QSynth - A Program Synthesis approach for Binary Code Deobfuscation

Binary Analysis Workshop - NDSS

Robin David <rdavid@quarkslab.com>

Luigi Coniglio <luigi.coniglio@studenti.unitn.it>

Mariano Ceccato <mariano.ceccato@univr.it>

February 23th, 2020 - San Diego, California

quarkslab

www.quarkslab.com

Talk Outline

Context:

- Need to address highly obfuscated binaries
- Few approaches address data obfuscation

Goal: deobfuscating expression *(obfuscated with data transformations)*

Talk Outline

Context:

- Need to address highly obfuscated binaries
- Few approaches address data obfuscation

Goal: deobfuscating expression *(obfuscated with data transformations)*

Takeway

We provide a **synthesis approach**
addressing various obfuscations
and that **supersede** the state-of-the-art in both
speed and **accuracy**

Table of Contents

Obfuscation types

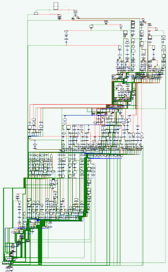
Control-Flow Obfuscation

Hiding the **logic** and algorithm of the program

Virtualization, Opaque predicates, CFG-flattening, Split, Merge, Packing, Implicit Flow, MBA, Loop-Unrolling...

Example

$\Rightarrow$



Obfuscation types

Control-Flow Obfuscation

Hiding the **logic** and algorithm of the program

Virtualization, Opaque predicates, CFG-flattening, Split, Merge, Packing, Implicit Flow, MBA, Loop-Unrolling...

Data-Flow Obfuscation

Hiding data, constants, strings, APIs, keys etc.

Data encoding, MBA, Arithmetic Encoding, Whitebox, Array Split, Fold and Merge, Variable Splitting...

Example

$$a + b \quad \Longrightarrow \quad (((((((a \wedge \neg b) + b) << 1) \wedge \neg((a \vee b) - (a \wedge b))) << 1) - ((((a \wedge \neg b) + b) << 1) \oplus ((a \vee b) - (a \wedge b))))$$

Obfuscation types

Control-Flow Obfuscation

Hiding the **logic** and algorithm of the program

Virtualization, Opaque predicates, CFG-flattening, Split, Merge, Packing, Implicit Flow, MBA, Loop-Unrolling...

Data-Flow Obfuscation

Hiding data, constants, strings, APIs, keys etc.

Data encoding, MBA, Arithmetic Encoding, Whitebox, Array Split, Fold and Merge, Variable Splitting...

Example

$$a + b \quad \Longrightarrow \quad ((((((a \wedge \neg b) + b) << 1) \wedge \neg((a \vee b) - (a \wedge b))) << 1) - ((((a \wedge \neg b) + b) << 1) \oplus ((a \vee b) - (a \wedge b))))$$

Problem: Reverting an obfuscating transformation is hard.

Deobfuscation

Let's focus on two deobfuscation techniques:

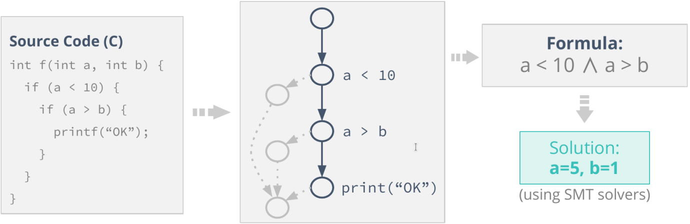
Dynamic Symbolic Execution

Program Synthesis

Symbolic Execution

Definition

Mean of executing a program using **symbolic values** *(logical symbols)* rather than real values *(bitvectors)* in order to obtain an **in-out relationship of a path**



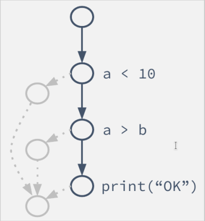
Symbolic Execution

Definition

Mean of executing a program using **symbolic values** *(logical symbols)* rather than real values *(bitvectors)* in order to obtain an **in-out relationship of a path**

Source Code (C)

```
int f(int a, int b) {
  if (a < 10) {
    if (a > b) {
      printf("OK");
    }
  }
}
```



Formula:
$a < 10 \wedge a > b$

Solution:
a=5, b=1
(using SMT solvers)

Dynamic Symbolic Execution (a.k.a. concolic)

- **Properties:** work on **dynamic paths** and **use runtime values**
- **Advantages:** path sure to be **feasible** and **thwart various obfuscations**

Symbolic Execution: Example

⇒ In this context used to extract symbolic expressions *(e.g. b)*

```
if (a > 0){
    b += (a | -1) - 1;
    b -= ((~ a) & -1);
} else {
    b += (a | -3) + 1;
}
b -= 1 + ((b * (b + 1)) % 2);
```

Symbolic State

Symbolic Execution: Example

⇒ In this context used to extract symbolic expressions *(e.g. b)*

```
if (a > 0){
    b += (a | -1) - 1;
    b -= ((~ a) & -1);
} else {
    b += (a | -3) + 1;
}
b -= 1 + ((b * (b + 1)) % 2);
```

Symbolic State
$\phi_b = b$

Symbolic Execution: Example

$\Rightarrow$ In this context used to extract symbolic expressions *(e.g. b)*

```
if (a > 0){
    b += (a | -1) - 1;
    b -= ((~ a) & -1);
} else {
    b += (a | -3) + 1;
}
b -= 1 + ((b * (b + 1)) % 2);
```

Symbolic State
$\phi_b = b$
$\phi_b = b + (a \mid -1) - 1$

Symbolic Execution: Example

$\Rightarrow$ In this context used to extract symbolic expressions *(e.g. b)*

```
if (a > 0){
    b += (a | -1) - 1;
    b -= ((~ a) & -1);
} else {
    b += (a | -3) + 1;
}
b -= 1 + ((b * (b + 1)) % 2);
```

