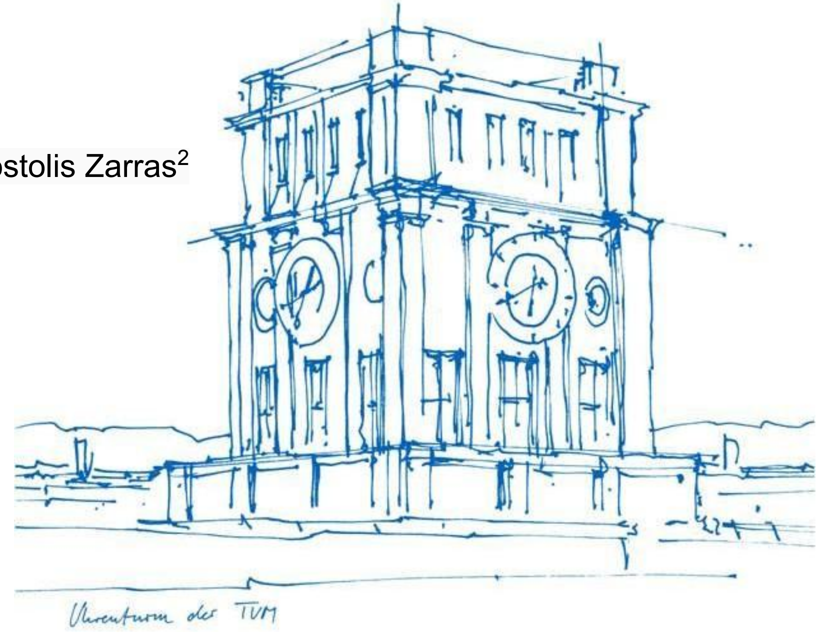
ISC 2021
Information Security Conference

Hybroid: Toward Android Malware Detection and Categorization with Program Code and Network Traffic

Mohammad Reza Norouzian[1], **Peng Xu[1]**, Claudia Eckert[1], and Apostolis Zarras[2]

[1] Technical University of Munich

[2] Delft University of Technology



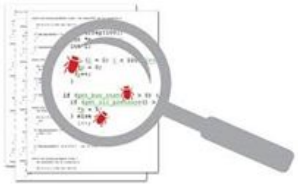
Agenda

- Introduction
- System design
- Evaluation
- Limitation and future works
- Summery

Introduction

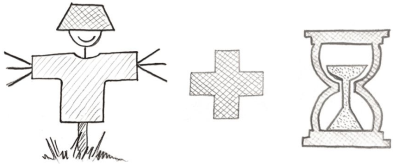
- Problem
  - Finding a robust and efficient way to detect Android malware

- Traditional solutions
  - **Static**: reviews the source code and binaries in order to find suspicious patterns.
  - **Dynamic**: involves the execution app in an isolated environment while monitoring and tracing its behaviour.



Static and Dynamic Analysis Approaches

- Static
  - Traditionally: signatures
  - Patterns in: **binary file**, **API calls**, **op-codes**
  - Methods: manual analysis or machine learning
  - Challenge: **obfuscated** applications, processing **speed**



- Dynamic
  - Executing in the isolated environment
  - **System-level** behavior or **networking** behavior: monitoring battery, op-codes, API calls, **network traffic**, etc.
  - Methods - signature based or machine learning

