深圳大学计算机与软件学院
College of Computer Science and Software Engineering of Shenzhen University

ORCAS: Obfuscation-Resilient Binary Code Similarity Analysis using Dominance Enhanced Semantic Graph

Yufeng Wang, Yuhong Feng, Yixuan Cao, Haoran Li, Haiyue Feng, Yifeng Wang
College of Computer Science and Software Engineering,
Shenzhen University

Yufeng Wang
November 12, 2025

Background

Binary Code Similarity Analysis (BCSA)

- To score the semantic similarity of two binary code snippets.

Applications

- Malware identification, vulnerability detection, and plagiarism detection, etc.

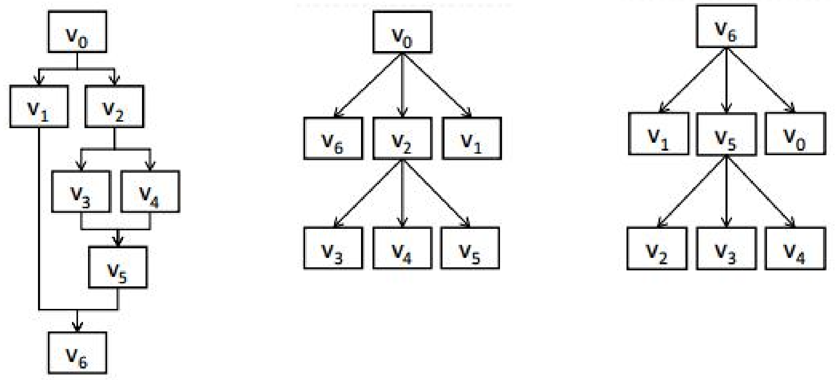
Code Obfuscation

- **Obscures** the code's control flow and basic blocks while preserving the function semantics.
- **Key Techniques** includes Instruction Substitution (SUB), Bogus Control Flow (BCF), and Control Flow Flattening (FLA).
- **Dual Purpose:** Protects software (e.g., 50% of top Google Play apps) and conceals malware.

Motivation

Challenges of code Obfuscation

- Simple instructions are replaced with complex instructions.
- The numbers of basic block surges (e.g., 4x), and control flow becomes unstable.



(a) A CFG G (b) G's D-Tree (c) G's PD-Tree

Fig 1. Before obfuscation.

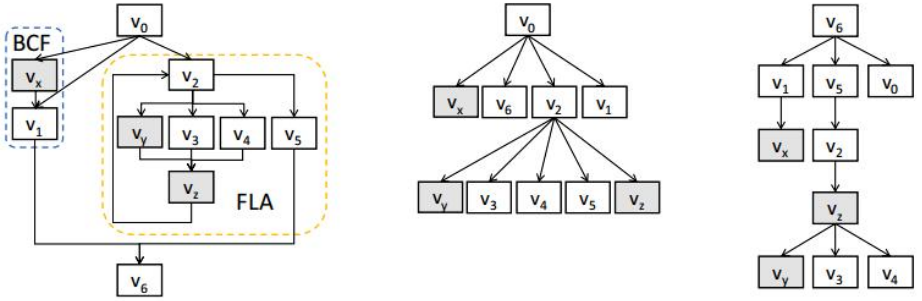
Dominator Tree (D-Tree)

- Identify the key nodes from the **entry node** to a specific node.

Post-Dominator Tree (PD-Tree)

- Identify the key nodes from a specific node to the **exit node**.

Superior Stability



(d) $\tilde{G}$: The Obfuscated G (e) $\tilde{G}$'s D-Tree (f) $\tilde{G}$'s PD-Tree

Fig 2. After obfuscation.

Quantify stability difference

- D-Trees demonstrate superior stability, with a 23.6% lower average GED[1] than CFGs.

Tab 1. Average GED across different sizes of basic blocks.

