Towards a malware family classification model using static call graph instruction visualization

Attila Mester, Zalán Bodó, Vinod P., Mauro Conti

{*attila.mester*, *zalan.bodo*}@*ubbcluj.ro*
{*vinod.puthuvath*, *mauro.conti*}@*unipd.it*

Babeș–Bolyai University, Romania
University of Padua, Italy

20th November 2024

Content

Problem definition: attribution

- aim: classify family and/or actor(s) behind an attack (**attribution**)
- motivation: improved threat intelligence, faster incidence response, tailored defense strategies
- simple feature: the binary **PE** file
  - byte sequences; distribution; hashes
  - system calls e.g. *GetFileInformationByHandle*
  - executable's static call graph
- goal: **malware family/author detection** using **static call graph**

Malware analysis – the textbook approach

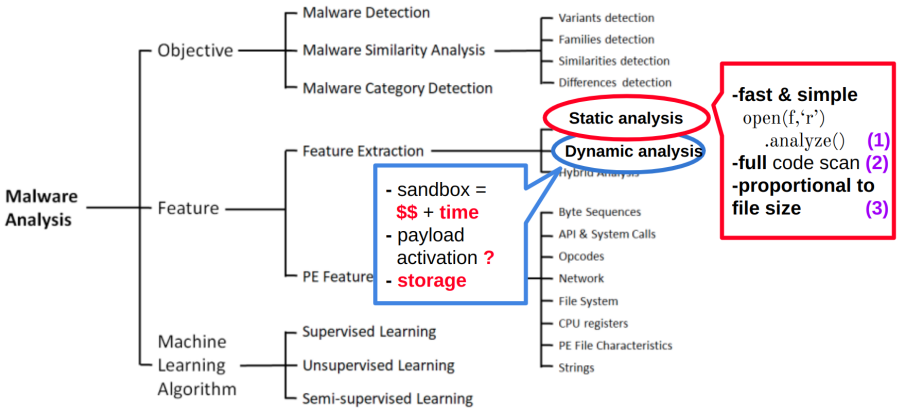


Figure 1: Malware analysis – taxonomy according to (Ucci, Aniello, and Baldoni 2018).

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

- static = obtained by disassembly (vs. dynamic = obtained by sandbox)
- function execution sequence = call graph
  - node = function (DLL / subroutine)
  - link = function call
- generation method: **Radare2** disassembler tool

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

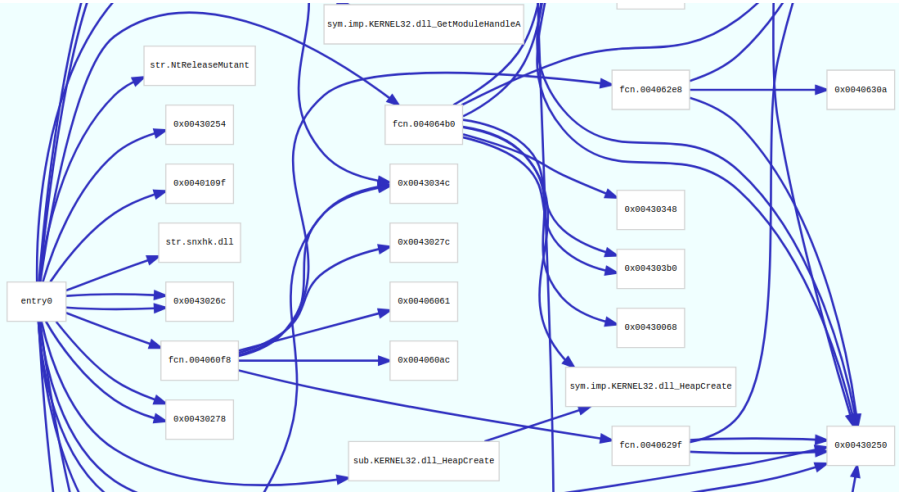
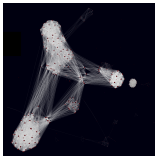


Figure 2: Sample call graph obtained by Radare2.