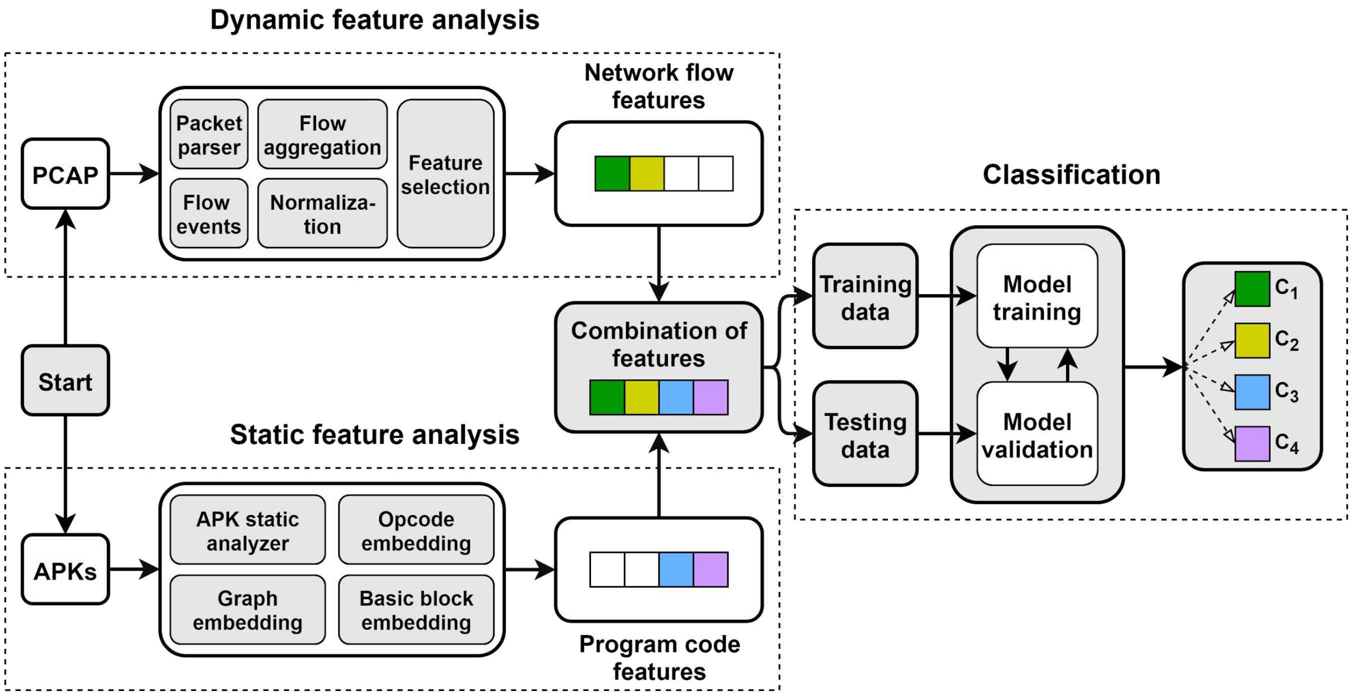
Our Approach

- Utilizing **static** and **dynamic** behavioral analysis
- Hybrid = program code structures + network traffic + machine learning
- **Binary** classification and **multi-label** classification
- Android malware **detection** and **categorization**

Our Contribution

- We present Hybroid, a **hybrid framework** for Android malware detection and categorization based on static and dynamic features.
- We design and implement automatic extraction of **flow-based** features from the Android raw network traffic as a dynamic features.
- We leverage **NLP** and convert machine codes, functions, and programs to **opcode2vec**, **function2vec**, and **graph2vec** by embedding methods.
- We **evaluate** the accuracy of our approach using a real-world dataset and show that Hybroid outperforms nearly all state-of-the-art solutions.

System Overview

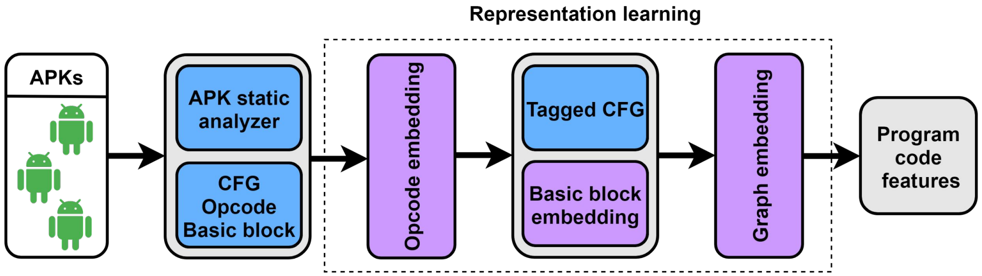


Static Features Preparation

- Extract the **opcode**, **basic block**, and **CFG** from the Android APKs.
- Extract the **CFG** by utilizing the Androguard[1] framework.
- Iterate each function in the program to get the **basic block**.
- Analyze each instruction and take **opcode** as our basic term.
- The entire process includes three main steps:
  - Opcode embedding
  - Basic block embedding
  - Graph embedding

1- https://github.com/androguard

Static Features Preparation Cont'd



- Opcode embedding
  - Converts the machine instructions into vectors
- Basic block embedding
  - Transforms a basic block of the program into a vector
- Graph embedding
  - Modifies the whole function call graph into a vector
- Representation learning