Basic Block Size Range	(0,50]	(50,100]	(100,150]	>150
CFG	207	553	916	1084
D-Tree	162	431	734	828

1 Graph Edit Distance (GED) measures the minimum number of edit operations required to transform one graph into another.

Contributions

- Propose an original Dominance Enhanced Semantic Graph (DESG) embedding for representing a binary function.
- Develop ORCAS, a novel Obfuscation-Resilient BCSA model, capturing more binaries’ implicit semantics without control flow structure, for more robust BCSA.
- Construct an original obfuscated real-world vulnerability dataset.
- ORCAS outperforms 8 baselines over the existing dataset and our constructed dataset.

ORCAS

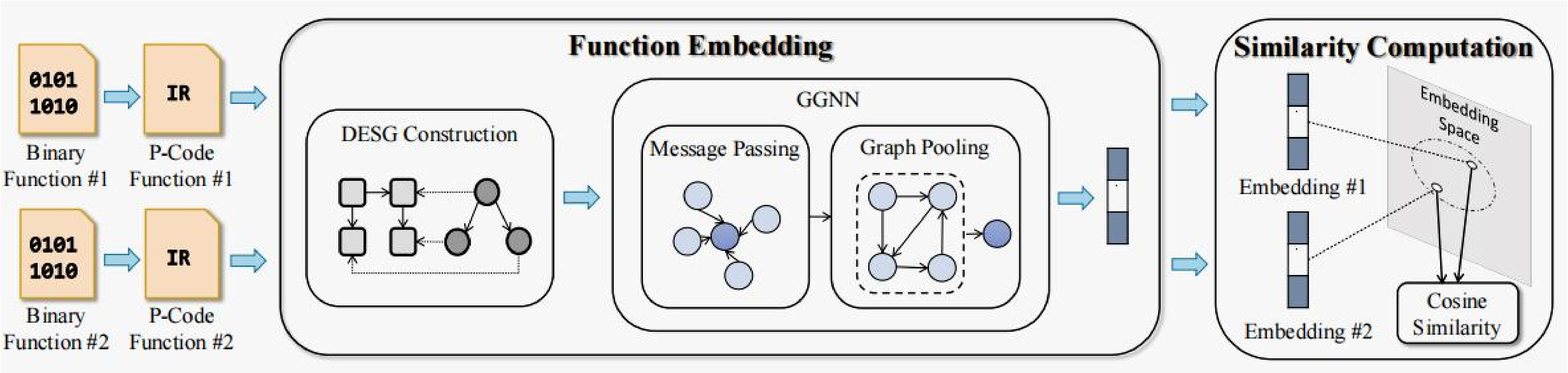


Fig 3: The framework of ORCAS.

DESG Construction

- Inter-basic block relations
  - Dominance
  - Post-Dominance
- Inter-instruction relations
  - Data
  - Effect
- Instruction-basic block relations
  - Contain

GGNN

$$\boldsymbol{x}_u^k = \mathrm{LayerNorm}(\mathrm{ReLU}(\mathrm{Linear}_k(\boldsymbol{h}_u^r)))$$

$$\boldsymbol{e}_g^k = \mathrm{Softmax}(\{\boldsymbol{x}_u^k \mid u \in \mathcal{N}\} \mid \beta^k)$$

$$\boldsymbol{e}_g = \mathrm{Linear}(\mathrm{Concat}(\boldsymbol{e}_g^1, \boldsymbol{e}_g^2, \ldots, \boldsymbol{e}_g^h))$$

Intermediate Representation

Ghidra

- Translate instructions into a sequence of P-Code operations.
- A P-Code operation takes one or more varnodes as input and may produce a single output varnode.

Varnode

- A generalized abstraction of a register or memory location, represented by a formal triple: (**address space type, offset, size**).

