black hat USA 2021

Greybox Program Synthesis: A New Approach to Attack Dataflow Obfuscation

Robin David
<rdavid@quarkslab.com>

Quarkslab

About me

- Software Security Engineer @ Quarkslab
- Primarily interested in attacking **obfuscation** and **automating bug discovery**

Agenda

I. Introduction

II. Synthesis Primer

- Usages
- Application to software deobfuscation

III. Greybox Synthesis

- Algorithm overview
- black-box I/O oracle
- whitebox AST search

IV. Table generation

V. Implementation in QSynthesis *(deobfuscation up-to **reassembled instructions**)*

- implementation & reassembly
- IDA integration

VI. Use-cases

VII. Conclusion

Introduction

(obfuscation techniques)

Obfuscation

What ?

Transformation of a program P in a **semantically equivalent** P'
harder to understand

Why ?

To protect **intellectual property** from
reverse-engineering

How ?

By hiding **valuable assets** of the program

(which are usually)

program logic
algorithms
(referred as control-flow)

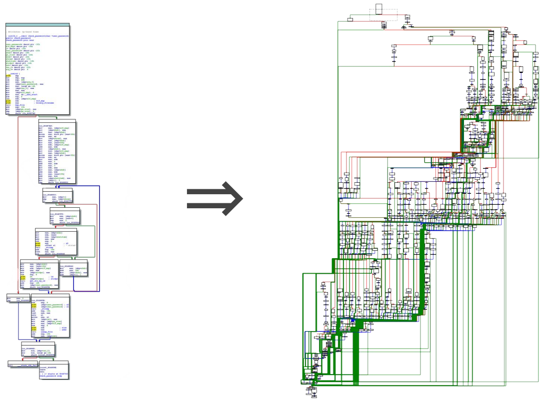
program data
keys, strings, constants...
(referred as data-flow)

Obfuscation Diversity

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

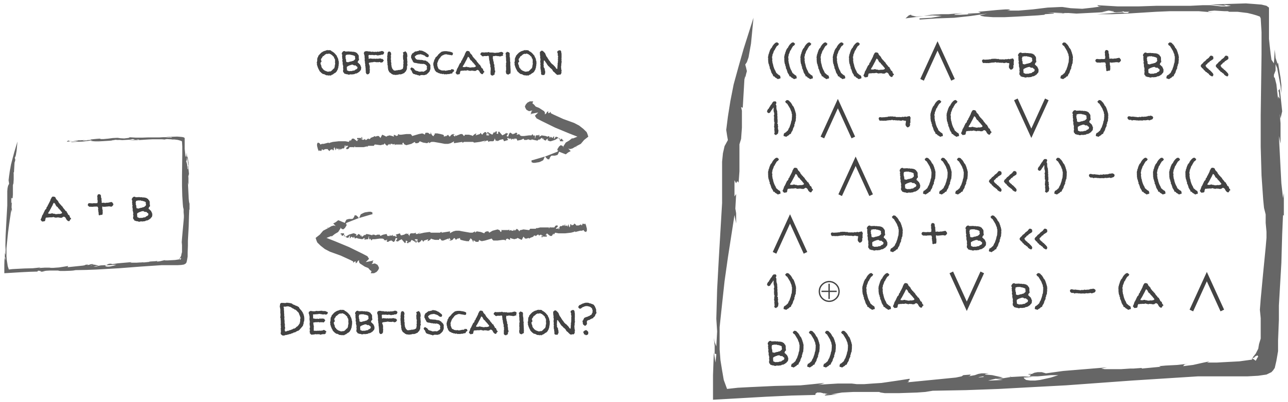
Hiding **data**: constants, strings, APIs, keys etc.

data encoding, MBA, arithmetic encoding, whitebox, array split/fold/merge, variable splitting…

$$a + b \Rightarrow ((((((a \land \neg b) + b) \ll 1) \land \neg ((a \lor b) - (a \land b))) \ll 1) - ((((a \land \neg b) + b) \ll 1) \oplus ((a \lor b) - (a \land b))))$$

Data Obfuscation *(data-flow)*

⇒ This work focuses on data-flow and more especially **MBA** (Mixed Boolean Arithmetic)
(but many other transformation exists like: data encoding, whitebox, variable splitting/merging ..)



! Problem Reversing the transformation is hard *(unlike many control-flow obfuscation, solution is not boolean)*

Deobfuscation Problems

Deobfuscating data-flow expressions on real-world obfuscated programs yield **two distinct** research problems.

PB #1

Locating the data to deobfuscate and knowing **what to deobfuscate** *(depends on what you're looking for in the binary)*.

(This is specific to each binary and is mostly manual)

PB #2

Deobfuscating the data obtained after it gets located *(in our context a data-flow expression)*.

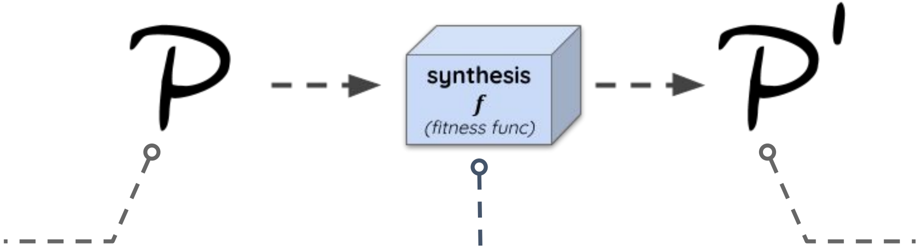
*(Synthesis **only** addresses this issue !)*

Synthesis primer

Program Synthesis

⇒ Program synthesis consists in automatically deriving a program from

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



Program *(for which the semantic is the specification)*

Synthesis constraint *(fitness function which optimizes speed, size, etc..)*

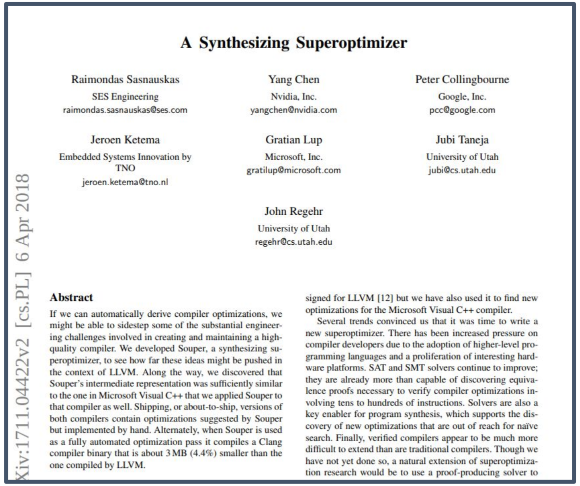
New program *(satisfying the specification and constraints)*

Synthesis for Superoptimization

Synthesis is used in a **variety** of **domains**.
Applied on program analysis it is mostly
used for **optimization** *(known as super-optimization)*
or **deobfuscation**.

AT CORE
LEVEL THE
SAME ISSUE

Superoptimizers



A Synthesizing Superoptimizer

Raimondas Sasnauskas
SES Engineering
raimondas.sasnauskas@ses.com

Yang Chen
Nvidia, Inc.
yangchen@nvidia.com

Peter Collingbourne
Google, Inc.
pcc@google.com

Jeroen Ketema
Embedded Systems Innovation by TNO
jeroen.ketema@tno.nl

Gratian Lup
Microsoft, Inc.
gratilup@microsoft.com

Jubi Taneja
University of Utah
jubi@cs.utah.edu

John Regehr
University of Utah
regehr@cs.utah.edu

Xiv:1711.04422v2 [cs.PL] 6 Apr 2018

Abstract

If we can automatically derive compiler optimizations, we might be able to sidestep some of the substantial engineering challenges involved in creating and maintaining a high-quality compiler. We developed Souper, a synthesizing superoptimizer, to see how far these ideas might be pushed in the context of LLVM. Along the way, we discovered that Souper's intermediate representation was sufficiently similar to the one in Microsoft Visual C++ that we applied Souper to that compiler as well. Shipping, or about-to-ship, versions of both compilers contain optimizations suggested by Souper but implemented by hand. Alternately, when Souper is used as a fully automated optimization pass it compiles a Clang compiler binary that is about 3 MB (4.4%) smaller than the one compiled by LLVM.

signed for LLVM [12] but we have also used it to find new optimizations for the Microsoft Visual C++ compiler.

