Detecting and Categorizing Android Malware with Graph Neural Networks

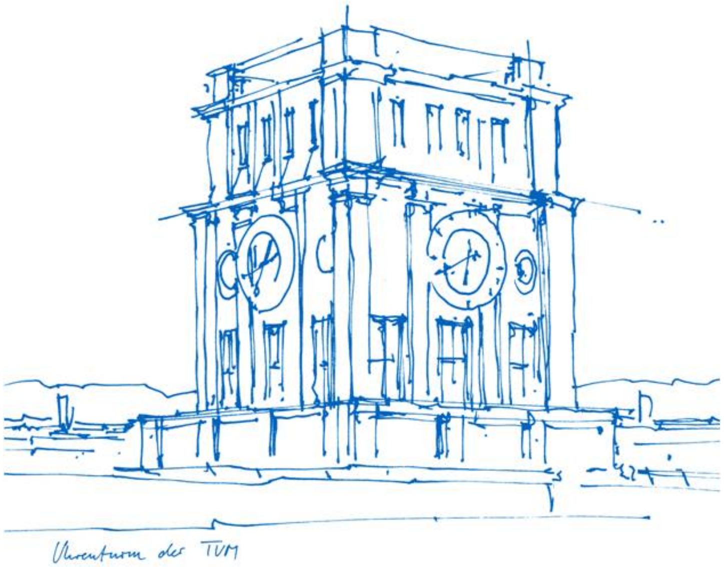
Peng Xu[1], Claudia Eckert[1], Apostolis Zarras[2]

{Peng,eckert}sec.in.tum.de

a.zarras@tudelft.nl

[1] Technical University of Munich

[2] Delft University of Technology



Motivation

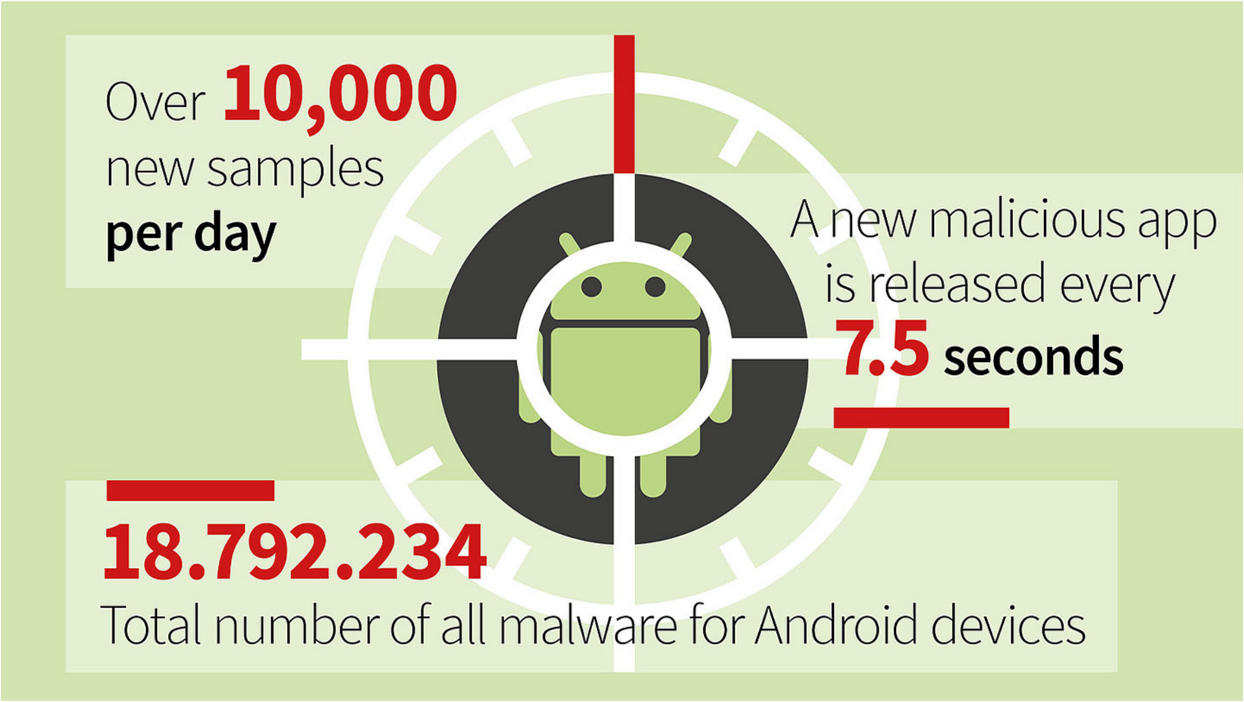
The year in figures

In 2020, Kaspersky mobile products and technologies detected:

- 5,683,694 malicious installation packages,
- 156,710 new mobile banking Trojans,
- 20,708 new mobile ransomware Trojans.

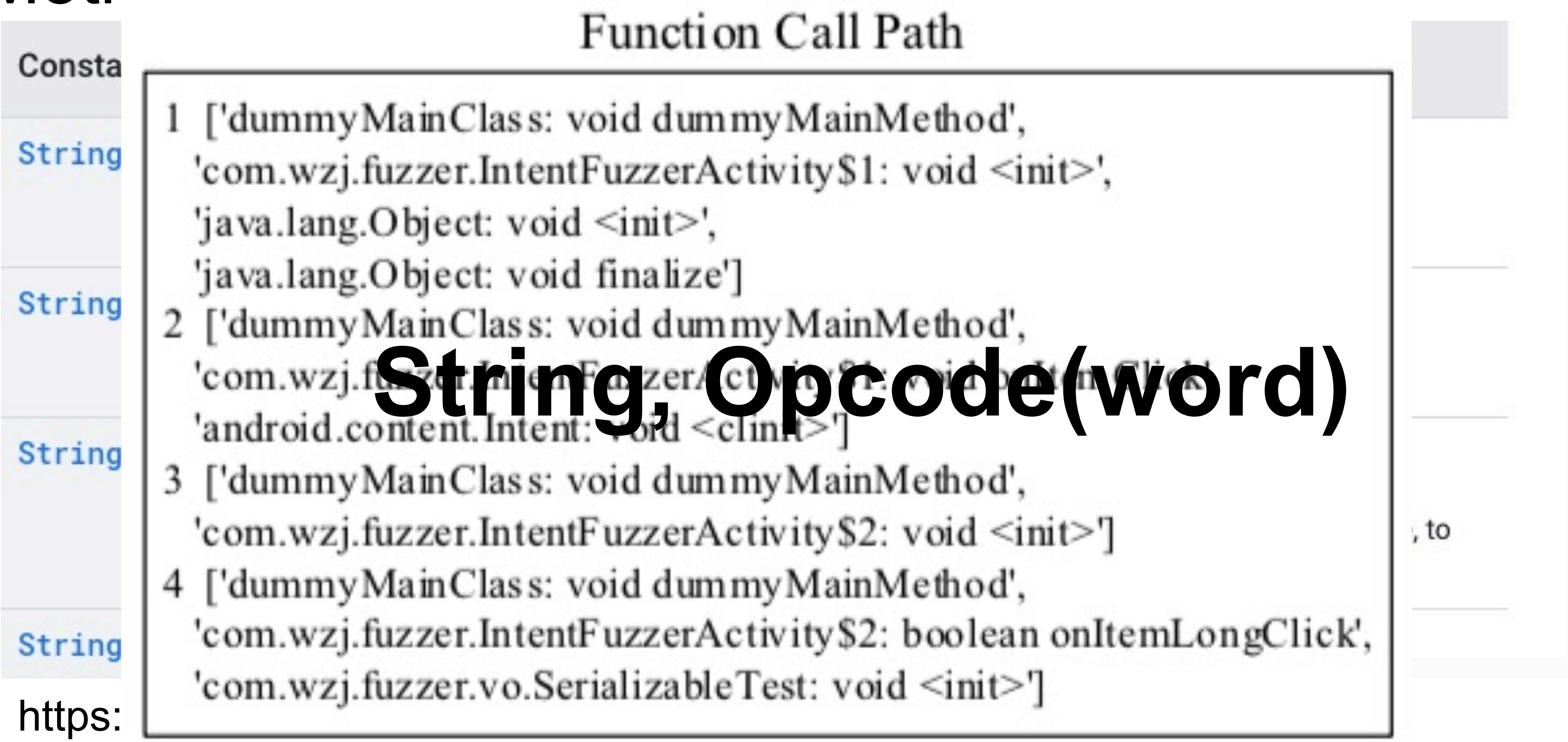
https://securelist.com/mobile-malware-evolution-2020/101029/

