Malware classification based on graph convolutional neural networks and static call graph features

Attila Mester and Zalán Bodó

{*attila.mester*, *zalan.bodo*}@*ubbcluj.ro*, *amester*@*bitdefender.com*

Babeş–Bolyai University of Cluj-Napoca

20th July 2022

Content

1. Problem definition

Attribution

Literature

Static call graph – IDA Pro

2. Graph convolutional neural networks

Literature

Scientific approach

Experiments and Results

3. Conclusions & Future work

Problem definition

- aim: classify family and/or actor(s) behind an attack (**attribution**)
- complex features: infrastructure, intrusion, infection method, events, etc.
- simple feature: the binary file – PE executable's static call graph
- goal: *malware family detection with high number of families*

Survey of previous researches

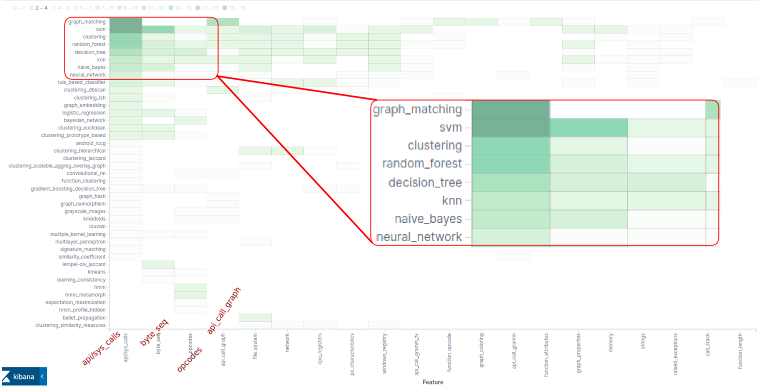
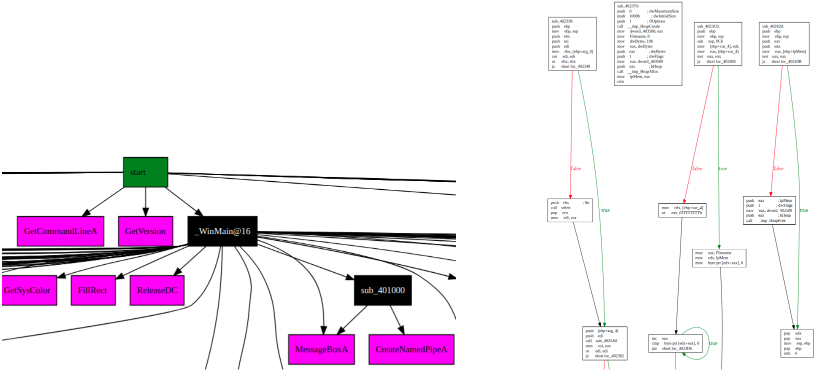


Figure 1: ≈ 100 research papers categorized according to *extracted features* and *algorithms*. Most frequent: *API/sys calls*.

What is a static call graph of an *.exe*?

- (dynamic = execution in sandbox)
- static = disassembler
- function execution sequence = call graph
  - node = function (black – local, blue – statically linked lib., purple – DLL)
  - link = function call
- *why not blacklist the hash of the graph?*
  - metamorphic viruses: code generations
  - common libraries, functions

Generating the static call graph



```
sub_402330:
push    ebp
mov     ebp, esp
push    ebx
push    esi
push    edi
mov     ebx, [ebp+arg_0]
xor     edi, edi
or      ebx, ebx
jz      short loc_402348
```

```
sub_402370:
push    0               ; dwMaximumSize
push    1000h           ; dwInitialSize
push    1               ; flOptions
call    __imp_HeapCreate
mov     dword_403500, eax
mov     Filename, 0
mov     dwBytes, 10h
mov     eax, dwBytes
push    eax             ; dwBytes
push    1               ; dwFlags
mov     eax, dword_403500
push    eax             ; hHeap
call    __imp_HeapAlloc
mov     lpMem, eax
retn
```

```
sub_4023C0:
push    ebp
mov     ebp, esp
sub     esp, 0Ch
mov     [ebp+var_4], edx
mov     eax, [ebp+var_4]
test    eax, eax
jz      short loc_402405
```

```
sub_402420:
push    ebp
mov     ebp, esp
push    eax
push    edx
mov     eax, [ebp+lpMem]
test    eax, eax
jz      short loc_40243B
```

```
push    ebx             ; Str
call    strlen
pop     ecx
mov     edi, eax
```

```
mov     edx, [ebp+var_4]
or      eax, 0FFFFFFFFh
```

```
push    eax             ; lpMem
push    1               ; dwFlags
mov     eax, dword_403500
push    eax             ; hHeap
call    __imp_HeapFree
```

```
mov     eax, Filename
mov     edx, lpMem
mov     byte ptr [edx+eax], 0
```

```
push    [ebp+arg_4]
push    edi
call    sub_4025A0
mov     esi, eax
or      edi, edi
jz      short loc_402362
```

```
inc     eax
cmp     byte ptr [edx+eax], 0
jnz     short loc_4023D6
```

```
pop     edx
pop     eax
mov     esp, ebp
pop     ebp
retn    4
```