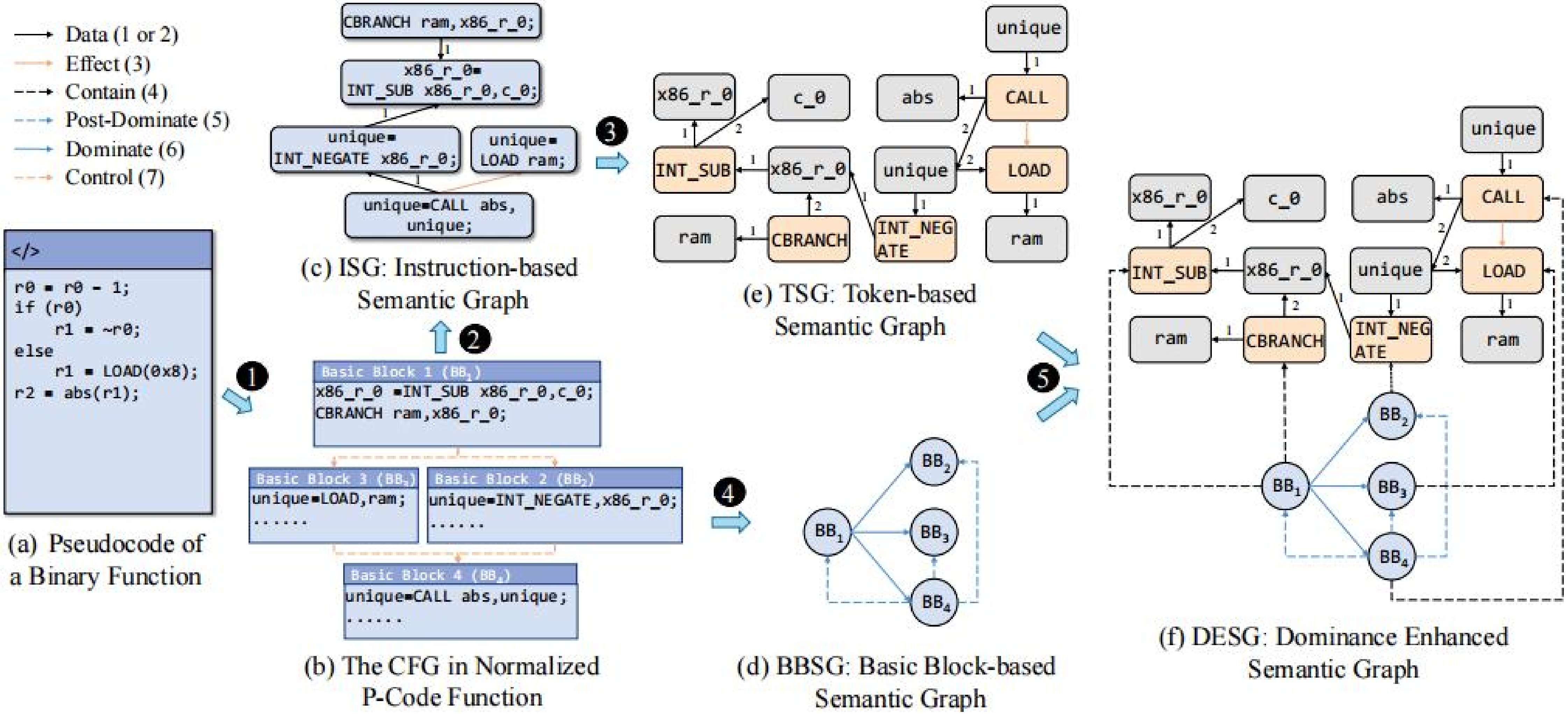
Tab 2: An example of varnode normalization strategies.

Address Space	Before Normalization	After Normalization
Ram	(ram,0x8,8)	ram or abs
Register	(register,0x0,4)	x86_r_0 or ARM_r_0
Constant	(const,0x0,4)	c_0
Unique	(unique,0x0,4)	unique
Stack	(stack,0xf,8)	stack

Varnode normalization

- Ram: Normalize to library function name or ram
- Register: Retain offset and incorporate an architecture identifier
- Constant: Normalize to c_offset
- Unique: Normalize to unique
- Stack: Normalize to stack

DESG Construction



Node

- Virtual basic block
- Opcode
- Oprand

Edge

- Data
- Effect
- Contain
- Dominance
- Post-Dominace

DESG Construction Algorithm

❶ Decompile into P-Code and normalize varnodes.

❷ Split basic blocks into individual instructions to build the ISG.

❸ Tokenize opcodes and operands to refine the ISG into the TSG.

❹ Create virtual basic block nodes with dominance and post-dominance relations to form the BBSG.

❺ Link TSG and BBSG with contain relations to construct the DESG.