Motivation



G DATA Mobile Malware Report 2019: New high for malicious Android apps

Motivation



String, Opcode(word)

https:

OpCode-Level Function Call Graph Based Android Malware Classification Using Deep Learning

1. permission-based Android Malware Detection systems (DREBIN, FM)
2. API-call-based Android Malware Detection systems (DroidNative)

Motivation

String(permission), API Call(word)

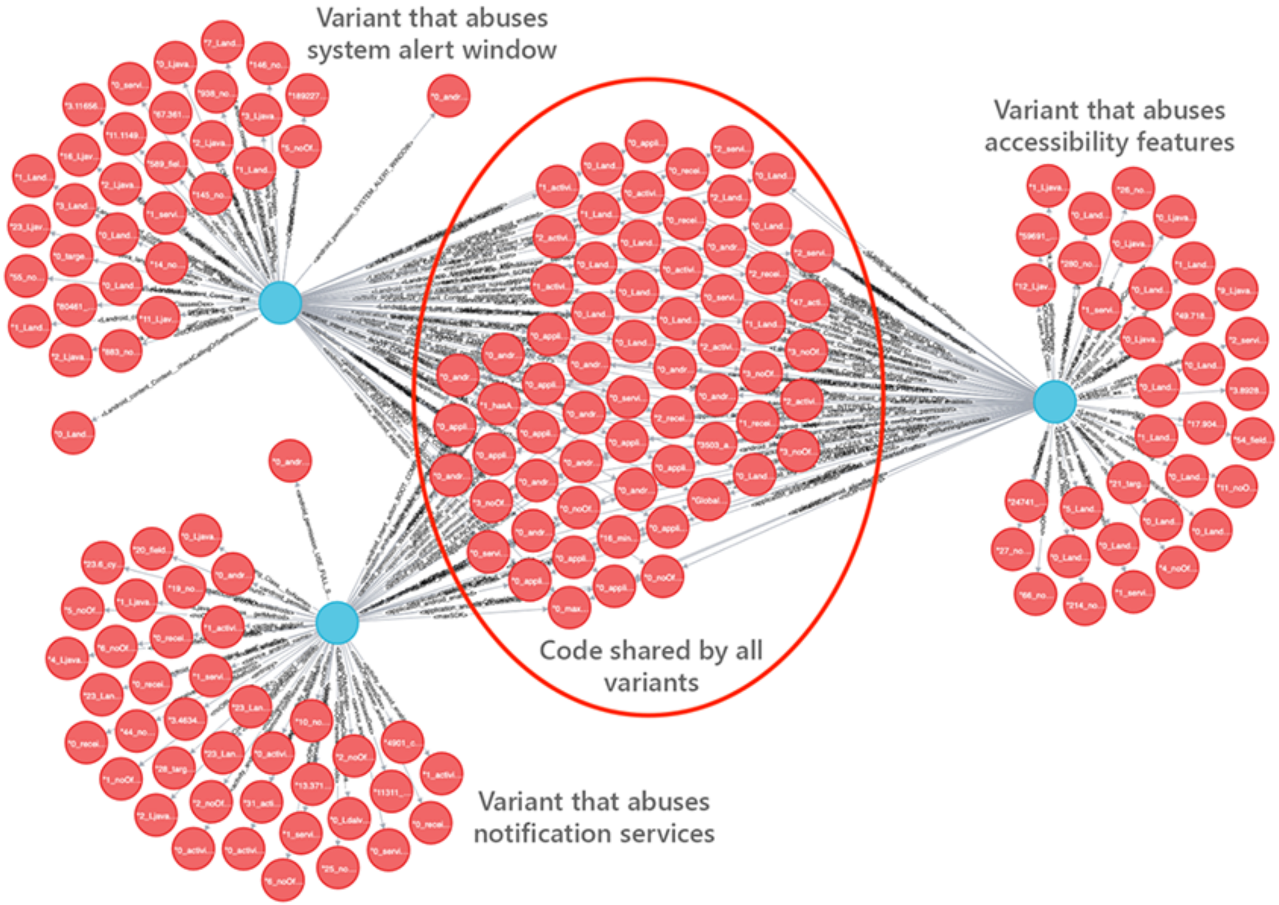
Obfuscation

String Obfuscation

- **Class Encryption**
- **String Encryption**
- **Reflection**
  - replace each invoke instruction with specific bytecode