GenCallGdl: call graph without instructions

GenFuncGdl: execution flowchart

Figure 2: IDA Pro disassembler

Generating the static call graph II

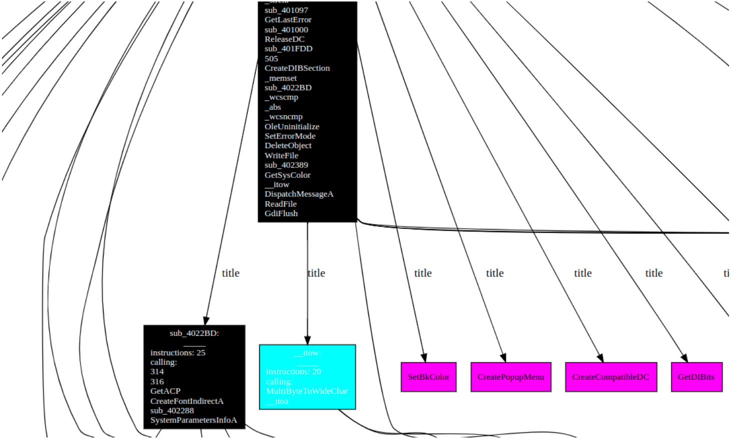


Figure 3: Static call graph of *totalcmd.exe* – merge method in (Mester and Bodó 2021)

Static call graph of 2 variations of a metamorphic virus

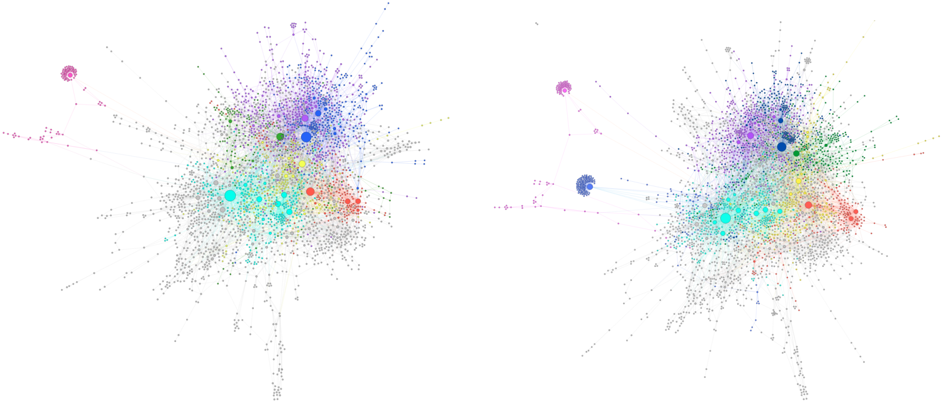


Figure 4: Static call graph of metamorphic generations (Gephi, Force Atlas)

How to extract info from this graph? I

- clustering problem: **signatures** (Mester and Bodó 2021)