Symbolic State
$\phi_b = b$
$\phi_b = b + (a \mid -1) - 1$
$\phi_b = b + (a \mid -1) - 1 - ((\sim a) \;\&\; -1)$

Symbolic Execution: Example

⇒ In this context used to extract symbolic expressions *(e.g. b)*

```
if (a > 0){
    b += (a | -1) - 1;
    b -= ((~ a) & -1);
} else {
    b += (a | -3) + 1;
}
b -= 1 + ((b * (b + 1)) % 2);
```

Symbolic State
$\phi_b = b$
$\phi_b = b + (a \mid -1) - 1$
$\phi_b = b + (a \mid -1) - 1 - ((\sim a)$ $\& -1)$
$\phi_b = b + (a \mid -1) - 1 - ((\sim a)$ $\& -1) - 1 + (((b + (a \mid -1)$ $-1 - ((\sim a) \& -1)) \times (b + \ldots$

Question: How to simplify the ϕ_b expression?

(Knowing that the quality of the result depends on the syntactic complexity of the obfuscated expression)

Program Synthesis

Definition

Program synthesis consists in automatically deriving a program from:

- a high-level specification *(typically its I/O behaviour)*
- additional constraints:
  - Compilation: a faster program
  - Deobfuscation: a smaller or more readable program

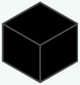
Program Synthesis

Definition

Program synthesis consists in automatically deriving a program from:

- a high-level specification *(typically its I/O behaviour)*
- additional constraints:
  - Compilation: a faster program
  - Deobfuscation: a smaller or more readable program

Example



Obfuscated Program

Input	**Output**

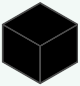
Program Synthesis

Definition

Program synthesis consists in automatically deriving a program from:

- a high-level specification *(typically its I/O behaviour)*
- additional constraints:
  - Compilation: a faster program
  - Deobfuscation: a smaller or more readable program

Example



Obfuscated Program

Input	**Output**
1, 2	3

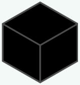
Program Synthesis

Definition

Program synthesis consists in automatically deriving a program from:

- a high-level specification *(typically its I/O behaviour)*
- additional constraints:
  - Compilation: a faster program
  - Deobfuscation: a smaller or more readable program

Example



Obfuscated Program

Input	Output
1, 2	3
2, 2	4

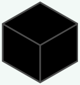
Program Synthesis

Definition

Program synthesis consists in automatically deriving a program from:

- a high-level specification *(typically its I/O behaviour)*
- additional constraints:
  - Compilation: a faster program
  - Deobfuscation: a smaller or more readable program

Example



Obfuscated Program

Input	Output
1, 2	3
2, 2	4
2, 3	5

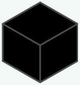
Program Synthesis

Definition

Program synthesis consists in automatically deriving a program from:

- a high-level specification *(typically its I/O behaviour)*
- additional constraints:
  - Compilation: a faster program
  - Deobfuscation: a smaller or more readable program

Example



Obfuscated Program

Input	**Output**
1, 2	3
2, 2	4
2, 3	5

$\Rightarrow$ $a + b$

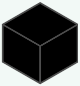
Program Synthesis

Definition

Program synthesis consists in automatically deriving a program from:

- a high-level specification *(typically its I/O behaviour)*
- additional constraints:
  - Compilation: a faster program
  - Deobfuscation: a smaller or more readable program

Example



Obfuscated Program

Input	Output
1, 2	3
2, 2	4
2, 3	5

$\Longrightarrow$ $a + b$

Problem

Synthesizing programs *(expressions)* with complex behaviors **is hard**.

