CSE 4800: Thesis

Binary Function Clone Detection Using MBA Simplification

Presented By -

Tasnimul Hasnat (190041113)

Sumit Alam Khan (190041207)

Md Abdullahil Kafi (190041224)

Supervised By -

Dr. Md Moniruzzaman,
Assistant Professor, IUT

Imtiaj Ahmed Chowdhury,
Lecturer, IUT

Table of Contents

Function Cloning

Function cloning is slightly **modifying** a function so that the changes do not affect the function's **behavior**.

- Function cloning can be broadly categorized into two types:
  - ➜ **Optimization**
  - ➜ **Obfuscation**

Function Cloning (Optimization)

```
func1:
        mov   -20[rbp], edi
        mov   -24[rbp], esi
        mov   -8[rbp], 10
        mov   eax, -8[rbp]
        xor   eax, -20[rbp]
        sub   eax, -24[rbp]
        mov   -4[rbp], eax
        mov   eax, -24[rbp]
        imul  eax, -4[rbp]
        mov   edx, eax
        mov   eax, -20[rbp]
        add   eax, edx
        ret
```

No Optimization

```c
int func1(int a, int b){
      int  x;
      int  y;
      x = 10;
      y = (x ^ a) - b;
      return b * y + a;
}
```

Source Function

```
func1:
        mov   eax, edi
        xor   eax, 10
        sub   eax, esi
        imul  esi, eax
        lea   eax, [rsi+rdi]
        ret
```

O1 Optimization

Function Cloning (Obfuscation)

```c
int func1(int a, int b) {
    int x;
    int y;
    x = 10;
    y = (((x ^ a) & ~b) << 1)
- ((x ^ a) ^ b);
    return ((b * y - ~a) -1);
}
```

Equivalent Obfuscated Function

```asm
func1:
        mov   edx, edi
        xor   edx, 10
        mov   eax, esi
        not   eax
        and   eax, edx
        add   eax, eax
        xor   edx, esi
        sub   eax, edx
        imul  esi, eax
        not   edi
        sub   esi, edi
        lea   eax, -1[rsi]
        ret
```

Optimized and obfuscated

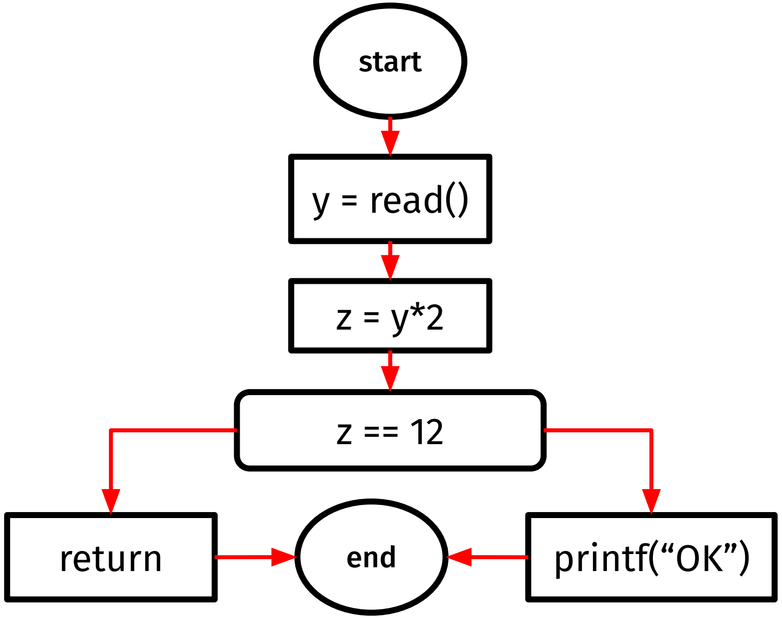
≈

```asm
func1:
        mov   eax, edi
        xor   eax, 10
        sub   eax, esi
        imul  esi, eax
        lea   eax, [rsi+rdi]
        ret
```

O1 Optimized Function

Control Flow Graph (CFG)

```
int f() {
  ...
  y = read();
  z = y * 2;
  if (z == 12) {
    return;
  } else {
    printf("OK");
  }
}
```

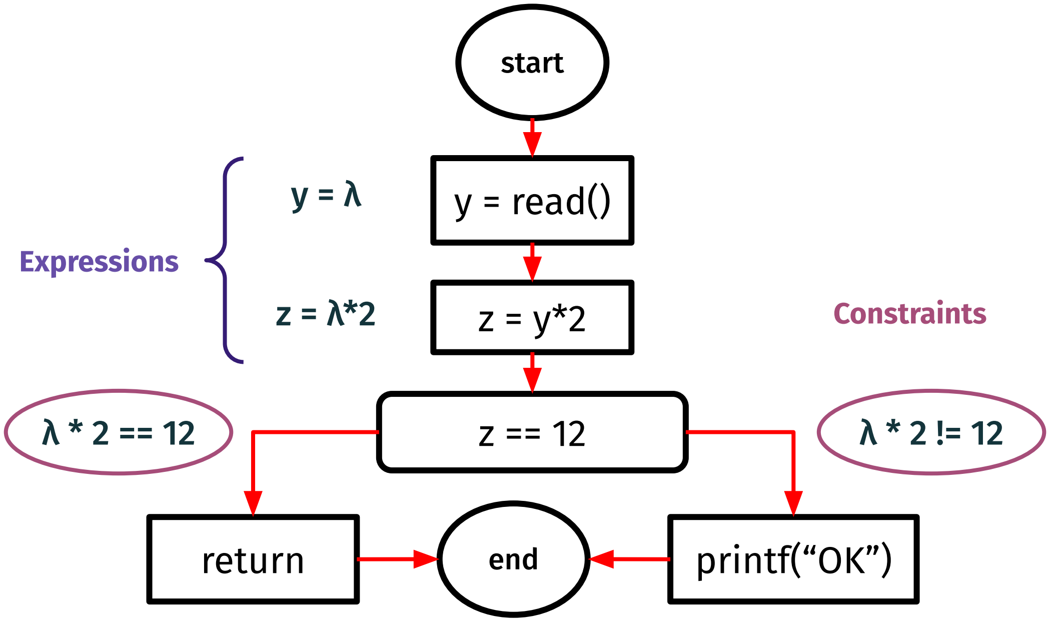


- Represents **flow of execution**
- **Nodes** → *Blocks of code*
- **Edges** → *Control flow*

Symbolic Execution

- **Explores** the CFG of a function with **symbolic** values rather than **concrete** values.

```
int f() {
  ...
  y = read();
  z = y * 2;
  if (z == 12) {
    return;
  } else {
    printf("OK");
  }
}
```



Mixed Boolean Arithmetic (MBA)

MBA Expression:

- Expression containing both traditional arithmetic operations (+, −, ×, . . .) and bitwise boolean operators (∧, ∨, ¬, ⊕, . . .)

$$\sum_{i \in I} a_i e_i(x_1, \ldots, x_t)$$

where,

x_t → variables

a_i → constant coefficient

$e_i(x_1 \ldots x_t)$ → bitwise expressions

Figure: Formal Definition of MBA Expression[1]

[1] **MBA-Blast: Unveiling and Simplifying Mixed Boolean-Arithmetic Obfuscation** in the Proceedings of USENIX Security Symposium, 2021.