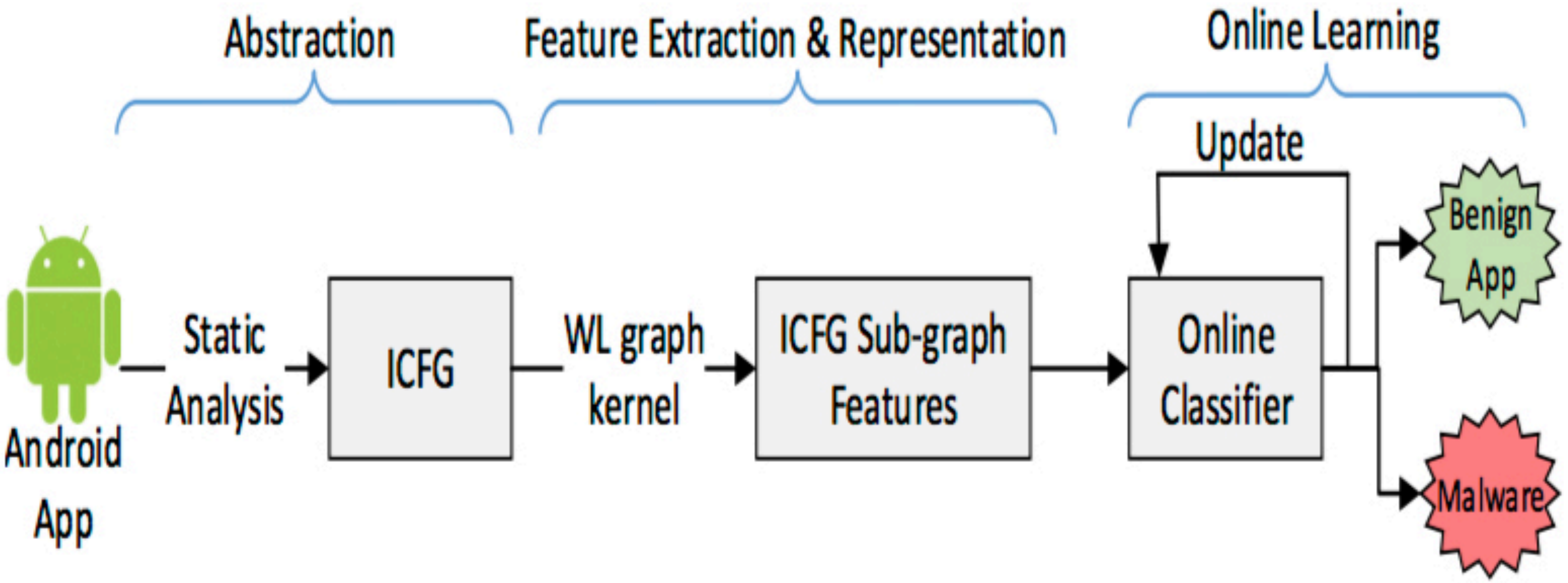
- **Trivial Obfuscation**
  - Only affects string, not bytecode
- **Trivial + String Encryption**
- **Trivial + StringEnc + Reflection**
- **Trivial + StringEnc + Reflection + ClassEnc**

Motivation



https://www.microsoft.com/security/

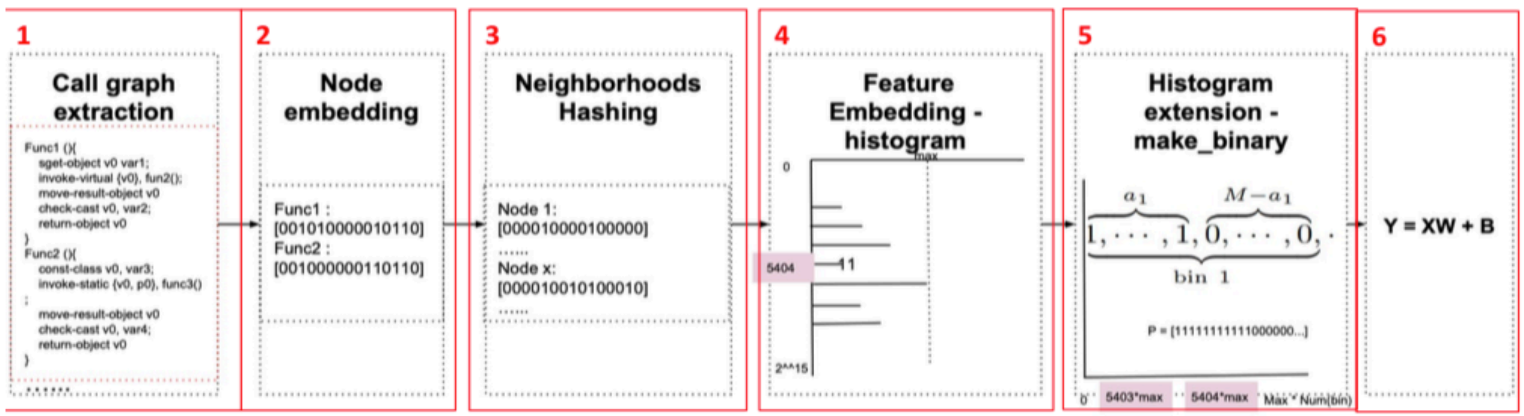
Motivation



DroidOL: Android malware detection based on online machine learning

Motivation

Android Malware Detection on Graph Structure



14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
binop	unop	invoke	staticop	instanceop	arrayop	branch	jump	throw	new	test	monitor	return	move	nop

Figure 1: Adagio malware detection. Top: Detection system includes six steps. Botton: 15-Dalvik instruction categories.

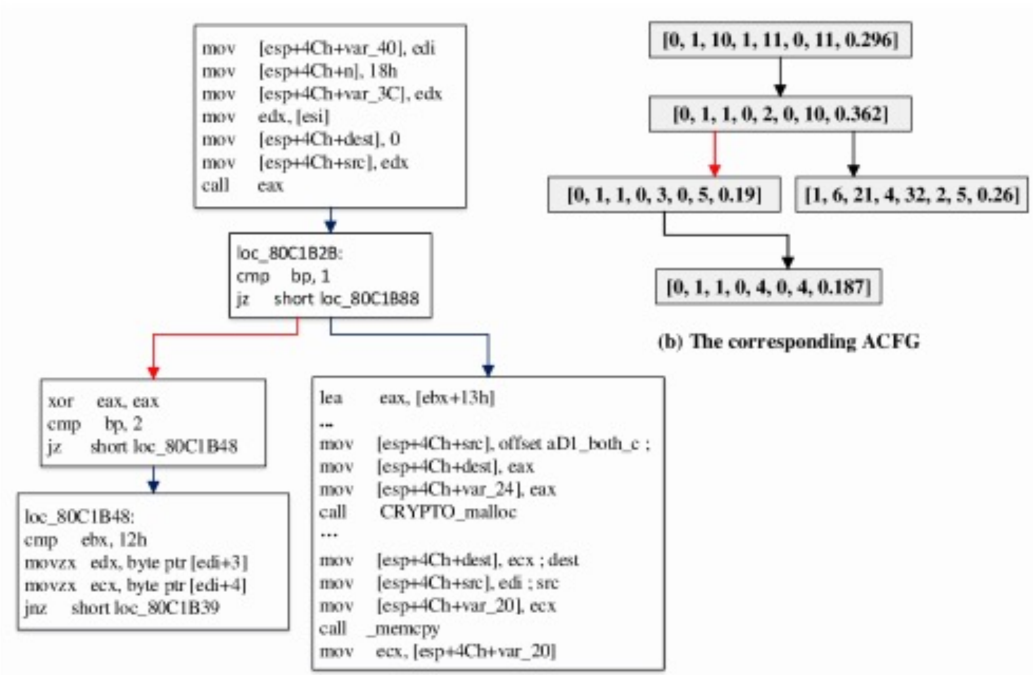
5404 -> [001-0101-0001-1100]

$$h(v) = r(\ell(v)) \oplus \left(\bigoplus_{z \in V_v} \ell(z) \right)$$

$\{[11111111111]_{11}[0]_{(max*5404-11)}$

Adagio: Structural Detection of Android Malware using Embedded Call-Graph
MANIS: evading malware detection system on graph structure