Table of Contents

Key Intuition

Symbolic Execution

+ Capture full semantic
- Influenced by syntactic complexity

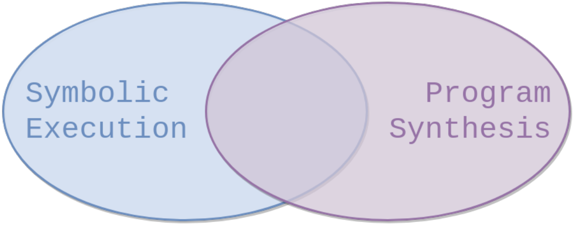
Key Intuition

Symbolic Execution

- \+ Capture full semantic
- \- Influenced by syntactic complexity

Program Synthesis

- \+ Only influenced by semantic complexity
- \- Black-box ⇒ big search space



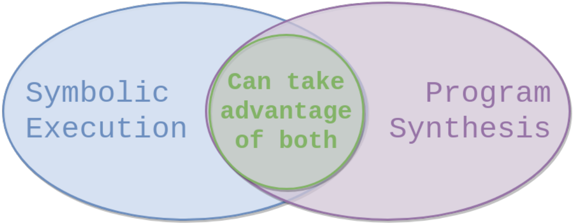
Key Intuition

Symbolic Execution

- \+ Capture full semantic
- \- Influenced by syntactic complexity

Program Synthesis

- \+ Only influenced by semantic complexity
- \- Black-box ⇒ big search space



Idea: Using symbolic execution to **reduce the synthesis search space**

Contributions

A synthesis approach using an

Offline Enumerative Search

based on pre-computed lookup tables

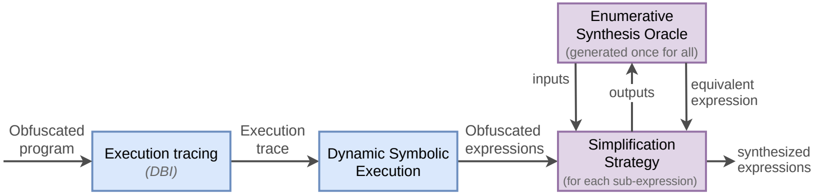
combined with an Abstract Syntax Tree

simplification algorithm

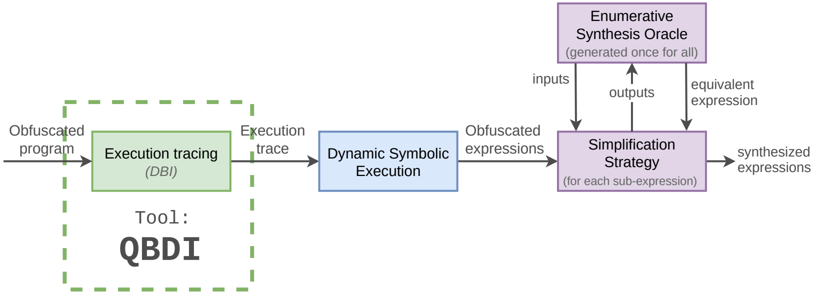
which **outperform** similar approach of

the state-of-the-art *(e.g. Syntia)*

QSynth: Overview



QSynth: Overview



Execution Tracing

Dynamic Binary Instrumentation

Using **QBDI**: QuarkslaB Dynamic binary Instrumentation *(similar to Pin, DynamoRIO)*

- \+ multi-architecture & platform
- \- no (direct) thread support

Qtracer (a qbditool like Pin ``pintools'')

- gather instruction executed with their concrete state *(registers and memory)*
- Data are consolidated in database *(SQLite, PostgresSQL etc.)*

Original

```
mov qword [0x000232c0], 8
mov r13, rax
test rax, rax
je 0x42a7
xor r8d, r8d
xor edx, edx
xor esi, esi
```

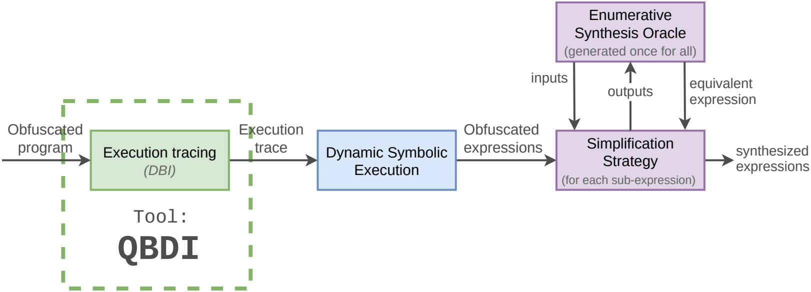
Instrumentation

```
mov qword [0x000232c0], 8
; Some code ...
mov r13, rax
; Some code ...
test rax, rax
; Some code ...
je <patched address>
; Some code ...
xor r8d, r8d
; Some code ...
xor edx, edx
; Some code ...
xor esi, esi
; Some code ...
```

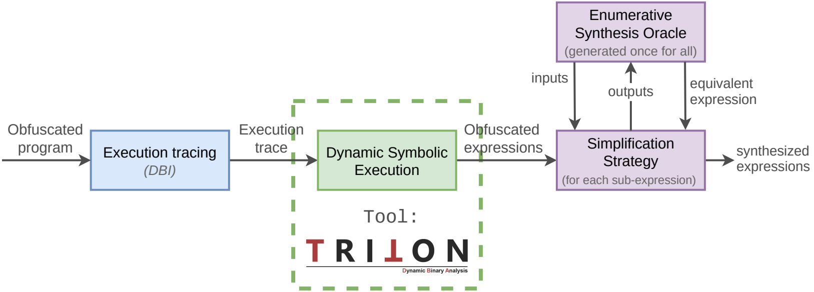
Instrumented

https://qbdi.quarkslab.com/

QSynth: Overview



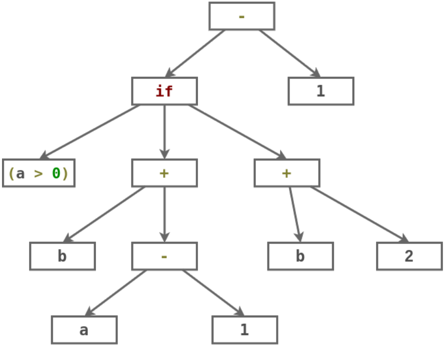
QSynth: Overview



DSE: Symbolic expression computation

⇒ Triton allows computing any **symbolic expression** along the trace by backtracking on data dependencies

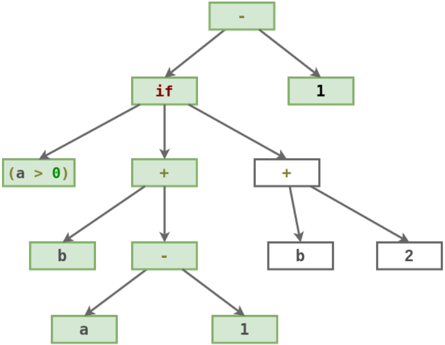
```
if (a > 0){
      b += a - 1;
} else {
      b += 2;
}
b -= 1;
```



DSE: Symbolic expression computation

⇒ Triton allows computing any **symbolic expression** along the trace by backtracking on data dependencies

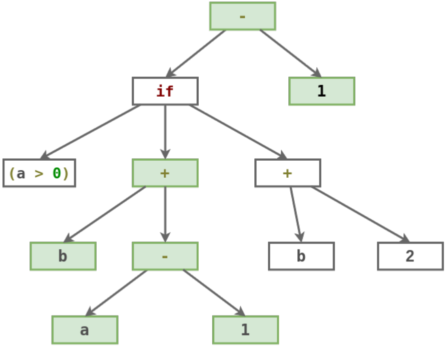
```
if (a > 0){
    b += a - 1;
} else {
    b += 2;
}
b -= 1;
```