Mixed Boolean Arithmetic (MBA)

MBA Expression:

- A sequence of assembly instructions can be represented as MBA expressions.

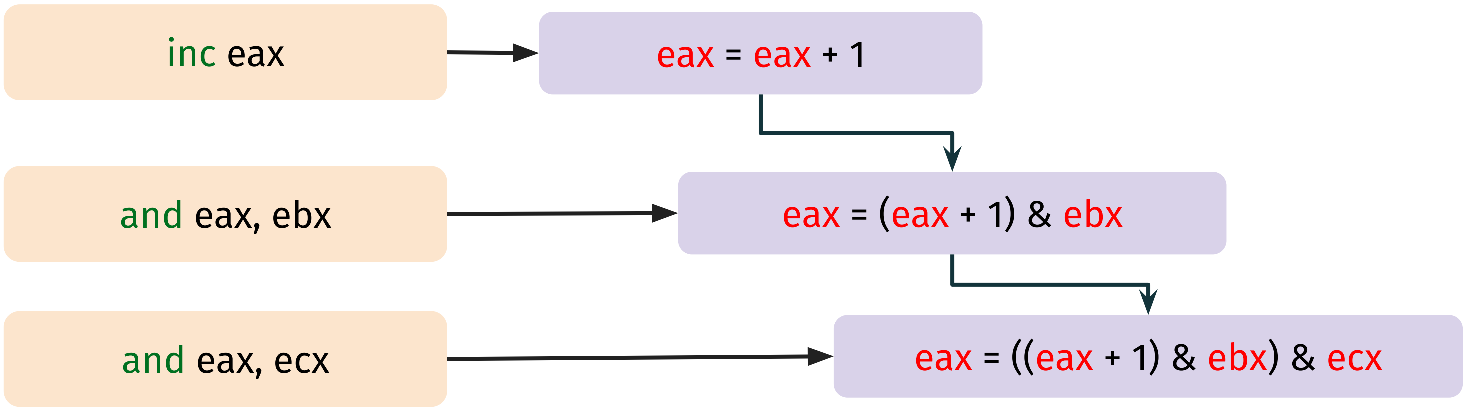


Figure: Symbolic Execution on Assembly Instructions representing MBA Expressions.

Satisfiability Solver

- Operates by taking a **logical expression** as input.
- Determines whether there exists an **assignment** to the variables that makes the logical expression **satisfiable or not**.
- Can **simplify** logical expressions while determining satisfiability.
- Can also show **equivalence** between different MBA expressions.

Limitations of existing literature

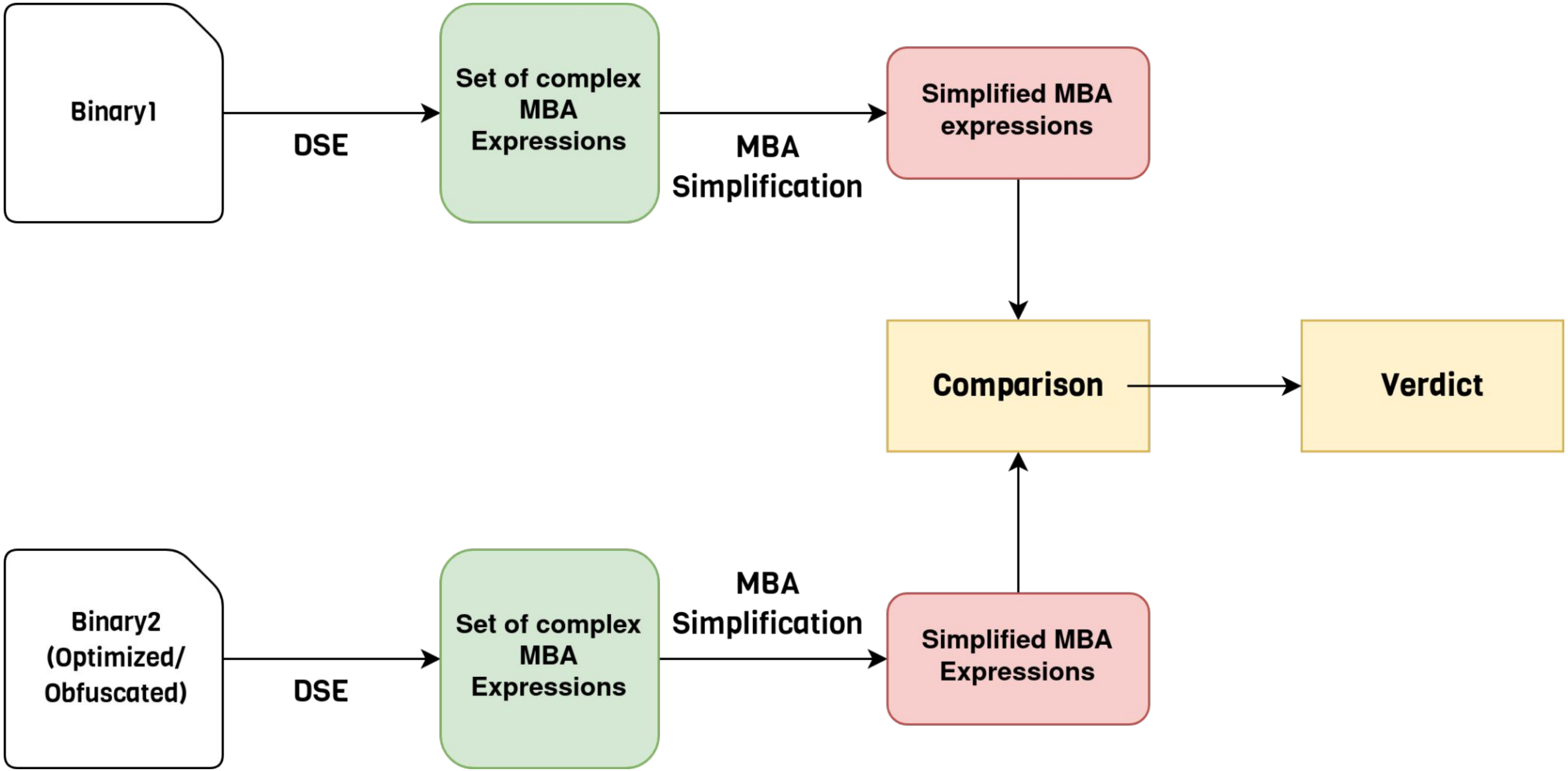
Current solutions are mostly semantic based **Machine Learning** algorithms.
They,

- Incur unnecessary **overhead**.
- Work only for **small** and **medium** sized binaries.
- Doesn’t work well against **obfuscation** techniques.
- Need to train the model from the **beginning** when **new data** is introduced.