Static Features Preparation Cont'd

- Opcode embedding
  - Word2vec
  - Opcode/Mnemonic
- Basic block embedding normalization
  - $x^1 = (x - min(x))/(max(x) - min(x))$
- Graph embedding.
  - Structure2vec
  - Vertices: functions/basic block
  - Edges: caller/callee, jump/return/jne instructions

Algorithm 1: Graph embedding

Input: Instruction embedding $v_i : i \in I$, control flow graph insider of a function g_f, parameter α

Output: Graph embedding $v_f : f \in F$

Initialize $\mu_v^0 = \vec{Rand}, forall v \in V$

for t=1 *to* T **do**

  for $v \in V$ **do**

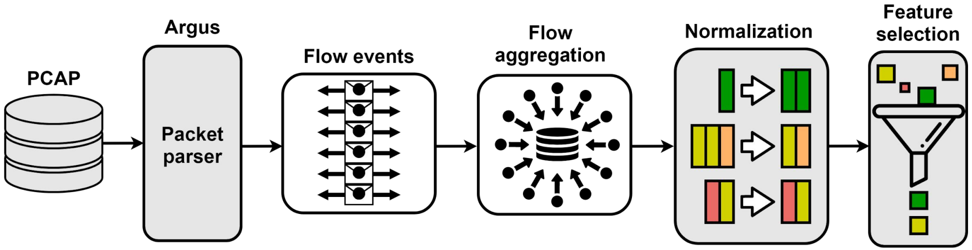
    $l_v = \sum_{u \in N(v)} \mu_u^{(t-1)}$

    $\mu_v^{(t)} = tanh(W_1 x_v + \sigma(l_v))$

$v_f = W_2(\sum_{v \in V} \mu_v^T)/len(V))$

return v_f

Dynamic Features Preparation



- Network flow generation
  - NetFlow data aggregated
- Normalization
  - $x^1 = (x - min(x))/(max(x) - min(x))$
- Feature selection
  - Complexity reduction
  - Noise reduction

Dynamic Feature Selection

- Feature selection algorithms
  - **Pearson** correlation, **Extra trees** classifier, **Univariate** feature selection
- Feature selection validation
  - **Kendall's** correlation method

Notation	Traffic Features
Mean	*Average duration of aggregated records*
sTos	*Source TOS byte value*
dTos	*Destination TOS byte value*
sTtl	*Source to destination TTL value*
dTtl	*Destination to source TTL value*
TotBytes	*Total transaction bytes*
SrcBytes	*Source to destination transaction bytes*
DstWin	*Destination TCP window advertisement*
SrcTCPBase	*Source TCP base sequence number*
DstTCPBase	*Destination TCP base sequence number*
Flgs_er	*State flag for Src loss/retransmissions*
Flgs_es	*State flag for Dst packets out of order*
Dir	*Direction of transaction*

List of network flow features

Dynamic network flow feature correlation scores

Observation of Malware Network Communications

- Observations on the entire encrypted data flows
- Initially more upload than download are more likely to be **malicious**.
  - Malware connects to a control server, identifies a client certificate
  - After the initial connection, the channel is often kept open but **idle**!
- The initial upload of normal connections usually
  - A **GET** request (little upload)
  - Large response in the form of web page from server
- Hybroid results show that analyzing flow metadata would be effective on encrypted flows too.

Category	HTTP Flow	TLS Flow
Adware	52.00%	8.00%
Ransomware	29.22%	0.00%
Scareware	61.38%	10.89%
SMSmalware	52.20%	10.28%

Type of malware category communication networks

Evaluation and Dataset

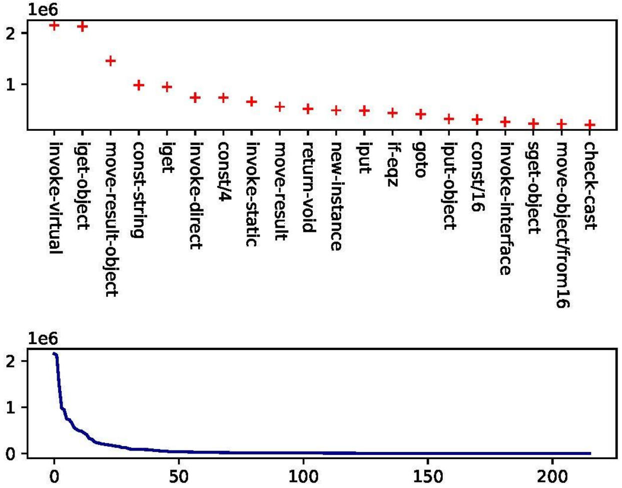
- We set up our experiments on our Euklid server with 32 Core Processor, 128 GB RAM, and 16 GB GPU.
- Python, Scikit-Learn, Tensorflow, and Keras.
- 5-fold cross-validation
  - we averaged the results of the cross-validation tests, executed each time with a new random dataset shuffle.