DSE: Symbolic expression computation

⇒ Triton allows computing any **symbolic expression** along the trace by backtracking on data dependencies

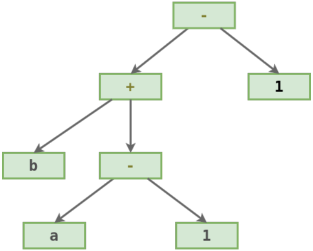
```
if (a > 0){
        b += a - 1;
} else {
        b += 2;
}
b -= 1;
```



DSE: Symbolic expression computation

⇒ Triton allows computing any **symbolic expression** along the trace by backtracking on data dependencies

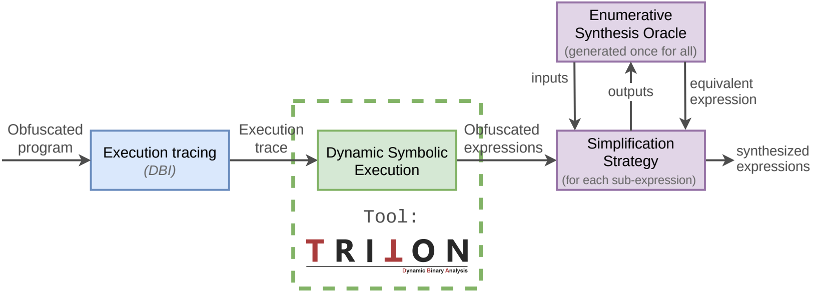
```
if (a > 0){
      b += a - 1;
} else {
      b += 2;
}
b -= 1;
```



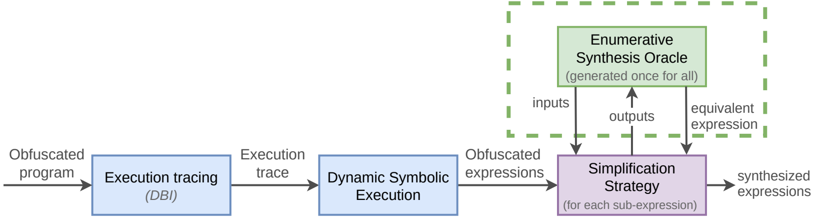
$$\varphi \triangleq (b + (a - 1)) - 1$$

O_φ the associated I/O oracle can be evaluated on different inputs

QSynth: Overview



QSynth: Overview



Synthesis Primitive

Definition

We call Synthesis Primitive any program $\mathcal{SP}$ taking as input parameters a **black-box oracle** $\mathcal{O}_\varphi$ and a **set of input parameters to the oracle** $\mathcal{I}$, and returning, in case of success, a program p, such that for any $i \in \mathcal{I}$ then $p(i) = \mathcal{O}_\varphi(i)$.

$$\mathcal{SP}(\mathcal{O}_\varphi, \mathcal{I}) \quad \Rightarrow \quad p \mid \forall i \in \mathcal{I}, p(i) \equiv \mathcal{O}_\varphi(i)$$

$$\mathcal{SP}(\mathcal{O}_\varphi, \mathcal{I}) \quad \Rightarrow \quad \varnothing$$

Offline Enumerative Search (synthesis primitive $\mathcal{SP}$)

Generate a set of programs based on a given **grammar**: *(operators & variables)*

$$a + b, a - b, a + a, b + b, a + a - b, \ldots$$

Offline Enumerative Search (synthesis primitive $\mathcal{SP}$)

Generate a set of programs based on a given **grammar**: *(operators & variables)*

$$a + b, a - b, a + a, b + b, a + a - b, \ldots$$

and with a set of **inputs**: *(pseudo-random)*

vector $\mathcal{I} = \{(1,1),(1,0),(2,1)\}$

Offline Enumerative Search (synthesis primitive $\mathcal{SP}$)

Generate a set of programs based on a given **grammar**: *(operators & variables)*

$$a + b, a - b, a + a, b + b, a + a - b, \ldots$$

and with a set of **inputs**: *(pseudo-random)*

$$\text{vector } \mathcal{I} = \{(1,1), (1,0), (2,1)\}$$

Evaluate all programs on $\mathcal{I}$ and create the synthesis oracle $\mathcal{SP}$: **outputs** $\rightarrow p$

Offline Enumerative Search (synthesis primitive $\mathcal{SP}$)

Generate a set of programs based on a given **grammar**: *(operators & variables)*

$$a + b, a - b, a + a, b + b, a + a - b, \ldots$$

and with a set of **inputs**: *(pseudo-random)*

$$\text{vector } \mathcal{I} = \{(1,1), (1,0), (2,1)\}$$

Evaluate all programs on $\mathcal{I}$ and create the synthesis oracle $\mathcal{SP}$: **outputs** $\rightarrow p$

Example:

Outputs	**p**
$2, 1, 3$	$a + b$

Offline Enumerative Search (synthesis primitive $\mathcal{SP}$)

Generate a set of programs based on a given **grammar**: *(operators & variables)*

$$a + b, a - b, a + a, b + b, a + a - b, \ldots$$

and with a set of **inputs**: *(pseudo-random)*

$$\text{vector } \mathcal{I} = \{(1, 1), (1, 0), (2, 1)\}$$

Evaluate all programs on $\mathcal{I}$ and create the synthesis oracle $\mathcal{SP}$: **outputs** $\rightarrow p$

Example:

Outputs	**p**
$2, 1, 3$	$a + b$
$0, 1, 1$	$a - b$

Offline Enumerative Search (synthesis primitive $\mathcal{SP}$)

