Progess of the Project

Tsung-Min Pai

2023/8/9

Outline

- **Graph - Data Analysis**
- **GNN**
  - GAT
  - GCN
- **TRAM**
- **Future Work**

Graph - Data Analysis

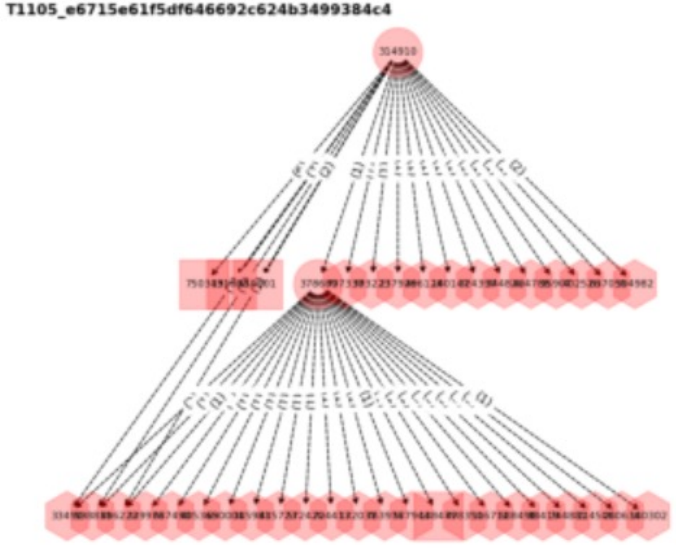
Graph

- Considering the **entity** of each nodes → Give each different shapes
  - **Process**: circle, **Registry**: hexagon, **File**: square, **Network**: diamond
- Plot a big graph contains **165 APs**
- Compare the real labels with the predicted labels of **Sigma** Rule
  - If matched: **red** nodes with red **solid** line
  - If not matched: half **transparent** red nodes with black **dotted** line

src	src_entity	dest	dest_entity	rel	label	sigma
665262	process	246678	registry	19	benign	benign
665262	process	619403	registry	11	benign	benign
665262	process	251142	registry	11	benign	benign
526287	file	433452	registry	18	T1005_720	benign

Graph

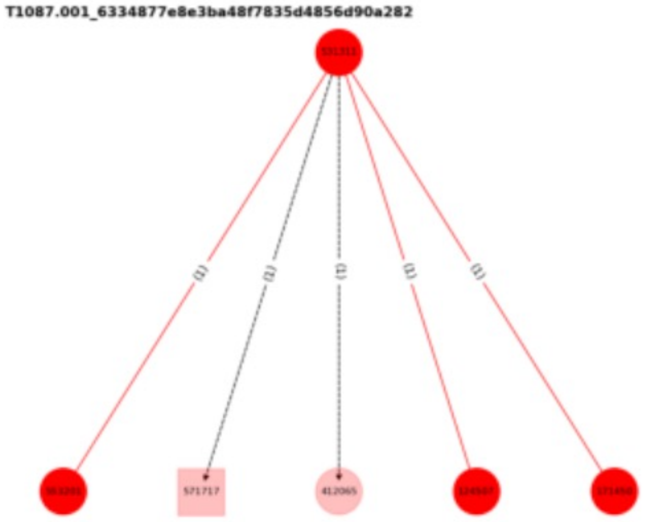
T1105_e67



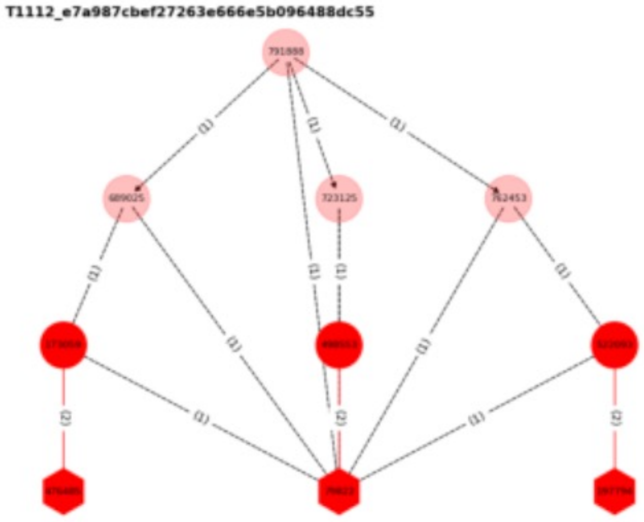
T1219_7da

Many of them are not matched!

