Tackling obfuscated code through variant analysis and Graph Neural Networks

SOSYSEC seminar 21/03/25

Roxane Cohen <rcohen@quarkslab.com>
Robin David <rdavid@quarkslab.com>
Riccardo Mori <rmori@quarkslab.com>
Florian Yger <florian.yger@lamsade.dauphine.fr>
Fabrice Rossi <rossi@ceremade.dauphine.fr>

Whoami

PhD subject: Graph representation learning for reverse-engineering

Affiliation: Quarkslab & Université Paris-Dauphine

Started: November 2022

End: November 2025

Topics: obfuscation, binary analysis, Machine & Deep Learning, graphs, Graph Neural Networks

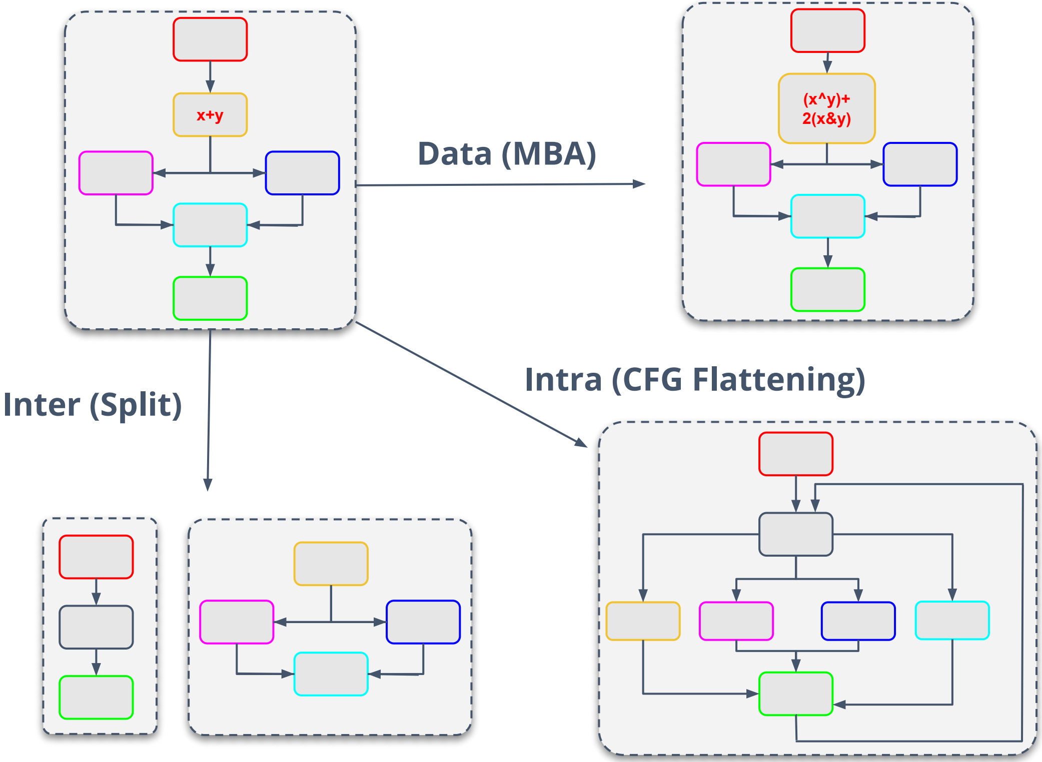
Agenda

1) Obfuscation introduction

2) Attacking obfuscation with binary variants

3) Locating and characterizing obfuscation

Obfuscation



Definition

All the techniques used to alter the syntactic properties of a program without modifying its semantics *(preserving soundness)*

Obfuscation types (static)

- Inter-procedural *(between functions)*
- Intra-procedural *(inside functions)*
- Data *(operations, constants, strings, etc.)*

Obfuscation analysis

Identification → Characterization → Attack (Deobfuscation) → "plain text" program

Determining if a binary / function has been obfuscated
⇒ Binary classification

Determining how it was obfuscated
⇒ Multi-class classification

Attack the obfuscation depending on its type
(academic papers start here)

Attacking obfuscation

Attacking head-on obfuscation ?

- May be costly (manually or with symbolic execution) [1, 2, 3]
- Where to look for ?

[1] You et al. **Deoptfuscator: Defeating Advanced Control-Flow Obfuscation Using Android Runtime (ART)**. IEEE Access, 2022

[2] Menguy, Grégoire, et al. **Search-Based Local Black-Box Deobfuscation: Understand, Improve and Mitigate**. CCS '21: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security, 2021

[3] Tofighi-Shirazi, R., Asavoae, I.M., Elbaz-Vincent, P., Le, T.H.: **Defeating opaque predicates statically through machine learning and binary analysis.** Proceedings of the 3rd ACM Workshop on Software Protection. 2019

Attacking obfuscation using binary variants

Another strategy: use program variants

Using multiple variants to transfer knowledge between binaries

➢ An attacker obtains a “plain” binary and one obfuscated newer variant

➢ An attacker gets its hands on two obfuscated variants *(of the same program)*

Core concept:

- Idea: Multiple binary variants can help to **draw correlations** between program content
- Advantage: Comparing binaries **without** having to deobfuscate them first.
- How: comparing different binaires, finding similarities and differences.
- Tips: multiple obfuscations alter specific program aspects **but not the overall program** *(because harder to put in practice)*

  ⇒ Use **resilient** binary features *(analyst knowledge)*

See ApkDiff: Matching Android App Versions Based on Class Structure, De Ghein and al., 2022

Binary diffing

Definition

Goal is **comparing** two *(or more)* binaries to analyze their differences. It usually done using functions with a `1-to-1` mapping computation.
(which can be problematic when functions are merged or split)

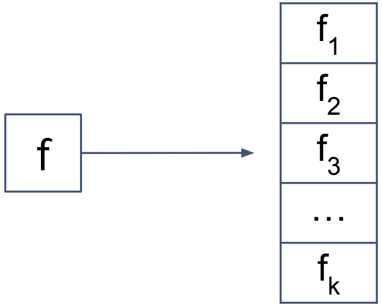
Use-cases:

- malware diffing *(analysing updates, or common components between two variants)*
- patch analysis / 1-day analysis *(understanding if patch is correct, or what is 1-day about)*
- anti-plagiarism
- statically linked libraries identification *(static binary against some libs)*
- symbol porting *(e.g: IDA annotations to a new version of a binary)*
- backdoor detection *(legitimate binary against a modified version)*
- cross-architecture diffing *(for symbol porting etc..)*

Diffing ain't Similarity

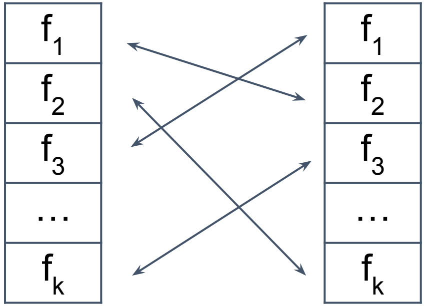
Similarity

Which function is the **most similar** to *f* among a pool of size *k* ?



Matching

What is the **best mapping** between functions of primary and secondary ?



primary *secondary*

Diffing = Similarity + Matching

(from similarity scores, create an assignment…)

Diffing solutions

- ✔ **decompiler**
- ✘ exporter
- ✔ precision
- ✘ recall

- ✔ **fast**
- ✘ no API
- ✔ now OSS
- ✔ disass agnostic

		Diaphora	Bindiff	Radiff2	Ghidriff
	Language	`Python`	`Java`	`C`	`Python`
Disassembler	IDA	✔	✔	✘	✘
	Ghidra	✘	✔	✘	✔
	Binja	✘	✔	✘	✘
	Radare2	✘	✘	✔	✘
	Exporter	`SQLite`	`Binexport`	`n/c`	`n/c`
	Scripting API	✔	✘	`n/c`	`n/c`
	Use decompiler	✔	✘	✘	`n/c`

Diffing obfuscated binaries requires modularity

QBindiff

Algorithm: Solve the Network Alignment Problem *(approximation)* using an optimization algorithm based on **message passing** (belief propagation) to arbitrate function similarity and call-graph topology.