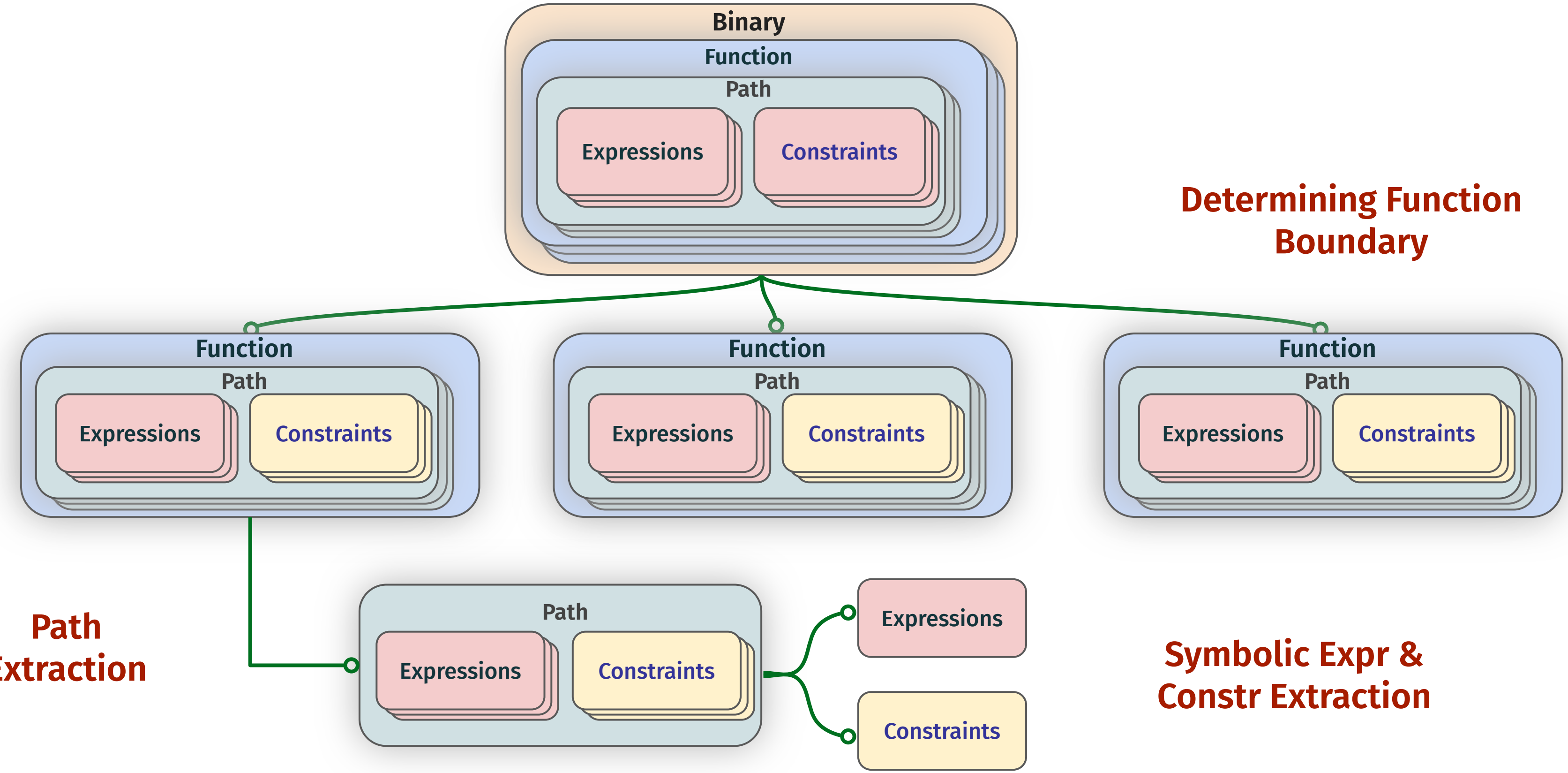
Problem Statement

Given two binary files identify binary similarity through function clone detection using MBA simplification

High Level Overview



Methodology

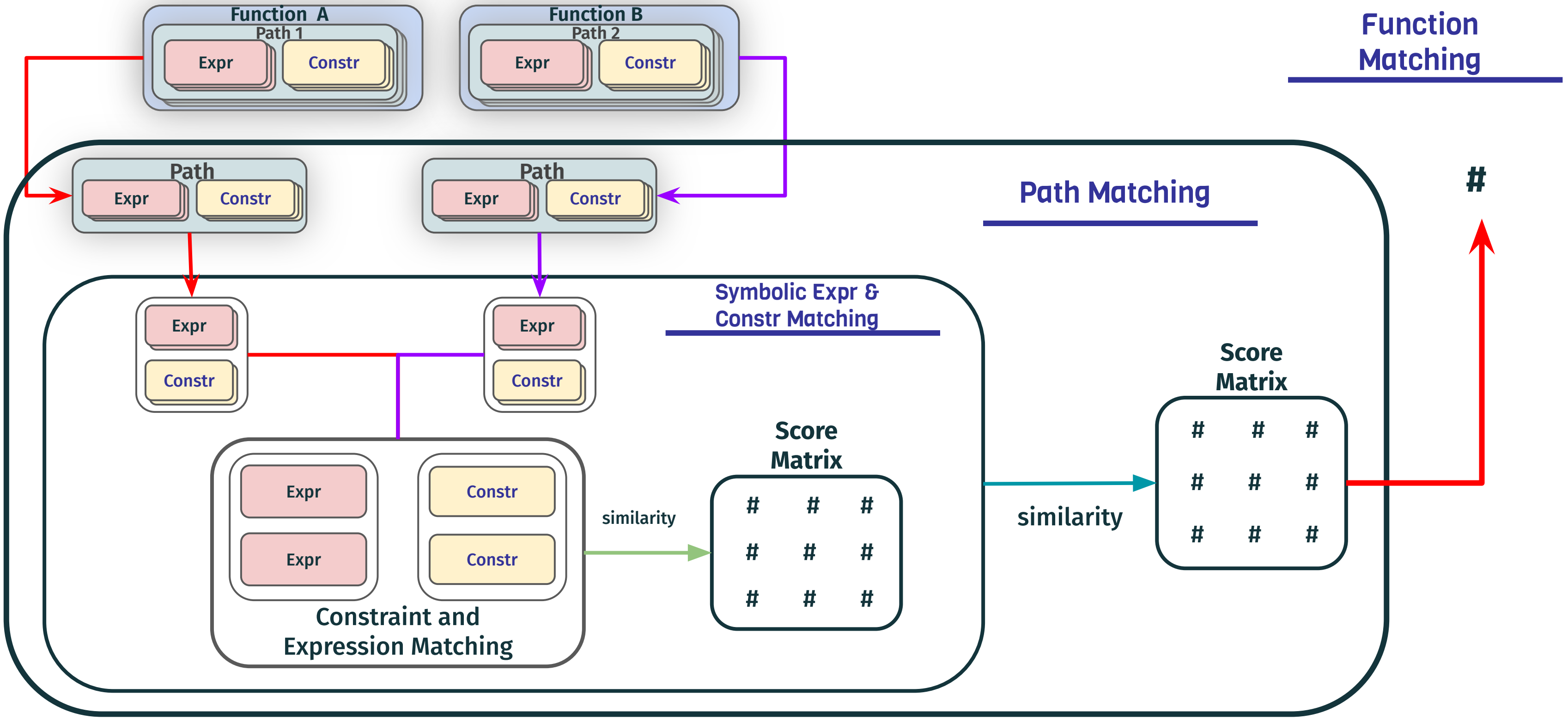


Determining Function Boundary

Path Extraction

Symbolic Expr & Constr Extraction

Methodology



Constraint and Expression Matching

- **Longest Common Subsequence (LCS)** Based Approach
- **Abstract Syntax Tree (AST) Isomorphism** Based Approach
- **Satisfiability Solver** Based Approach
    - **Angr Z3[2] Backend**