Algorithm 1: The construction algorithm of the DESG

Input: A binary function f.
Output: A DESG $\mathcal{G}_{\text{DESG}}$ for representing the binary function f.

Let $\mathcal{G}_{\text{DESG}}$, $\mathcal{G}_{\text{ISG}}$, and $\mathcal{G}_{\text{BBSG}}$ be empty graphs;
Decompile f into the CFG $\mathcal{G}_{\text{CFG}}$;
Apply varnode normalization strategies on $\mathcal{G}_{\text{CFG}}$;
for $block \in \mathcal{G}_{\text{CFG}}$ **do**
  for $inst \in$ GetInstruction($block$) **do**
    $\mathcal{G}_{\text{ISG}}$.addNode($inst$);
    for $op \in$ the source operands of $inst$ **do**
      if op is defined by instruction $defInst$ **then**
        $\mathcal{G}_{\text{ISG}}$.addDataEdge($inst$, $defInst$);
        if $defInst$ accesses memory **then**
          $\mathcal{G}_{\text{ISG}}$.addEffectEdge($inst$, $defInst$);
$\mathcal{G}_{\text{TSG}} \leftarrow \mathcal{G}_{\text{ISG}}$;
for $inst \in \mathcal{G}_{\text{TSG}}$ **do**
  Split the opcode of $inst$ as a node opc;
  Refine the effect edge of $inst$ into the effect edge of opc;
  Split the target operand of $inst$ as a node op;
  $\mathcal{G}_{\text{TSG}}$.addDataEdge(op, opc);
  for $op \in$ the source operands of $inst$ **do**
    Split op as a node;
    $\mathcal{G}_{\text{TSG}}$.addDataEdge(opc, op);
$\mathcal{T}_{\text{D}} \leftarrow$ getDominatorTree($\mathcal{G}_{\text{CFG}}$);
$\mathcal{T}_{\text{PD}} \leftarrow$ getPostDominatorTree($\mathcal{G}_{\text{CFG}}$);
for $block \in \mathcal{G}_{\text{CFG}}$ **do**
  Create a virtual basic block node bb_i for $block$;
  $\mathcal{G}_{\text{BBSG}}$.addNode($bb_i$);
  $bb_j \leftarrow \mathcal{T}_{\text{D}}$.getPredecessor($bb_i$);
  $\mathcal{G}_{\text{BBSG}}$.addDomainceEdge($bb_j$, bb_i);
  $bb_j \leftarrow \mathcal{T}_{\text{PD}}$.getPredecessor($bb_i$);
  $\mathcal{G}_{\text{BBSG}}$.addPostDominanceEdge($bb_j$, bb_i);
Merge $\mathcal{G}_{\text{TSG}}$ and $\mathcal{G}_{\text{BBSG}}$ into $\mathcal{G}_{\text{DESG}}$ using contain edges;

Model Training

Message Passing

$$\boldsymbol{m}_u^l = \sum_{v \in \mathrm{Edge}(u)} f(\boldsymbol{h}_u^l, \boldsymbol{h}_v^l, \boldsymbol{e}_{v,u}, \boldsymbol{e}_{u,v})$$

$$\boldsymbol{h}_u^{l+1} = \mathrm{GRU}(\boldsymbol{h}_u^l, \boldsymbol{m}_u^l)$$

Graph Pooling

$$\mathrm{Softmax}(\mathcal{H} \mid \beta) = \sum_{\boldsymbol{h}_u^l \in \mathcal{H}} \frac{\exp(\beta \odot \boldsymbol{h}_u^l)}{\sum_{\boldsymbol{h}_v^l \in \mathcal{H}} \exp(\beta \odot \boldsymbol{h}_v^l)} \odot \boldsymbol{h}_u^l$$

$$\boldsymbol{x}_u^k = \mathrm{LayerNorm}(\mathrm{ReLU}(\mathrm{Linear}_k(\boldsymbol{h}_u^r))) \qquad \boldsymbol{e}_g^k = \mathrm{Softmax}(\{\boldsymbol{x}_u^k \mid u \in \mathcal{N}\} \mid \beta^k)$$

$$\boldsymbol{e}_g = \mathrm{Linear}(\mathrm{Concat}(\boldsymbol{e}_g^1, \boldsymbol{e}_g^2, \ldots, \boldsymbol{e}_g^h))$$