Key Features:

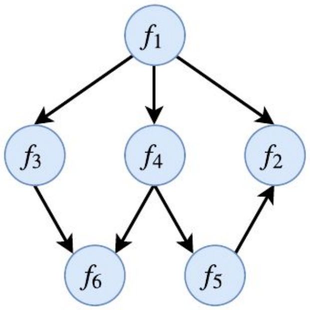
- Disassembler agnostic *(use exported representation)*
- Standalone program
- Python API *(to be used programmatically)*
- Two APIs:
  - High-level for binary diffing
  - Low-level for arbitrary diffing *(matrices as input)*
- Designed to be **modular**!

Blog ↗

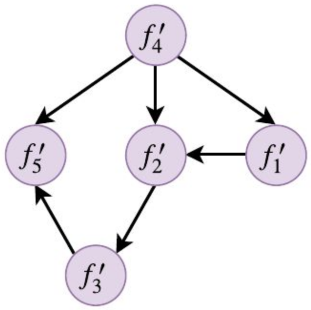
See **A modular differ to enhance binary diffing and graph alignment, SSTIC 2024**
Improving binary diffing through similarity and matching intricacies, CAID 2024

QBinDiff algorithm

Sample 1 (#M nodes)



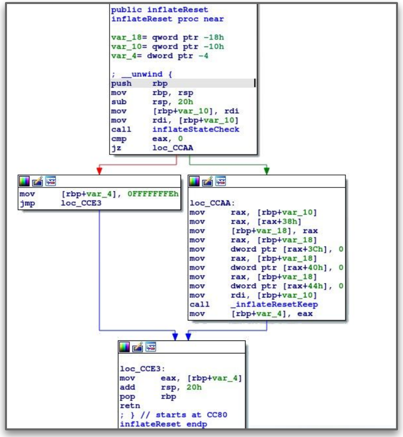
Sample 2 (#N nodes)

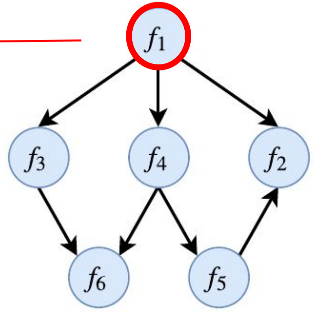


QBinDiff algorithm

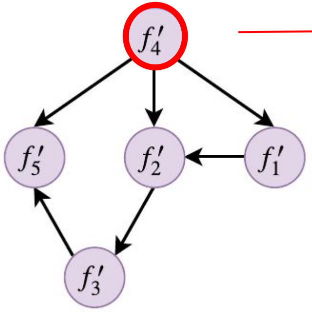
Sample 1 (#M nodes)

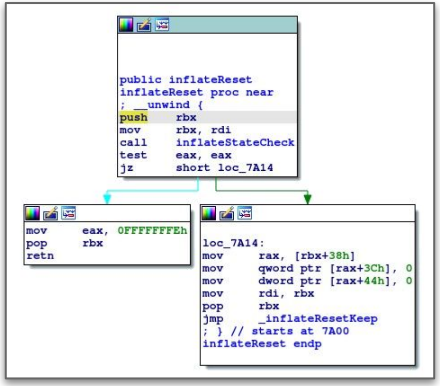
Sample 2 (#N nodes)





N





Features

(# nodes, # edges, cyclomatic complexity...)

4, 4, 2...

0 < Similarity < 1

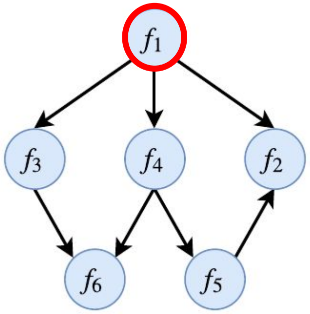
Features

(# nodes, # edges, cyclomatic complexity...)

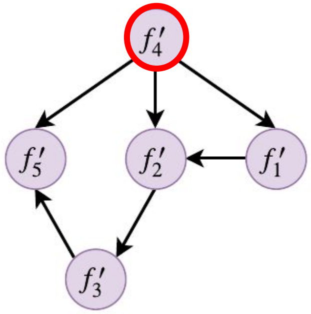
3, 2, 1...

QBinDiff algorithm

Sample 1 (#M nodes)



Sample 2 (#N nodes)



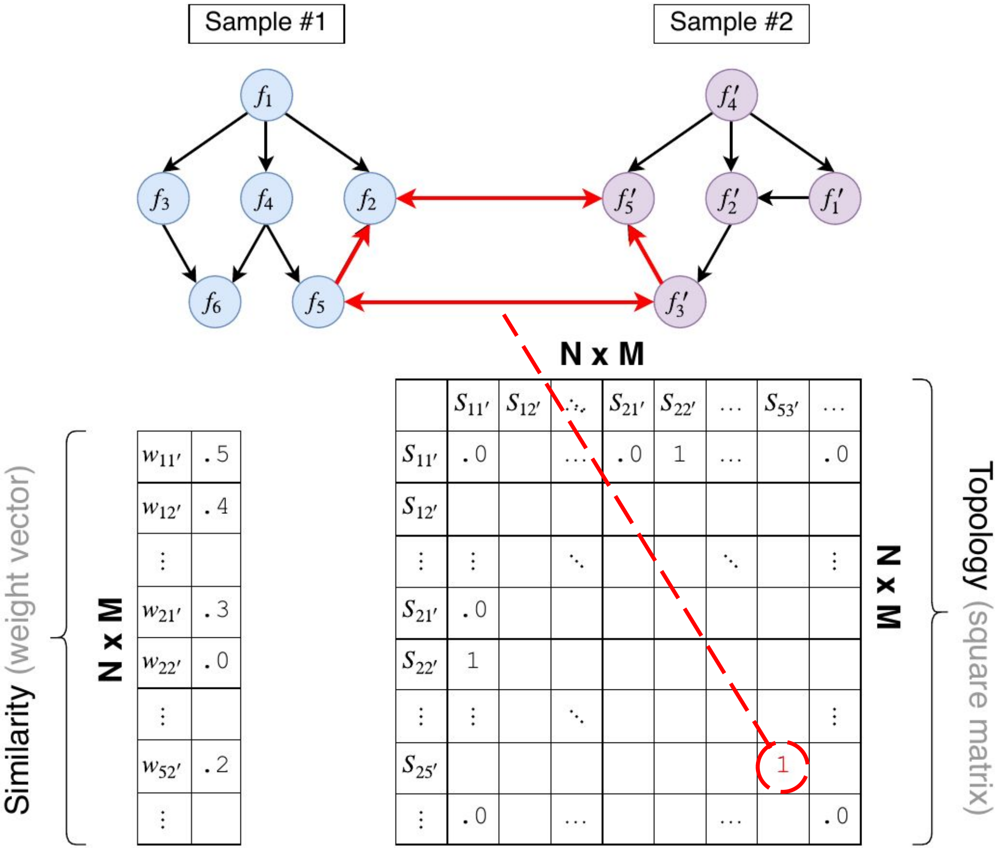
Similarity (weight matrix)