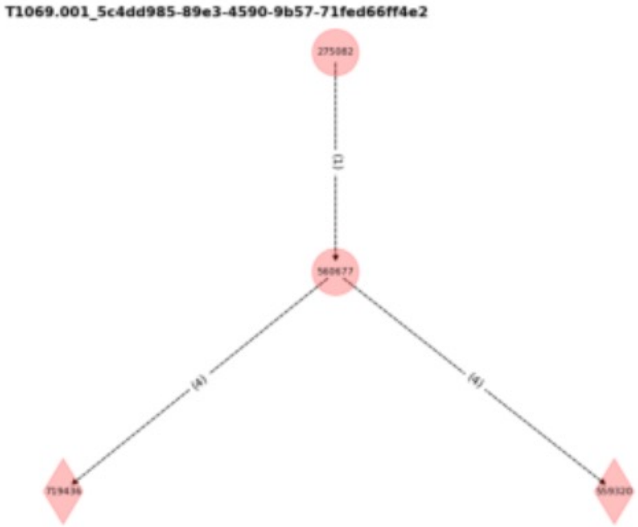
T1087.001_633



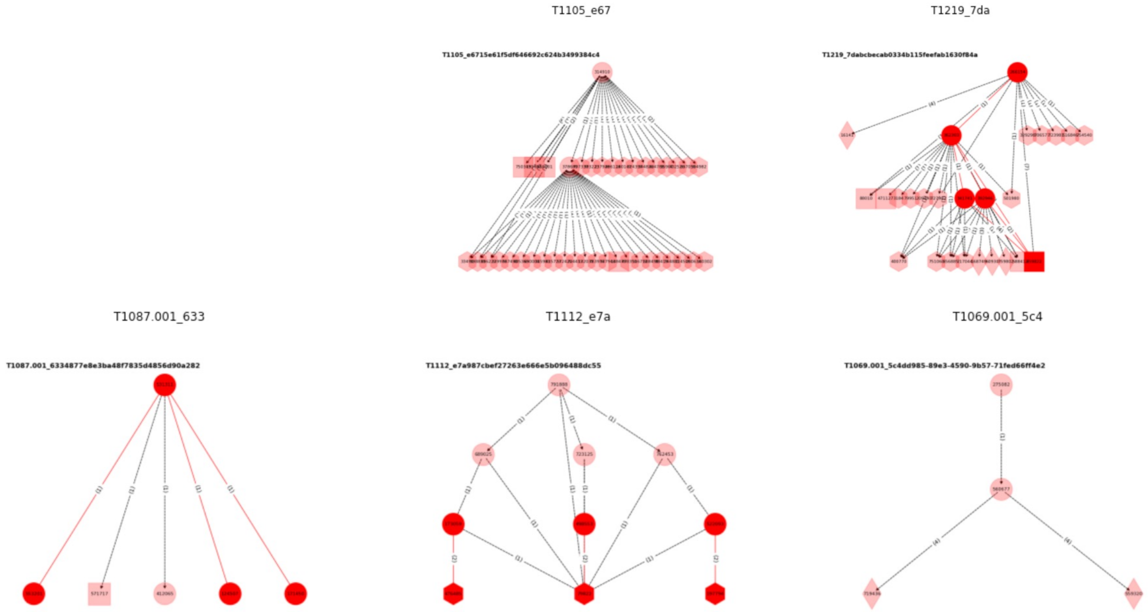
T1112_e7a



T1069.001_5c4



Graph



GNN

Data Format

- Old Version:

y (sequence)	num_nodes (int64)	node_feat (sequence)	edge_attr (sequence)	edge_index (sequence)
[76]	3	[[562981], [21], [328936]]	[[0], [0]]	[[0, 1], [1, 2]]
[0]	3	[[549132], [25], [257747]]	[[0], [0]]	[[0, 1], [1, 2]]
[0]	3	[[753794], [19], [659061]]	[[0], [0]]	[[0, 1], [1, 2]]