Generate a set of programs based on a given **grammar**: *(operators & variables)*

$$a + b, a - b, a + a, b + b, a + a - b, \ldots$$

and with a set of **inputs**: *(pseudo-random)*

$$\text{vector } \mathcal{I} = \{(1, 1), (1, 0), (2, 1)\}$$

Evaluate all programs on $\mathcal{I}$ and create the synthesis oracle $\mathcal{SP}$: **outputs** $\rightarrow p$

Example:

Outputs	**p**
2, 1, 3	$a + b$
0, 1, 1	$a - b$
2, 2, 4	$a + a$

Offline Enumerative Search (synthesis primitive $\mathcal{SP}$)

Generate a set of programs based on a given **grammar**: *(operators & variables)*

$$a + b, a - b, a + a, b + b, a + a - b, \ldots$$

and with a set of **inputs**: *(pseudo-random)*

$$\text{vector } \mathcal{I} = \{(1, 1), (1, 0), (2, 1)\}$$

Evaluate all programs on $\mathcal{I}$ and create the synthesis oracle $\mathcal{SP}$: **outputs** $\rightarrow p$

Example:

Outputs	**p**
2, 1, 3	$a + b$
0, 1, 1	$a - b$
2, 2, 4	$a + a$
$\ldots$	$\ldots$

Offline Enumerative Search (synthesis primitive $\mathcal{SP}$)

Generate a set of programs based on a given **grammar**: *(operators & variables)*

$$a + b, a - b, a + a, b + b, a + a - b, \ldots$$

and with a set of **inputs**: *(pseudo-random)*

$$\text{vector } \mathcal{I} = \{(1, 1), (1, 0), (2, 1)\}$$

Bad 👎

- Expressions derived grows exponentially *(but can still easily achieve 10 nodes AST expressions)*
- This primitive is **unsound** *(it is only sound wrt. $\mathcal{I}$)*

2, 1, 3	$a + b$
0, 1, 1	$a - b$
2, 2, 4	$a + a$
$\ldots$	$\ldots$

Offline Enumerative Search (synthesis primitive $\mathcal{SP}$)

Generate a set of programs based on a given **grammar**: *(operators & variables)*

$$a + b, a - b, a + a, b + b, a + a - b, \ldots$$

and with a set of **inputs**: *(pseudo-random)*

$$\text{vector } \mathcal{I} = \{(1, 1), (1, 0), (2, 1)\}$$

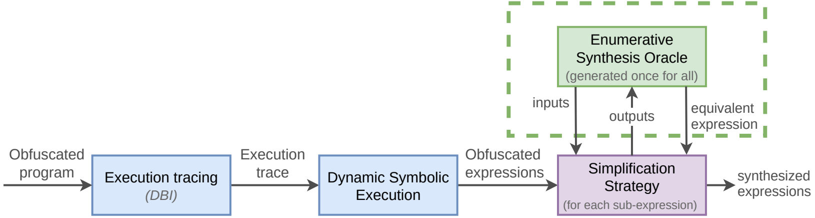
Bad 👎

- Expressions derived grows exponentially *(but can still easily achieve 10 nodes AST expressions)*
- This primitive is **unsound** *(it is only sound wrt. $\mathcal{I}$)*

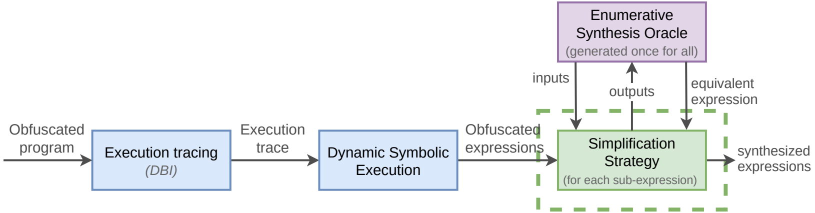
Good 👍

Generated **only once** and usable on different obfuscations and **across programs**

QSynth: Overview

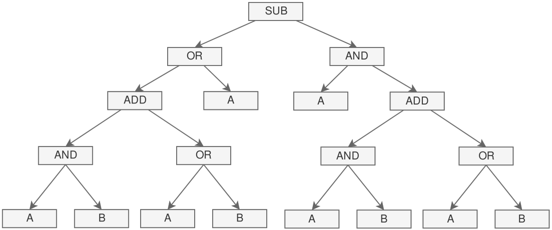


QSynth: Overview



AST simplification - Example

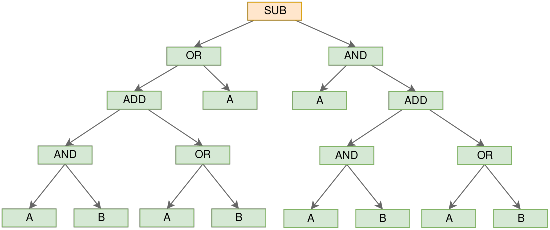
$$\varphi \triangleq (((A \vee B) + (A \wedge B)) \wedge A) - (((A \vee B) + (A \wedge B)) \vee A)$$



$$\mathcal{I} = \{(1,1),(1,0),(2,1)\}$$

AST simplification - Example

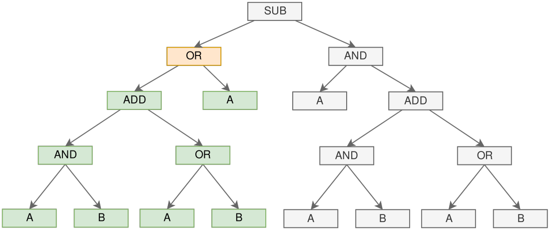
$$\varphi \triangleq (((A \vee B) + (A \wedge B)) \wedge A) - (((A \vee B) + (A \wedge B)) \vee A)$$



$\mathcal{I} = \{(1,1),(1,0),(2,1)\}$
$\mathcal{O}_\varphi$**outputs** $= \{3,0,1\}$
$\mathcal{SP}$[**outputs**]: **not found**