N

M

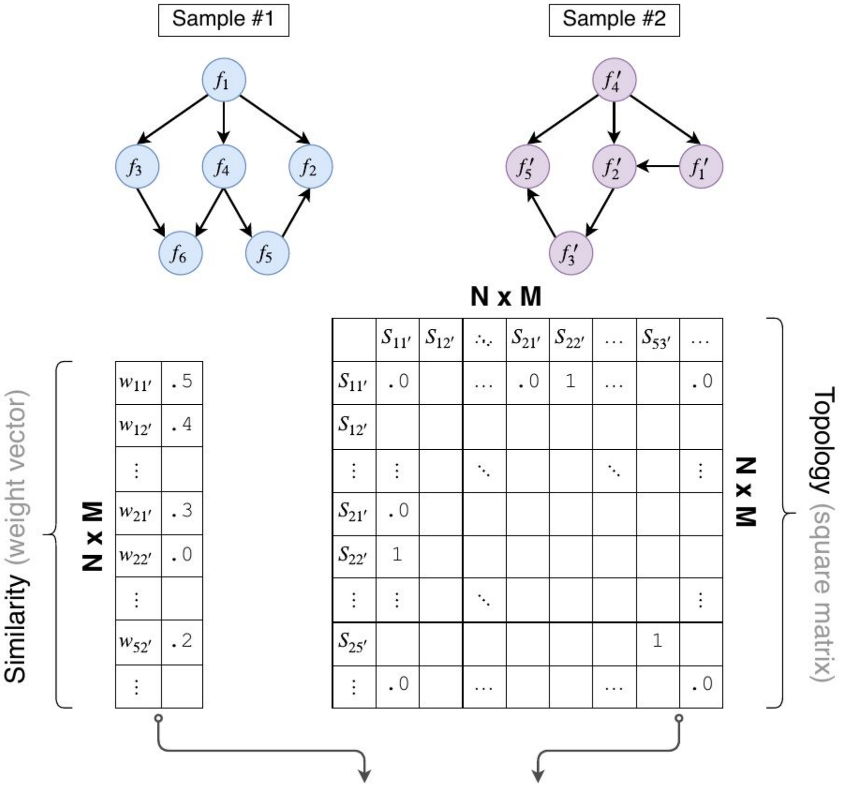
	f'_1	f'_2	$\cdots$	f'_4	f'_5
f_1	.5	.4		.6	.0
f_2	.3	.0		1	1
$\vdots$	$\vdots$		$\ddots$		
f_5	.6	.2			
f_6	.6				

QBinDiff algorithm



QBinDiff algorithm

Any data represented as **a similarity matrix** and **graph adjacency** can be aligned



$$\alpha x^{\mathsf{T}} w + \beta x^{\mathsf{T}} S x$$

Goal: **Arbitrate** between **function similarity** and **call-graph topology** to be more resilient if one of them is altered *(+ still use imported functions as anchors)*

Binary similarity solutions

Diffing = similarity + matching

- ➤ Use binary similarity approaches *(state-of-the-art but costly ~ deep learning)*
- ➤ Combine with a matching algorithm *(Hungarian algorithm)*

	GMN [1]	Asm2vec [2]	PalmTree [3]	jTrans [4]	SOG [5]
Language	Python	Python	Python	Python	Python / Java
Technique	GNN	word2vec	transformer	transformer	GNN

[1] Li and al.**Graph Matching Networks for Learning the Similarity of Graph Structured Objects.** 2019
[2] Ding and al. **Asm2Vec: Boosting Static Representation Robustness for Binary Clone Search against Code Obfuscation and Compiler Optimization.** 2019
[3] Li and al. **PalmTree: Learning an Assembly Language Model for Instruction Embedding.** 2021
[4] Wang and al. **jTrans: Jump-Aware Transformer for Binary Code Similarity.** 2022
[5] He and al. **Code is not Natural Language: Unlock the Power of Semantics-Oriented Graph Representation for Binary Code Similarity Detection.** 2023

Experiments

Current limitations

- No satisfactory dataset (not enough data, code snippet, only OLLVM…)
- Limited work on diffing in an obfuscated setting

Goal

- Creating a realistic and large obfuscated dataset
- Evaluating an obfuscation / obfuscator robustness according to its ability to prevent knowledge transfer between binaries using relevant metrics
- Showing and comparing differs ability to perform knowledge transfer with obfuscated binaries in two settings : **plain-vs-obfuscated** and **obfuscated-vs-obfuscated**

Dataset

	Passes	Pass type	zlib	lz4	minilua	sqlite	freetype
Tigress	Copy	Inter	✔	✔	✔	✔	✔
	Split	Inter	✔	✔	✔	✔	✔
	Merge	Inter	✔	✔	✗	✗	~
	CFF	Intra	✔	✔	✔	✔	✔
	Virtualize	Intra	✔	✔	~	~	✗
	Opaque	Intra	✔	✔	✔	✗	~
	EncodeArithmetic (Enc.A)	Data	✔	✔	✔	✔	✔
	EncodeLiterals (Enc.L)	Data	✔	✔	✔	✔	✔
	Mix	Intra & Data	✔	✔	✔	~	~
	Mix + Split	All	✔	✔	✔	~	~
OLLVM-14	CFF	Intra	✔	✔	✔	✔	✔
	Opaque	Intra	✔	✔	✔	✔	✔
	EncodeArithmetic (Enc.A)	Data	✔	✔	✔	✔	✔
	Mix	Intra & Data	✔	✔	✔	✔	✔

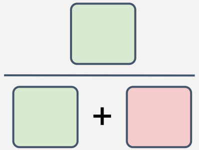
- Major constraint: using only projects that stand in a unique C file (Tigress)
- OLLVM is based on LLVM-4 (-O2 optimization removes a lot of it)
- Tigress may be capricious
- Can be found at: *https://github.com/quarkslab/diffing_obfuscation_dataset*

Diffing evaluation

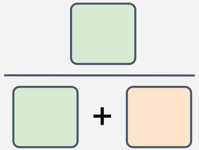
How can we compare the functions pairs that should be matched *(Ground-Truth)* **and the functions that are matched by a differ on stripped binaries ?**

True Positives	**False Positives**	**True Negative**	**False Negative**
good match correctly identified	wrong match identified	Not a match considered as-is	Good match **not** identified

Precision =



Recall =



$\Rightarrow$ **F1-score** = $\mathbf{2 \times \frac{P \times R}{P + R}}$

Recap

Attacker model : plain-vs-obfuscated

- A plain binary against an obfuscated variant
- Attack the obfuscation by diffing the two executables to recover an assignment
- Evaluating the diffing relevance (f1-score, the higher the better)
- **High f1-score = an attacker transfers knowledge between binaries and can weaken the obfuscation**

Plain-vs-obfuscated

	Attacker $\mathcal{A}$ (differ)	OLLVM-14				Tigress									
		Mix	CFF	Opaque	Enc.A	Mix	Mix + Split	Copy	Merge	Split	CFF	Virtualize	Opaque	Enc.A	Enc.L
10%	BinDiff	**0.98**	**0.99**	**0.98**	**0.99**	0.88	0.87	0.84	0.83	0.78	0.90	0.87	0.87	0.91	0.90
	Diaphora3	0.93	0.94	0.95	0.96	0.78	0.77	0.78	0.80	0.72	0.79	0.76	0.80	0.81	0.81
	GMN	0.86	0.87	0.88	0.92	0.53	0.52	0.57	0.54	0.43	0.53	0.45	0.55	0.55	0.59
	Asm2vec	0.64	0.65	0.68	0.72	0.46	0.42	0.46	0.55	0.41	0.45	0.42	0.49	0.51	0.53
	QBinDiff	0.94	0.97	0.97	0.98	**0.90**	**0.89**	**0.91**	0.88	0.86	0.92	0.87	**0.90**	**0.95**	**0.94**
	QBinDiff_s	-	0.97	0.97	0.95	-	-	0.89	**0.90**	**0.87**	**0.94**	**0.94**	0.86	0.88	0.89
50%	BinDiff	**0.94**	**0.98**	0.95	**0.99**	**0.75**	0.63	0.65	0.68	0.48	0.85	0.80	0.75	0.90	0.90
	Diaphora3	0.79	0.86	0.87	0.96	0.62	0.55	0.72	0.68	0.45	0.66	0.50	0.74	0.79	0.80
	GMN	0.59	0.63	0.67	0.81	0.32	0.30	0.47	0.38	0.23	0.31	0.28	0.40	0.47	0.58
	Asm2vec	0.40	0.46	0.54	0.72	0.26	0.23	0.34	0.39	0.24	0.26	0.18	0.40	0.48	0.48
	QBinDiff	0.86	0.96	0.94	0.98	0.73	**0.69**	**0.77**	0.70	0.58	0.82	0.74	**0.81**	**0.94**	**0.94**
	QBinDiff_s	-	0.97	**0.97**	0.93	-	-	0.76	**0.72**	**0.60**	**0.93**	**0.89**	0.75	0.88	0.89
100%	BinDiff	0.77	0.96	0.83	**0.99**	0.37	0.17	0.42	0.41	0.21	0.73	0.67	0.53	0.89	0.86
	Diaphora3	0.51	0.68	0.74	0.96	0.28	0.17	**0.67**	0.37	0.26	0.52	0.10	0.66	0.75	0.78
	GMN	0.28	0.34	0.42	0.69	0.08	0.08	0.40	0.20	0.09	0.07	0.11	0.24	0.37	0.58
	Asm2vec	0.19	0.29	0.40	0.72	0.08	0.08	0.22	0.14	0.12	0.08	0.02	0.33	0.39	0.53
	QBinDiff	**0.78**	0.93	0.91	0.98	**0.44**	**0.34**	0.65	0.42	0.37	0.72	0.59	**0.70**	**0.93**	**0.93**
	QBinDiff_s	-	**0.97**	**0.96**	0.93	-	-	0.64	**0.43**	**0.40**	**0.91**	**0.84**	0.64	0.88	0.87