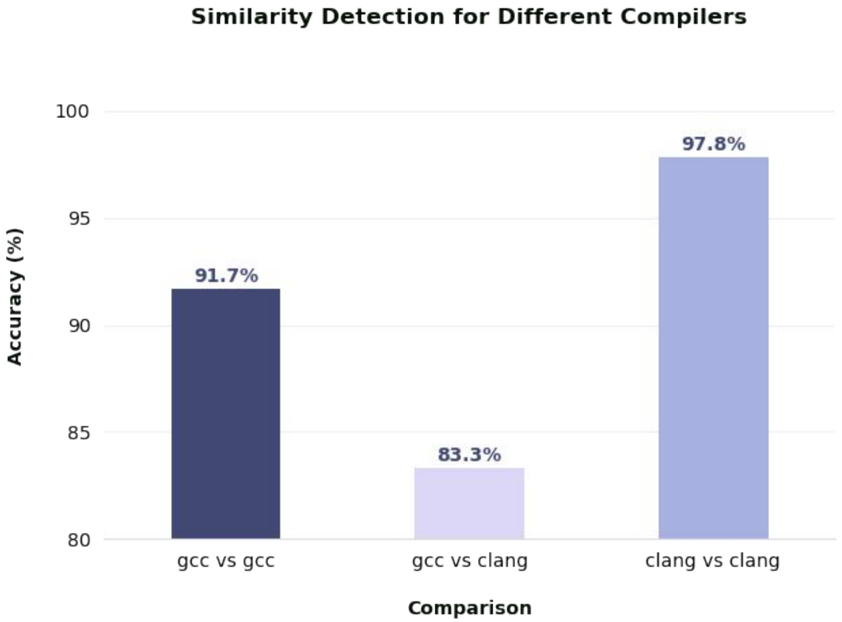
[2] **Z3: An Efficient SMT Solver**
in the Proceedings of International Conference on Tools and Algorithms for the Construction and Analysis of Systems, 2008.

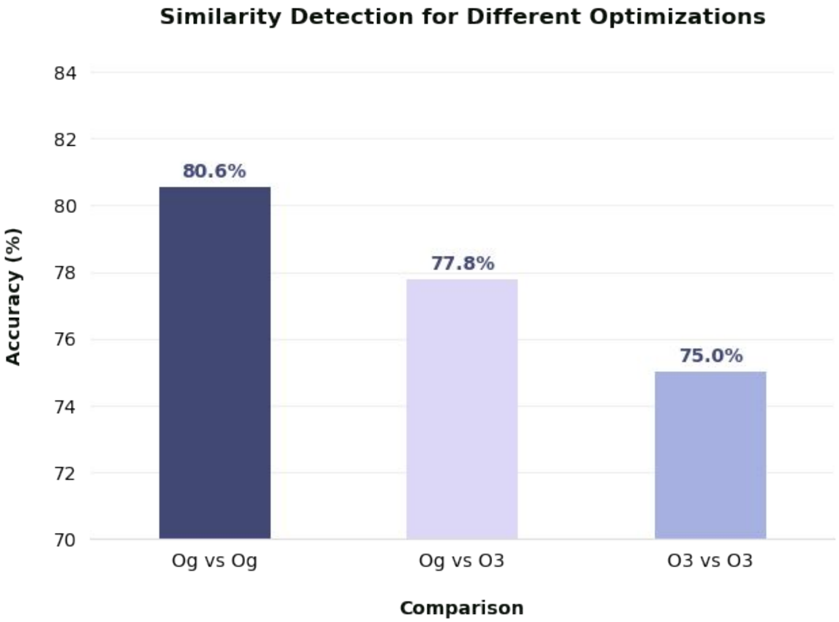
Experimental Results

Experimental Setup

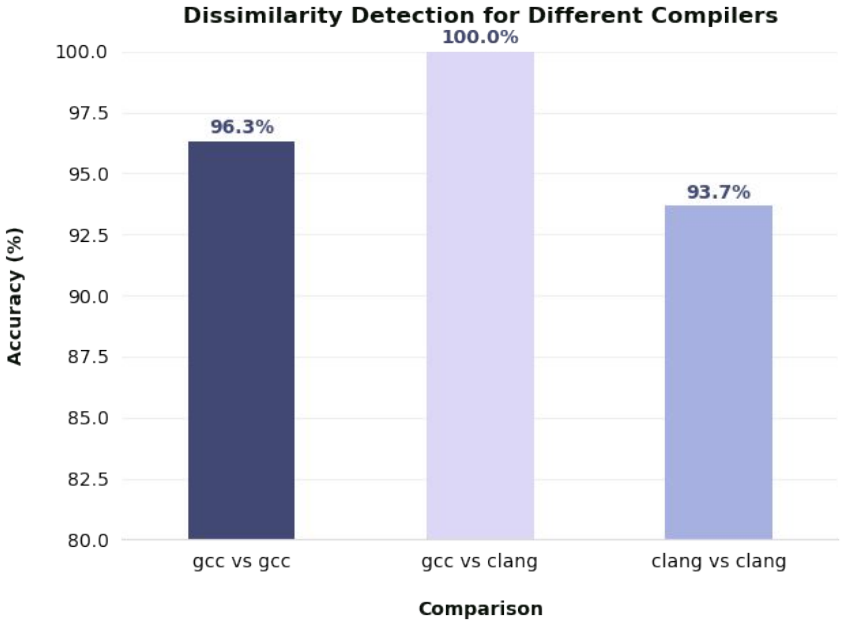
1. Collected source codes from the **Coreutils** library.
2. Compiled binaries using both **gcc** and **clang** compiler.
3. Compiled the binaries with multiple **optimization** levels **(Og, O1, O3)** for each compilers.
4. Collected obfuscated binaries from various sources e.g custom codes, CTF challenges etc.
5. The experiments were ran on a computer with **128GB RAM** and **Intel core i9 12900K cpu**.