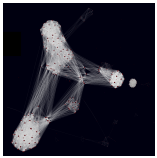
How to use this graph?



clustering signatures

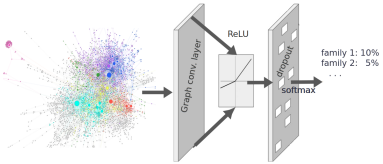
(Mester and Bodó 2021)

How to use this graph?



clustering signatures

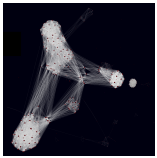
(Mester and Bodó 2021)



GCN

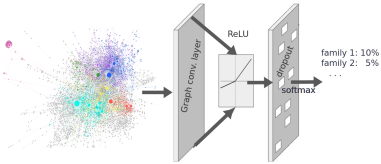
(Mester and Bodó 2022)

How to use this graph?



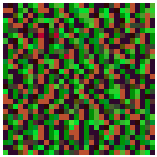
clustering signatures

(Mester and Bodó 2021)



GCN

(Mester and Bodó 2022)



CNN

current work

Motivation

- **helpful**: wide range of tools: CNN, visualization, patterns
  - DFS traversal – instruction patters – image patterns
- classification based on image derived from instructions
- aiming method stability: altering instructions is not trivial

Literature

- first work in 2011: **MalImg** dataset (Nataraj et al. 2011)
- working with grayscale images (Cui et al. 2018; Kalash et al. 2018)
- SimHash images: locality-sensitive binary hash generated by SimHash (Charikar 2002) for the opcode sequences, converted into binary image (Ni, Qian, and Zhang 2018)
- Markov images: visualizing transition probabilities between bytes, opcodes (Yuan et al. 2020; Deng et al. 2023)

Datasets

Table 1: Public malware datasets (Yang et al. 2021).

Dataset		Published	Binaries	Families	Samples(mal.)	EMBER	Disasm.	Image
MalImg	Nataraj et al. 2011	2011	❍	25	9458	❍ ✪	❍ ✪	● ✪
MS BIG	Ronen et al. 2018	2015	❍	9	10 868	❍	●	❍
EMBER	Anderson and Roth 2018	2018	❍	◗	800 000	●	❍	❍
UCSB	Aghakhani et al. 2020	2020	●	❍	232 415	❍	❍	❍
SOREL-20m	Harang and Rudd 2020	2020	●	❍	9 962 820	●	❍	❍
BODMAS	Yang et al. 2021	2021	●	581	57 293	●	❍ ✪	❍ ✪
Malflow	**(this)**	**2024**	❍	**47**	**18 756**	✪	✪	✪

Our contribution

- **original method**: generate RGB images for PE files based on disassembled call graph instruction flow
  - ignores noise from the binary
  - learns from images based solely on instruction information
- **training** CNN models: 0.88 micro F_1 on BODMAS with 57 families
- **publishing** a new dataset on Kaggle: Internal Bitdefender Dataset – IBD
- **publishing** packer information, static call graph, images and instruction data for BODMAS, MalIMG and IBD
- **comparing** our method to the state-of-the-art EMBER features

Generating the call graph & RGB image

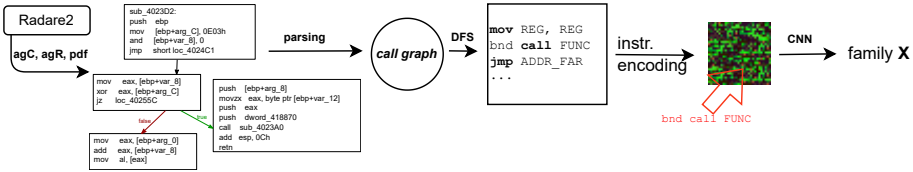


Figure 3: Pipeline of our proposed method.

- Radare2 (5.8.8 release): `agCd` and `agRd` – global call and reference graphs
- each function: `pdfj`
- DFS on functions
- **[bnd?] [prefix?] mnemonic [param1 [param2, ...]]**

Example encoding

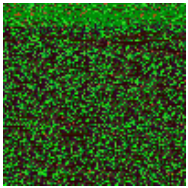
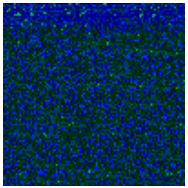
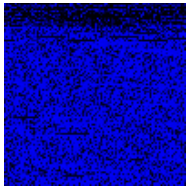
Simple hex dump	Encoding **FE**	Encoding **PE1**	Encoding **PE2**
			
BODMAS sample 53c4c900e03eb6e94c0fe18091591904			

Figure 4: Comparing different instruction encodings and simple hex dump image.