Loss Function

$$\min_\theta \mathcal{L}(\theta) = \sum_{\langle \boldsymbol{p}, \boldsymbol{q} \rangle \in \mathcal{B}} (\mathrm{positive}(\boldsymbol{p}, \boldsymbol{q}) + \mathrm{negative}(\boldsymbol{p}))$$

$$\mathrm{positive}(\boldsymbol{p}, \boldsymbol{q}) = \max(1 - \cos(\boldsymbol{p}, \boldsymbol{q}) - m, 0)$$

$$\mathrm{negative}(\boldsymbol{p}) = \max(\cos(\boldsymbol{p}, \mathrm{sampling}(\boldsymbol{p})) - m, 0)$$

Experiment

Baselines

- SAFE [1]
- Gemini [2]
- Asm2vec [3]
- Graph Mathcing Network (GMN) [4]
- Trex [5]
- jTrans [6]
- CRABS-former [7]
- HermesSim [8]

Environment

- **OS:** Ubuntu 20.04 LTS
- **CPU:** Intel Xeon 32-core 2.90GHz
- **RAM:** 256GB
- **GPU:** 2 NVIDIA GeForce RTX 3090

[1] Luca Massarelli, et al. 2022. Function Representations for Binary Similarity. IEEE Transactions on Dependable and Secure Computing 19, 4 (July 2022), 2259–2273.

[2] Xiaojun Xu, et al. 2017. Neural network-based graph embedding for cross-platform binary code similarity detection. In Proceedings of the 2017 ACM SIGSAC conference on computer and communications security. 363–376.

[3] Steven HH Ding, et al. 2019. Asm2vec: Boosting static representation robustness for binary clone search against code obfuscation and compiler optimization. In 2019 ieee symposium on security and privacy (sp). IEEE, 472–489.

[4] Yujia Li, et al. 2019. Graph matching networks for learning the similarity of graph structured objects. In International conference on machine learning. PMLR, 3835–3845.

[5] Kexin Pei, et al. 2023. Learning Approximate Execution Semantics From Traces for Binary Function Similarity. IEEE Trans. Softw. Eng. 49, 4 (Apr. 2023), 2776–2790.

[6] Hao Wang, et al. 2022. jTrans: Jump-aware transformer for binary code similarity detection. In Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis. 1–13.

[7] Yuhong Feng, et al. 2024. CRABS-former: CRoss-Architecture Binary Code Similarity Detection based on Transformer. In Proceedings of the 15th Asia-Pacific Symposium on Internetware. 11–20.

[8] Haojie He, et al. 2024. Code is not Natural Language: Unlock the Power of Semantics-Oriented Graph Representation for Binary Code Similarity Detection. In 33rd USENIX Security Symposium (USENIX Security 24). Philadelphia, PA, 1759–1776.

Evaluation

- **RQ1**: How performance is ORCAS on one-to-many similarity searching for binary functions?
- **RQ2**: How accurate is ORCAS on one-to-one similarity matching of obfuscated binary function?
- **RQ3**: How performance is the impact of incorporating dominance analysis on the performance of ORCAS?
- **RQ4**: How effective is ORCAS for real-world vulnerability detection?

RQ1: Searching against all Binary Functions

Subtask

- Cross-ISA (XA)
- Cross-optimization level (XO)
- Cross-ISA, optimization level, and obfuscation option (XM).