f1-score comparison in a plain-obfuscated setting in -O0

(the higher, the better the differ)

Same trends, with lower scores, in the **obfuscated-vs-obfuscated**

Plain-vs-obfuscated

	Attacker $\mathcal{A}$ (differ)	OLLVM-14				Tigress									
		Mix	CFF	Opaque	Enc.A	Mix	Mix + Split	Copy	Merge	Split	CFF	Virtualize	Opaque	Enc.A	Enc.L
10%	BinDiff	**0.98**	**0.99**	**0.98**	**0.99**	0.88	0.87	0.84	0.83	0.78	0.90	0.87	0.87	0.91	0.90
	Diaphora3	0.93	0.94	0.95	0.96	0.78	0.77	0.78	0.80	0.72	0.79	0.76	0.80	0.81	0.81
	GMN	0.86	0.87	0.88	0.92	0.53	0.52	0.57	0.54	0.43	0.53	0.45	0.55	0.55	0.59
	Asm2vec	0.64	0.65	0.68	0.72	0.46	0.42	0.46	0.55	0.41	0.45	0.42	0.49	0.51	0.53
	QBinDiff	0.94	0.97	0.97	0.98	**0.90**	**0.89**	**0.91**	0.88	0.86	0.92	0.87	**0.90**	**0.95**	**0.94**
	QBinDiff_s	-	0.97	0.97	0.95	-	-	0.89	**0.90**	**0.87**	**0.94**	**0.94**	0.86	0.88	0.89
50%	BinDiff	**0.94**	**0.98**	0.95	**0.99**	**0.75**	0.63	0.65	0.68	0.48	0.85	0.80	0.75	0.90	0.90
	Diaphora3	0.79	0.86	0.87	0.96	0.62	0.55	0.72	0.68	0.45	0.66	0.50	0.74	0.79	0.80
	GMN	0.59	0.63	0.67	0.81	0.32	0.30	0.47	0.38	0.23	0.31	0.28	0.40	0.47	0.58
	Asm2vec	0.40	0.46	0.54	0.72	0.26	0.23	0.34	0.39	0.24	0.26	0.18	0.40	0.48	0.48
	QBinDiff	0.86	0.96	0.94	0.98	0.73	**0.69**	**0.77**	0.70	0.58	0.82	0.74	**0.81**	**0.94**	**0.94**
	QBinDiff_s	-	0.97	**0.97**	0.93	-	-	0.76	**0.72**	**0.60**	**0.93**	**0.89**	0.75	0.88	0.89
100%	BinDiff	0.77	0.96	0.83	**0.99**	0.37	0.17	0.42	0.41	0.21	0.73	0.67	0.53	0.89	0.86
	Diaphora3	0.51	0.68	0.74	0.96	0.28	0.17	**0.67**	0.37	0.26	0.52	0.10	0.66	0.75	0.78
	GMN	0.28	0.34	0.42	0.69	0.08	0.08	0.40	0.20	0.09	0.07	0.11	0.24	0.37	0.58
	Asm2vec	0.19	0.29	0.40	0.72	0.08	0.08	0.22	0.14	0.12	0.08	0.02	0.33	0.39	0.53
	QBinDiff	**0.78**	0.93	0.91	0.98	**0.44**	**0.34**	0.65	0.42	0.37	0.72	0.59	**0.70**	**0.93**	**0.93**
	QBinDiff_s	-	**0.97**	**0.96**	0.93	-	-	0.64	**0.43**	**0.40**	**0.91**	**0.84**	0.64	0.88	0.87

OLLVM obfuscations are easily mitigated

f1-score comparison in a plain-obfuscated setting in -O0

(the higher, the better the differ)

Plain-vs-obfuscated

	Attacker $\mathcal{A}$ (differ)	OLLVM-14				Tigress									
		Mix	CFF	Opaque	Enc.A	Mix	Mix + Split	Copy	Merge	Split	CFF	Virtualize	Opaque	Enc.A	Enc.L
10%	BinDiff	**0.98**	**0.99**	**0.98**	**0.99**	0.88	0.87	0.84	0.83	0.78	0.90	0.87	0.87	0.91	0.90
	Diaphora3	0.93	0.94	0.95	0.96	0.78	0.77	0.78	0.80	0.72	0.79	0.76	0.80	0.81	0.81
	GMN	0.86	0.87	0.88	0.92	0.53	0.52	0.57	0.54	0.43	0.53	0.45	0.55	0.55	0.59
	Asm2vec	0.64	0.65	0.68	0.72	0.46	0.42	0.46	0.55	0.41	0.45	0.42	0.49	0.51	0.53
	QBinDiff	0.94	0.97	0.97	0.98	**0.90**	**0.89**	**0.91**	0.88	0.86	0.92	0.87	**0.90**	**0.95**	**0.94**
	QBinDiff_s	-	0.97	0.97	0.95	-	-	0.89	**0.90**	**0.87**	**0.94**	**0.94**	0.86	0.88	0.89
50%	BinDiff	**0.94**	**0.98**	0.95	**0.99**	**0.75**	0.63	0.65	0.68	0.48	0.85	0.80	0.75	0.90	0.90
	Diaphora3	0.79	0.86	0.87	0.96	0.62	0.55	0.72	0.68	0.45	0.66	0.50	0.74	0.79	0.80
	GMN	0.59	0.63	0.67	0.81	0.32	0.30	0.47	0.38	0.23	0.31	0.28	0.40	0.47	0.58
	Asm2vec	0.40	0.46	0.54	0.72	0.26	0.23	0.34	0.39	0.24	0.26	0.18	0.40	0.48	0.48
	QBinDiff	0.86	0.96	0.94	0.98	0.73	**0.69**	**0.77**	0.70	0.58	0.82	0.74	**0.81**	**0.94**	**0.94**
	QBinDiff_s	-	0.97	**0.97**	0.93	-	-	0.76	**0.72**	**0.60**	**0.93**	**0.89**	0.75	0.88	0.89
100%	BinDiff	0.77	0.96	0.83	**0.99**	0.37	0.17	0.42	0.41	0.21	0.73	0.67	0.53	0.89	0.86
	Diaphora3	0.51	0.68	0.74	0.96	0.28	0.17	**0.67**	0.37	0.26	0.52	0.10	0.66	0.75	0.78
	GMN	0.28	0.34	0.42	0.69	0.08	0.08	0.40	0.20	0.09	0.07	0.11	0.24	0.37	0.58
	Asm2vec	0.19	0.29	0.40	0.72	0.08	0.08	0.22	0.14	0.12	0.08	0.02	0.33	0.39	0.53
	QBinDiff	**0.78**	0.93	0.91	0.98	**0.44**	**0.34**	0.65	0.42	0.37	0.72	0.59	**0.70**	**0.93**	**0.93**
	QBinDiff_s	-	**0.97**	**0.96**	0.93	-	-	0.64	**0.43**	**0.40**	**0.91**	**0.84**	0.64	0.88	0.87