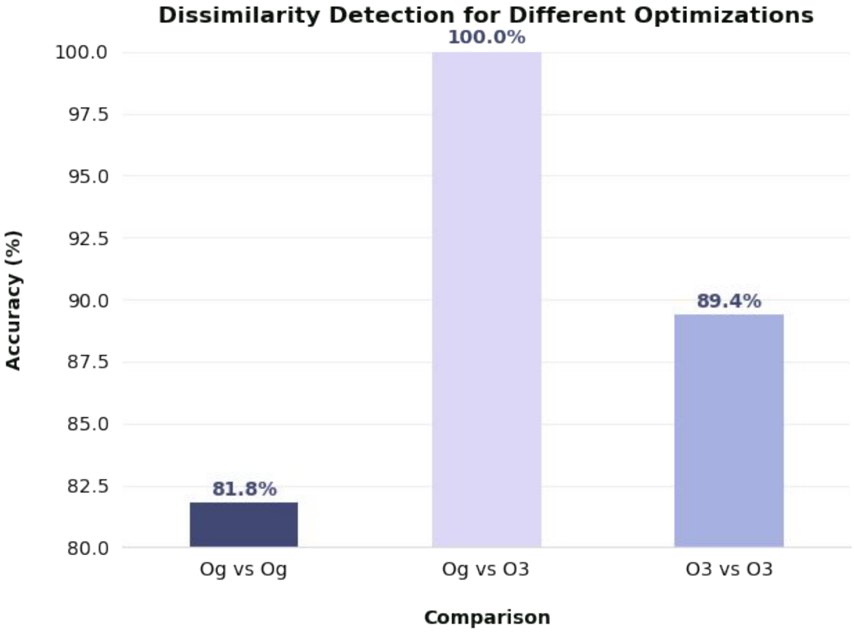
Similarity Detection





Dissimilarity Detection





Comparison with Existing Works

BinDiff	Graphlet	GeneDiff	LSH-S	Our Approach
0.938	0.695	0.961	0.86	0.8787

- BinDiff, GeneDiff and LSH-S has better accuracy but due to being machine learning based incurs huge overhead.
- Graphlet is a graph based approach but it is quite old. Our algorithm outperforms it significantly.

Conclusion

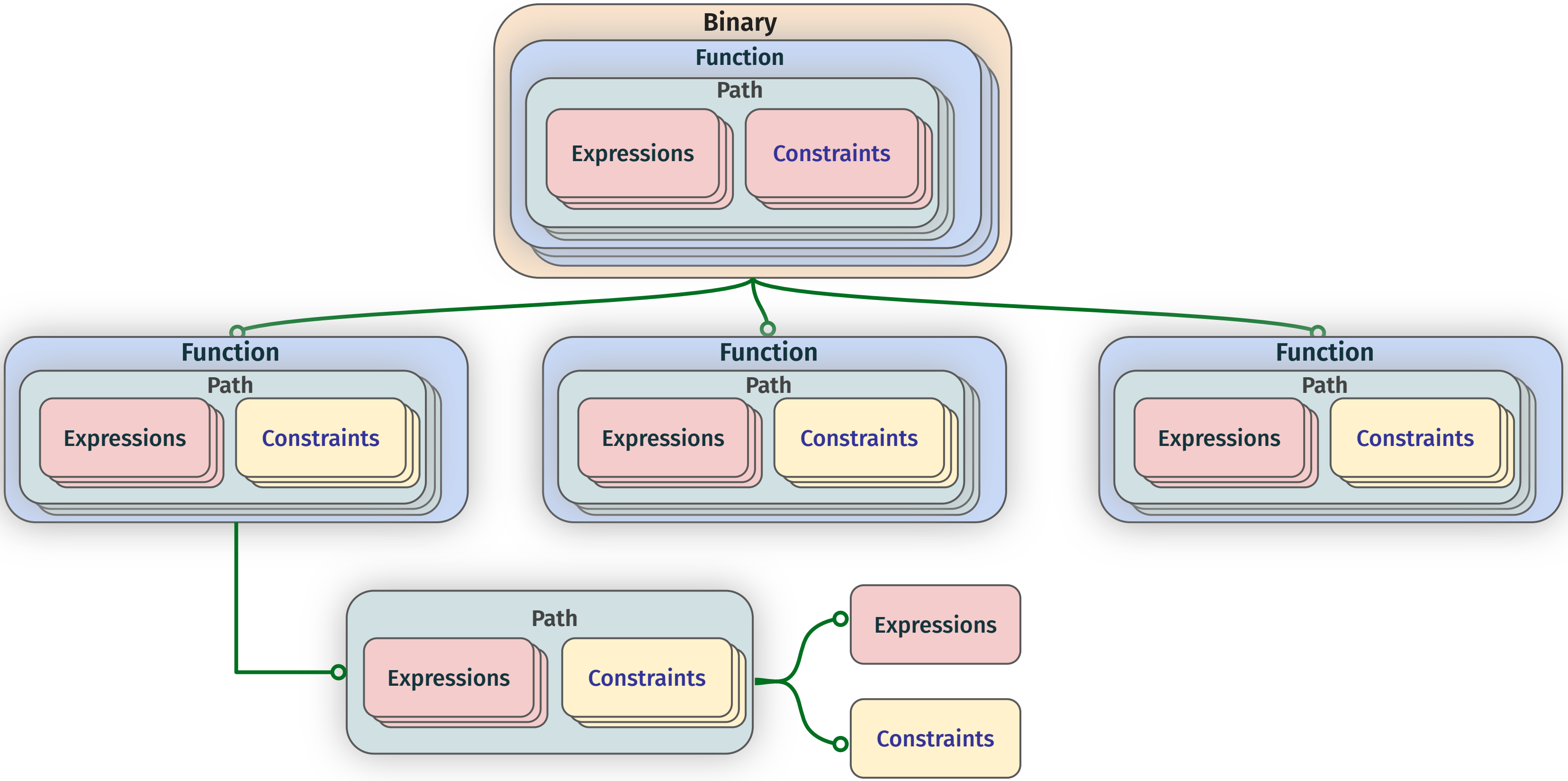
1. Managed to identify binary similarities with an overall accuracy of **87.87%**.
2. Identified but not solve for **conditional branching** with **side effects**, which is a scope for future works.
3. Decompiling 03 optimized binaries is very **error-prone** reducing accuracy.
4. Most of the time was spent to **prove satisfiability** between expressions. In future better SAT solvers can reduce operation time.

