Scribe Note: Neural Network-based Graph Embedding for Cross-Platform Binary Code Similarity Detection[2]

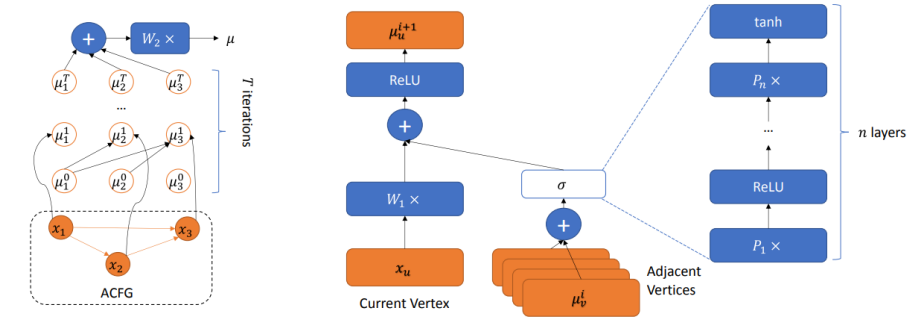
Presenter: Faizan Ahmad, Scribe: Ji Gao

4/5/2019

1 Research Question

- **Research question:** Cross-platform binary code similarity detection.
- **Previous work:** Graph matching based algorithm. Can be slow, inaccurate, and hard to scale to other tasks.
- **Method:** Use deep graph models to generate embedding for the graph. In addition, use Siamese network to learn the difference.

2 Overview Figure



a) Graph Embedding Network Overview

(b) One Layer (Iteration) of the Graph Embedding Network

Figure 3: Graph Embedding Network

3 Model

- Problem: Generate an embdding for the Attributed Control Flow Graph(ACFG).

- Structure2Vec model:

$$\mu_v^{(t+1)} = \mathcal{F}(x_v, \sum_{u \in N(v)} \mu_u^{(t)}), \forall v \in \mathcal{V} \tag{1}$$

$\mathcal{F}$ is a nonlinear transformation function.

- This paper:

$$\mu_v^{(t+1)} = \tanh(W_1 x_v + \sigma(\sum_{u \in N(v)} \mu_u^{(t)})) \tag{2}$$

Where σ is a n layer Fully connected neural network.

4 Siamese Learning

- Idea: learn how to generate embeddings by matching the generated embeddings of two similar inputs in training.

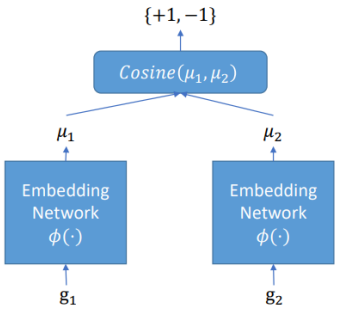


Figure 4: Siamese Architecture

- Use cosine similarity as the importance measure.

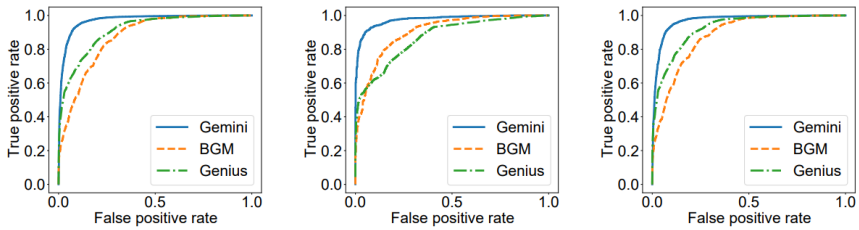
$$\cos(\phi(g), \phi(g')) = \frac{< \phi(g), \phi(g') >}{||\phi(g)|| \cdot ||\phi(g')||} \tag{3}$$

5 Pre-Training:

- Learn on a general default task first: whether two binary functions are compiled from the same source code.
- Fine-tuning on specific tasks: Add additional data with specific task labels (i.e., A pair of graphs should be similar) into the data. Sample 50 times more often of the new data