QBinDiff *(and Bindiff)* are the best differs

f1-score comparison in a plain-obfuscated setting in -O0

(the higher, the better the differ)

Plain-vs-obfuscated

	Attacker $\mathcal{A}$ (differ)	OLLVM-14				Tigress									
		Mix	CFF	Opaque	Enc.A	Mix	Mix + Split	Copy	Merge	Split	CFF	Virtualize	Opaque	Enc.A	Enc.L
10%	BinDiff	**0.98**	**0.99**	**0.98**	**0.99**	0.88	0.87	0.84	0.83	0.78	0.90	0.87	0.87	0.91	0.90
	Diaphora3	0.93	0.94	0.95	0.96	0.78	0.77	0.78	0.80	0.72	0.79	0.76	0.80	0.81	0.81
	GMN	0.86	0.87	0.88	0.92	0.53	0.52	0.57	0.54	0.43	0.53	0.45	0.55	0.55	0.59
	Asm2vec	0.64	0.65	0.68	0.72	0.46	0.42	0.46	0.55	0.41	0.45	0.42	0.49	0.51	0.53
	QBinDiff	0.94	0.97	0.97	0.98	**0.90**	**0.89**	**0.91**	0.88	0.86	0.92	0.87	**0.90**	**0.95**	**0.94**
	QBinDiff_s	-	0.97	0.97	0.95	-	-	0.89	**0.90**	**0.87**	**0.94**	**0.94**	0.86	0.88	0.89
50%	BinDiff	**0.94**	**0.98**	0.95	**0.99**	**0.75**	0.63	0.65	0.68	0.48	0.85	0.80	0.75	0.90	0.90
	Diaphora3	0.79	0.86	0.87	0.96	0.62	0.55	0.72	0.68	0.45	0.66	0.50	0.74	0.79	0.80
	GMN	0.59	0.63	0.67	0.81	0.32	0.30	0.47	0.38	0.23	0.31	0.28	0.40	0.47	0.58
	Asm2vec	0.40	0.46	0.54	0.72	0.26	0.23	0.34	0.39	0.24	0.26	0.18	0.40	0.48	0.48
	QBinDiff	0.86	0.96	0.94	0.98	0.73	**0.69**	**0.77**	0.70	0.58	0.82	0.74	**0.81**	**0.94**	**0.94**
	QBinDiff_s	-	0.97	**0.97**	0.93	-	-	0.76	**0.72**	**0.60**	**0.93**	**0.89**	0.75	0.88	0.89
100%	BinDiff	0.77	0.96	0.83	**0.99**	0.37	0.17	0.42	0.41	0.21	0.73	0.67	0.53	0.89	0.86
	Diaphora3	0.51	0.68	0.74	0.96	0.28	0.17	**0.67**	0.37	0.26	0.52	0.10	0.66	0.75	0.78
	GMN	0.28	0.34	0.42	0.69	0.08	0.08	0.40	0.20	0.09	0.07	0.11	0.24	0.37	0.58
	Asm2vec	0.19	0.29	0.40	0.72	0.08	0.08	0.22	0.14	0.12	0.08	0.02	0.33	0.39	0.53
	QBinDiff	**0.78**	0.93	0.91	0.98	**0.44**	**0.34**	0.65	0.42	0.37	0.72	0.59	**0.70**	**0.93**	**0.93**
	QBinDiff_s	-	**0.97**	**0.96**	0.93	-	-	0.64	**0.43**	**0.40**	**0.91**	**0.84**	0.64	0.88	0.87

Tigress obfuscation, especially inter-procedural, offers more protection

f1-score comparison in a plain-obfuscated setting in -O0

(the higher, the better the differ)

Plain-vs-obfuscated

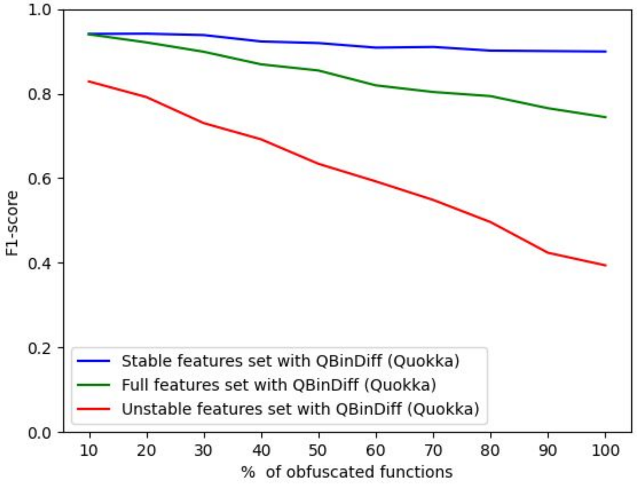
Attacker $\mathcal{A}$ (differ)		OLLVM-14				Tigress									
		Mix	CFF	Opaque	Enc.A	Mix	Mix + Split	Copy	Merge	Split	CFF	Virtualize	Opaque	Enc.A	Enc.L
10%	BinDiff	**0.98**	**0.99**	**0.98**	**0.99**	0.88	0.87	0.84	0.83	0.78	0.90	0.87	0.87	0.91	0.90
	Diaphora3	0.93	0.94	0.95	0.96	0.78	0.77	0.78	0.80	0.72	0.79	0.76	0.80	0.81	0.81
	GMN	0.86	0.87	0.88	0.92	0.53	0.52	0.57	0.54	0.43	0.53	0.45	0.55	0.55	0.59
	Asm2vec	0.64	0.65	0.68	0.72	0.46	0.42	0.46	0.55	0.41	0.45	0.42	0.49	0.51	0.53
	QBinDiff	0.94	0.97	0.97	0.98	**0.90**	**0.89**	**0.91**	0.88	0.86	0.92	0.87	**0.90**	**0.95**	**0.94**
	QBinDiff_s	-	0.97	0.97	0.95	-	-	0.89	**0.90**	**0.87**	**0.94**	**0.94**	0.86	0.88	0.89
50%	BinDiff	**0.94**	**0.98**	0.95	**0.99**	**0.75**	0.63	0.65	0.68	0.48	0.85	0.80	0.75	0.90	0.90
	Diaphora3	0.79	0.86	0.87	0.96	0.62	0.55	0.72	0.68	0.45	0.66	0.50	0.74	0.79	0.80
	GMN	0.59	0.63	0.67	0.81	0.32	0.30	0.47	0.38	0.23	0.31	0.28	0.40	0.47	0.58
	Asm2vec	0.40	0.46	0.54	0.72	0.26	0.23	0.34	0.39	0.24	0.26	0.18	0.40	0.48	0.48
	QBinDiff	0.86	0.96	0.94	0.98	0.73	**0.69**	**0.77**	0.70	0.58	0.82	0.74	**0.81**	**0.94**	**0.94**
	QBinDiff_s	-	0.97	**0.97**	0.93	-	-	0.76	**0.72**	**0.60**	**0.93**	**0.89**	0.75	0.88	0.89
100%	BinDiff	0.77	0.96	0.83	**0.99**	0.37	0.17	0.42	0.41	0.21	0.73	0.67	0.53	0.89	0.86
	Diaphora3	0.51	0.68	0.74	0.96	0.28	0.17	**0.67**	0.37	0.26	0.52	0.10	0.66	0.75	0.78
	GMN	0.28	0.34	0.42	0.69	0.08	0.08	0.40	0.20	0.09	0.07	0.11	0.24	0.37	0.58
	Asm2vec	0.19	0.29	0.40	0.72	0.08	0.08	0.22	0.14	0.12	0.08	0.02	0.33	0.39	0.53
	QBinDiff	**0.78**	0.93	0.91	0.98	**0.44**	**0.34**	0.65	0.42	0.37	0.72	0.59	**0.70**	**0.93**	**0.93**
	QBinDiff_s	-	**0.97**	**0.96**	0.93	-	-	0.64	**0.43**	**0.40**	**0.91**	**0.84**	0.64	0.88	0.87

Binary similarity tools (+ matching) show limited performances

f1-score comparison in a plain-obfuscated setting in -O0

(the higher, the better the differ)

Feature impact on diffing



QBinDiff feature impact : stable, full and unstable features

(Control-Flow Graph Flattening f1-score evolution)

Characterize the obfuscation => adapt the features for better diffing results

What if we cannot find multiple variants?