Motivation



(b) The corresponding ACFG

(a) Partial control flow graph of dtls1_process_heartbeat

Type	Attribute name
Block-level attributes	String Constants Numeric Constants No. of Transfer Instructions No. of Calls No. of Instructions No. of Arithmetic Instructions
Inter-block attributes	No. of offspring Betweenness

Table 1: Basic-block attributes

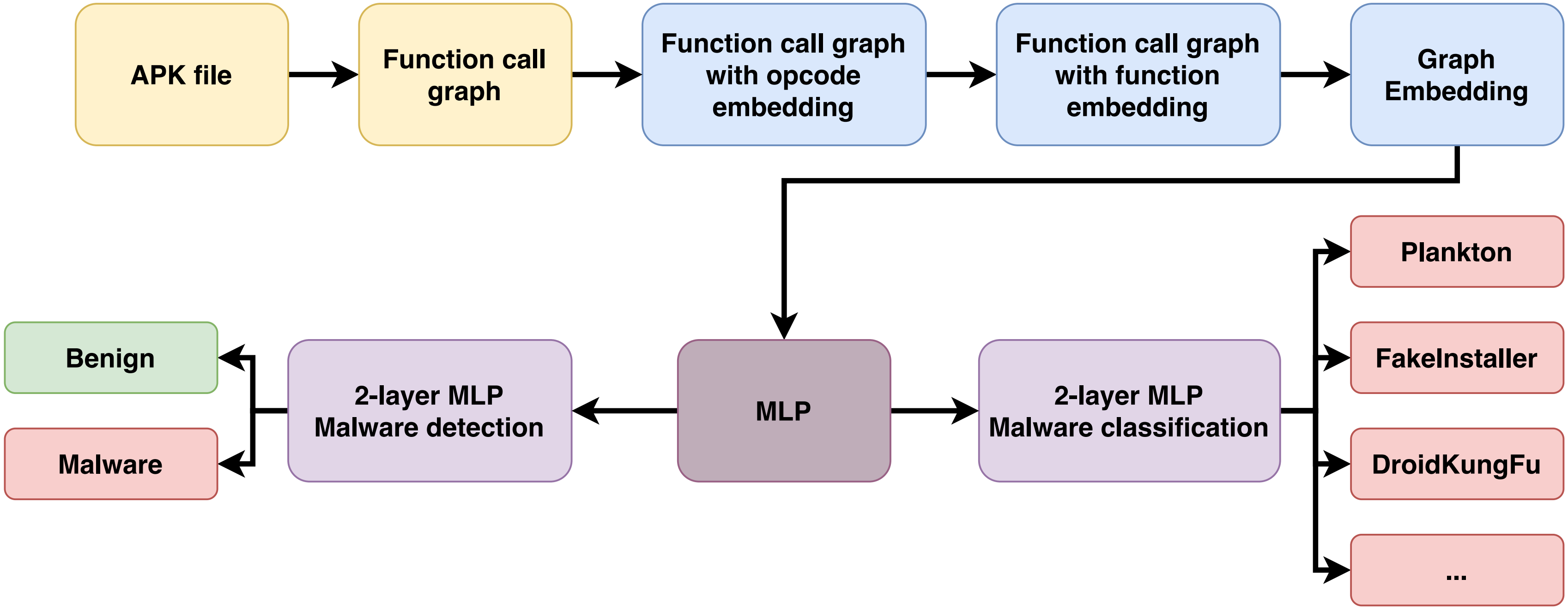
Neural Network-based Graph Embedding for Cross-Platform Binary Code Similarity Detection

Motivation

Table 1: Comparsion with previous works

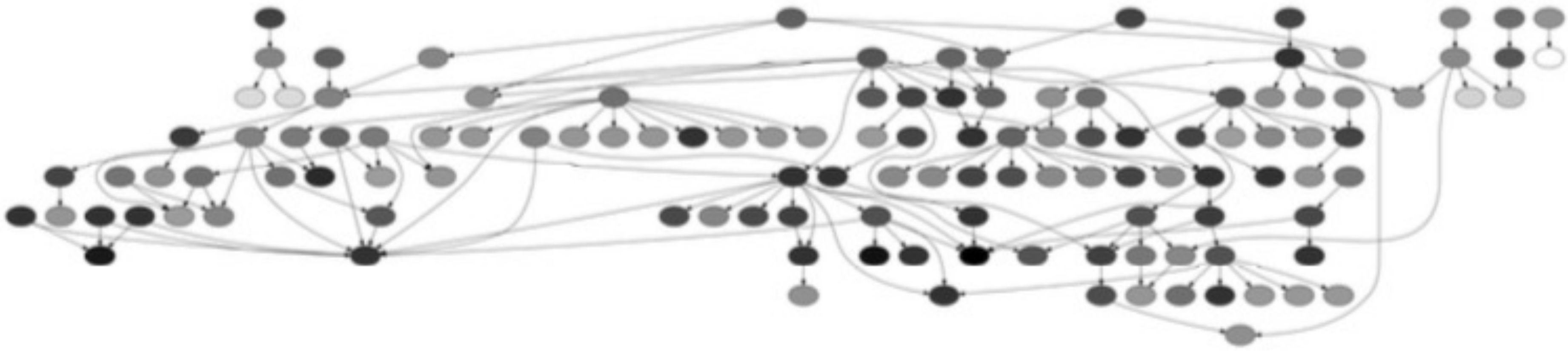
	Approach	**Description**
Permission	Drebin [4]	Permission
	FM [14]	Permission + Factorization Machine
CFG	Gemini [8]	CFG + Manually indicated features (frequency of 8 instructions(Bytecode))
	Adagio [9]	CFG + Manually indicated one-hot embedding features (15 categories instructions(Bytecode))
Byte Sequences	McLaughlin et. al. [16]	Convolutional + trainable embedding + Bytecode
	Droidnative [1]	N-gram + CFG + native code
Our Solution	GE	CFG + N-gram trainable instructions + Bytecode

Overview



APK file
Function call graph
Function call graph with opcode embedding
Function call graph with function embedding
Graph Embedding
Plankton
Benign
2-layer MLP Malware detection
MLP
2-layer MLP Malware classification
FakeInstaller
Malware
DroidKungFu
...