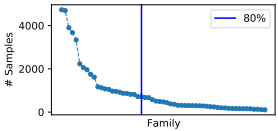
Training CNN models

- ResNet18/50, Resnet1D, MobileNetV3, GoogleNet, EfficientNet, DenseNet
- images formats: hex dump-based grayscale images, FE, PE1, PE2 encodings
- augmentation: generating 7000 metamorphic variants with pymetangine[1]

[1] https://hub.docker.com/r/attilamester/pymetangine

Training CNN models

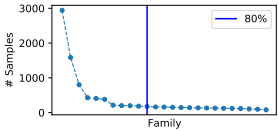
- ResNet18/50, Resnet1D, MobileNetV3, GoogleNet, EfficientNet, DenseNet
- images formats: hex dump-based grayscale images, FE, PE1, PE2 encodings
- augmentation: generating 7000 metamorphic variants with pymetangine[1]



BODMAS

57 families

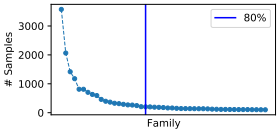
57k samples



MalImg

23 families

9k samples



IBD

47 families

18k samples

[1] https://hub.docker.com/r/attilamester/pymetangine

Model	Batch	Img.	Acc.	F_1 micro	F_1 macro
BODMAS, 57 families					
ResNet18	20	224 × 224	0.820	0.820	0.797
ResNet18	20	100 × 100	0.887	0.887	0.859
ResNet50	32	100 × 100	0.812	0.812	0.786
ResNet1d	20	100 × 100	0.784	0.784	0.727
MobileNetV3	32	30 × 30	0.790	0.790	0.733
GoogleNet	32	30 × 30	0.790	0.790	0.731
EfficientNet	32	30 × 30	0.786	0.786	0.737
DenseNet121	32	30 × 30	0.790	0.790	0.746
BODMAS, 57 families, hex dump images					
ResNet18	20	100 × 100	0.881	0.881	0.873
MalImg, 23 families					
ResNet18	20	100 × 100	0.722	0.744	0.737
ResNet18	20	30 × 30	0.761	0.761	0.749
IBD, 47 families					
ResNet18	20	100 × 100	0.872	0.872	0.784
ResNet18	32	30 × 30	0.849	0.849	0.776

Comparing our method with EMBER

- motivation: measure our model's performance compared to EMBER features (Anderson and Roth 2018)
- EMBER: 2381-long feature vector: bytes, histograms, entropy, section info, strings, import-export directory info, etc.
- **comparison methodology**: instruction mnemonic histogram vector for each file (2019-dimensional integer vector) vs. EMBER
- **compared models**: DecisionTree, RandomForest, LinearSVC, SGD, LogReg

Model	EMBER (1)		Mnem. hist. (2)		(1) + (2)	
	F_1 m.	F_1 M.	F_1 m.	F_1 M.	F_1 m.	F_1 M.
BODMAS, full dataset – 51 974 samples, 57 families						
DecisionTree	0.907	0.886	0.172	0.058	0.904	0.877
RandomForest	0.924	0.905	0.174	0.056	0.924	0.900
BODMAS, only non-packed samples – 33 807 samples, 44 families						
DecisionTree	0.930	0.917	0.879	0.831	0.932	0.922
RandomForest	0.945	0.938	0.884	0.839	0.944	0.939
MalImg, full dataset – 9138 samples, 23 families						
DecisionTree	0.990	0.975	0.753	0.841	0.998	0.978
RandomForest	0.996	0.990	0.786	0.832	0.993	0.982
IBD, full dataset – 18 756 samples, 47 families						
DecisionTree	0.923	0.862	0.897	0.838	0.923	0.870
RandomForest	0.961	0.936	0.924	0.877	0.963	0.940
IBD, only non-packed samples – 16 088 samples, 41 families						
DecisionTree	0.922	0.866	0.892	0.828	0.916	0.867
RandomForest	0.960	0.938	0.918	0.866	0.961	0.941

Observations

- packed samples distort the model trained on the mnemonic histogram vector, but do not have so heavy effect on binary file-based features, i.e. the EMBER
- EMBER's performance originates mainly from the byte- and byte entropy histogram information

Table 2: Comparison between the full EMBER vector and its subsets, on BODMAS

Model	[0..2381]		[0..255]		[256..511]		[0..511]	
	F_1 m.	F_1 M.	F_1 m.	F_1 M.	F_1 m.	F_1 M.	F_1 m.	F_1 M.
BODMAS, full dataset – 51 974 samples, 57 families								
DecisionTree	0.907	0.886	0.864	0.829	0.887	0.852	0.889	0.850
RandomForest	0.924	0.905	0.906	0.880	0.914	0.891	0.914	0.895