Several trends convinced us that it was time to write a new superoptimizer. There has been increased pressure on compiler developers due to the adoption of higher-level programming languages and a proliferation of interesting hardware platforms. SAT and SMT solvers continue to improve; they are already more than capable of discovering equivalence proofs necessary to verify compiler optimizations involving tens to hundreds of instructions. Solvers are also a key enabler for program synthesis, which supports the discovery of new optimizations that are out of reach for naïve search. Finally, verified compilers appear to be much more difficult to extend than are traditional compilers. Though we have not yet done so, a natural extension of superoptimization research would be to use a proof-producing solver to

Souper: superoptimizer for LLVM IR
(backed by SMT solving)

STOKE

A stochastic superoptimizer and program synthesizer

STOKE is a stochastic optimizer and program synthesizer for the x86-64 instruction set. STOKE uses random search to explore the extremely high-dimensional space of all possible program transformations. Although any one random transformation is unlikely to produce a code sequence that is desirable, the repeated application of millions of transformations is sufficient to produce novel and non-obvious code sequences. STOKE can be used in many different scenarios, such as optimizing code for performance or size, synthesizing an implementation from scratch or to trade accuracy of floating point computations for performance. As a superoptimizer, STOKE has been shown to outperform the code produced by general-purpose and domain-specific compilers, and in some cases expert hand-written code.

Publications

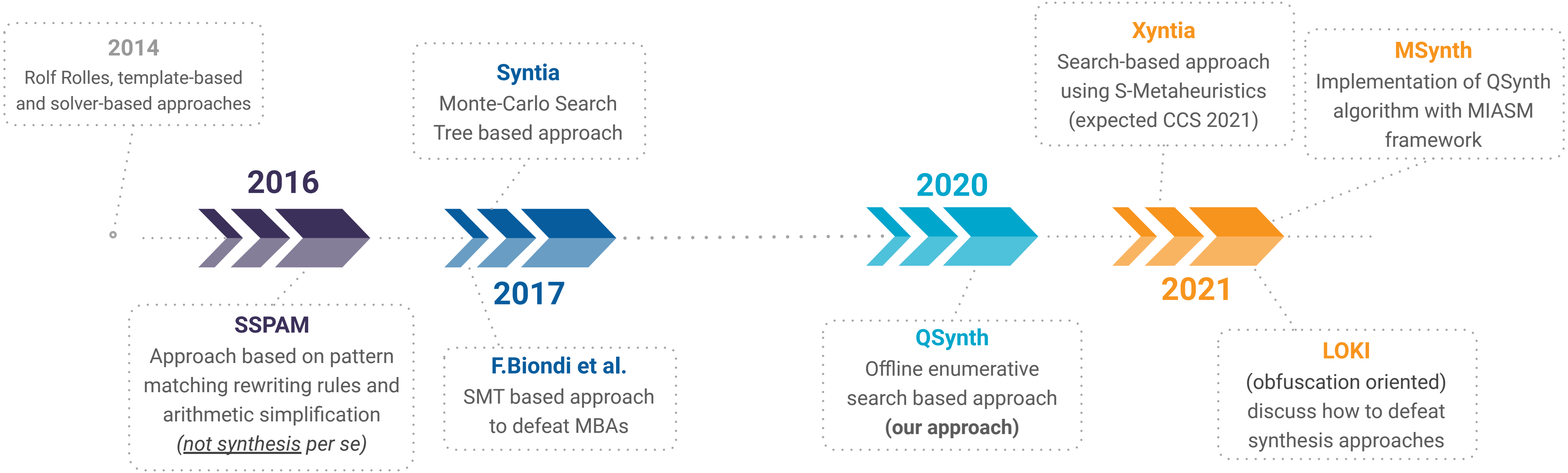
STOKE has appeared in a number of publications.

- **Stochastic Superoptimization** – ASPLOS 2013
- **Data-Driven Equivalence Checking** – OOPSLA 2013
- **Stochastic Optimization of Floating-Point Programs with Tunable Precision** – PLDI 2014
- **Conditionally Correct Superoptimization** – OOPSLA 2015
- **Stochastic Program Optimization** – CACM 2016
- **Stratified Synthesis: Automatically Learning the x86-64 Instruction Set** – PLDI 2016
- **Sound Loop Superoptimization for Google Native Client** – ASPLOS 2017

STOKE: stochastic superoptimizer at assembly level *(x86_64)*

Synthesis for Deobfuscation

Multiple approaches exist, **templates**, **stochastics** *(e.g MCTS)*, **solver-based**, **enumerative** approaches, **search-based** *(S-Metaheuristics)* etc…



2014
Rolf Rolles, template-based and solver-based approaches

2016

SSPAM
Approach based on pattern matching rewriting rules and arithmetic simplification *(not synthesis per se)*

Syntia
Monte-Carlo Search Tree based approach

2017

F.Biondi et al.
SMT based approach to defeat MBAs

2020

QSynth
Offline enumerative search based approach **(our approach)**

Xyntia
Search-based approach using S-Metaheuristics (expected CCS 2021)

MSynth
Implementation of QSynth algorithm with MIASM framework

2021

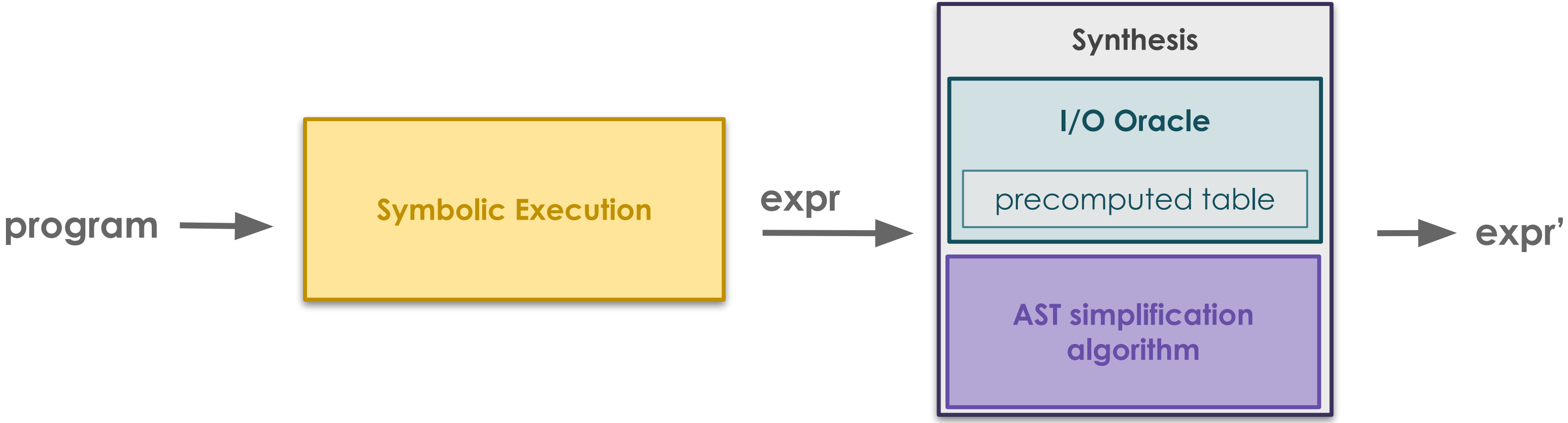
LOKI
(obfuscation oriented) discuss how to defeat synthesis approaches

Greybox Synthesis

(design & principles of our algorithm)

Synthesis algorithm

Our algorithm is based on an **enumerative approach** backed by **symbolic execution** and a **synthesis** *(itself based on two sub-components)*



Symbolic Execution

⇒ We use symbolic execution as a means of extracting **data-flow expressions** of registers or memory at arbitrary locations in the program. The symbolic execution can either be **static** or **dynamic**.