AST simplification - Example

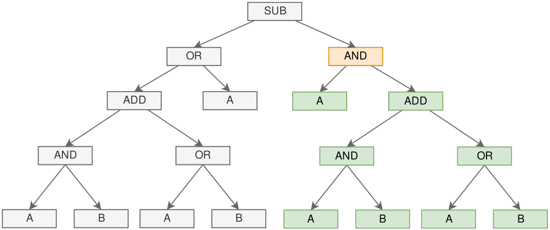
$$\varphi \triangleq (((A \vee B) + (A \wedge B)) \wedge A) - (((A \vee B) + (A \wedge B)) \vee A)$$



$\mathcal{I} = \{(1,1),(1,0),(2,1)\}$
$\mathcal{O}_\varphi$**outputs** $= \{3,1,3\}$
$\mathcal{SP}$[**outputs**]: **not found**

AST simplification - Example

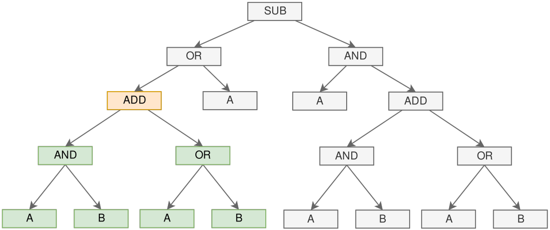
$\varphi \triangleq (((A \vee B) + (A \wedge B)) \wedge A) - (((A \vee B) + (A \wedge B)) \vee A)$



$\mathcal{I} = \{(1,1),(1,0),(2,1)\}$
$\mathcal{O}_{\varphi}$**outputs** $= \{0,1,2\}$
$\mathcal{SP}$[**outputs**]: **not found**

AST simplification - Example

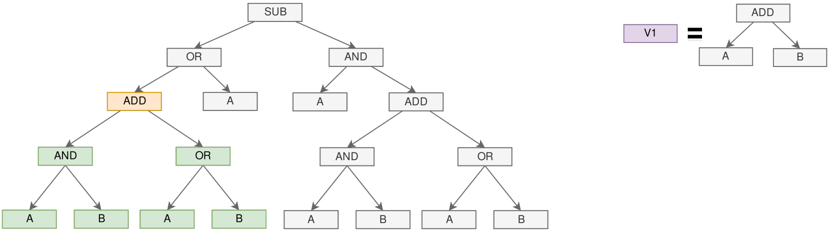
$$\varphi \triangleq (((A \vee B) + (A \wedge B)) \wedge A) - (((A \vee B) + (A \wedge B)) \vee A)$$



$\mathcal{I} = \{(1,1),(1,0),(2,1)\}$
$\mathcal{O}_{\varphi}$**outputs** $= \{2,1,3\}$
$\mathcal{SP}$[**outputs**]: **found** $\Rightarrow A + B$

AST simplification - Example

$$\varphi \triangleq (((A \vee B) + (A \wedge B)) \wedge A) - (((A \vee B) + (A \wedge B)) \vee A)$$



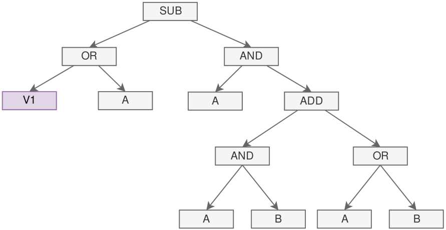
$\mathcal{I} = \{(1,1),(1,0),(2,1)\}$

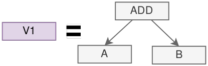
$\mathcal{O}_{\varphi}$**outputs** $= \{2,1,3\}$

$\mathcal{SP}$[**outputs**]: **found** $\Rightarrow A + B$

AST simplification - Example

$$\varphi \triangleq (((A \vee B) + (A \wedge B)) \wedge A) - (((A \vee B) + (A \wedge B)) \vee A)$$





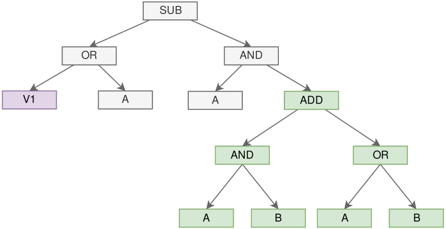
$\mathcal{I} = \{(1,1),(1,0),(2,1)\}$

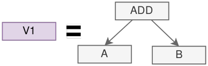
$\mathcal{O}_\varphi$**outputs** $= \{2,1,3\}$

$\mathcal{SP}$[**outputs**]: **found** $\Rightarrow A + B$

AST simplification - Example

$$\varphi \triangleq (((A \vee B) + (A \wedge B)) \wedge A) - (((A \vee B) + (A \wedge B)) \vee A)$$

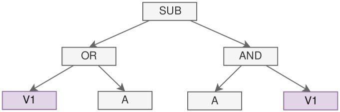


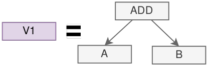


$\mathcal{I} = \{(1,1), (1,0), (2,1)\}$
$\mathcal{O}_\varphi$**outputs** $= \{2, 1, 3\}$
$\mathcal{SP}$[**outputs**]: **found** $\Rightarrow A + B$

AST simplification - Example

$$\varphi \triangleq (((A \vee B) + (A \wedge B)) \wedge A) - (((A \vee B) + (A \wedge B)) \vee A)$$

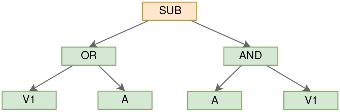


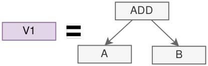


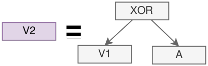
$\mathcal{I} = \{(1,1),(1,0),(2,1)\}$
$\mathcal{O}_\varphi$**outputs** $= \{2,1,3\}$
$\mathcal{SP}$[**outputs**]: **found** $\Rightarrow A + B$