6 Evaluation

- **Baselines:**
  - Bipartite Graph Matching(BGM): Match the graphs directly using Bipartite Graph Matching algorithms.
  - Codebook-based Graph Embedding (Genius)[1]: Previous graph embedding method.
- **Experiments:**
  - Task 1: Compile OpenSSL on 3 different architectures, get 129,365 ACFGs.
  - Task 2: Large scale data with with 33,045 firmware images.
  - Task 3: vulnerable functions obtained from the vulnerability dataset
  - Task 4: Efficiency evaluation with different size of ACFGs.
- Result:
  - ROC curve:

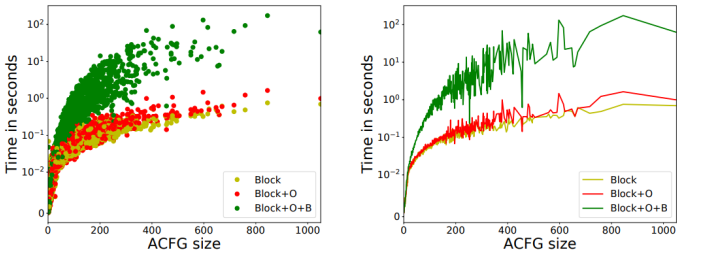


(a) Results on the similarity testing set (b) Results on the large-graph subset (c) Results on the small-graph subset

Figure 5: ROC curves for different approaches evaluated on the testing similarity dataset.

ROC curve shows Gemini outperforms baselines.

  - Efficiency: It's faster than Genius[1].



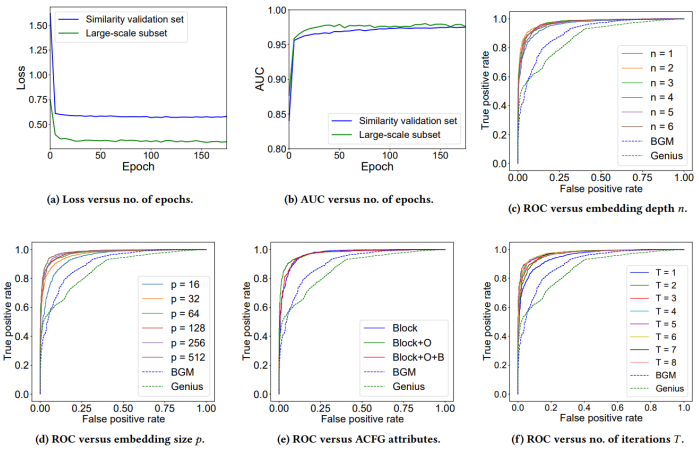
(a) ACFG Extraction time

(b) Average ACFG extraction time

(c) Feature vector generation time

(d) Average feature vector generation time

– Hyperparameters:



(a) Loss versus no. of epochs.

(b) AUC versus no. of epochs.

(c) ROC versus embedding depth n.

(d) ROC versus embedding size p.

(e) ROC versus ACFG attributes.

(f) ROC versus no. of iterations T.

Figure 7: Effectiveness of different hyperparameters of NN. Figure 7a and Figure 7b are evaluated over the similarity validation set. Figure 7c, 7d, 7e, and 7f are evaluated over large-graph subset of the similarity testing set.

* Adding number of layers of MLP in the model doesn't help much.
* When T=5(Number of stacked embedding layers), the model gets best performance.

References

[1] Qian Feng, Rundong Zhou, Chengcheng Xu, Yao Cheng, Brian Testa, and Heng Yin. Scalable graph-based bug search for firmware images. In *Pro-*

ceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pages 480–491. ACM, 2016.

[2] Xiaojun Xu, Chang Liu, Qian Feng, Heng Yin, Le Song, and Dawn Song. Neural network-based graph embedding for cross-platform binary code similarity detection. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 363–376. ACM, 2017.