Quarkslab

Last chance: deobfuscation

Deobfuscation

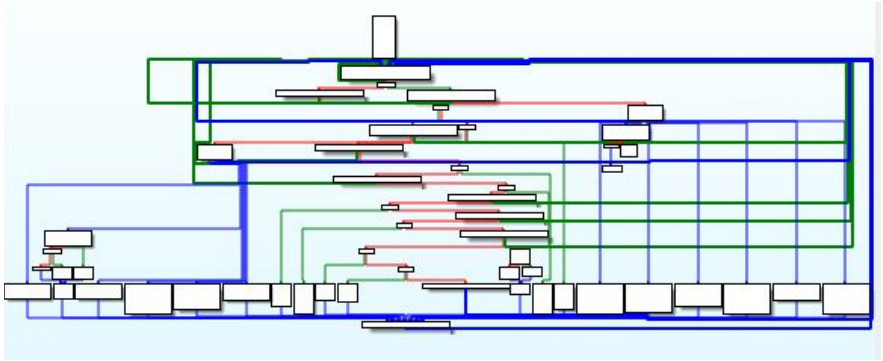
- ➢ Locating obfuscation inside a binary (program / function level)
- ➢ Characterizing it (MBA, CFF ?)
- ➢ Stealth property of an obfuscation

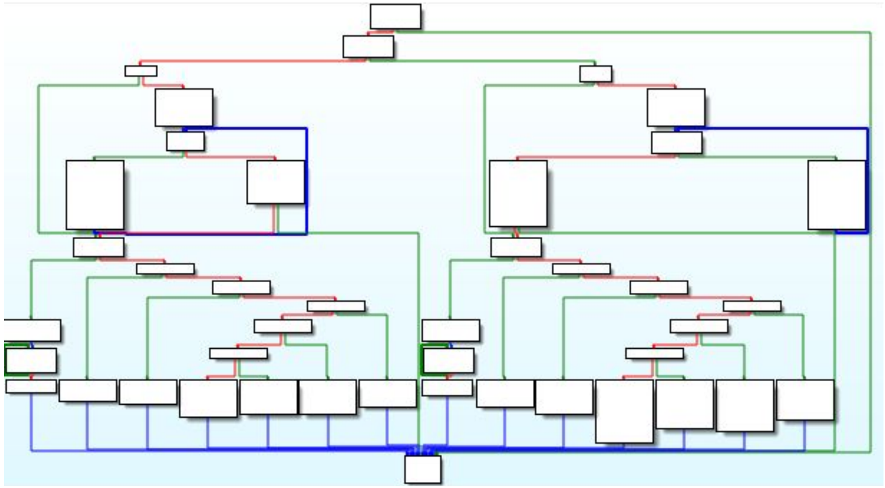
Obfuscation detection:

1) Identifying obfuscation at the function level *(time-saver for deobfuscation)*
2) Characterizing the applied obfuscation
3) Launch deobfuscation algorithms *(against MBA, OpaquePredicates…)*

See Identifying Obfuscated Code through Graph-Based Semantic Analysis of Binary Code, ComplexNetworks 2024

How can we recognize an obfuscated function ?





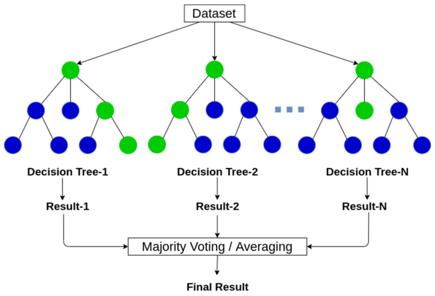
Which function is obfuscated ? How it is obfuscated ?

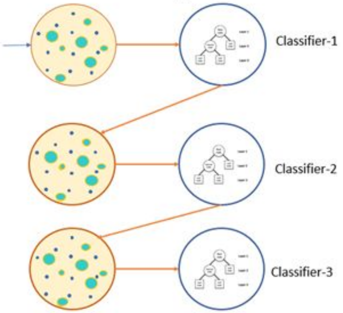
Use graph-based Machine Learning

Graph-based ML

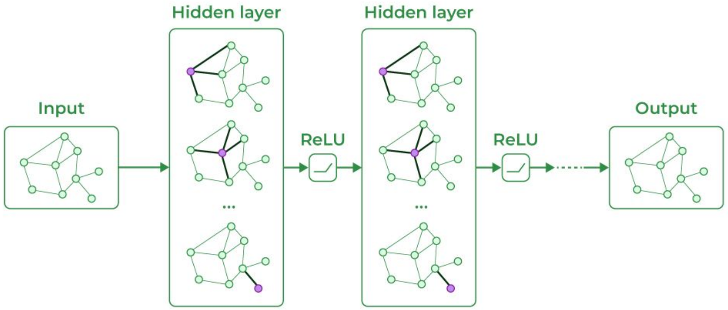
- ➢ Functions are naturally represented by Control-Flow Graph (CFG)
- ➢ CFG are attributed graphs containing part of the function semantics
- ➢ Combining CFG structure and attributes to infer obfuscation location / type

Elementary ML





Graph Neural Networks



Graph Neural Networks

Definition

- Neural networks adapted to non-euclidean data
- Invariant to permutation
- Iteratively update initial node feature given the node neighborhood

$$a_v^{(k)} = AGGREGATE^{(k)}\left(\{h_u^{(k-1)} : u \in \mathcal{N}(v)\}\right)$$

$$h_v^{(k)} = COMBINE^{(k)}\left(h_v^{(k-1)}, a_v^{(k)}\right)$$

$$h_G = READOUT(\{h_v^{(K)} | v \in G\})$$

Xu et al. **How powerful are graph neural networks?** International Conference on Learning Representations (2019)

Graph Neural Networks

GCN	$\mathbf{x}'_i = \mathbf{\Theta}^\top \sum_{j \in \mathcal{N}(i) \cup \{i\}} \frac{e_{j,i}}{\sqrt{\hat{d}_j \hat{d}_i}} \mathbf{x}_j$	$\hat{d}_i = 1 + \sum_{j \in \mathcal{N}(i)} e_{j,i}$
SAGE	$\mathbf{x}'_i = \mathbf{W}_1 \mathbf{x}_i + \mathbf{W}_2 \cdot \mathrm{mean}_{j \in \mathcal{N}(i)} \mathbf{x}_j$	
GIN	$\mathbf{x}'_i = h_{\mathbf{\Theta}} \left((1 + \epsilon) \cdot \mathbf{x}_i + \sum_{j \in \mathcal{N}(i)} \mathbf{x}_j \right)$	
GAT	$\mathbf{x}'_i = \sum_{j \in \mathcal{N}(i) \cup \{i\}} \alpha_{i,j} \mathbf{\Theta}_t \mathbf{x}_j,$	$\alpha_{i,j} = \frac{\exp\left(\mathrm{LeakyReLU}\left(\mathbf{a}_s^\top \mathbf{\Theta}_s \mathbf{x}_i + \mathbf{a}_t^\top \mathbf{\Theta}_t \mathbf{x}_j\right)\right)}{\sum_{k \in \mathcal{N}(i) \cup \{i\}} \exp\left(\mathrm{LeakyReLU}\left(\mathbf{a}_s^\top \mathbf{\Theta}_s \mathbf{x}_i + \mathbf{a}_t^\top \mathbf{\Theta}_t \mathbf{x}_k\right)\right)}$

Comparison of GNN convolution.
GIN offers the best theoretical guarantees *(as powerful as the 1-WL test)*

Experiments

Current limitations

- ➢ Little or no study on GNN potential for obfuscation detection
- ➢ Limited obfuscation set available

Goal

- ➢ Use the previous dataset (with lot of obfuscation) and split it in 2 (Dataset-1 & Dataset-2) *(easier to harder)*
- ➢ Evaluating 1) Graph representation 2) Features 3) Models 4) Data in the context of obfuscation detection
- ➢ Binary classification vs multi-class classification (11 classes !)