AST simplification - Example

$$\varphi \triangleq (((A \vee B) + (A \wedge B)) \wedge A) - (((A \vee B) + (A \wedge B)) \vee A)$$



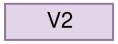


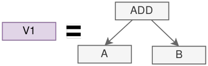


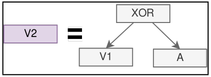
$\mathcal{I} = \{(1,1), (1,0), (2,1)\}$
$\mathcal{O}_\varphi$**outputs** $= \{0, 1, 3\}$
$\mathcal{SP}$[**outputs**]: **found** $\Rightarrow V1 \oplus A$

AST simplification - Example

$$\varphi \triangleq (((A \vee B) + (A \wedge B)) \wedge A) - (((A \vee B) + (A \wedge B)) \vee A)$$

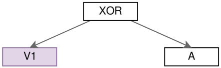


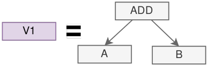


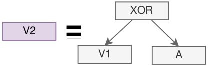


AST simplification - Example

$$\varphi \triangleq (((A \vee B) + (A \wedge B)) \wedge A) - (((A \vee B) + (A \wedge B)) \vee A)$$

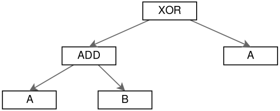


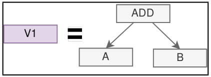


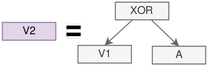


AST simplification - Example

$$\varphi \triangleq (((A \vee B) + (A \wedge B)) \wedge A) - (((A \vee B) + (A \wedge B)) \vee A)$$

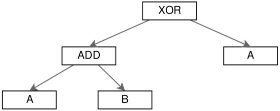


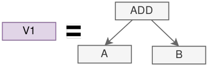




AST simplification - Example

$$\varphi \triangleq (((A \vee B) + (A \wedge B)) \wedge A) - (((A \vee B) + (A \wedge B)) \vee A)$$





Result

Obfuscated:

$$(((A \vee B) + (A \wedge B)) \wedge A) - (((A \vee B) + (A \wedge B)) \vee A)$$

$$\Downarrow$$

Deobfuscated:

$$(A + B) \oplus A$$

Table of Contents

Dataset

⇒ Datasets are built with Tigress 2.2 and the `EncodeArithmetic` **(EA)**, `EncodeData` **(ED)** and `Virtualization` **(VR)**.

⇒ In each dataset: 500 obfuscated functions *(except 239 for EA-ED)*

Dataset

⇒ Datasets are built with Tigress 2.2 and the `EncodeArithmetic` **(EA)**, `EncodeData` **(ED)** and `Virtualization` **(VR)**.

⇒ In each dataset: 500 obfuscated functions *(except 239 for EA-ED)*

	Mean size φ *(in node)*	
	Original	Obfuscated
#1: Syntia †	3.97	203.19
#2: EA	13.5	131.56
#3: VR-EA	13.5	443.64
#4: EA-ED	13.5	9223.46

†use EA-ED *(with 5 derivations max, other are 21 max)*

Dataset

⇒ Datasets are built with Tigress 2.2 and the `EncodeArithmetic` **(EA)**, `EncodeData` **(ED)** and `Virtualization` **(VR)**.

⇒ In each dataset: 500 obfuscated functions *(except 239 for EA-ED)*

	Mean size φ *(in node)*	
	Original	Obfuscated
#1: Syntia †	3.97	203.19
#2: EA	13.5	131.56
#3: VR-EA	13.5	443.64
#4: EA-ED	13.5	9223.46

†use EA-ED *(with 5 derivations max, other are 21 max)*

 # lookup table ($\mathcal{SP}$): 3,358,709 expressions *(14 sets of 3 vars & 5 operators each)*

input vector size $\mathcal{I}$ (for $\mathcal{SP}$): 15

Syntia benchmark

Simplification

	Mean expr. size			Simplification			Mean scale factor	
	Orig	Obf_B	Synt	∅	Partial	Full	Obf_S/Orig	Synt/Orig
Syntia	/	/	/	52	0	448	/	/
QSynth	3.97	203.19	3.71	0	500	**500**	x35.03	**x0.94**

Orig, Obf_S, Obf_B, Synt are rsp. original, obfuscated (source, binary level) and synthesized exprs

Syntia benchmark

Simplification

	Mean expr. size			Simplification			Mean scale factor	
	Orig	Obf_B	Synt	∅	Partial	Full	Obf_S/Orig	Synt/Orig
Syntia	/	/	/	52	0	448	/	/
QSynth	3.97	203.19	3.71	0	500	**500**	x35.03	**x0.94**

Orig, Obf_S, Obf_B, Synt are rsp. original, obfuscated (source, binary level) and synthesized exprs

Accuracy & Speed

	Semantic	Time			
		Sym.Ex	Synthesis	Total	per fun.
Syntia	/	/	/	34 min	4.08s
QSynth	**500**	1m20s	15s	**1m35s**	0.19s

Tigress benchmark

Simplification

	Mean expr. size			Simplification			Mean Scale factor	
	Orig	Obf_B	Synt	∅	Partial	Full	Obf_S/Orig	Synt/Orig
Dataset 2 EA	13.5	245.81	21.92	0	**500**	354 (**70.80%**)	x18.34	**x1.64**
Dataset 3 VR-EA	13.5	443.64	25.42	0	**500**	375 (**75.00%**)	-	**x1.90**
Dataset 4 EA-ED	13.5	9223.46	3812.84	5	234	133 (55.65%)	x405.25	**x234.44**