- New Version:


```
{"label": 10, "num_nodes": 3, "node_feat": [205565, 733769, 250773], "edge_attr": [23, 23], "edge_index": [[0, 0], [1, 2]]}
{"label": 11, "num_nodes": 3, "node_feat": [470650, 663446, 627322], "edge_attr": [23, 23], "edge_index": [[0, 0], [1, 2]]}
{"label": 15, "num_nodes": 2, "node_feat": [9863, 103498], "edge_attr": [23], "edge_index": [[0], [1]]}
{"label": 16, "num_nodes": 2, "node_feat": [157277, 753159], "edge_attr": [23], "edge_index": [[0], [1]]}
{"label": 22, "num_nodes": 36, "node_feat": [83068, 614681, 444724, 266227, 121794, 623948, 116790, 769462, 255741, 169794,
{"label": 0, "num_nodes": 7, "node_feat": [150942, 396371, 507529, 763634, 318841, 126686, 88192], "edge_attr": [11, 19, 15, 25,
{"label": 0, "num_nodes": 3, "node_feat": [264800, 229960, 554967], "edge_attr": [19, 15], "edge_index": [[0, 0], [1, 2]]}
{"label": 0, "num_nodes": 4, "node_feat": [416286, 466370, 94352, 15536], "edge_attr": [9, 9, 9, 9], "edge_index": [[0, 2, 0, 2],
```


- 165 APs + 35 benign (connected subgraph)
- Label is come from the **LabelEncoder**

- Multiply 200 times
- Train:Validation:Test = 3:1:1

Data Format

- New Version:


```
{"label": 10, "num_nodes": 3, "node_feat": [205565, 733769, 250773], "edge_attr": [23, 23], "edge_index": [[0, 0], [1, 2]]}
{"label": 11, "num_nodes": 3, "node_feat": [470650, 663446, 627322], "edge_attr": [23, 23], "edge_index": [[0, 0], [1, 2]]}
{"label": 15, "num_nodes": 2, "node_feat": [9863, 103498], "edge_attr": [23], "edge_index": [[0], [1]]}
{"label": 16, "num_nodes": 2, "node_feat": [157277, 753159], "edge_attr": [23], "edge_index": [[0], [1]]}
{"label": 22, "num_nodes": 36, "node_feat": [83068, 614681, 444724, 266227, 121794, 623948, 116790, 769462, 255741, 169794,
```