Can backtrack expressions up to program entry

Avoid having to execute the program

Assembly

```
mov     rax, rsi
xor     rax, 0xFFFFFFFFFFFFFFFF
or      rax, rdi
mov     rcx, rdi
xor     rcx, 0xFFFFFFFFFFFFFFFF
and     rcx, rsi
mov     rdx, rdi
and     rdx, rsi
xor     rdx, 0xFFFFFFFFFFFFFFFF
or      rdi, rsi
add     rax, rcx
sub     rax, rdx
add     rax, rdi
retn
```

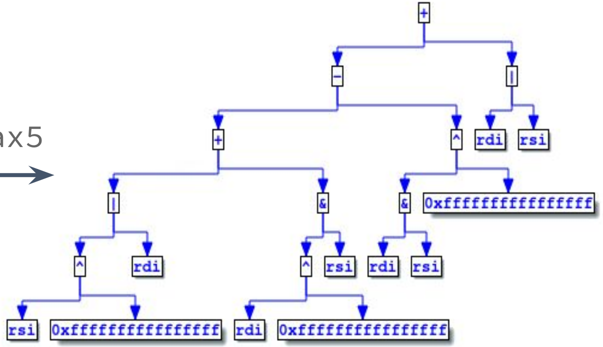
SE →

Intermediate Representation

```
rax0 := rsi
rax1 := rax ⊕ 0xFFFFFFFFFFFFFFFF
rax2 := rax1 | rdi
rcx0 := rdi
rcx1 := rcx0 ⊕ 0xFFFFFFFFFFFFFFFF
rcx2 := rcx1 & rsi
rdx0 := rdi
rdx1 := rdx0 & rsi
rdx2 := rdx1 ⊕ 0xFFFFFFFFFFFFFFFF
rdi0 := rdi | rsi
rax3 := rax2 + rcx2
rax4 := rax3 - rdx2
rax5 := rax4 + rdi0
```

rax5 →

AST

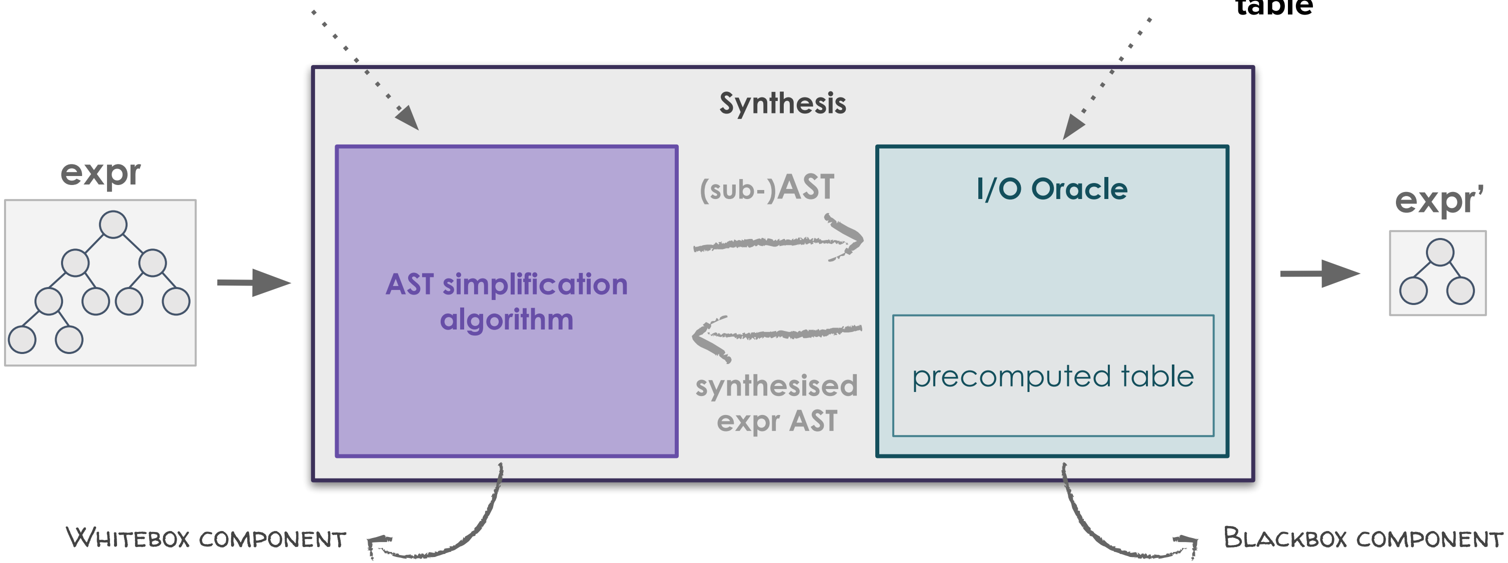


Our synthesis algorithm

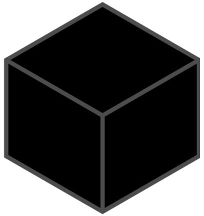
Our algorithm is a **greybox synthesizer** based on two components

An **AST simplification** algorithm that can use **various strategies**

An **I/O oracle** based on an **offline enumerative search** backed by a **pre-computed table**



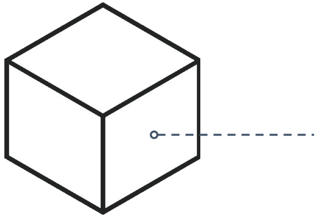
Blackbox vs Whitebox in Synthesis *(for deobfuscation)*



Blackbox

relates to approaches considering expressions to synthesize as blackboxes and only interacting with them through their **input/ouput behavior**

+ only influenced by semantic complexity
- large search space
- boolean result *(fully synthesized or not at all)*



$$((((((a \land \neg b) + b) \ll 1) \land \neg ((a \lor b) - (a \land b))) \ll 1) - ((((a \land \neg b) + b) \ll 1) \oplus ((a \lor b) - (a \land b))))$$

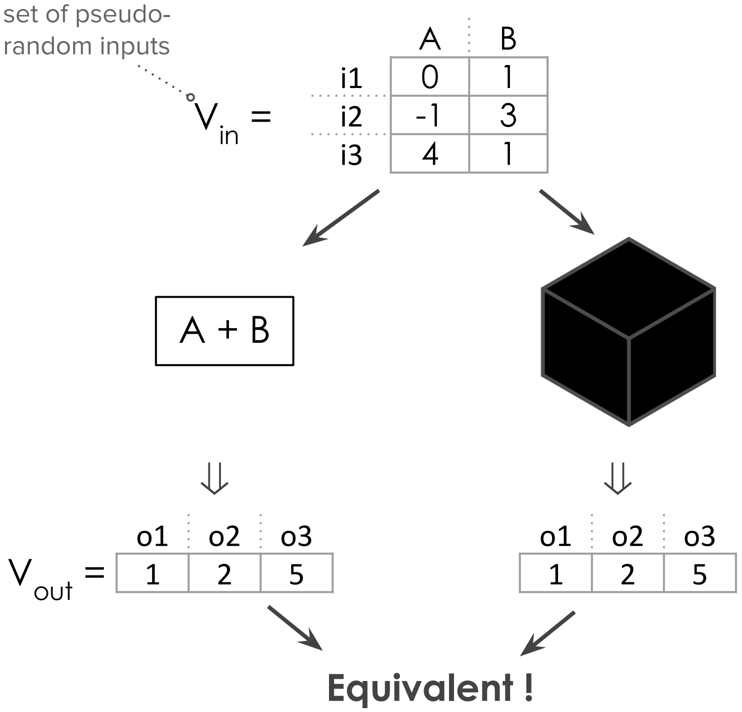
Whitebox

relates to approaches manipulating the semantic of the expression through its syntactic representation *(usually the AST of the semantic)*

+ the exact semantic is considered
- influenced by syntactic complexity
+ enable sub-expressions synthesis

Blackbox I/O Synthesis Oracle

Blackbox I/O Oracle



V_{in} =

	A	B
i1	0	1
i2	-1	3
i3	4	1

⇓ V_{out} =

o1	o2	o3
1	2	5

⇓

o1	o2	o3
1	2	5

Equivalent !