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

Tigress benchmark

Simplification

	Mean expr. size			Simplification			Mean Scale factor	
	Orig	Obf_B	Synt	∅	Partial	Full	Obf_S/Orig	Synt/Orig
Dataset 2 EA	13.5	245.81	21.92	0	**500**	354 (**70.80%**)	x18.34	**x1.64**
Dataset 3 VR-EA	13.5	443.64	25.42	0	**500**	375 (**75.00%**)	-	**x1.90**
Dataset 4 EA-ED	13.5	9223.46	3812.84	5	234	133 (55.65%)	x405.25	**x234.44**

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

Accuracy & Speed

	Semantic	Time			
		Sym.Ex	Synthesis	Total	per fun.
Dataset 2 EA	OK: 413 KO: 4	1m7s	1m42s	2m49s	0.34s
Dataset 3 VR-EA	OK: 401 KO: 43	17m10s	2m46s	19m56s	2.39s
Dataset 4 EA-ED	-	13m18s	2h7m	2h21m	35.47s

Conclusion

Challenge

$\Rightarrow$ Deobfuscating some data-flow based *(composite)* obfuscations

Conclusion

Challenge

$\Rightarrow$ Deobfuscating some data-flow based *(composite)* obfuscations

Results

$\Rightarrow$ A scalable synthesis algorithm improving the state-of-the-art in both **speed** and **accuracy**

Conclusion

Challenge

$\Rightarrow$ Deobfuscating some data-flow based *(composite)* obfuscations

Results

$\Rightarrow$ A scalable synthesis algorithm improving the state-of-the-art in both **speed** and **accuracy**

Limitation:

- synthesizing expressions using constants
- addressing encoded-data *(which scale)*

Conclusion

Challenge

⇒ Deobfuscating some data-flow based *(composite)* obfuscations

Results

⇒ A scalable synthesis algorithm improving the state-of-the-art in both **speed** and **accuracy**

Limitation:

- synthesizing expressions using constants
- addressing encoded-data *(which scale)*

Future work:

- experimenting other synthesis primitives & simplification strategies *(D&C..)*
- combining with other approach *(not necessarily synthesis-based)*
- testing against other obfuscators

Thank you!

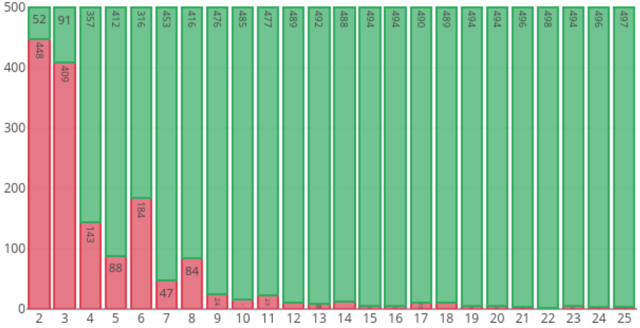
References

- Susmit Jha, Sumit Gulwani, Sanjit A Seshia, and Ashish Tiwari.
  Oracle-guided component-based program synthesis.
  Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1, pages 215-224. ACM, 2010.
  Synthesis time: **31 seconds in average**
- Fabrizio Biondi, Sébastien Josse, Axel Legay, and Thomas Sirvent.
  Effectiveness of synthesis in concolic deobfuscation.
  Computers & Security, 70:500-515, 2017.
  Synthesis time: **96 bits in 20 seconds ca.**
- Tim Blazytko, Moritz Contag, Cornelius Aschermann, and Thorsten Holz.
  Syntia: Synthesizing the semantics of obfuscated code.
  26th USENIX Security Symposium (USENIX Security 17), pages 643-659, 2017.
  Synthesis time: **4 seconds in average**

Presetting pre-computed synthesis lookup tables

Goal: Finding the **smallest discriminative input vector size**

How: Checking equivalence by SMT with synthesized expr. (on EA)

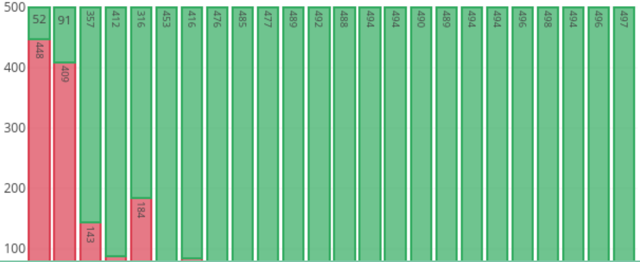


x axis: input vector size, **y axis:** Function number

Presetting pre-computed synthesis lookup tables

Goal: Finding the **smallest discriminative input vector size**

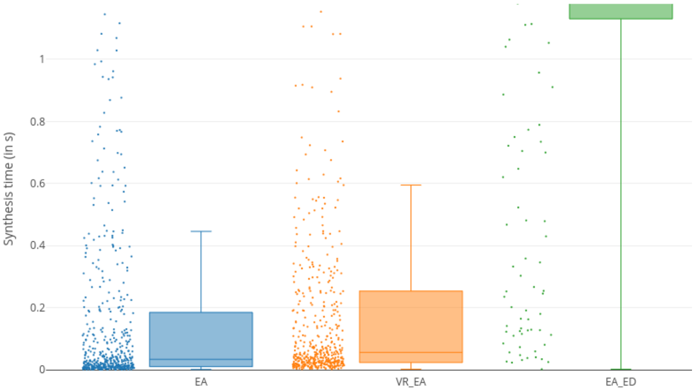
How: Checking equivalence by SMT with synthesized expr. (on EA)



Conclusion

We chose **15** as a good trade-of between **semantic accuracy** and **evaluation speed**.

Synthesis time distribution (on EA)



Synthesis simplification (on EA)