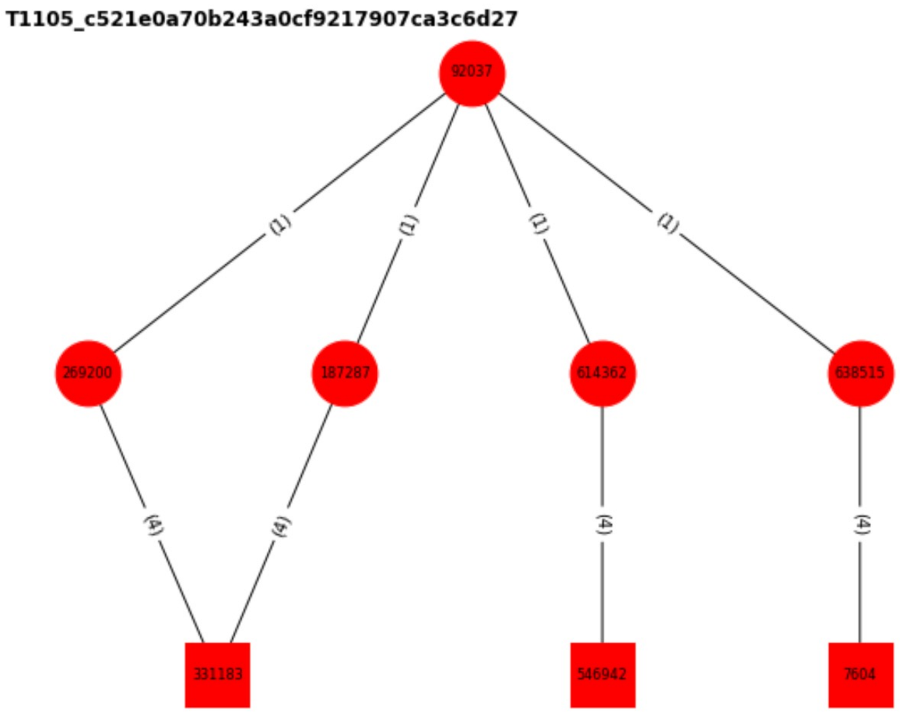

- Since assigning the node id as the node feature is a little bit weird
  - Map the **node_feat** to its **embedding** based on the **transR_50.vec.json** → node id correspond to a vector of **50-dim**

- We have the maps recording the:
  1. **label** to original **name** of AP
  2. **Real node id** to the **DGL** (Deep Graph Library) node id


```
"node_feat": [[-0.004259330220520496, 0.020233182236552224, 0
37990725412965, -0.01702643744647503, 0.0013664175057783723,
65514039993, -0.00885914359241724, -0.0010546729899942875, -
9137960374355, -0.009182648733258247, -0.008827704936265945,
9938482046127, -0.0014976661186665297, 0.008676833473145962,
```

Data Format

Current version:



Q: How to label?

Graph Convolutional Network - GCN

Model:

```
class GCN(nn.Module):
    def __init__(self, in_feats, hidden_size, num_classes):
        super(GCN, self).__init__()
        self.conv1 = GraphConv(in_feats, hidden_size)
        self.conv2 = GraphConv(hidden_size, num_classes)

    def forward(self, g, inputs):
        h = self.conv1(g, inputs)
        h = torch.relu(h)
        h = self.conv2(g, h)

        g.ndata['h'] = h
        hg = dgl.mean_nodes(g, 'h')
        return hg
```

- Use the **old** verison of the dataset
- Use **DGL** to be our library
- DGL data format:

```
batched_g is like:
Graph(num_nodes=96, num_edges=160, ndata_schemes={'feat': Scheme(shape=(1,), dtype=torch.int64)}, edata_schemes={})
num_nodes = 3*batch_size, num_edges = 5*batch_size

labels is like: tensor([ 76,   0,   0,   0,   0,   0,   0,   0,   0,  76,   0,  76,   0,   0,
                          0,   0,  76,   0,  30,  92,   0,   0,  76,   0,   0,   0,   0,   0,
                        116,   0,  76,  76])
```

Result:

```
  0%|          | 0/120 [00:00<?, ?it/s]
Epoch 0 | Train Loss: 2625.5943 | Train Accuracy: 0.4763
  1%|          | 1/120 [00:56<1:52:21, 56.65s/it]
Validation Loss: 494.0275 | Validation Accuracy: 0.6642
 99%|█████████▉| 119/120 [1:51:06<00:55, 55.13s/it]
Validation Loss: 0.9964 | Validation Accuracy: 0.6642
Epoch 119 | Train Loss: 0.9625 | Train Accuracy: 0.6644
100%|██████████| 120/120 [1:52:03<00:00, 56.03s/it]
Validation Loss: 0.9965 | Validation Accuracy: 0.6642

Test Accuracy: 66 %
```