Pre-computed tables

Given a grammar with some **operators** *(+, -, |, &, ⊕..)*, and **variables** *(a, b, c..)*, derives all possible expressions *(up to a given bound)* and evaluate them on V_{in} to obtain a function:

$$V_{out} \mapsto expr$$

V_{out}	**expr**
`<1, 2, 5>`	A + B
`<-1,-4, 3>`	A - B
`<1, -1, 5>`	A | B
....	....

- generated once, and ensures **O(log(n))** synthesis
- Unsound but equivalence can be checked by SMT

⇒ What happens if it cannot synthesize the root node ?

Whitebox AST search

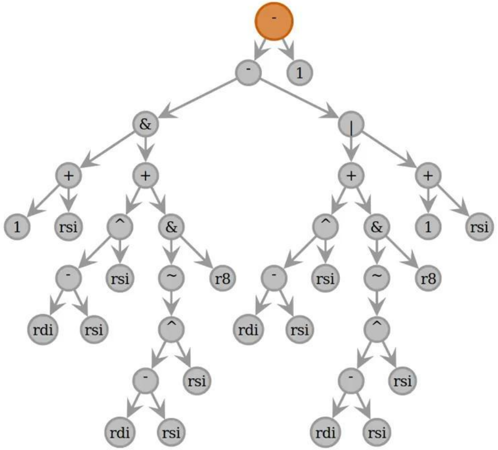
- If it cannot synthesize root node it aims at simplifying sub-expressions to obtain **at least a partial synthesis** *(while with an I/O oracle the result is boolean)*.

- Thus an **AST search algorithm** will iterate through the graph looking for sub-nodes to synthesize.

Algorithm

1. Search a node to synthesize
2. if find one, replaces it by a temporary placeholder
3. if not, replaces it also
4. repeat the search until having substituted all nodes
5. recursively replace placeholders by the corresponding AST (synthesized or original)

Original strategy



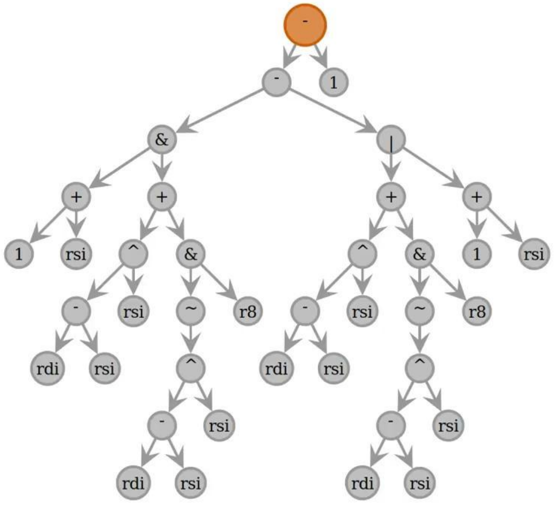
https://youtu.be/ID_PEVseecI

This simplification strategy have some **complexity issues** *(yet it provides optimal results)*

New AST search strategies

Top-Down *(Divide & Conquer)*

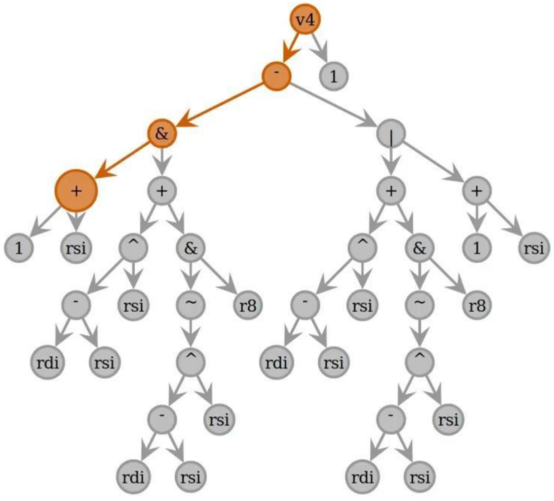
Single **DFS** traversal of the AST. Ensures linearity of the simplification of the algorithm *(while original one was quadratic in the worst case)*.



https://youtu.be/VQRg3LHC6Lw

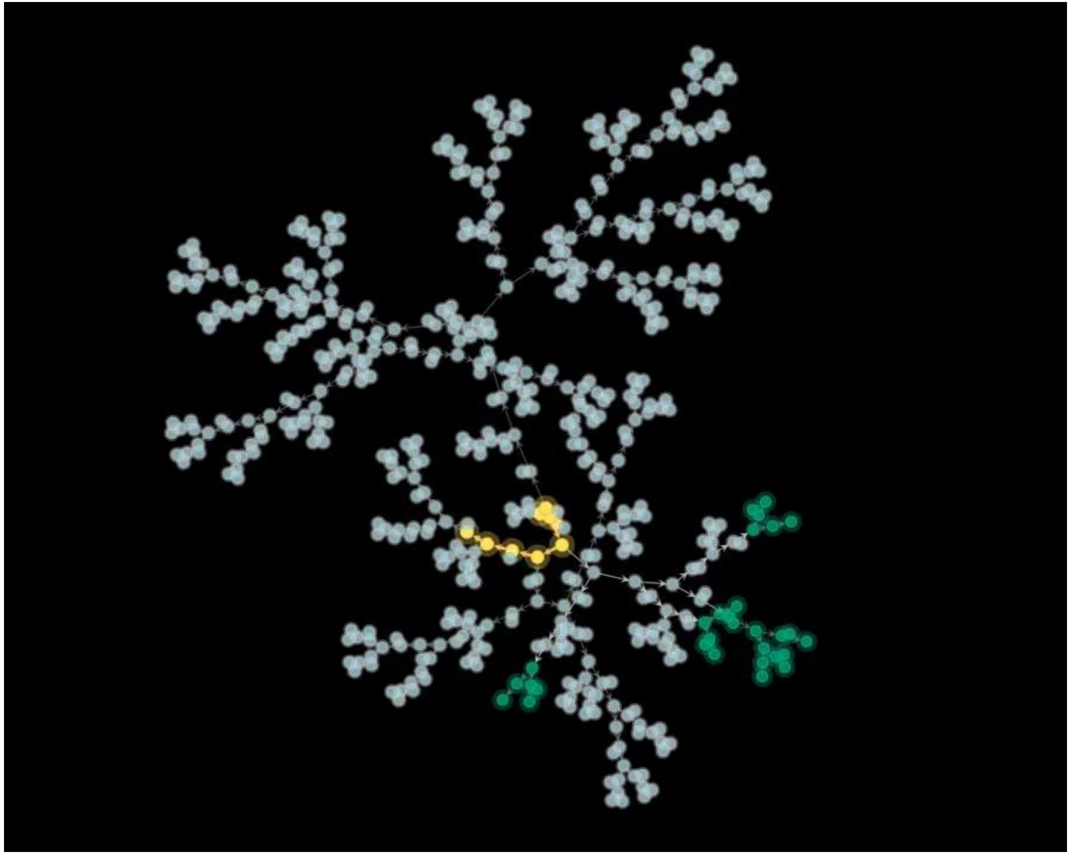
Top-Down & Bottom-Up

Like Top-Down but if a node gets synthesized attempts to re-synthesize its parents by means of reducing the variable cardinal.