Dataset

Dataset

- **projects**: zlib, lz4, minilua, sqlite, freetype
- **obfuscator**: OLLVM, Tigress
- **obfuscations**:
  - intra *(CFF, Opaque, Virtualization)*
  - inter *(Split, Merge, Copy)*
  - data *(EncodeArithmetic, EncodeLiterals)*
  - mix1 *(intra & data)*
  - mix2 (intra & inter & data)
- High class unbalance

Dataset-1

- Split per function
- Randomly assign functions *(and their obfuscations variants)* to a set *(training, validation, testing)*
- "Easy" setup as two functions belonging to the same program may be close

Dataset-2

- Split per binary
- Assign all the functions of zlib/lz4/minilua *(and their obfuscations variants)* to the training set, sqlite/freetype to the validation/test set
- "Harder" setup: it must generalize to completely unseen binaries

Elementary ML

Reminder

- 1 function = 1 CFG = 1 graph
- Elementary ML : **1 graph = 1 feature vector** *(1, d)*

CFG
(functions)

Features

Graph-based features
Extract various graph features
(#nodes, #edges, cyclomatic complexity, density)

Assembly mnemonic TF-IDF
Use the TF-IDF feature of the 128-most frequent mnemonics inside the function assembly

ML models

Random Forest → classification results

Gradient Boosting → classification results

Graph Neural Networks

Reminder

- 1 function = 1 CFG = 1 graph
- GNN : **1 graph = 1 feature vector per node** !

Features

- Identity feature *(vector filled with 1's)*
- Coarse assembly feature : counting the number of assembly classes *(floating-point mnemonics, data-transfer mnemonics…)*
- "Semantic" assembly feature : counting the assembly mnemonics *(mov, lea, …)*
- "Semantic" Pcode feature : counting the Pcode mnemonics *(BRANCH, STORE,…)*
- Transformer-based embedding : PalmTree *("Palmtree: learning an assembly language model for instruction embedding", Li and al., 2021)*

Pcode is an intermediary representation that translates an assembly instruction into an architecture-agnostic language

⇓

Advantage

Only 72 Pcode mnemonics !

(More than 1800 for x86 assembly)

Evaluation

True Positives

False Positives

True Negatives

False Negatives

Recall = $\frac{\square}{\square + \square}$ $\Rightarrow$ balanced accuracy = $\frac{\text{Recall(c0)} + \ldots + \text{Recall(cn)}}{n}$

Binary classification

Graph	Features	Algorithm	Balanced accuracy	
			Dataset-1	*Dataset-2*
CFG	Graph features & assembly (Dim: **#23**)	RandomForest	0.702	0.60
		GradientBoosting	0.725	0.649
	TF-IDF on assembly mnemonics (Dim: **#128**)	RandomForest	0.76	0.607
		GradientBoosting	0.80	0.683
	Identity (Dim: **#1**)	GCN	0.634	0.608
		Sage	0.615	0.574
		GIN	0.603	0.531
		GAT	0.589	0.539
		UNet	0.616	0.555
	Counting mnemonic classes (Dim: **#27**)	GCN	0.659	0.658
		Sage	0.694	0.66
		GIN	0.701	0.673
		GAT	0.655	0.667
		UNet	0.66	0.654
	Semantic & counting PCode mnemonics (Dim: **#77**)	GCN	0.761	0.733
		Sage	0.782	0.70
		GIN	0.775	0.69
		GAT	0.77	0.73
		UNet	0.753	0.724
	Semantic & counting assembly mnemonics (Dim: **#1839**)	GCN	0.792	0.758
		Sage	0.802	0.727
		GIN	0.793	0.727
		GAT	0.797	0.729
		UNet	0.785	0.701
	PalmTree on assembly code (Dim : **#128**)	GCN	0.763	-
		Sage	0.718	-
		GIN	0.715	-
		GAT	0.773	-
		UNet	0.768	-

Binary classification

Stable baselines, with better scores using GB and mnemonic TF-IDF

Dataset-1 has higher score than Dataset-2

Graph	Features	Algorithm	Balanced accuracy	
			Dataset-1	*Dataset-2*
CFG	Graph features & assembly (Dim: **#23**)	RandomForest	0.702	0.60
		GradientBoosting	0.725	0.649
	TF-IDF on assembly mnemonics (Dim: **#128**)	RandomForest	0.76	0.607
		GradientBoosting	0.80	0.683
	Identity (Dim: **#1**)	GCN	0.634	0.608
		Sage	0.615	0.574
		GIN	0.603	0.531
		GAT	0.589	0.539
		UNet	0.616	0.555
	Counting mnemonic classes (Dim: **#27**)	GCN	0.659	0.658
		Sage	0.694	0.66
		GIN	0.701	0.673
		GAT	0.655	0.667
		UNet	0.66	0.654
	Semantic & counting PCode mnemonics (Dim: **#77**)	GCN	0.761	0.733
		Sage	0.782	0.70
		GIN	0.775	0.69
		GAT	0.77	0.73
		UNet	0.753	0.724
	Semantic & counting assembly mnemonics (Dim: **#1839**)	GCN	0.792	0.758
		Sage	0.802	0.727
		GIN	0.793	0.727
		GAT	0.797	0.729
		UNet	0.785	0.701
	PalmTree on assembly code (Dim : **#128**)	GCN	0.763	-
		Sage	0.718	-
		GIN	0.715	-
		GAT	0.773	-
		UNet	0.768	-

Binary classification

Graph	Features	Algorithm	Balanced accuracy	
			Dataset-1	*Dataset-2*
CFG	Graph features & assembly (Dim: **#23**)	RandomForest	0.702	0.60
		GradientBoosting	0.725	0.649
	TF-IDF on assembly mnemonics (Dim: **#128**)	RandomForest	0.76	0.607
		GradientBoosting	0.80	0.683
	Identity (Dim: **#1**)	GCN	0.634	0.608
		Sage	0.615	0.574
		GIN	0.603	0.531
		GAT	0.589	0.539
		UNet	0.616	0.555
	Counting mnemonic classes (Dim: **#27**)	GCN	0.659	0.658
		Sage	0.694	0.66
		GIN	0.701	0.673
		GAT	0.655	0.667
		UNet	0.66	0.654
	Semantic & counting PCode mnemonics (Dim: **#77**)	GCN	0.761	0.733
		Sage	0.782	0.70
		GIN	0.775	0.69
		GAT	0.77	0.73
		UNet	0.753	0.724
	Semantic & counting assembly mnemonics (Dim: **#1839**)	GCN	0.792	0.758
		Sage	0.802	0.727
		GIN	0.793	0.727
		GAT	0.797	0.729
		UNet	0.785	0.701
	PalmTree on assembly code (Dim : **#128**)	GCN	0.763	-
		Sage	0.718	-
		GIN	0.715	-
		GAT	0.773	-
		UNet	0.768	-

GNN with coarse features give disappointing results.