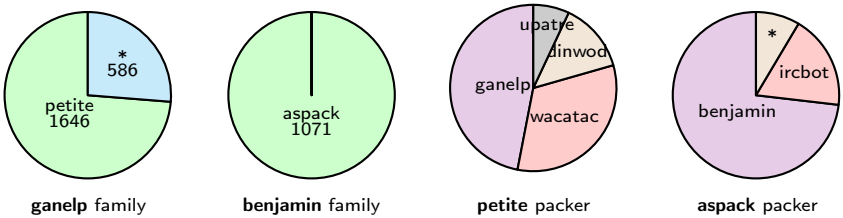
On correlation between packers and families

- BODMAS: 57k samples, 18k packed (33%)
- MalImg: 9.4k samples, 1.7k packed (18%)
- IBD: 18.7k samples, 2k packed (12%)

Table 3: Cramér's V association score between family, packed status, and packer tool.

	BODMAS	MalImg	IBD
Malware family – Packed	0.755	0.981	0.736
Malware family – Packer	0.547	0.590	0.525

Family – packer association



ganelp family **benjamin** family **petite** packer **aspack** packer

Figure 5: BODMAS: packer-family association, showing the number of samples.

Contribution summary

- **goal**: malware family classification
- **proposed model**: RGB images based on call graph instructions
- CNN architectures tested, but performance did not surpass the EMBER baseline's F1 score
- **discovered** strong correlation between families and packers
  - causing potential overfitting
  - packed samples may mislead ML models,
  - models may learn packer traits instead of malware behavior
- **published** on Kaggle: disassembled call graphs, images, packer info on BODMAS, MalImg and IBD

Future work

- combine instruction-based model with binary features to match/surpass EMBER baseline
- build explainable model on instruction images
- map malware families to instruction-level traits

Funded by:

- Babeș-Bolyai University, Romania
- HORIZON Europe Framework Programme – "OPTIMA - Organization sPecific Threat Intelligence Mining and sharing" (101063107)
- Bitdefender

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

References I

[1] Daniele Ucci, Leonardo Aniello, and Roberto Baldoni. "Survey of Machine Learning Techniques for Malware Analysis". In: (2018).

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

[3] Attila Mester and Zalán Bodó. "Malware Classification Based on Graph Convolutional Neural Networks and Static Call Graph Features". In: *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*. Springer. 2022, pages 528–539.

References II

[4] Lakshmanan Nataraj et al. "Malware images: visualization and automatic classification". In: *VizSec*. 2011, pages 1–7.

[5] Zhihua Cui et al. "Detection of malicious code variants based on deep learning". In: *IEEE Transactions on Industrial Informatics* 14.7 (2018), pages 3187–3196.

[6] Mahmoud Kalash et al. "Malware classification with deep convolutional neural networks". In: *NTMS*. IEEE. 2018, pages 1–5.

[7] M. S. Charikar. "Similarity estimation techniques from rounding algorithms". In: *STOC*. Montréal, Québec, Canada: ACM, 2002, pages 380–388.

References III

[8] Sang Ni, Quan Qian, and Rui Zhang. "Malware identification using visualization images and deep learning". In: *Computers & Security* 77 (2018), pages 871–885.

[9] Baoguo Yuan et al. "Byte-level malware classification based on markov images and deep learning". In: *Computers & Security* 92 (2020), page 101740.

[10] Huaxin Deng et al. "MCTVD: A malware classification method based on three-channel visualization and deep learning". In: *Computers & Security* 126 (2023), page 103084.

References IV

[11] Limin Yang et al. "BODMAS: An Open Dataset for Learning based Temporal Analysis of PE Malware". In: *SPW*. May 2021, pages 78–84.

[12] Royi Ronen et al. *Microsoft malware classification challenge*. arXiv preprint arXiv:1802.10135. 2018.

[13] Hyrum S. Anderson and Phil Roth. *EMBER: an open dataset for training static pe malware machine learning models*. arXiv preprint arXiv:1804.04637. 2018.

[14] Hojjat Aghakhani et al. "When Malware is Packin' Heat; Limits of Machine Learning Classifiers Based on Static Analysis Features". In: *NDSS*. 2020.

References V

[15] Richard Harang and Ethan M. Rudd. *SOREL-20M: A large scale benchmark dataset for malicious PE detection*. arXiv preprint arXiv:2012.07634. 2020.