Name	Number	Description	Distribution(%)
APK files	2,126	All program code files	100%
PCAP files	2,126	All the raw network traffic files	100%
Benign APKs	1,700	No. of benign APK	80%
Adware APKs	124	No. of Adware category APK	5.9%
Ransomware APKs	112	No. of Ransomware category APK	5.2%
Scareware APKs	109	No. of Scareware category APK	5.2%
SMSmalware APKs	101	No. of SMSmalware category APK	4.7%

CICAndMal2017 Dataset[1]

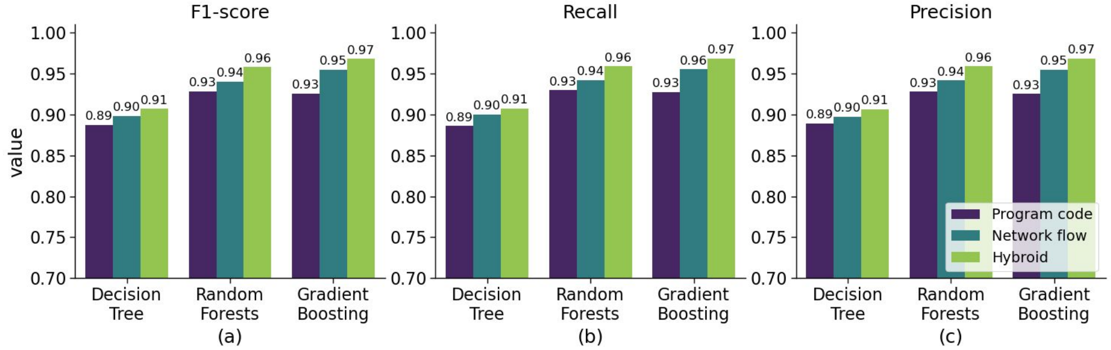
1- https://www.unb.ca/cic/datasets/andmal2017.html