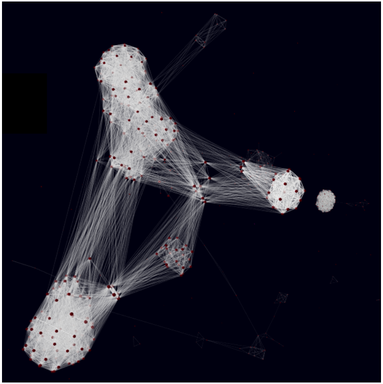


Figure 5: 600 malicious files, having 24 000 signatures

How to extract info from this graph? II

- classification problem: **neural networks**

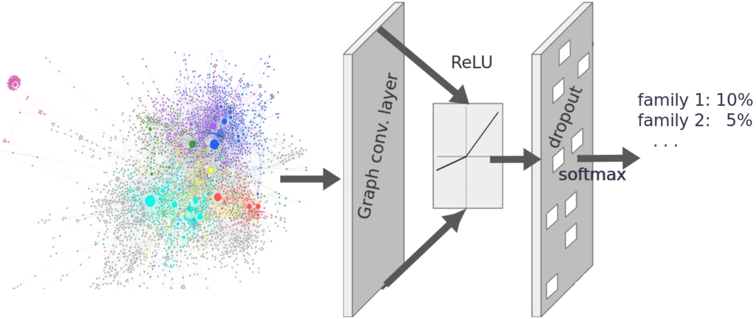


Figure 6: Malware family classification with GCN

Graph convolutional neural networks

- CNN – convolutional operator
- GCN – specialized CNN for graph input type
  - spatial: neighbourhood info used for embedding
  - spectral: eigenvectors of graph Laplacian
- Laplacian smoothing: averaging the points in the neighbourhood (Kipf and Welling 2016) – nodes in same cluster, similar vector representaion

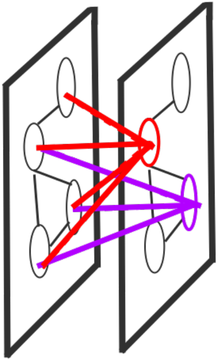


Figure 7: Laplacian smoothing: averaging the neighbourhood information

- propagation rule:

$$\mathbf{H}^{(i+1)} = \sigma\left(\tilde{\mathbf{A}}\mathbf{H}^{(i)}\mathbf{W}^{(i)}\right) \tag{1}$$

- **H** – embedded data representation, $H^0 = X$ (input feature matrix)
- $\tilde{\mathbf{A}}$ – normalized adj. matrix
- **W** – weights of neural network
- σ – activation function (e.g. ReLU)
- usecases:
  - node classification
  - **graph classification**
  - link classification, edge prediction

Literature

- *Android*: (Cai et al. 2021) – first on Android GCN: app's runtime behaviour - function calls - embedding - SVM ; (John, Thomas, and Emmanuel 2020)
- dynamic analysis (Oliveira and Sassi 2021) – **not scalable**
- static API calls, graph, GCN (Dam and Touili 2017; Hong, S. Park, et al. 2018; Phan et al. 2018; Hong, S.-J. Park, et al. 2019)
- node / graph embedding (Jiang, Turki, and Wang 2018; Hong, S.-J. Park, et al. 2019; J. Yan, G. Yan, and Jin 2019)

What do we do differently?

- size of dataset in literature vs. our dataset (D)
- # of classes in literature vs. # of families in D
- node-level features: LSH on function's instruction n-gram distribution (Mester and Bodó 2021)
- 223 families, 8620 samples
  - 6 – 12 families in (Hong, S. Park, et al. 2018; Hong, S.-J. Park, et al. 2019; Tang and Qian 2019; J. Yan, G. Yan, and Jin 2019)
- *training on call graph, not CFG* (e.g. J. Yan, G. Yan, and Jin 2019)

Summary

- scan with call graph: IDA Pro, *GenCallGdl*, *GenFuncGdl* (A)
- obtain LSH codewords of subroutines – random projection method (X)
- training the GCN on A
- training the GCN on A and X
- training the GCN on A and (J. Yan, G. Yan, and Jin 2019)
- training a MLP on X
- training a MLP on (J. Yan, G. Yan, and Jin 2019)

```
ModuleList(
    (0): GCNConv(8, 128)
    (1): ReLU()
    (2): Dropout(p=0.5)
    (3): GCNConv(128, 128)
    (4): ReLU()
    (5): Dropout(p=0.5)
    (6): GCNConv(128, 128)
    (7): ReLU()
    (8): Dropout(p=0.5)
    (9): GCNConv(128, 128)
    (10): Dropout(p=0.5)
  )
  (f): Linear(in_features=128, out_features=223, bias=True))
```

Figure 8: GCN model used in the experiments

```
(stack): Sequential(
    (0): Linear(in_features=8, out_features=128, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.5)
    (3): Linear(in_features=128, out_features=128, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.5)
    (6): Linear(in_features=128, out_features=223, bias=True)
  )
```