Thank You

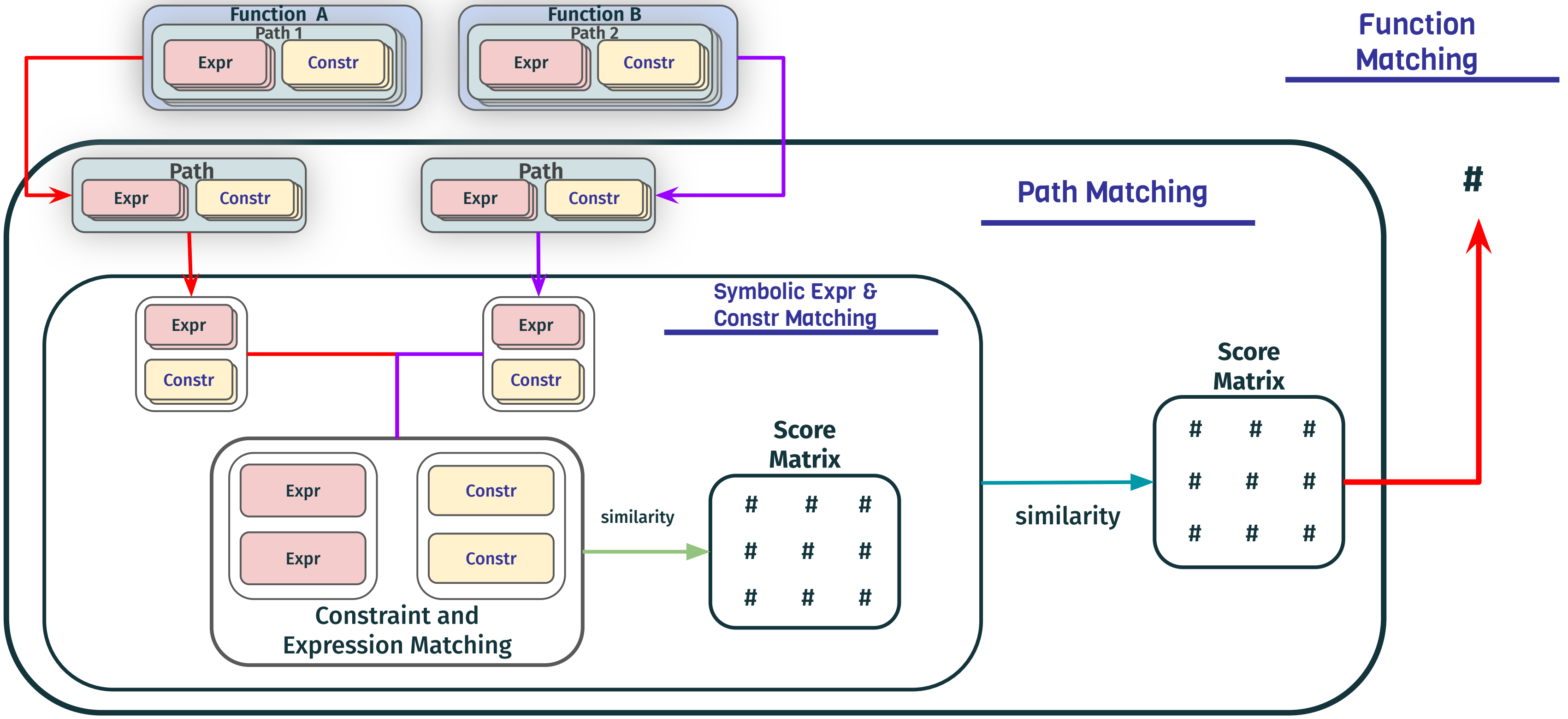
Challenges (WIP)

- Extracting MBA expressions from branches and loops.
- Comparing equivalence of set of simplified expressions
- Choosing the appropriate form of verdict (binary vs similarity metrics).

Methodology



Methodology



Experimental Results

Accuracy: 83.33%

33.33% True Positive	0% False Positive
50% True Negative	20% False Negative

Figure: Confusion Matrix for only gcc (Og,O3) binaries

Accuracy: 91.67%

41.67% True Positive	0% False Positive
50% True Negative	8.33% False Negative

Figure: Confusion Matrix for only clang (Og,O3) binaries

Experimental Results

Accuracy: 87.56%

33.67% True Positive	0% False Positive
50% True Negative	16.67% False Negative

Figure: Confusion Matrix for Og optimization on both gcc and clang binaries

Accuracy: 84.97%

41.67% True Positive	0% False Positive
50% True Negative	8.33% False Negative

Figure: Confusion Matrix for O3 optimization on both gcc and clang binaries

Experimental Results (WIP)

Accuracy: 87.56%

33.67% True Positive	0% False Positive
50% True Negative	16.67% False Negative

Figure: Confusion Matrix for both compilers and optimization levels

Motivation

- **Copyright infringement** is a serious issue in the software development industry.
- Verification and code **reuse identification** in closed-source applications is significantly difficult.
- Cybersecurity challenges increase as attackers **modify** the existing malwares to evade antivirus detection.
- Existing methodologies for finding function clone is mostly **Machine learning based** which is difficult to scale.
- A systematic method to find function clones from binary files could be **scalable** and easy to use.

Relevant Concepts

Reference

1 **MBA-Blast: Unveiling and Simplifying Mixed Boolean-Arithmetic Obfuscation**
in the Proceedings of USENIX Security Symposium, 2021.

2 **Binary Function Clone Search in the Presence of Code Obfuscation and Optimization over Multi-CPU Architectures**
ACM Asia Conference on Computer and Communications Security, July 2023

3 **Qsynth- A Program Synthesis based Approach for Binary Code Deobfuscation**
NDSS Symposium Binary Analysis Research Workshop, 2020

4 **Simplifying Mixed Boolean-Arithmetic Obfuscation by Program Synthesis and Term Rewriting**
ACM SIGSAC Conference on Computer and Communications Security, November 2023

Methodology (WIP)

1. Create testing dataset from the **TREX** binary database.
2. Determine the function boundaries and extract the functions.
3. Symbolically execute
4. Extraction of complex MBA expressions from obfuscated binary through dynamic symbolic execution.
5. Simplify the set of MBA expressions.
6. Check for **equivalency** between set of expressions using possible means e.g. SMT, fuzzing, program synthesis, graph isomorphism etc.

Existing Literature

Binary Function Clone Search in the Presence of Code Obfuscation and Optimization over Multi-CPU Architectures

ASIA CCS '23: Proceedings of the 2023 ACM Asia Conference on Computer and Communications Security, July 2023

Abdullah Qasem, *Concordia University*

Mourad Debbabi, *Concordia University*

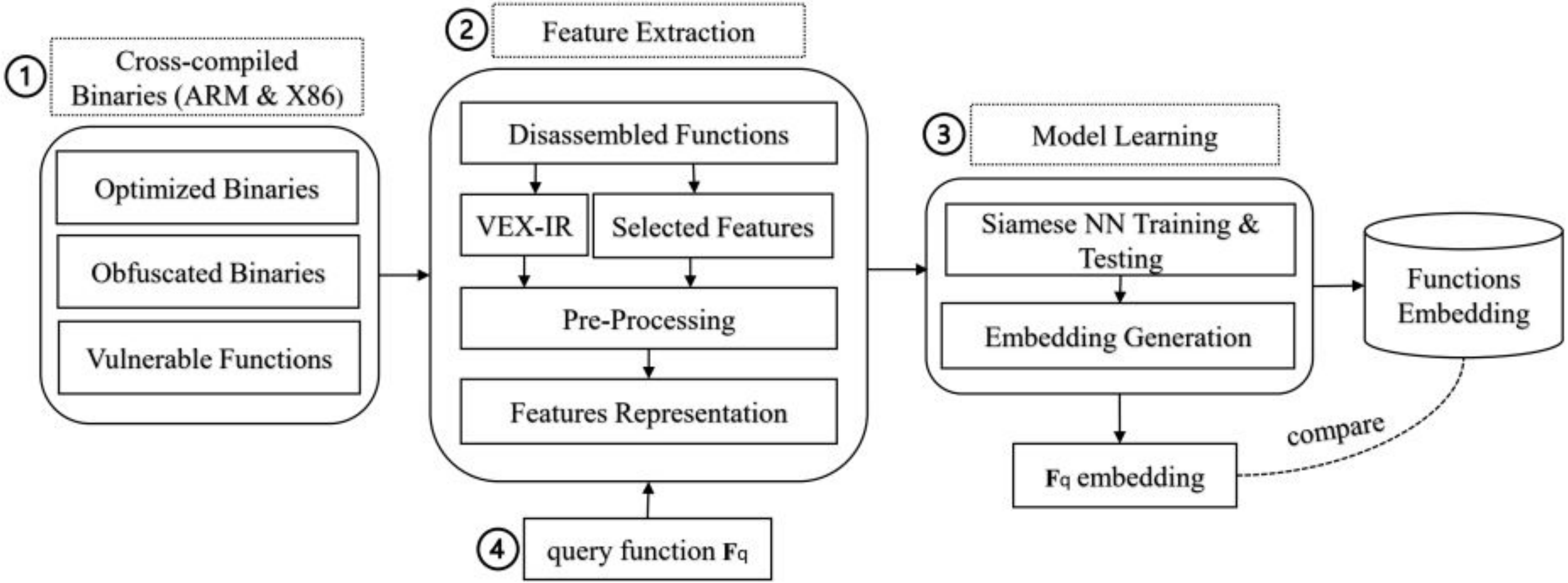
Bernard Lebel, *Thales Research and Technologies*