Pool size: 10 - 2000

Metric

- $\text{Recall@1} = \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}\left(\text{Rank}\left(f_i^{gt}\right) \leq 1\right)$

$$\mathbb{I}(x) = \begin{cases} 1, \text{ if } x = \text{True} \\ 0, \text{ if } x = \text{False} \end{cases}$$

- Mean Reciprocal Rank (MRR)

$$\text{MRR} = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{\text{Rank}\left(f_i^{gt}\right)}$$

Improvement

- XA : 9.6% ↑
- XO: 12.8% ↑
- XM: 9.6% ↑

Improvement

- XA : 7.4% ↑
- XO: 11.0% ↑
- XM: 8.4% ↑

Tab 3: Results on XA

Models	Pool size = 10		Pool size = 100		Pool size = 1,000		Pool size = 2,000	
	Recall@1	MRR	Recall@1	MRR	Recall@1	MRR	Recall@1	MRR
SAFE	0.506	0.685	0.133	0.269	0.023	0.069	0.011	0.042
Gemini	0.741	0.845	0.367	0.516	0.133	0.232	0.099	0.173
Asm2vec	-	-	-	-	-	-	-	-
GMN	0.788	0.879	0.326	0.509	0.069	0.168	0.040	0.108
Trex	0.455	0.594	0.193	0.305	-	-	-	-
jTrans	0.467	0.668	0.089	0.225	0.014	0.049	0.008	0.030
CRABS-former	0.836	0.904	0.549	0.671	0.308	0.412	0.242	0.337
HermesSim	0.948	0.968	0.870	0.909	0.654	0.761	0.544	0.680
ORCAS	**0.961**	**0.974**	**0.913**	**0.937**	**0.744**	**0.825**	**0.640**	**0.754**

Tab 4: Results on XO subtask.

Models	Pool size = 10		Pool size = 100		Pool size = 1,000		Pool size = 2,000	
	Recall@1	MRR	Recall@1	MRR	Recall@1	MRR	Recall@1	MRR
SAFE	0.405	0.613	0.085	0.210	0.010	0.044	0.007	0.027
Gemini	0.527	0.681	0.144	0.274	0.033	0.080	0.015	0.048
Asm2vec	0.433	0.646	0.100	0.210	0.034	0.065	0.022	0.046
GMN	0.610	0.755	0.197	0.359	0.032	0.100	0.020	0.064
Trex	0.387	0.532	0.122	0.211	-	-	-	-
jTrans	0.401	0.604	0.111	0.224	0.019	0.060	0.031	0.060
CRABS-former	0.624	0.761	0.293	0.424	0.120	0.201	0.085	0.147
HermesSim	0.900	0.934	0.788	0.842	0.579	0.677	0.476	0.595
ORCAS	**0.937**	**0.957**	**0.860**	**0.896**	**0.690**	**0.769**	**0.604**	**0.705**

Tab 5: Results on XM subtask.

Models	Pool size = 10		Pool size = 100		Pool size = 1,000		Pool size = 2,000	
	Recall@1	MRR	Recall@1	MRR	Recall@1	MRR	Recall@1	MRR
SAFE	0.510	0.685	0.123	0.252	0.030	0.070	0.016	0.045
Gemini	0.473	0.646	0.157	0.275	0.058	0.103	0.046	0.078
Asm2vec	-	-	-	-	-	-	-	-
GMN	0.726	0.832	0.319	0.481	0.055	0.150	0.032	0.097
Trex	0.302	0.460	0.154	0.240	-	-	-	-
jTrans	0.277	0.511	0.036	0.128	0.003	0.019	0.001	0.013
CRABS-former	0.509	0.682	0.160	0.292	0.036	0.089	0.027	0.069
HermesSim	0.908	0.939	0.816	0.859	0.642	0.723	0.557	0.660
ORCAS	**0.946**	**0.963**	**0.878**	**0.907**	**0.737**	**0.804**	**0.653**	**0.744**

RQ2: Matching Obfuscated Binary Functions

Purpose

- Evaluate the ability to match obfuscated binary functions with their normal version.