Figure 9: MLP model used for learning only on node-level features

Tech stack + hardware

- Python3, IDA Pro 6, GraphViz, PyTorch 1.10.0, Pytorch Geometric (pyg) 2.0.2, Tensorboard
- Intel Xeon E5-2697A v4, 64 GB RAM, GeForce RTX 2080 Ti
    - **sincere thanks to Bitdefender**

Dataset

- 15 375 samples from 967 families
- after filtering: 8620 samples from 223
- call graph nodes: max. $76k$, avg. $1k$
- call graph links: max $245k$, avg. $3.4k$

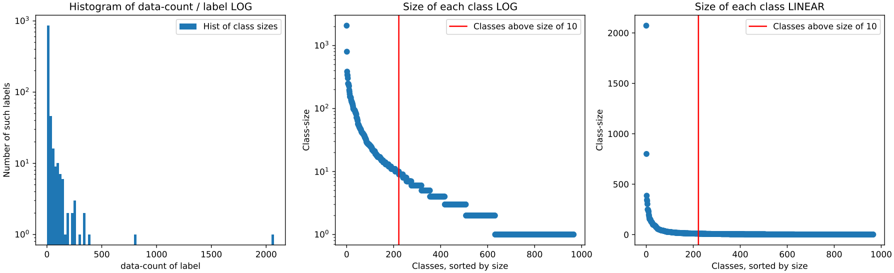
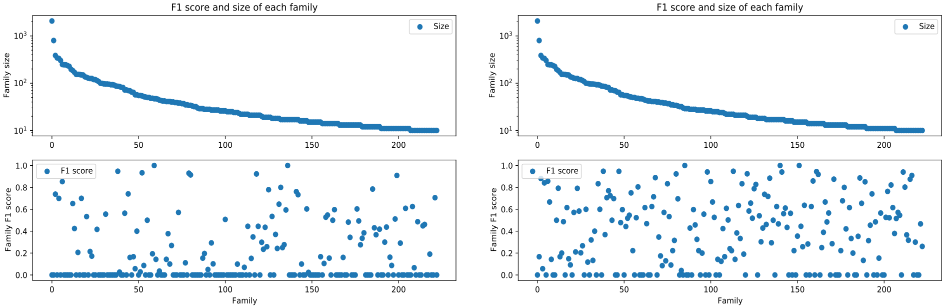


Figure 10: Distribution of family sizes within the dataset of 15k samples.

Hyperparameters

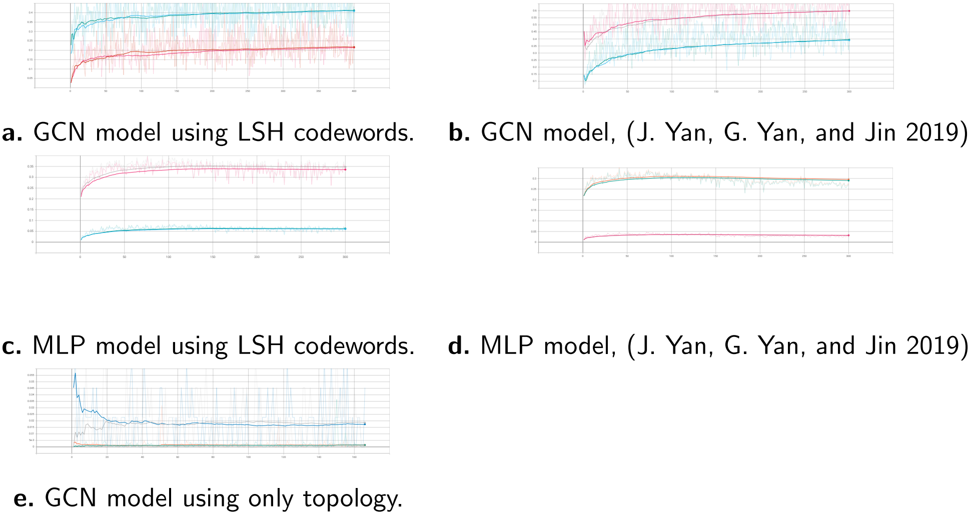
- number of hidden layers: 1 – 4,
- size of hidden GCN layers: 64, 128 or 256,
- dropout probability: 0.2, 0.4 or 0.5,
- dropout only after the last GCN layer or after each of them

Results



a. GCN model on CG, with LSH codes **b.** GCN model on CFG – literature

Figure 11: F_1-score of each class, plotted against the size of the family. The input of GCN is CG and CFG (J. Yan, G. Yan, and Jin 2019), respectively.



a. GCN model using LSH codewords.

b. GCN model, (J. Yan, G. Yan, and Jin 2019)

c. MLP model using LSH codewords.

d. MLP model, (J. Yan, G. Yan, and Jin 2019)

e. GCN model using only topology.