Power Law

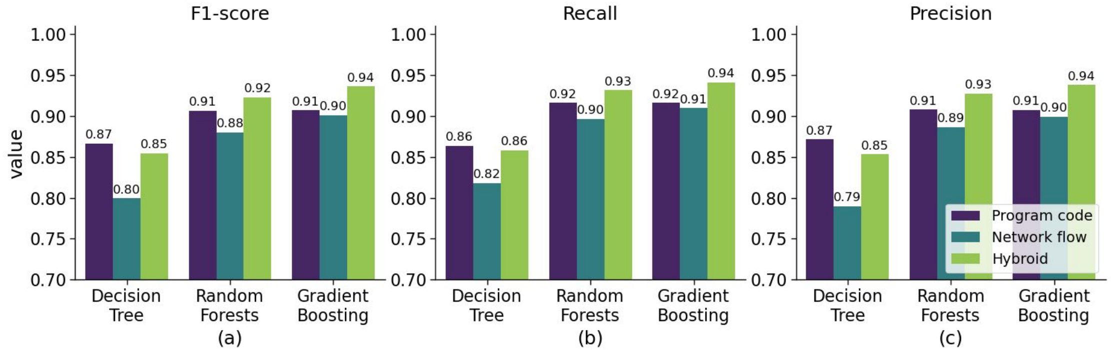


Power-law distribution for Dalivk opcodes

Results Performance

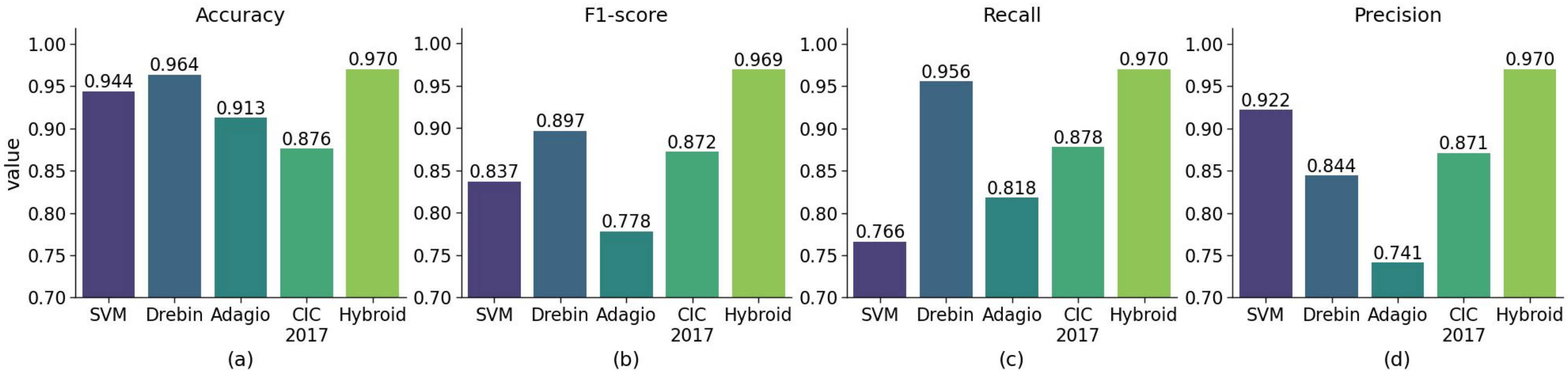


Binary Classification - Detection



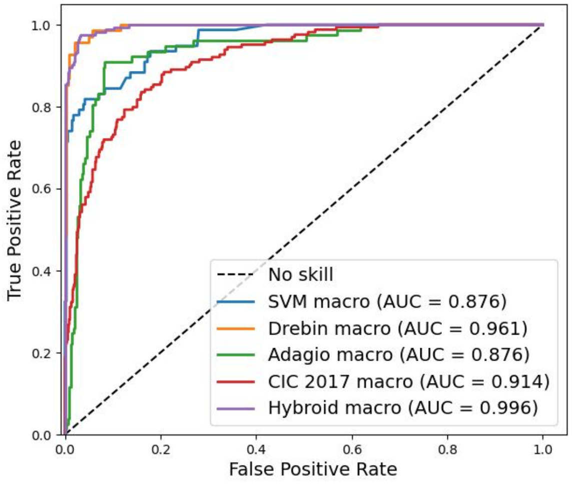
Multi-label Classification - Categorization

Related Works on Binary Classification

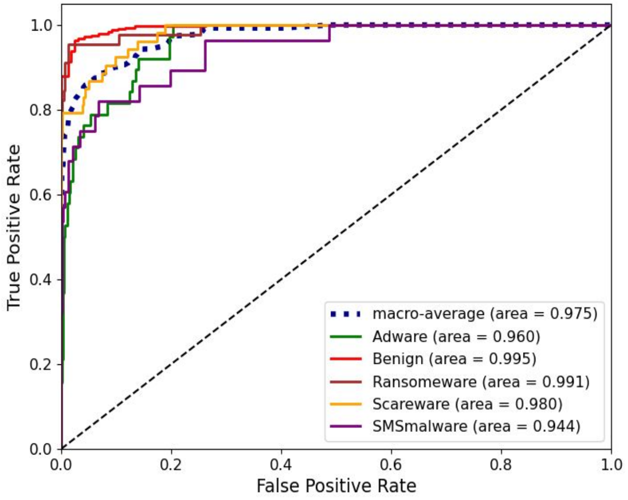


Malware detection overall performance of different related works

ROC Curve Results



Malware detection ROC curve of different related works



Malware categorization ROC curve of gradient boosting

Limitation and Future Work

- Lack of labeled data for CICAndMal2017
- Larger dataset
- Tested Hybroid on 45,592 malware and 90,313 benign samples
  - AndroidZoo[1]
  - VirusTotal[2]
  - VirusShare[3]
  - The accuracy and F1-score of 95.0% and 96.0% respectively

1- https://androzoo.uni.lu/
2- https://www.virustotal.com/gui/
3- https://virusshare.com/

Conclusion

- Summary
- Limitations
    - Lack of labeled data for CICAndMal2017
    - Larger dataset
    - Tested Hybroid on 56000 samples from

Acknowledgement

- It has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreements No. 830892 (SPARTA), No. 883275 (HEIR), and No. 833115 (PREVISION).

Thanks!!!

Discussions?