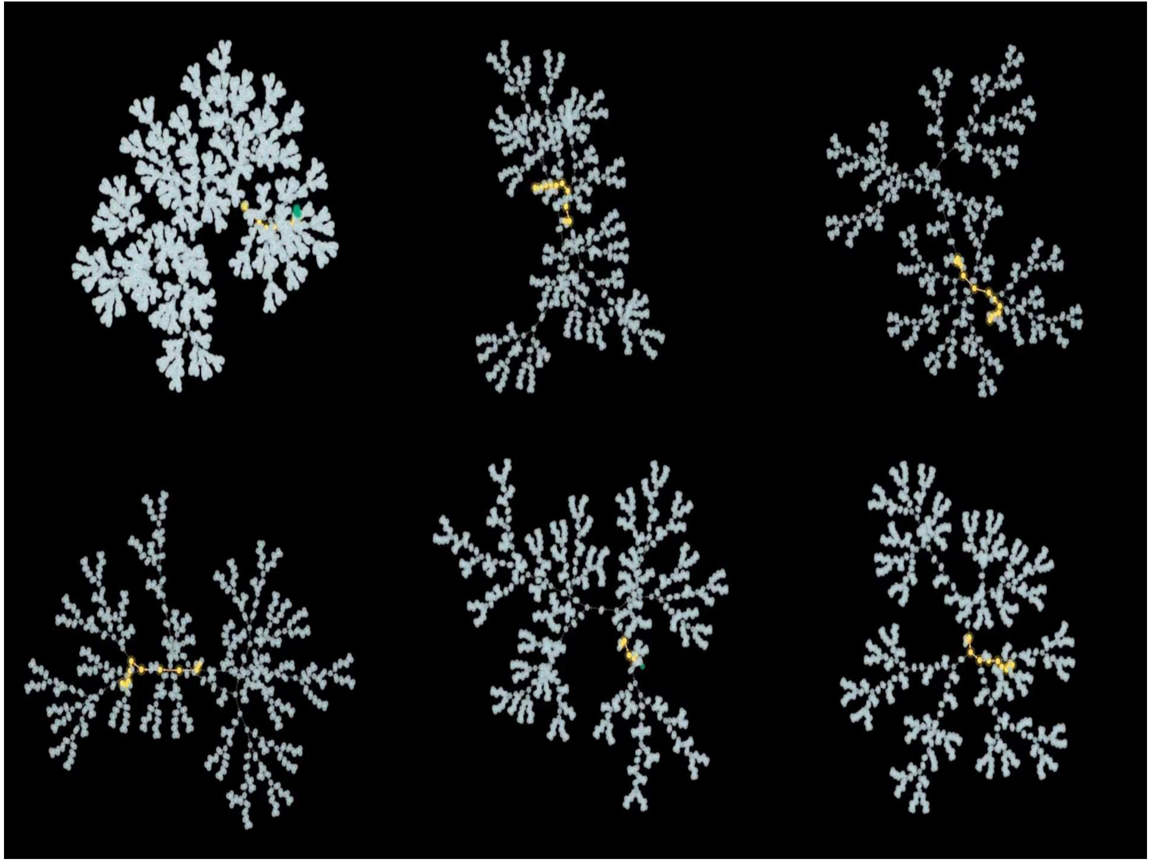
https://youtu.be/G1lBOqmwLaI

Algorithm Visualization



https://youtu.be/Nz8KC1HtgiI

Algorithm Visualization



https://youtu.be/9MHeGtc3Uhc

Algorithm Visualization

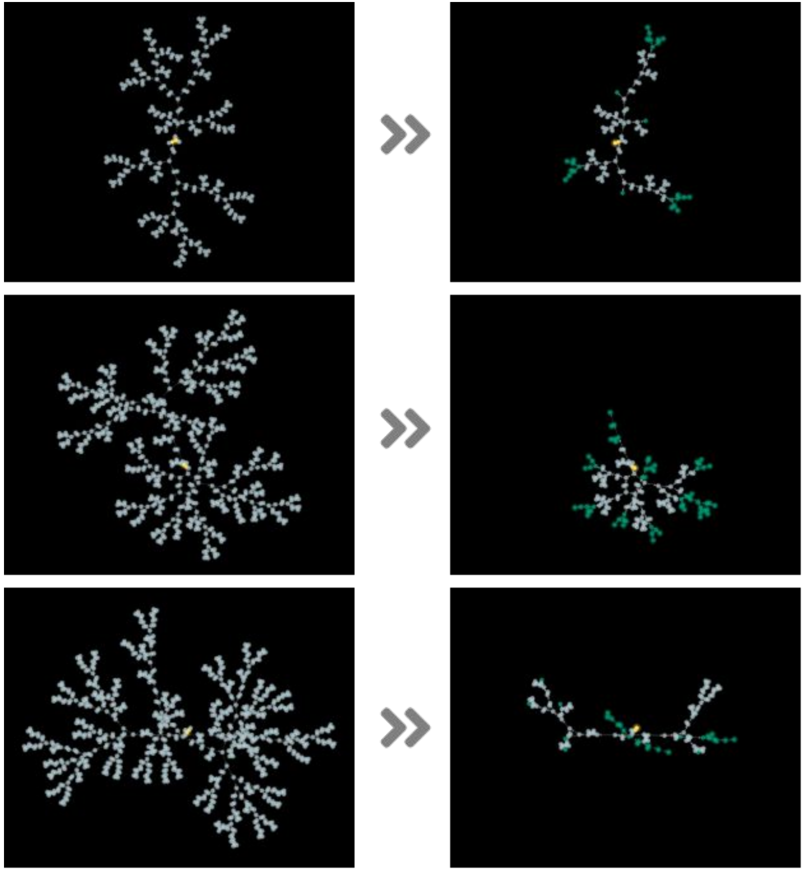


Table generation

(aka generating a potent I/O oracle)

Table Generation

⇒ Table generation requires **evaluating millions** of expressions and keeping millions of V_{out} vectors to ignore identical ones *(by construction we generate from smaller to larger expressions)*.

Improvements:

- ○ **Memoization** of all evaluated expressions *(thus A+B is evaluated only once for all, when combined with another expression like A+B-C the memoïzed result is reused for evaluation)*
- ○ **JITTing** of expressions evaluation. Evaluation made on native integers *(not using Python)*. For that uses **dragonFFI** *(could also have used numpy)*.

reach
25K exprs/sec

⇓

We now have a table with **375 million entries**
(last year we had ~3 millions)

(Generated with a 235 GB RAM machine :p)

Table Storage

`pickle`

Python object serialization module

- Requires loading the whole table
- Parsing is slow on large object

⇒ Ok for small tables but limited for larger ones

(format used by MSynth)

⇒

PONY

Python ORM for databases like sqlite

- If V_{out} primary key, insertion is linear in number of entries.
- If not, lookup is linear in the number of entries

⇒ Not suitable for such large tables

⇒

levelDB

Key Value database *(by Google)*

- Store keys as "tries" to ensure **O(log(n))** access
- Automatic caching mechanism

⇒ Best suited for our need

→ 122 µs

⇒ We also made a REST API *(using FastAPI)* to serve Level-DB database content

Expression Normalization

⇒ Tables are limited by the enumerative approach, combining some variables *(a, b, c..)* with some operators *(+, -, & …)*. Thus no constants in sight. To improve expression diversity we performed two experiments.

Expression Linearization

Goal: Representing expressions as **normalized equations**. For that, uses SymPy a library for symbolic maths.

SymPy

Original	Linearized
a − (c − a)	2∗a − c
(a−b) − (a + a)	−a − b
a + (b ∗ b)	b^2 + a
...	...

⇒

Pros/Cons:

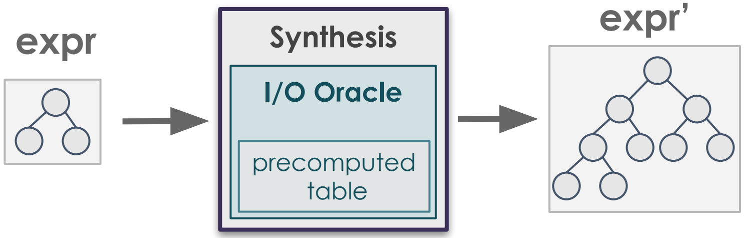
- introduces constants !
- **annihilates generation performances**
- introduces power operators
- only works on pure arithmetic expressions

we thus do not use it in practice

Expression Learning

Problem

What if the synthesized expression is larger than the one in input ?



expr

Synthesis

I/O Oracle

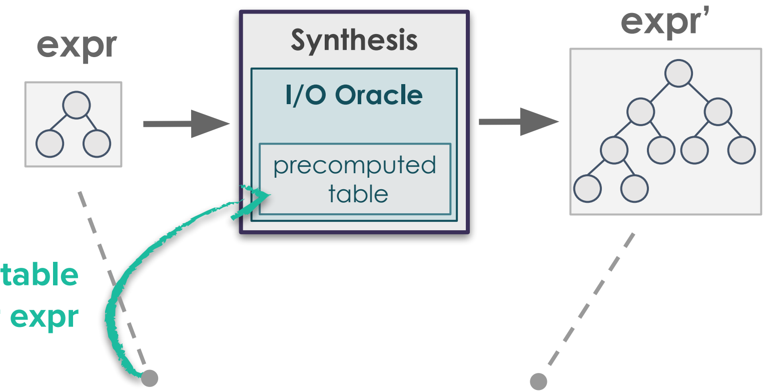
precomputed table

expr'

Expression Learning

Problem

What if the synthesized expression is **larger** than the one in input ?