Meaningful features (*containing part of the function semantics*) outperform baselines

Binary classification

Assembly feature outperforms Pcode feature but is significantly **more costly** (#78 instead of #1839) and **not CPU-agnostic.**

Graph	Features	Algorithm	Balanced accuracy	
			Dataset-1	*Dataset-2*
CFG	Graph features & assembly (Dim: **#23**)	RandomForest	0.702	0.60
		GradientBoosting	0.725	0.649
	TF-IDF on assembly mnemonics (Dim: **#128**)	RandomForest	0.76	0.607
		GradientBoosting	0.80	0.683
	Identity (Dim: **#1**)	GCN	0.634	0.608
		Sage	0.615	0.574
		GIN	0.603	0.531
		GAT	0.589	0.539
		UNet	0.616	0.555
	Counting mnemonic classes (Dim: **#27**)	GCN	0.659	0.658
		Sage	0.694	0.66
		GIN	0.701	0.673
		GAT	0.655	0.667
		UNet	0.66	0.654
	Semantic & counting PCode mnemonics (Dim: **#77**)	GCN	0.761	0.733
		Sage	0.782	0.70
		GIN	0.775	0.69
		GAT	0.77	0.73
		UNet	0.753	0.724
	Semantic & counting assembly mnemonics (Dim: **#1839**)	GCN	0.792	0.758
		Sage	0.802	0.727
		GIN	0.793	0.727
		GAT	0.797	0.729
		UNet	0.785	0.701
	PalmTree on assembly code (Dim : **#128**)	GCN	0.763	-
		Sage	0.718	-
		GIN	0.715	-
		GAT	0.773	-
		UNet	0.768	-

Binary classification

Transformers are fancy but **do not always give the best result.** Very costly*

Graph	Features	Algorithm	Balanced accuracy	
			Dataset-1	*Dataset-2*
CFG	Graph features & assembly (Dim: **#23**)	RandomForest	0.702	0.60
		GradientBoosting	0.725	0.649
	TF-IDF on assembly mnemonics (Dim: **#128**)	RandomForest	0.76	0.607
		GradientBoosting	0.80	0.683
	Identity (Dim: **#1**)	GCN	0.634	0.608
		Sage	0.615	0.574
		GIN	0.603	0.531
		GAT	0.589	0.539
		UNet	0.616	0.555
	Counting mnemonic classes (Dim: **#27**)	GCN	0.659	0.658
		Sage	0.694	0.66
		GIN	0.701	0.673
		GAT	0.655	0.667
		UNet	0.66	0.654
	Semantic & counting PCode mnemonics (Dim: **#77**)	GCN	0.761	0.733
		Sage	0.782	0.70
		GIN	0.775	0.69
		GAT	0.77	0.73
		UNet	0.753	0.724
	Semantic & counting assembly mnemonics (Dim: **#1839**)	GCN	0.792	0.758
		Sage	0.802	0.727
		GIN	0.793	0.727
		GAT	0.797	0.729
		UNet	0.785	0.701
	PalmTree on assembly code (Dim : **#128**)	GCN	0.763	-
		Sage	0.718	-
		GIN	0.715	-
		GAT	0.773	-
		UNet	0.768	-

(-) indicates OOM

** ~ 1 week for PalmTree / few hours for the other GNN*

Binary classification

Graph	Features	Algorithm	Balanced accuracy	
			Dataset-1	*Dataset-2*
CFG	Graph features & assembly (Dim: **#23**)	RandomForest	0.702	0.60
		GradientBoosting	0.725	0.649
	TF-IDF on assembly mnemonics (Dim: **#128**)	RandomForest	0.76	0.607
		GradientBoosting	0.80	0.683
	Identity (Dim: **#1**)	GCN	0.634	0.608
		Sage	0.615	0.574
		GIN	0.603	0.531
		GAT	0.589	0.539
		UNet	0.616	0.555
	Counting mnemonic classes (Dim: **#27**)	GCN	0.659	0.658
		Sage	0.694	0.66
		GIN	0.701	0.673
		GAT	0.655	0.667
		UNet	0.66	0.654
	Semantic & counting PCode mnemonics (Dim: **#77**)	GCN	0.761	0.733
		Sage	0.782	0.70
		GIN	0.775	0.69
		GAT	0.77	0.73
		UNet	0.753	0.724
	Semantic & counting assembly mnemonics (Dim: **#1839**)	GCN	0.792	0.758
		Sage	0.802	0.727
		GIN	0.793	0.727
		GAT	0.797	0.729
		UNet	0.785	0.701
	PalmTree on assembly code (Dim : **#128**)	GCN	0.763	-
		Sage	0.718	-
		GIN	0.715	-
		GAT	0.773	-
		UNet	0.768	-

Better generalization capabilities of **GNN** compared to baselines

Multi-class classification (11 classes)

Graph	Features	Algorithm	Balanced accuracy	
			Dataset-1	*Dataset-2*
CFG	Graph features & assembly (Dim: **#23**)	RandomForest	0.65	0.57
		GradientBoosting	0.66	0.594
	TF-IDF on assembly mnemonics (Dim: **#128**)	RandomForest	0.697	0.593
		GradientBoosting	0.724	0.579
	Identity (Dim: **#1**)	GCN	0.323	0.326
		Sage	0.341	0.347
		GIN	0.414	0.407
		GAT	0.192	0.195
		UNet	0.362	0.299
	Counting mnemonic classes (Dim: **#27**)	GCN	0.431	0.462
		Sage	0.498	0.499
		GIN	0.488	0.474
		GAT	0.45	0.342
		UNet	0.439	0.448
	Semantic & counting PCode mnemonics (Dim: **#77**)	GCN	0.699	0.693
		Sage	0.611	0.729
		GIN	0.706	0.71
		GAT	0.684	0.65
		UNet	0.704	0.627
	Semantic & counting assembly mnemonics (Dim: **#1839**)	GCN	0.723	0.633
		Sage	0.718	0.535
		GIN	0.713	0.427
		GAT	0.723	0.646
		UNet	0.709	0.611
	PalmTree on assembly code (Dim : **#128**)	GCN	0.696	-
		Sage	0.698	-
		GIN	0.693	-
		GAT	0.685	-
		UNet	0.67	-

Multi-class classification (11 classes)

Graph	Features	Algorithm	Balanced accuracy	
			Dataset-1	*Dataset-2*
CFG	Graph features & assembly (Dim: #**23**)	RandomForest	0.65	0.57
		GradientBoosting	0.66	0.594
	TF-IDF on assembly mnemonics (Dim: #**128**)	RandomForest	0.697	0.593
		GradientBoosting	0.724	0.579
	Identity (Dim: #**1**)	GCN	0.323	0.326
		Sage	0.341	0.347
		GIN	0.414	0.407
		GAT	0.192	0.195
		UNet	0.362	0.299
	Counting mnemonic classes (Dim: #**27**)	GCN	0.431	0.462
		Sage	0.498	0.499
		GIN	0.488	0.474
		GAT	0.45	0.342
		UNet	0.439	0.448
	Semantic & counting PCode mnemonics (Dim: #**77**)	GCN	0.699	0.693
		Sage	0.611	0.729
		GIN	0.706	0.71
		GAT	0.684	0.65
		UNet	0.704	0.627
	Semantic & counting assembly mnemonics (Dim: #**1839**)	GCN	0.723	0.633
		Sage	0.718	0.535
		GIN	0.713	0.427
		GAT	0.723	0.646
		UNet	0.709	0.611
	PalmTree on assembly code (Dim : #**128**)	GCN	0.696	-
		Sage	0.698	-
		GIN	0.693	-
		GAT	0.685	-
		UNet	0.67	-

Same trend than in the binary case !

Results are very promising given the high number of classes

Real-World example : XTunnel

XTunnel

- Malware designed by APT-28
- Used to exfiltrate data from a compromised device
- Obfuscated with Opaque Predicates [1]
- Handmade ground-truth *(costly)*

	Binary balanced accuracy	Multi-class balanced accuracy
Sample C637E	0.726	0.533
Sample 99B45	0.711	0.55

[1] Bardin and al. **Backward-bounded dse: Targeting infeasibility questions on obfuscated codes**. 2017

Conclusion

Resilient binary diffing

- Using multiple program variants weakens the applied obfuscation
- Differs and especially Qbindiff work well (even for 100% of obfuscation)
- Intra-procedural obfuscation and data obfuscation are sensitive to this attack
- Similarity matrix & graph adjacency => **diff anything !**

Obfuscation detection and classification

- Promising results, with satisfactory baselines
- GNN with a strong generalization power
- High results, both for the binary and multi-class classification

Thank you

Contact information:

Email: contact@quarkslab.com

Phone: +33 1 58 30 81 51

Website: quarkslab.com

@quarkslab

Quarkslab