- GAT applied on the old data has the similar result

Graph Attention Network - GAT

Model:

```
class GAT(nn.Module):
    def __init__(self, in_dim, hidden_dim, out_dim, num_heads, dropout_prob=0.2):
        super(GAT, self).__init__()
        # do not check the zero in_degree since we have all the complete graph
        self.layer1 = GATConv(in_dim, hidden_dim, num_heads=num_heads, activation=F.relu, allow_zero_in_degree=True)
        self.layer2 = GATConv(hidden_dim * num_heads, out_dim, num_heads=num_heads, allow_zero_in_degree=True)

        # Adding Dropout for regularization
        self.dropout = nn.Dropout(dropout_prob)

    def forward(self, g, h):
        # Apply GAT layers
        h = self.layer1(g, h)
        h = h.view(h.shape[0], -1)
        h = F.relu(h)
        h = self.dropout(h)
        h = self.layer2(g, h).squeeze(1)

        # Store the output as a new node feature
        g.ndata['h_out'] = h

        # Use mean pooling to aggregate this new node feature
        h_agg = dgl.mean_nodes(g, feat='h_out')
        return h_agg
```

- Use the **new** verison of the dataset

Graph Attention Network - GAT

```
Graph(num_nodes=156, num_edges=284,
      ndata_schemes={'feat': Scheme(shape=(50,), dtype=torch.float32)}
      edata_schemes={'feat': Scheme(shape=(50,), dtype=torch.float32)}),
tensor([164,  84, 152,  62, 127,  30,  11,  60,   0,   2, 158,  98, 146, 123,
        157, 147,  35, 140,  89,  70, 158,  55, 110,  64,  75, 105,  97,   0,
          0,  21, 138, 119])
```

```
  4%|▌         | 26/600 [13:13<4:53:13, 30.65s/it]
Validation Loss: 4.4224 | Validation Accuracy: 0.1764
total count: 750
Epoch 26 | Train Loss: 4.4137 | Train Accuracy: 0.1770
  4%|▌         | 27/600 [13:44<4:53:53, 30.77s/it]
Validation Loss: 4.3966 | Validation Accuracy: 0.1812
```

```
 15%|█▌        | 91/600 [46:32<4:19:50, 30.63s/it]
Validation Loss: 2.9832 | Validation Accuracy: 0.4744
total count: 750
Epoch 91 | Train Loss: 2.9960 | Train Accuracy: 0.4608
 15%|█▌        | 92/600 [47:03<4:19:47, 30.68s/it]
Validation Loss: 2.9678 | Validation Accuracy: 0.4796
total count: 750
```

```
100%|█████████▉| 599/600 [5:08:29<00:30, 30.96s/it]
Validation Loss: 2.8581 | Validation Accuracy: 0.4796
total count: 750
Epoch 599 | Train Loss: 2.8743 | Train Accuracy: 0.4730
100%|██████████| 600/600 [5:08:59<00:00, 30.90s/it]
Validation Loss: 2.8581 | Validation Accuracy: 0.4796
```

- Validation accuracy stop at 0.4796 since epoch 92
- Model? Dataset? Preprocessing? Training steps?

TRAM

Automation

- Successfully **upload** the pdf files
- Successfully **export** the pdf files
  - Click 3 times and then scroll $\frac{1}{3}$ of the window size

```
if count % 3 == 0:
    driver.execute_script(f"window.scrollBy(0, {window_height/3});")
    time.sleep(1)
```

