function based on Random Walk

$$\begin{aligned}\hat{h}^l &= PROP(h^l) \\ &= Ph^l \\ &= D^{-1}Ah^l \\ &= diag(A\mathbf{1})^{-1}Ah^l\end{aligned} \tag{1}$$

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

$$
\begin{aligned}
\hat{h}^l &= PROP(h^l) \\
&= Ph^l \\
&= D^{-1}Ah^l \\
&= diag(A\mathbf{1})^{-1}Ah^l
\end{aligned} \tag{1}
$$

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.

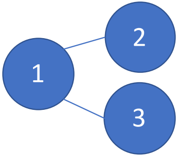
Implication: weighted sum of the contexts' current representation.

$\hat{h}^l = \sum_{u \in N(v_t)} P_{uv_t} h^l$, $\mathcal{F} = \{N(v_t)\}$: receptive field

$P \in \mathcal{R}^{N \times N}$: converged stationary distribution of the Markov process.

i^{th} row: likelihood of diffusion from entity.

Propagation Matrix Example



$$A = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} \quad D = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad D^{-1} = \begin{bmatrix} \frac{1}{2} & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

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

Figure: Propagation matrix example.

Outline

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:

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

Each channel $i = 1, 2, ..., |C|$
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
σ: nonlinear gating function.

Joint Representation of All Channels

Joint representation of all channels:

$$h_{G_{Join}} = \sum_{i=1}^{|C|} \boxed{ATT(h_{G_i})}\, h_{G_i}$$

Outline

Program Reidentification

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

Method	Settings	Evaluation Criteria				
		ACC	F-1	AUC	Precision	Recall
LR	*fea-1*	0.693	0.755	0.699	0.632	0.948
	fea-2	0.705	0.770	0.703	0.655	0.950
	fea-3	0.724	0.772	0.727	0.675	0.948
SVM	*fea-1*	0.502	0.662	0.502	0.505	0.970
	fea-2	0.795	0.778	0.725	0.701	0.935
	fea-3	0.504	0.652	0.504	0.505	**0.975**
XGB	*fea-1*	0.775	0.802	0.776	0.732	0.930
	fea-2	0.833	0.860	0.846	0.821	0.936
	fea-3	0.855	0.866	0.856	0.827	0.937
$MLP_{shallow}$	*fea-1*	0.633	0.745	0.643	0.626	0.938
	fea-2	0.775	0.808	0.779	0.724	0.932
	fea-3	0.778	0.808	0.780	0.726	0.932
MLP_{deep}	*fea-1*	0.633	0.743	0.653	0.625	0.945
	fea-2	0.801	0.830	0.805	0.769	0.921
	fea-3	0.815	0.831	0.816	0.778	0.923
DeepRe-ID$_{shallow}$	/	0.905	0.929	0.908	0.905	0.933
DeepRe-ID$_{deep}$	/	**0.929**	**0.961**	**0.935**	**0.932**	0.936

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.

Discussions

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.