Marthe Kassouf, *Hydro-Québec Research Institute*

Contribution

- Developed and implemented **BinFinder.**
- Identified **resilient**, interpretable binary function features against code optimizations and obfuscations on **multi-CPU architectures**.
- Designed **Siamese** neural network architecture for training a model using proposed features to generate binary function **embeddings**.
- **Evaluated** BinFinder in 3 scenarios- x86 with different optimization and obfuscation, multi-cpu architecture with and without optimization and obfuscation.

BinFinder - Methodology



[2] **Binary Function Clone Search in the Presence of Code Obfuscation and Optimization over Multi-CPU Architectures**
ACM Asia Conference on Computer and Communications Security, July 2023

BinFinder - Evaluation

Approach	AUC				XM				
	XC	XC+XB	XA	XM	small	medium	large	MRR10	Recall@1
BinFinder	**0.98**	**0.97**	**0.98**	**0.98**	**0.98**	**0.98**	**0.93**	**0.8**	**0.73**
GMN_OPC-200_e16	0.86	0.85	0.86	0.86	0.89	0.82	0.79	0.53	0.45
GNN-s2v_GeminiNN_OPC-200_e5	0.78	0.81	0.82	0.81	0.84	0.77	0.79	0.36	0.28
SAFE_ASM-list_e5	0.8	0.8	0.81	0.81	0.83	0.77	0.77	0.17	0.27
Zeek	0.84	0.84	0.85	0.84	0.85	0.83	0.87	0.28	0.13
asm2vec	0.62	0.81	0.74	0.69	0.63	0.7	0.78	0.12	0.07

XA → Different Architecture, Same Obfuscation, Different Optimization.
XB → Same Architecture, Different Obfuscation.
XC → Different Architecture, Same Obfuscation, Same Optimization.
XM → Different Architecture, Different Obfuscation, Different Optimization.

[2] **Binary Function Clone Search in the Presence of Code Obfuscation and Optimization over Multi-CPU Architectures**
ACM Asia Conference on Computer and Communications Security, July 2023

Qsynth-
A Program Synthesis based Approach for Binary Code Deobfuscation

NDSS Symposium Binary Analysis Research Workshop, 2020

Robin David, *Quarkslab*

Luigi Coniglio, *University of Trento*

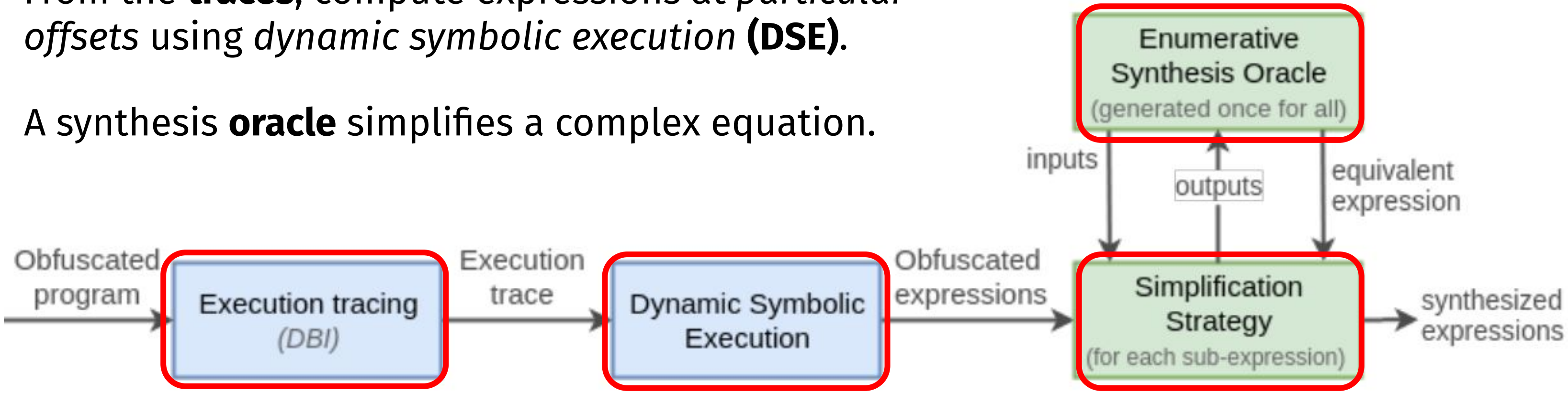
Mariano Ceccato, *University of Verona*

Contribution

- Introduced a combination of ***dynamic symbolic execution***, ***dataflow graph extraction***, and ***program synthesis*** against obfuscation.
- Proposed a **black-box synthesis** method based on ***offline enumerative search***.
- Utilized precomputed lookup tables for near **constant-time** expression synthesis.
- **Iteratively** simplified complex obfuscated expressions.
- Empirical **validation** through a comparison with the deobfuscation tool **Syntia**, demonstrating superior accuracy and speed (20x faster).

Qsynth - Methodology

1. Uses *dynamic binary instrumentation* **(DBI)** to trace paths in the binary.
2. From the **traces**, compute expressions at *particular offsets* using *dynamic symbolic execution* **(DSE)**.
3. A synthesis **oracle** simplifies a complex equation.