We can update the table with the smaller expr

IT
INTRODUCES
CONSTANTS !

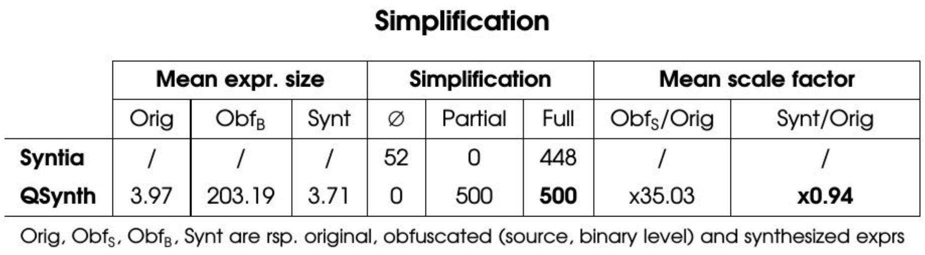
Input Expr		**Output Expr'** *(in table)*
(a*a) - 1	⇒	~~((a*a) + a) + (~a)~~
-1 + a	⇒	~~(~a + a) + (~(-a))~~
(b ^ a) - 1	⇒	~~(~a + a) + (b ^ a)~~

⇒ We also now introduce simple constants in our table generation process

Benchmarks

Paper benchmarks

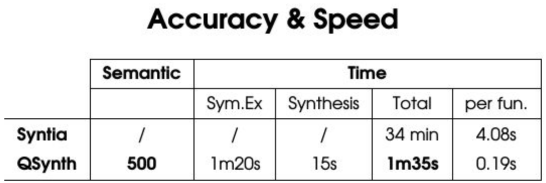
Comparison with Syntia



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**

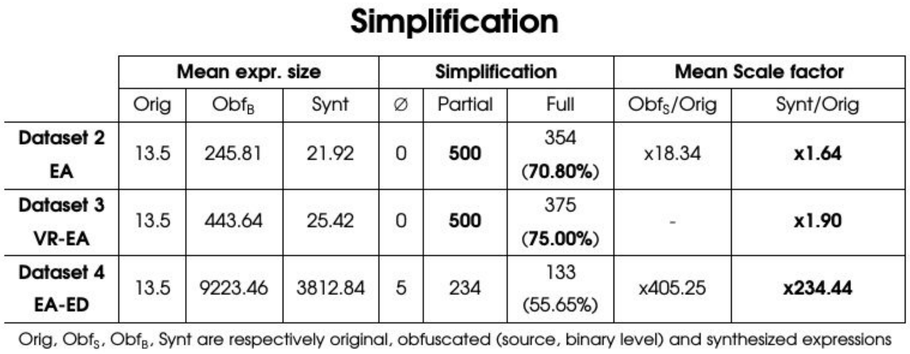
Orig, Obf_S, Obf_B, Synt are rsp. original, obfuscated (source, binary level) and synthesized exprs