Function Call Graph



Androguard to get Function call graph(e.g, Adagio, MANIS)

Opcode Embedding

- Instruction: Opcode + Operands
- Why only consider Opcode?
  - Other works: Address, Register are replaced by specific symbols
  - Move Instruction: move-wide vA, vB[04 12x], move-wide/from16 vAA, vBBBB[05 22x]
  - Invoke Instruction: invoke-super, invoke-direct, invoke-static, and invoke-interface
- Word Embedding

Function Embedding

- Weighted Mean Function Embedding

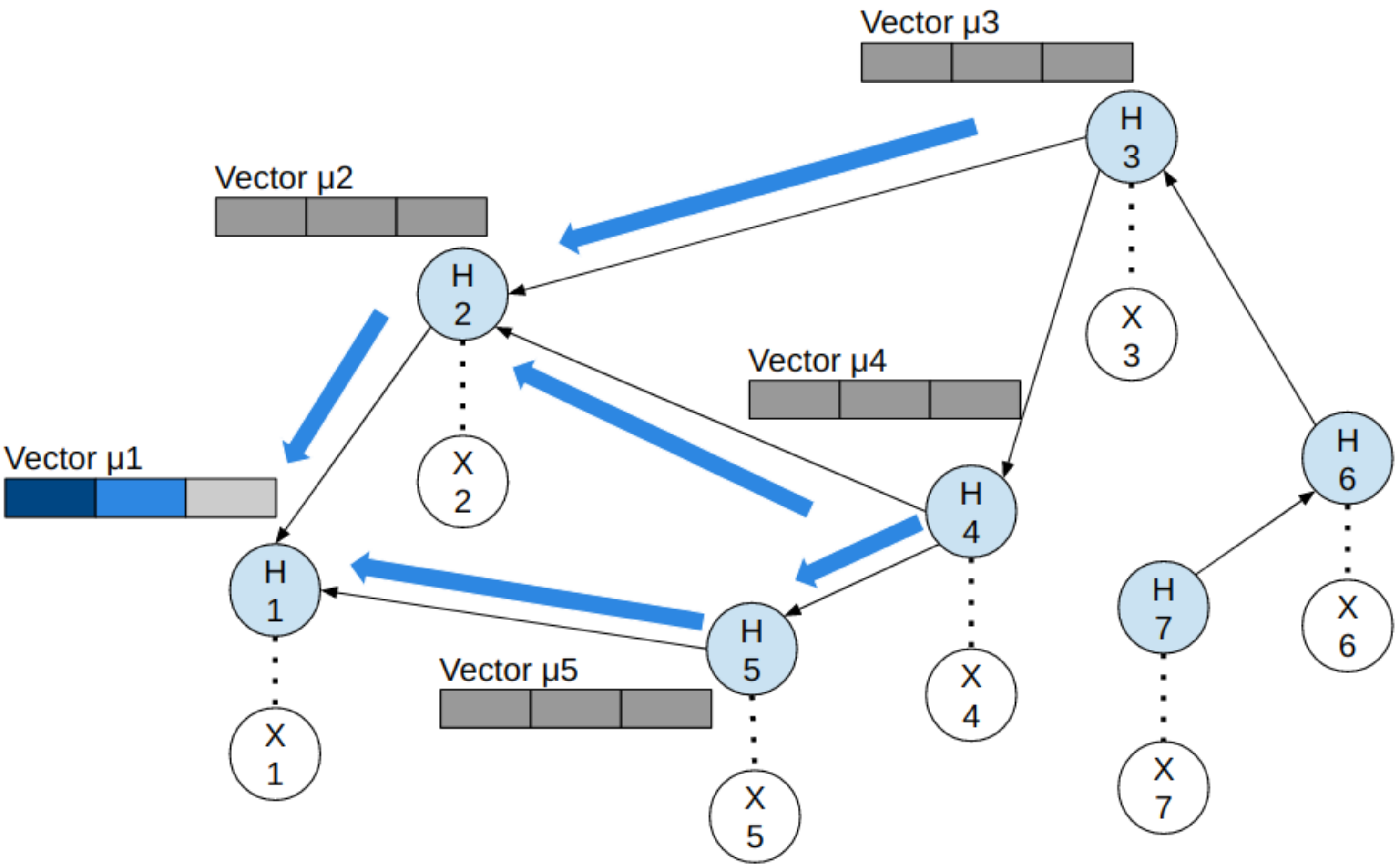
$$\vec{f} = \frac{\sum_{i=1}^{n} w_i x_i}{\sum_{i=1}^{n} w_i}$$

- SIF-Invoked Function Embedding
  - SIF: A simple but tough-to-beat baseline for sentence embeddings.

$$Pr[i \notin f | \tilde{c_f}] = \alpha p(i) + (1 - \alpha) \frac{exp(< c_f, v_i >)}{Z_{\tilde{c_f}}},$$

where $c_f = \beta c_0 \tilde{+} (1 - \beta) c_f, c_0 \perp c_f$, α and β are scalar hyperparameters, and $Z_{\tilde{c_f}} = \sum_{i \in f} exp(< \tilde{c_f}, v_i >)$ the normalizing constant.

Graph Embedding



MLP Classifier

- Malware Classification:

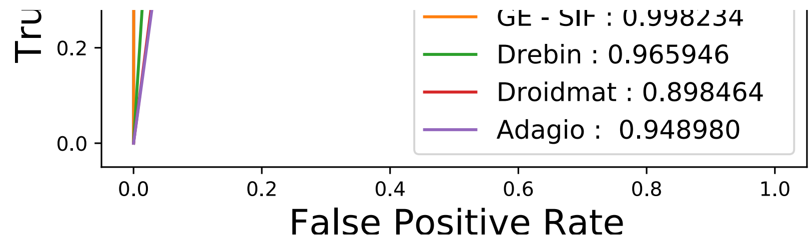
$$f(G_h) =< (< g_i, w_{i1} > +b_{i1}), w_{i2} > +b_{i2}$$

- Malware Categorization:

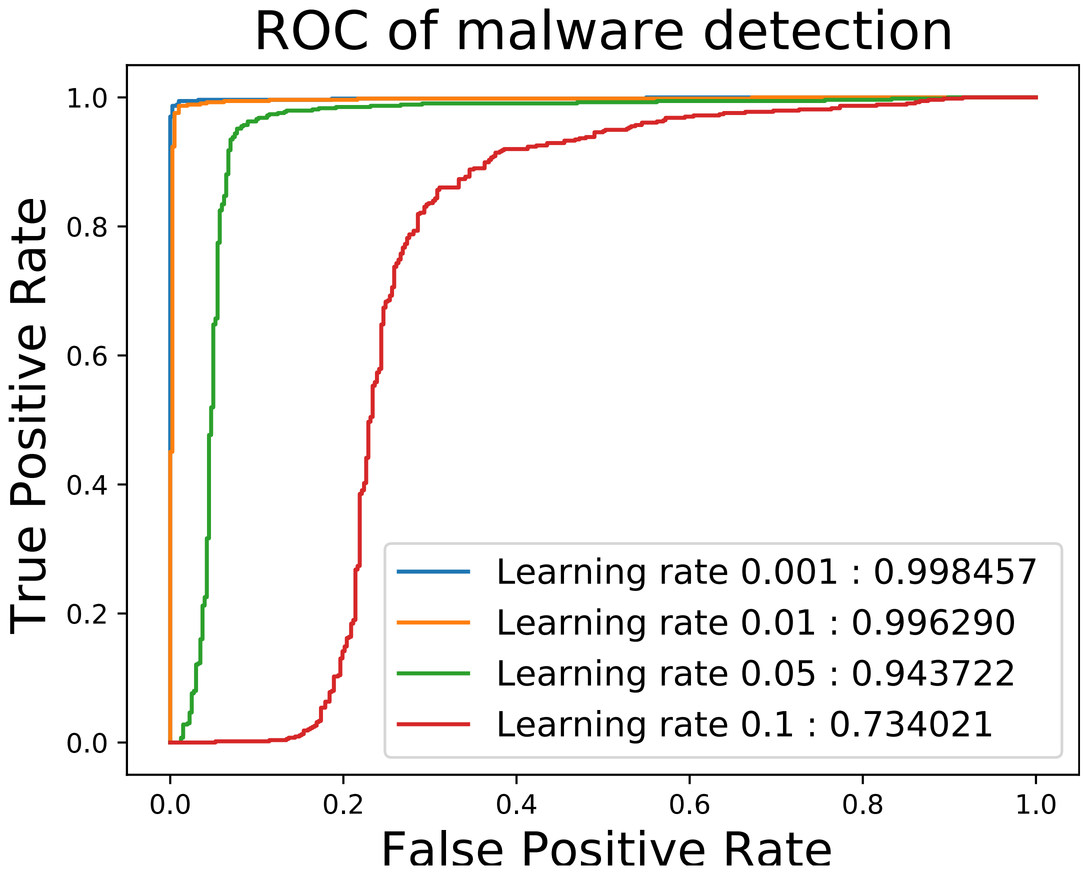
$$f(G_h) = softmax(< (< g_i, w_{i1} > +b_{i1}), w_{i2} > +b_{i2})$$

Evaluation

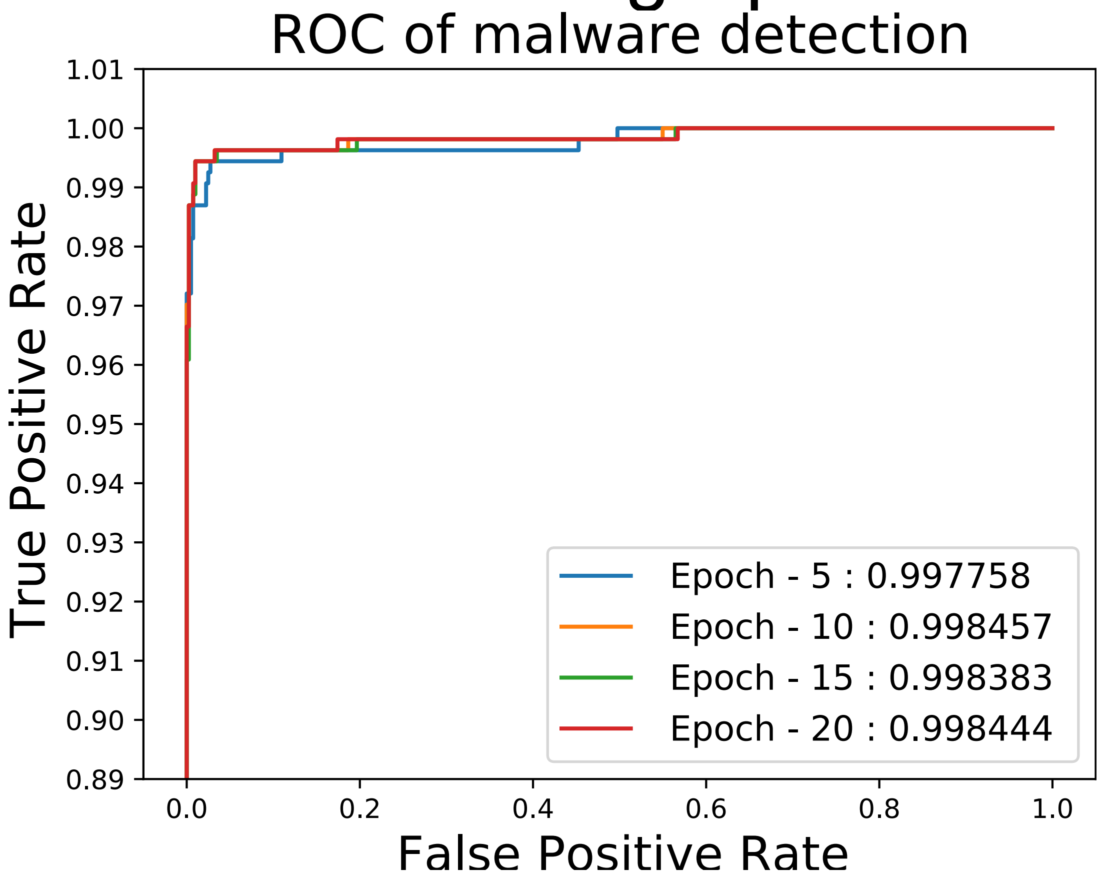
Algorithm	Accuracy (%)	Precision (%)	Recall (%)	F1 (%)	FPR (%)
Ge-SIF	99.86	99.75	99.75	99.42	0.7
Ge-Mean	99.74	99.92	99.63	99.78	0.4
Drebin	96.58	95.37	97.85	96.59	2.35
Droidmat	89.87	90.89	88.28	89.56	4.36
Adagio	95.0	91.07	100	95.32	5.0