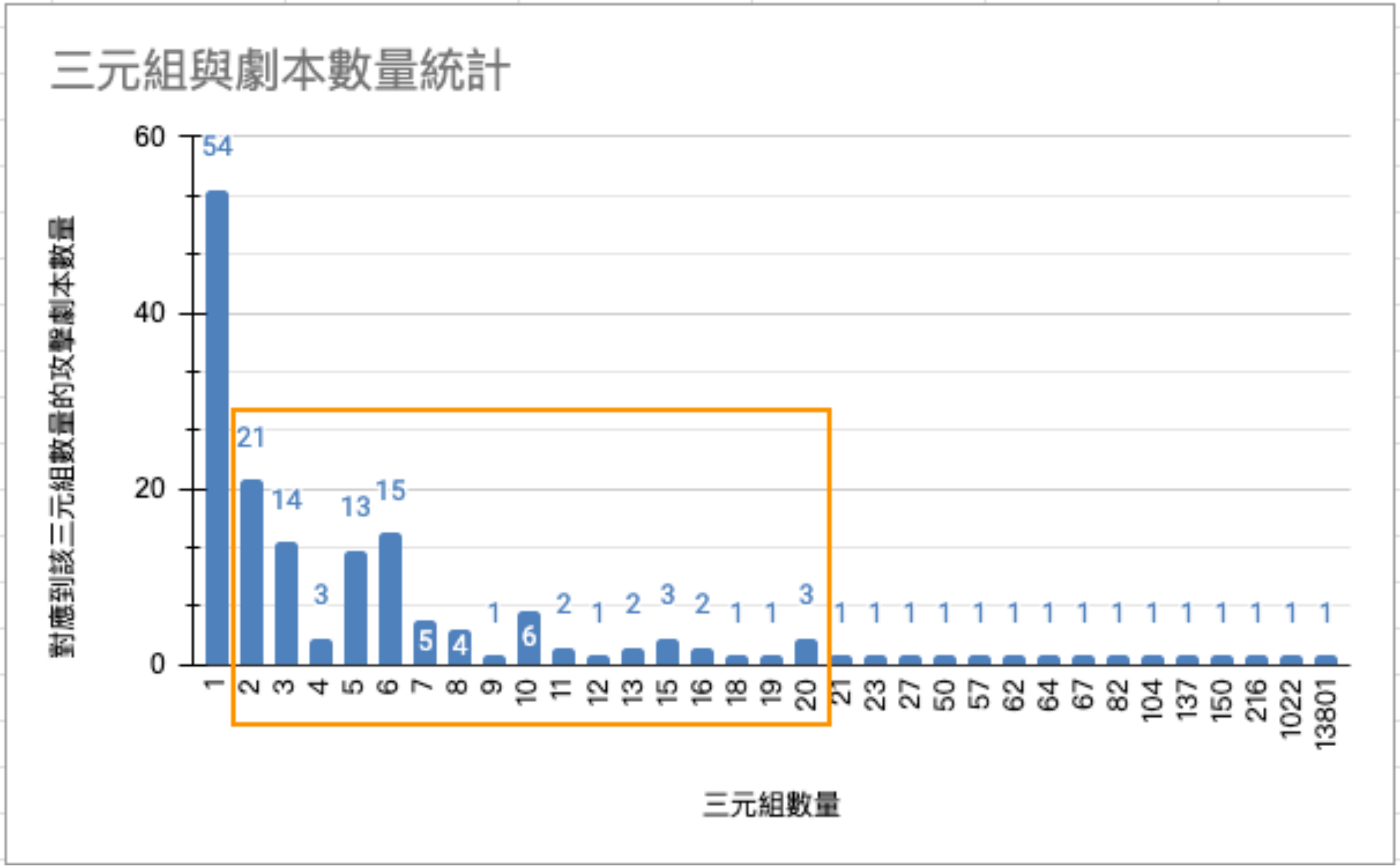
Future Work

Future Work

- **GNN**
  - Figure out the reason causing the currently bad performance on both GCN, GAT
  - Read some paper about these GNN models
  - Try the GraphSAGE
- **TRAM**
  - Delete.py
  - Try to upload and export HTML files
  - Transfer them into labeled data
- **Graphormer** (if available)
  - Write the trainer (training part)

Thanks!!

Appendix



三元組與劇本數量統計
對應到該三元組數量的攻擊劇本數量
60
40
20
0
54
21
14
3
13
15
5
4
1
6
2
1
2
3
2
1
1
3
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
2
3
4
5
6
7
8
9
10
11
12
13
15
16
18
19
20
21
23
27
50
57
62
64
67
82
104
137
150
216
1022
13801
三元組數量