Accuracy & Speed

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

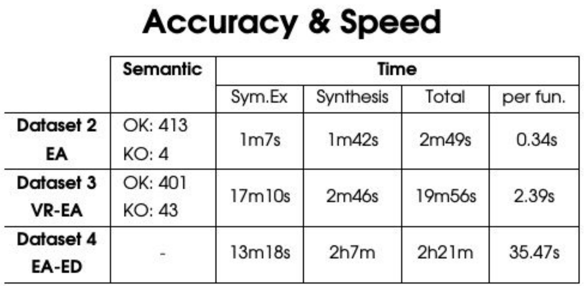
Against Tigress



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

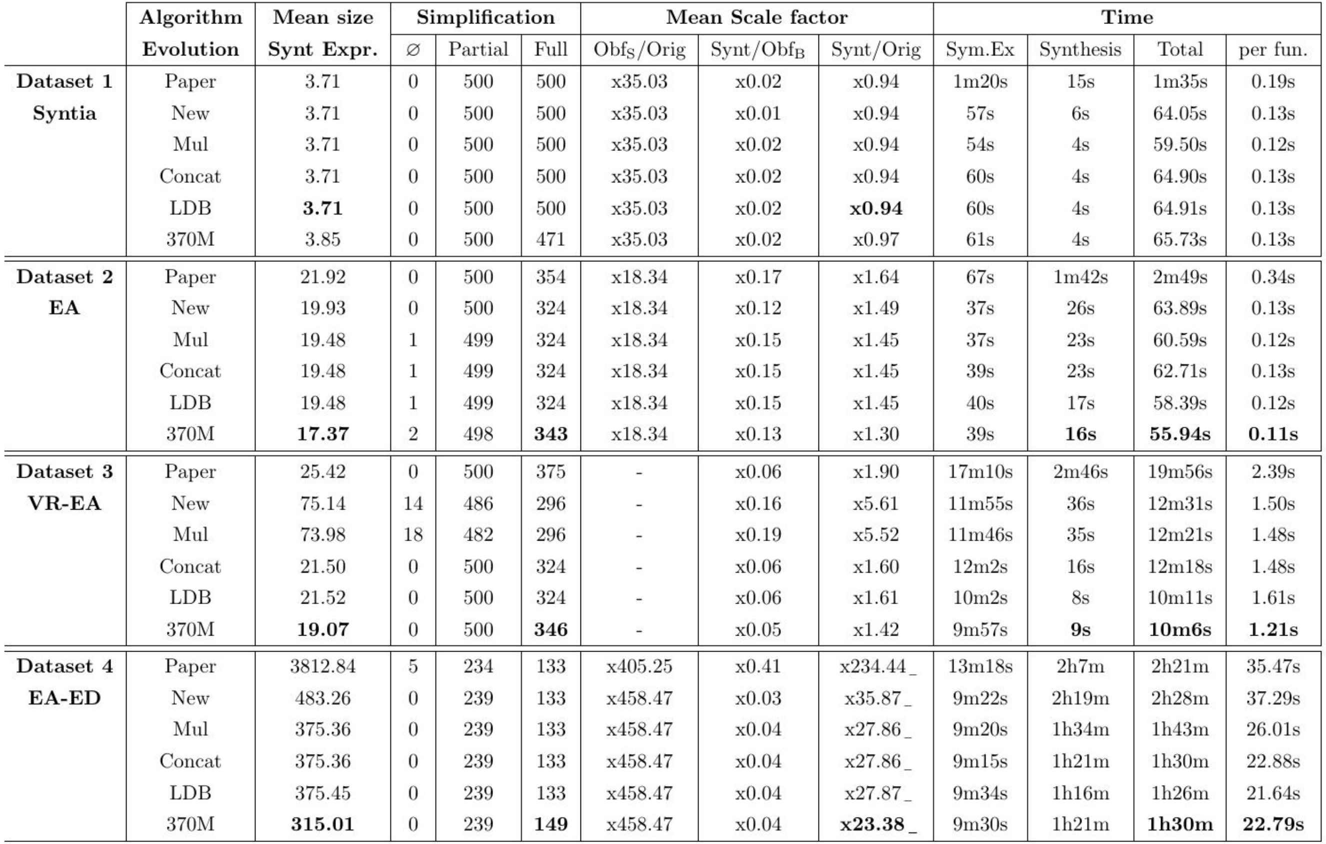


Accuracy & Speed

	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

⇒ *Results were promising !*

Benchmarks improvements

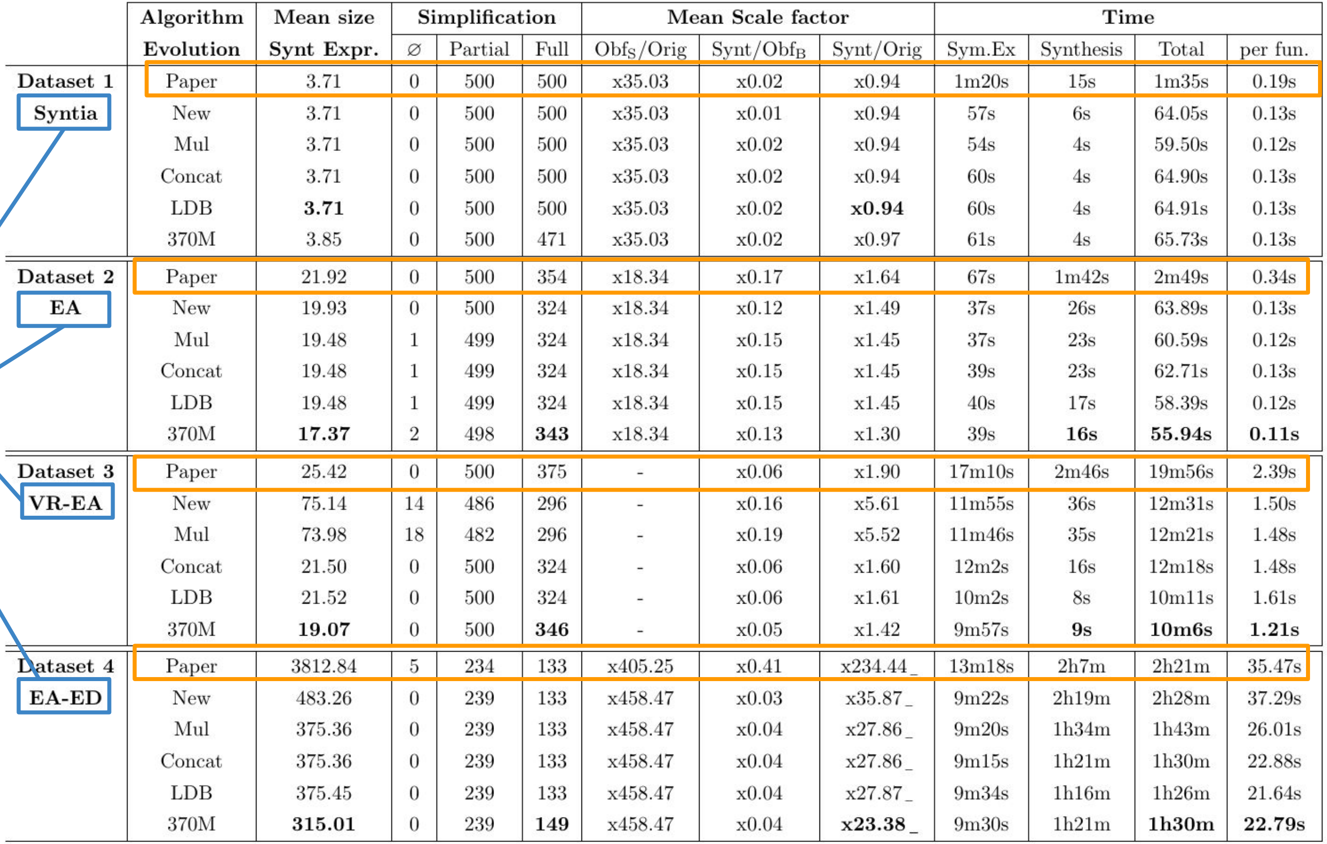


	Algorithm	Mean size	Simplification			Mean Scale factor			Time			
	Evolution	Synt Expr.	∅	Partial	Full	Obf$_S$/Orig	Synt/Obf$_B$	Synt/Orig	Sym.Ex	Synthesis	Total	per fun.
Dataset 1 **Syntia**	Paper	3.71	0	500	500	x35.03	x0.02	x0.94	1m20s	15s	1m35s	0.19s
	New	3.71	0	500	500	x35.03	x0.01	x0.94	57s	6s	64.05s	0.13s
	Mul	3.71	0	500	500	x35.03	x0.02	x0.94	54s	4s	59.50s	0.12s
	Concat	3.71	0	500	500	x35.03	x0.02	x0.94	60s	4s	64.90s	0.13s
	LDB	**3.71**	0	500	500	x35.03	x0.02	**x0.94**	60s	4s	64.91s	0.13s
	370M	3.85	0	500	471	x35.03	x0.02	x0.97	61s	4s	65.73s	0.13s
Dataset 2 **EA**	Paper	21.92	0	500	354	x18.34	x0.17	x1.64	67s	1m42s	2m49s	0.34s
	New	19.93	0	500	324	x18.34	x0.12	x1.49	37s	26s	63.89s	0.13s
	Mul	19.48	1	499	324	x18.34	x0.15	x1.45	37s	23s	60.59s	0.12s
	Concat	19.48	1	499	324	x18.34	x0.15	x1.45	39s	23s	62.71s	0.13s
	LDB	19.48	1	499	324	x18.34	x0.15	x1.45	40s	17s	58.39s	0.12s
	370M	**17.37**	2	498	**343**	x18.34	x0.13	x1.30	39s	**16s**	**55.94s**	**0.11s**
Dataset 3 **VR-EA**	Paper	25.42	0	500	375	-	x0.06	x1.90	17m10s	2m46s	19m56s	2.39s
	New	75.14	14	486	296	-	x0.16	x5.61	11m55s	36s	12m31s	1.50s
	Mul	73.98	18	482	296	-	x0.19	x5.52	11m46s	35s	12m21s	1.48s
	Concat	21.50	0	500	324	-	x0.06	x1.60	12m2s	16s	12m18s	1.48s
	LDB	21.52	0	500	324	-	x0.06	x1.61	10m2s	8s	10m11s	1.61s
	370M	**19.07**	0	500	**346**	-	x0.05	x1.42	9m57s	**9s**	**10m6s**	**1.21s**
Dataset 4 **EA-ED**	Paper	3812.84	5	234	133	x405.25	x0.41	x234.44_	13m18s	2h7m	2h21m	35.47s
	New	483.26	0	239	133	x458.47	x0.03	x35.87_	9m22s	2h19m	2h28m	37.29s
	Mul	375.36	0	239	133	x458.47	x0.04	x27.86_	9m20s	1h34m	1h43m	26.01s
	Concat	375.36	0	239	133	x458.47	x0.04	x27.86_	9m15s	1h21m	1h30m	22.88s
	LDB	375.45	0	239	133	x458.47	x0.04	x27.87_	9m34s	1h16m	1h26m	21.64s
	370M	**315.01**	0	239	**149**	x458.47	x0.04	**x23.38**_	9m30s	1h21m	**1h30m**	**22.79s**

Benchmarks improvements

- **Paper**: Original results

- **Syntia**: ED + EA (very simple)
- **EA**: EncodeArithmetic ⇒ MBA
- **VR-EA**: Virtualization + EA
- **EA-ED**: EA + EncodeData