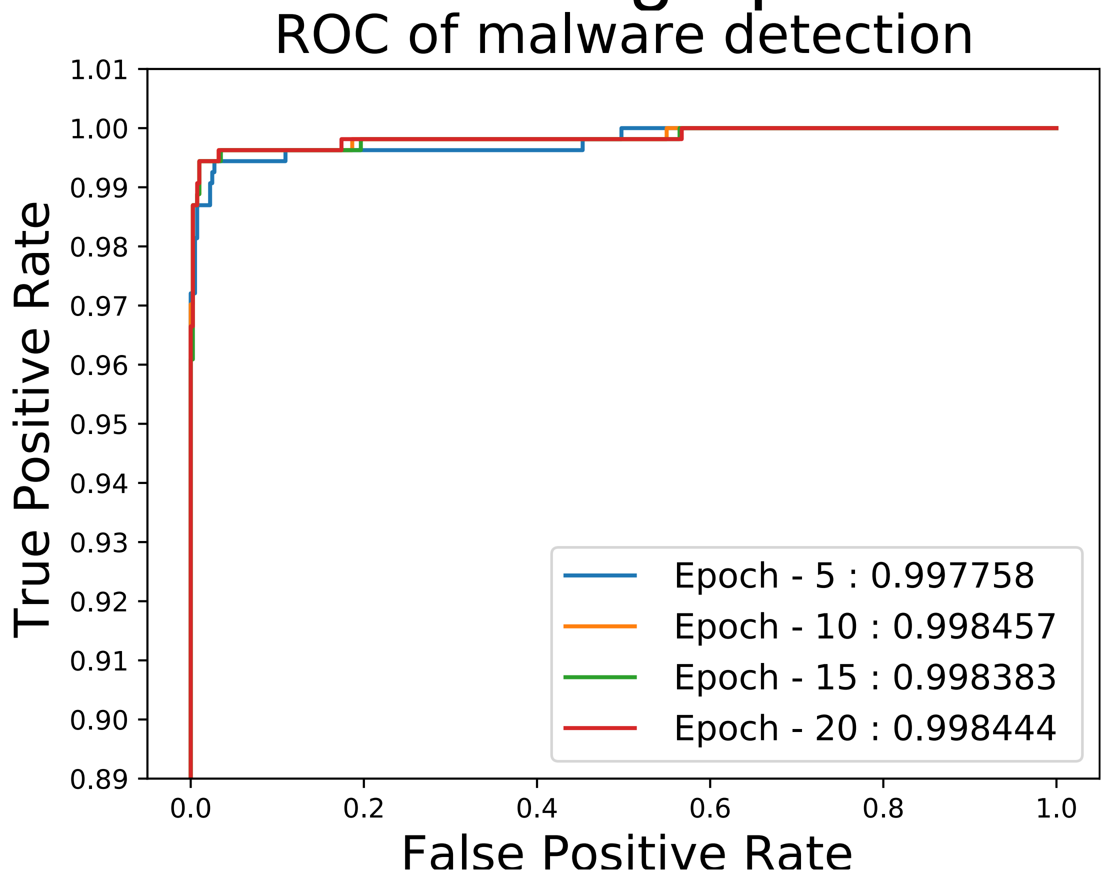
Evaluation – Various learning rate



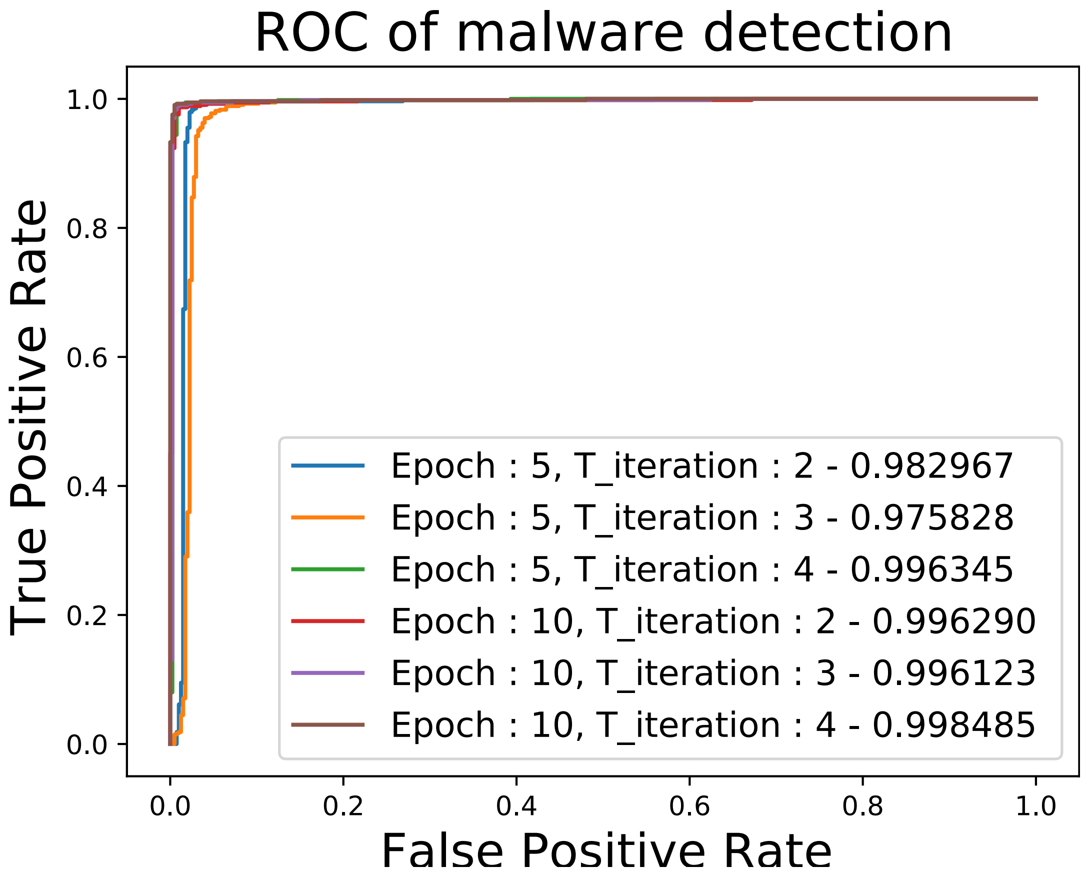
Evaluation – Various training Epoch



Evaluation – Various training Epoch



Evaluation – Various n-hop neighbors

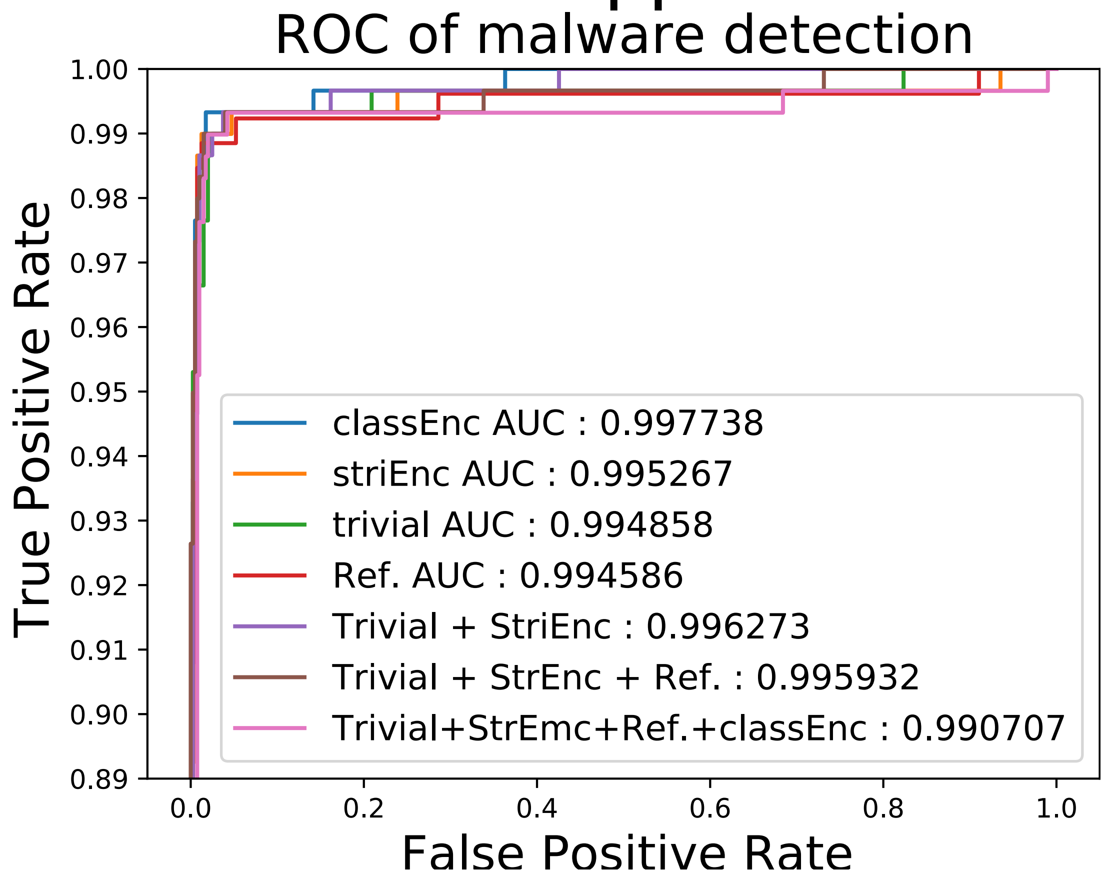


Evaluation – Obfuscated Application

Table 2: Detection rate of obfuscated APK

	ClassEnc.	StrEnc.	Refl.	Triv.	Triv.-Str.	Triv.-Ref.-Str.	Triv.-Ref.-Str.-Class.
PRAGuard[7]	38.0	64.0	96	90.0	50.0	44.0	32.0
Drebin	99.12	98.99	86.58	98.32	98.99	99.32	96.98
Our framework	99.33	98.99	86.58	98.32	98.99	99.32	96.98

Evaluation – Obfuscated Application



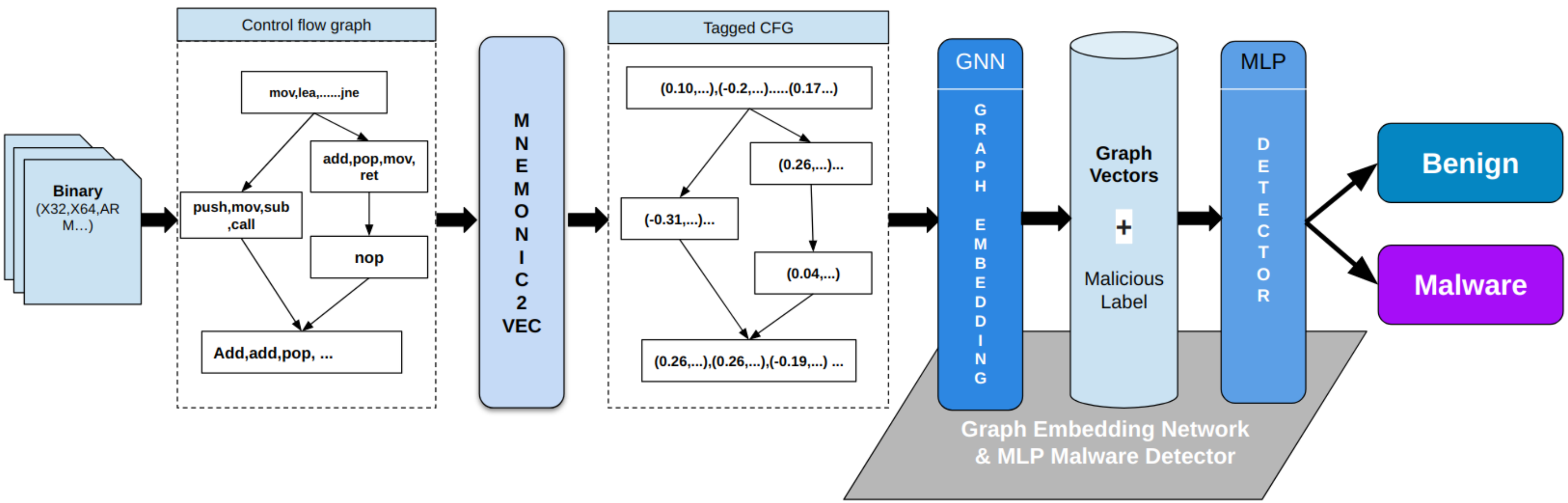
Evaluation – Categorization/Family Classification

Table 3: Family classification results

			5-epoch					10-epoch				
	Family	**Samples**	**Accuracy**	**Precision**	**Recall**	**F1**	**FPR**	**Accuracy**	**Precision**	**Recall**	**F1**	**FPR**
Mean	FakeInstaller	925	99.61	98.78	98.90	99.39	0.57	99.21	98.78	98.78	99.39	0.58
	DroidKungFu	667	99.60	98.10	98.10	99.04	0.50	99.20	98.10	98.06	99.04	0.5
	Plankton	624	99.65	92.31	92.31	96.00	0.37	99.29	92.31	92.31	96.0	0.37
	Opfake	613	99.35	97.21	97.05	98.50	0.82	99.08	97.87	97.86	98.92	0.58