[3] **Qsynth- A Program Synthesis based Approach for Binary Code Deobfuscation**
NDSS Symposium Binary Analysis Research Workshop, 2020

Qsynth - Evaluation

	Mean expr. size			Simplification			Mean scale factor			Sem.	Time			
	Orig	Obf_B	Synt	∅	Partial	Full	Obf_S/Orig	Synt/Obf_B	Synt/Orig		Sym.Ex	Synthesis	Total	per fun.
Syntia	/	/	/	52	0	448	/	/	/	/	/	/	34 min	4.08s
QSynth	3.97	203.19	3.71	0	500	500	x35.03	x0.02	x0.94	500	1m20s	15s	1m35s	0.19s

Orig, Obf_S, Obf_B, Synt are respectively original, obfuscated (source, binary level) and synthesized expressions

[3] Qsynth- A Program Synthesis based Approach for Binary Code Deobfuscation
NDSS Symposium Binary Analysis Research Workshop, 2020

Simplifying Mixed Boolean-Arithmetic Obfuscation by Program Synthesis and Term Rewriting

CCS '23: Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, November 2023. Pages 2351–2365

Jaehyung Lee, *Hanyang University, Ansan, Korea*

Woosuk Lee, *Hanyang University, Ansan, Korea*

Contributions

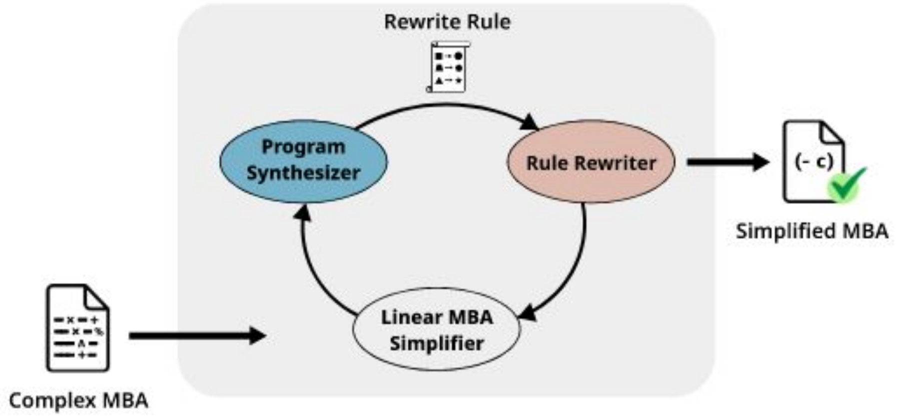
- Proposes a novel and versatile method called ***ProMBA*** for deobfuscating MBA expressions.
- Combines ***program synthesis***, ***term rewriting***, and ***algebraic simplification*** methods.
- Unlike existing techniques, ProMBA can deobfuscate a much **broader class** of MBA expressions.
- Implemented the proposed method in an ***open source tool***.
- The experimental results show that ProMBA outperforms the state-of-the-art deobfuscation method, ***MBASolver*** by a large margin.

ProMBA - Methodology

1 Proposed method **first** simplifies **linear** MBA sub-expressions using an off-the-shelf deobfuscator.

2 **Recursively** simplifies **non-linear** sub-expressions by synthesizing simpler sub-expressions.

3 Applies the resulting rewrite rules to other sub-expressions **until no** further **simplification** is possible.

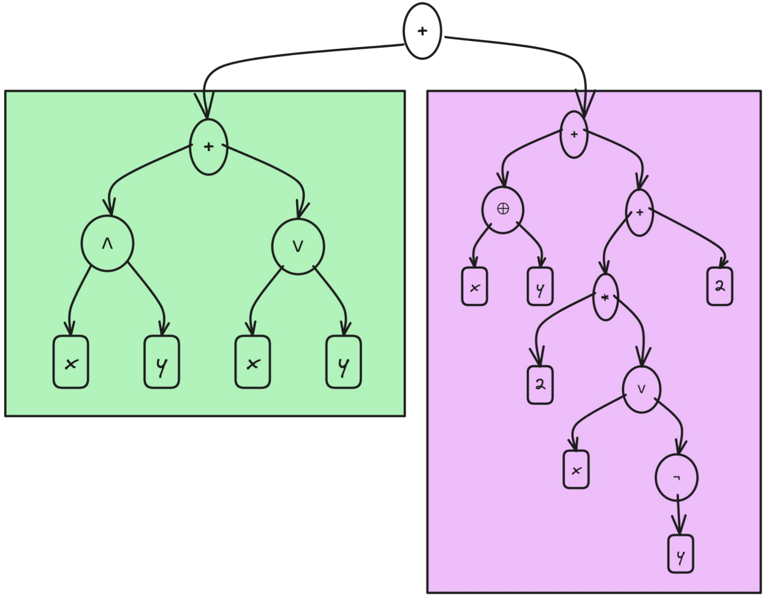


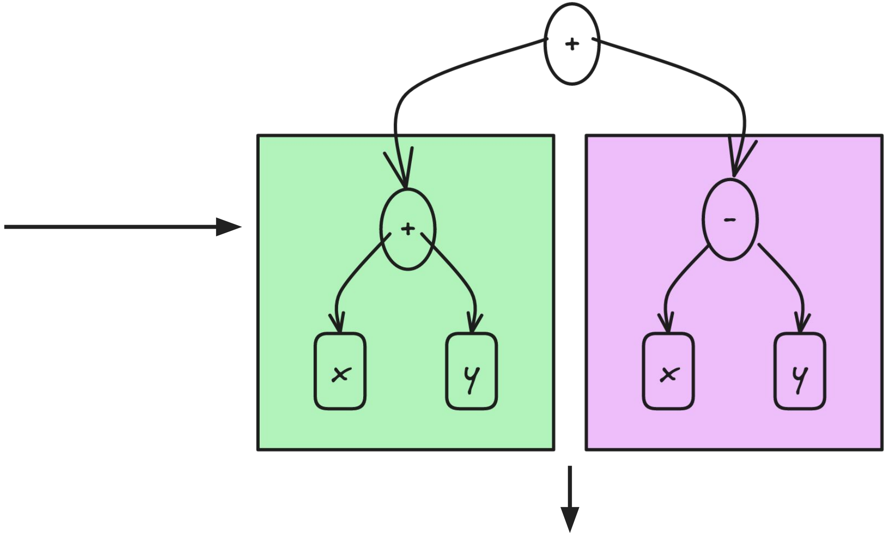
[4] **Simplifying Mixed Boolean-Arithmetic Obfuscation by Program Synthesis and Term Rewriting**
ACM SIGSAC Conference on Computer and Communications Security, November 2023

MBA Simplification

Known Rules:

1. $x + y == (x \land y) + (x \lor y)$
2. $x - y == x \oplus y + 2 * (x \lor \neg y) + 2$





$\{(x \land y) + (x \lor y)\} + \{x \oplus y + 2 * (x \lor \neg y) + 2\}$

$x + y + x - y \Rightarrow 2x$

ProMBA - Evaluation

Dataset	Size (Avg.)			Success Rate			Time (Avg.)		
	ProMBA	MBASolver	Syntia	ProMBA	MBASolver	Syntia	ProMBA	MBASolver	Syntia
MBA-Solver	11.76	21.67	**4.61**	**80.31%**	25.16%	17.45%	65.64s	**6.3s**	12.95s
QSynth	17.48	77.71	**4.72**	**62.8%**	4.2%	22.8%	241s	64.83s	**12.37s**
Loki	3.51	866.25	**3.1**	**97.2%**	0.13%	74.4%	100.03s	347.77s	**2.07s**
Total	9.39	344.5	**3.55**	**84.44%**	13.19%	39.42%	100.36s	141.29s	**8.81s**

[4] **Simplifying Mixed Boolean-Arithmetic Obfuscation by Program Synthesis and Term Rewriting**
ACM SIGSAC Conference on Computer and Communications Security, November 2023