Figure 12: F_1-score of the GCN and MLP models using various features.

Evaluation metrics

- F_1 – harmonic mean of precision and recall
- *micro* (considers label imbalance) and *macro*-averaged F_1

Model	Micro-F_1	Macro-F_1
GCN model with LSH codes	0.381	0.189
GCN model with features of (J. Yan, G. Yan, and Jin 2019)	0.614	0.392
GCN model without node-level features	0.204	0.003
MLP model with LSH codes	0.313	0.050
MLP model with features of (J. Yan, G. Yan, and Jin 2019)	0.242	0.020

Table 1: F_1-scores of each model on the test dataset.

Conclusions

- task: malware → family classification
- malware feature: static call graph (node = function)
- feature extraction: LSH codewords of instruction n-grams
- models: GCN and MLP
- best model: call graph topology + node-level features

Future work

- subroutine feature selection
  - mnemonic histogram options
  - simple instruction statistics
- GCN models, parameter options
- other disassembler tools, e.g. Radare2

Special Thanks to Zalán, George, Ovidiu, Ciprian, my managers and colleagues!

Funded by:

- Bitdefender
- the Hungarian Academy of Sciences, via *Domus 86/18/2022/HTMT* project
- Babeș–Bolyai University of Cluj-Napoca

`https://attilamester.github.io/call-graph/`

References I

[1] Attila Mester and Zalán Bodó. "Validating static call graph-based malware signatures using community detection methods". In: *Proceedings of ESANN*. 2021.

[2] Thomas N. Kipf and Max Welling. *Semi-supervised classification with graph convolutional networks*. arXiv preprint arXiv:1609.02907. 2016.

[3] Minghui Cai et al. "Learning features from enhanced function call graphs for Android malware detection". In: *Neurocomputing* 423 (2021), pages 301–307.

References II

[4] Teenu S. John, Tony Thomas, and Sabu Emmanuel. "Graph convolutional networks for Android malware detection with system call graphs". In: *Third ISEA Conference on Security and Privacy (ISEA-ISAP)*. IEEE. 2020, pages 162–170.

[5] Angelo Schranko de Oliveira and Renato José Sassi. "Behavioral malware detection using deep graph convolutional neural networks". In: *International Journal of Computer Applications* 174 (2021).

References III

[6] Khanh-Huu-The Dam and Tayssir Touili. “Malware detection based on graph classification”. In: *Proceedings of the 3rd International Conference on Information Systems Security and Privacy, SCITEPRESS-Science and Technology Publications*. 2017.

[7] Jiwon Hong, Sanghyun Park, et al. “Classifying malwares for identification of author groups”. In: *Concurrency and Computation: Practice and Experience* 30.3 (2018), e4197.

[8] Anh Viet Phan et al. “DGCNN: A convolutional neural network over large-scale labeled graphs”. In: *Neural Networks* 108 (2018), pages 533–543.

References IV

[9] Jiwon Hong, Sung-Jun Park, et al. “Malware classification for identifying author groups: a graph-based approach”. In: *Proceedings of the Conference on Research in Adaptive and Convergent Systems*. 2019, pages 169–174.

[10] Haodi Jiang, Turki Turki, and Jason T.L. Wang. “DLGraph: Malware detection using deep learning and graph embedding”. In: *2018 17th IEEE international conference on machine learning and applications (ICMLA)*. IEEE. 2018, pages 1029–1033.

References V

[11] Jiaqi Yan, Guanhua Yan, and Dong Jin. "Classifying malware represented as control flow graphs using deep graph convolutional neural network". In: *49th annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE. 2019, pages 52–63.

[12] Mingdong Tang and Quan Qian. "Dynamic API call sequence visualisation for malware classification". In: *IET Information Security* 13.4 (2019), pages 367–377.