	GinMaster	339	99.64	95.92	95.92	97.91	0.38	99.29	92.31	95.92	97.92	0.39
	BseBridge	330	99.61	96.62	96.62	98.28	0.44	99.14	96.31	96.31	98.12	0.48
SIF	FakeInstaller	925	99.5	98.45	98.45	99.22	0.74	99.0	98.45	97.59	97.60	0.74
	DroidKungFu	667	99.53	97.76	97.76	98.87	0.59	99.06	97.76	98.21	97.16	0.59
	Plankton	624	99.64	92.59	92.59	96.15	0.37	99.29	92.59	97.18	97.30	0.37
	Opfake	613	99.44	97.38	97.38	98.67	0.72	99.22	98.20	97.16	97.16	0.49
	GinMaster	339	99.50	94.8	94.4	97.32	0.55	98.76	92.8	97.87	97.86	0.70
	BseBridge	330	99.47	95.07	95.38	97.63	0.6	99.01	95.69	96.85	96.89	0.56

Question?
Thanks!

Backup



Control flow graph
mov,lea,......jne
add,pop,mov,
ret
push,mov,sub
,call
nop
Add,add,pop, ...
Binary
(X32,X64,AR
M...)
M
N
E
M
O
N
I
C
2
VEC
Tagged CFG
(0.10,...),(-0.2,...).....(0.17...)
(0.26,...)...
(-0.31,...)...
(0.04,...)
(0.26,...),(0.26,...),(-0.19,...) ...
GNN
GRAPH EMBEDDING
Graph
Vectors
+
Malicious
Label
MLP
DETECTOR
Benign
Malware
Graph Embedding Network
& MLP Malware Detector

Backup – Structure2vec

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

$$F(x_v, \sum_{u \in N_v} \mu_u^{(t)}) = tanh(W_1 x_v + \sigma(\sum_{u \in N(v)} \mu_u))$$

$$\sigma(l) = P_1 * ReLU(P_2 * \ldots ReLU(P_n l))$$