Obfuscation Technique

- BCF
- SUB
- FLA
- SUB+BCF+FLA (ALL)

1:100 ratio of positive to negative sample

Metric - Area Under the PR Curve

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

Improvement

- BCF: 4.3% ↑
- FLA: 4.7% ↑
- SUB: 2.3% ↑
- ALL: 12.1% ↑

Tab 6: PR-AUC socres of matching against obfuscated binary functions

Models	Bogus Control Flow (BCF)					Control Flow Flattening (FLA)					Instruction Substitution (SUB)					SUB+BCF+FLA (ALL)				
	Readline	Sed	Sharutils	Tar	Avg.	Readline	Sed	Sharutils	Tar	Avg.	Readline	Sed	Sharutils	Tar	Avg.	Readline	Sed	Sharutils	Tar	Avg.
SAFE	0.072	0.079	0.095	0.078	0.081	0.080	0.094	0.075	0.074	0.080	0.142	0.142	0.153	0.153	0.147	0.054	0.059	0.055	0.046	0.053
Gemini	0.062	0.089	0.083	0.063	0.074	0.026	0.026	0.025	0.022	0.024	0.515	0.557	0.570	0.548	0.547	0.018	0.020	0.017	0.016	0.017
Asm2vec	0.013	0.021	0.028	0.027	0.022	0.011	0.012	0.013	0.010	0.011	0.292	0.422	0.371	0.499	0.396	0.007	0.007	0.007	0.007	0.007
GMN	0.329	0.282	0.359	0.301	0.317	0.165	0.180	0.229	0.184	0.189	0.513	0.488	0.604	0.529	0.533	0.164	0.137	0.184	0.092	0.144
jTrans	0.055	0.067	0.126	0.108	0.089	0.063	0.071	0.100	0.097	0.082	0.373	0.459	0.524	0.570	0.481	0.015	0.022	0.020	0.027	0.021
CRABS-former	0.040	0.070	0.057	0.088	0.063	0.015	0.016	0.012	0.028	0.017	0.620	0.686	0.755	0.772	0.708	0.014	0.014	0.011	0.011	0.012
HermesSim	0.805	0.892	0.905	0.913	0.878	0.827	0.914	0.910	0.917	0.892	0.908	0.925	0.981	0.959	0.943	0.684	0.850	0.823	0.699	0.764
ORCAS	**0.840**	**0.935**	**0.965**	**0.946**	**0.921**	**0.881**	**0.967**	**0.971**	**0.939**	**0.939**	**0.947**	**0.957**	**0.993**	**0.969**	**0.966**	**0.792**	**0.935**	**0.933**	**0.800**	**0.865**

RQ3: Ablation Experiment

ORCAS-RD

- Remove the dominance and post-dominance relations in the DESG.
- Retain all other components.

Metric - Area Under the PR Curve

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

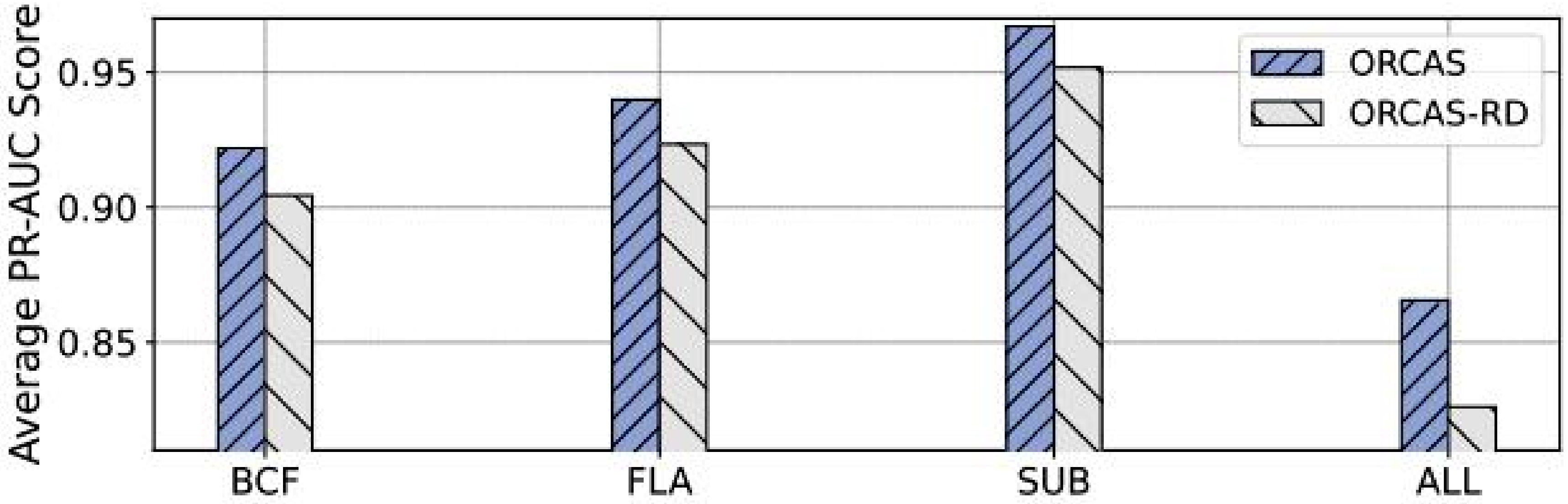


Fig 4: Average PR-AUC socres of ORCAS, ORCAS-RD

Improvement

- BCF : 1.7% ↑
- FLA: 1.7% ↑
- SUB: 1.6% ↑
- ALL: 4.0% ↑

RQ4: Real-World Vulnerability Search

Obfuscated real-world vulnerability dataset[2]

Setting

- 10 variants
  - 2 ISAs (x86-64 and ARM64)
  - 5 obfuscation options (none, SUB, BCF, FLA, and ALL)
- -O3 optimization option

Project

- openjpeg - 18,719 functions
- libgit2 - 57,673 functions

Recall@10 (m/10)

- Pool: All functions per CVE
- Search: 1 vulnerable variant as query
- Result: m = vulnerable variants in top-10

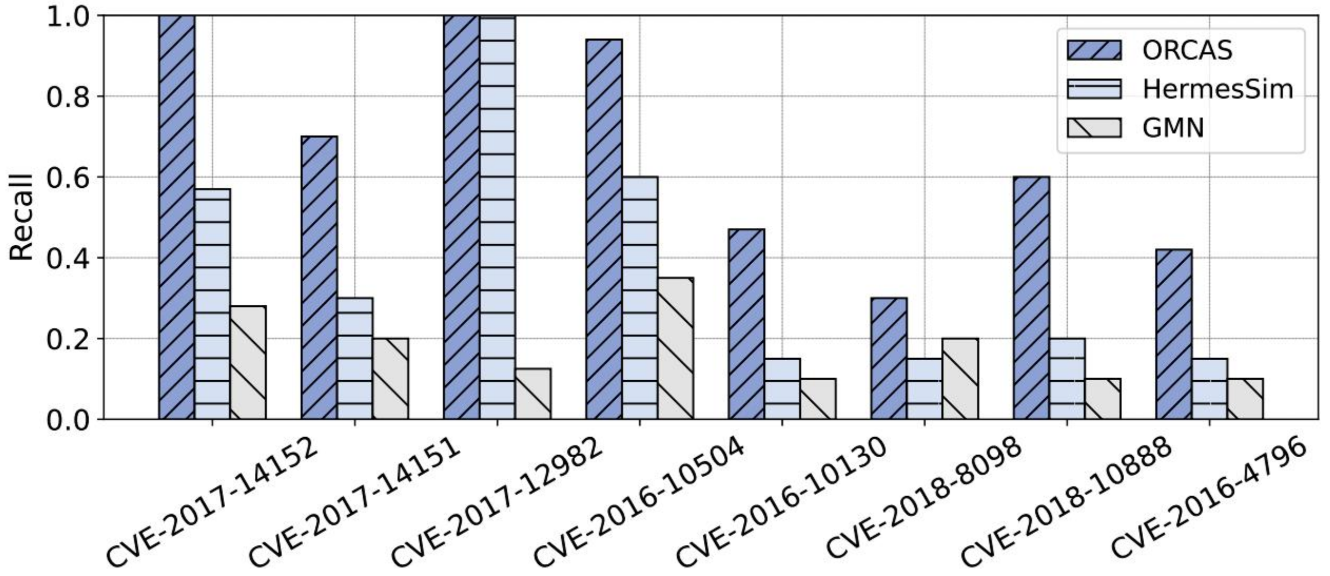


Fig 5: Recall rate of real-world vulnerability search.

2 https://github.com/Cao-Wuhui/ORCAS

Thanks for Listening
Q & A