	Algorithm	Mean size	Simplification			Mean Scale factor			Time			
	Evolution	Synt Expr.	∅	Partial	Full	Obfs/Orig	Synt/ObfB	Synt/Orig	Sym.Ex	Synthesis	Total	per fun.
Dataset 1	Paper	3.71	0	500	500	x35.03	x0.02	x0.94	1m20s	15s	1m35s	0.19s
Syntia	New	3.71	0	500	500	x35.03	x0.01	x0.94	57s	6s	64.05s	0.13s
	Mul	3.71	0	500	500	x35.03	x0.02	x0.94	54s	4s	59.50s	0.12s
	Concat	3.71	0	500	500	x35.03	x0.02	x0.94	60s	4s	64.90s	0.13s
	LDB	**3.71**	0	500	500	x35.03	x0.02	**x0.94**	60s	4s	64.91s	0.13s
	370M	3.85	0	500	471	x35.03	x0.02	x0.97	61s	4s	65.73s	0.13s
Dataset 2	Paper	21.92	0	500	354	x18.34	x0.17	x1.64	67s	1m42s	2m49s	0.34s
EA	New	19.93	0	500	324	x18.34	x0.12	x1.49	37s	26s	63.89s	0.13s
	Mul	19.48	1	499	324	x18.34	x0.15	x1.45	37s	23s	60.59s	0.12s
	Concat	19.48	1	499	324	x18.34	x0.15	x1.45	39s	23s	62.71s	0.13s
	LDB	19.48	1	499	324	x18.34	x0.15	x1.45	40s	17s	58.39s	0.12s
	370M	**17.37**	2	498	**343**	x18.34	x0.13	x1.30	39s	**16s**	**55.94s**	**0.11s**
Dataset 3	Paper	25.42	0	500	375	-	x0.06	x1.90	17m10s	2m46s	19m56s	2.39s
VR-EA	New	75.14	14	486	296	-	x0.16	x5.61	11m55s	36s	12m31s	1.50s
	Mul	73.98	18	482	296	-	x0.19	x5.52	11m46s	35s	12m21s	1.48s
	Concat	21.50	0	500	324	-	x0.06	x1.60	12m2s	16s	12m18s	1.48s
	LDB	21.52	0	500	324	-	x0.06	x1.61	10m2s	8s	10m11s	1.61s
	370M	**19.07**	0	500	**346**	-	x0.05	x1.42	9m57s	**9s**	**10m6s**	**1.21s**
Dataset 4	Paper	3812.84	5	234	133	x405.25	x0.41	x234.44_	13m18s	2h7m	2h21m	35.47s
EA-ED	New	483.26	0	239	133	x458.47	x0.03	x35.87_	9m22s	2h19m	2h28m	37.29s
	Mul	375.36	0	239	133	x458.47	x0.04	x27.86_	9m20s	1h34m	1h43m	26.01s
	Concat	375.36	0	239	133	x458.47	x0.04	x27.86_	9m15s	1h21m	1h30m	22.88s
	LDB	375.45	0	239	133	x458.47	x0.04	x27.87_	9m34s	1h16m	1h26m	21.64s
	370M	**315.01**	0	239	**149**	x458.47	x0.04	**x23.38_**	9m30s	1h21m	**1h30m**	**22.79s**

Benchmarks improvements

	Algorithm	Mean size	Simplification			Mean Scale factor			Time			
	Evolution	Synt Expr.	∅	Partial	Full	Obf_S/Orig	Synt/Obf_B	Synt/Orig	Sym.Ex	Synthesis	Total	per fun.
Dataset 1 **Syntia**	Paper	3.71	0	500	500	x35.03	x0.02	x0.94	1m20s	15s	1m35s	0.19s
	New	3.71	0	500	500	x35.03	x0.01	x0.94	57s	6s	64.05s	0.13s
	Mul	3.71	0	500	500	x35.03	x0.02	x0.94	54s	4s	59.50s	0.12s
	Concat	3.71	0	500	500	x35.03	x0.02	x0.94	60s	4s	64.90s	0.13s
	LDB	**3.71**	0	500	500	x35.03	x0.02	**x0.94**	60s	4s	64.91s	0.13s
	370M	3.85	0	500	471	x35.03	x0.02	x0.97	61s	4s	65.73s	0.13s
Dataset 2 **EA**	Paper	21.92	0	500	354	x18.34	x0.17	x1.64	67s	1m42s	2m49s	0.34s
	New	19.93	0	500	324	x18.34	x0.12	x1.49	37s	26s	63.89s	0.13s
		19.48	1	499	324	x18.34	x0			23s	60.59s	0.12s
	at	19.48	1	499	324	x18.34	x0			23s	62.71s	0.13s
	B	19.48	1	499	324	x18.34	x0			17s	58.39s	0.12s
		17.37	2	498	**343**	x18.34	x0			**16s**	**55.94s**	**0.11s**
	r	25.42	0	500	375	-	x0			m46s	19m56s	2.39s
		75.14	14	486	296	-	x0			36s	12m31s	1.50s
	Mul	73.98	18	482	296	-	x0.19	x5.52	11m46s	35s	12m21s	1.48s
	Concat	21.50	0	500	324	-	x0.06	x1.60	12m2s	16s	12m18s	1.48s
	LDB	21.52	0	500	324	-	x0.06	x1.61	10m2s	8s	10m11s	1.61s
	370M	**19.07**	0	500	**346**	-	x0.05	x1.42	9m57s	**9s**	**10m6s**	**1.21s**
Dataset 4 **EA-ED**	Paper	3812.84	5	234	133	x405.25	x0.41	x234.44_	13m18s	2h7m	2h21m	35.47s
	New	483.26	0	239	133	x458.47	x0.03	x35.87_	9m22s	2h19m	2h28m	37.29s
	Mul	375.36	0	239	133	x458.47	x0.04	x27.86_	9m20s	1h34m	1h43m	26.01s
	Concat	375.36	0	239	133	x458.47	x0.04	x27.86_	9m15s	1h21m	1h30m	22.88s
	LDB	375.45	0	239	133	x458.47	x0.04	x27.87_	9m34s	1h16m	1h26m	21.64s
	370M	**315.01**	0	239	**149**	x458.47	x0.04	**x23.38_**	9m30s	1h21m	**1h30m**	**22.79s**

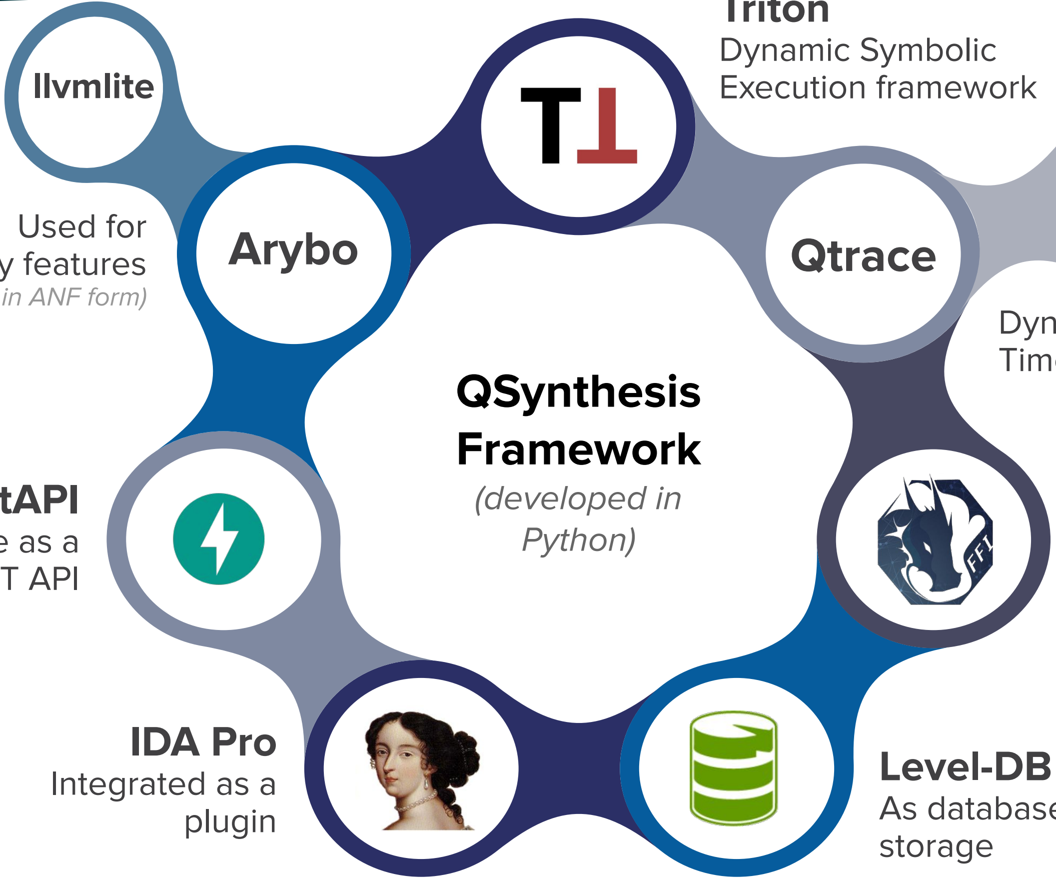
Better average simplification than original implementation
(90% for EA-ED)

Speed improvement ranging from 31% to 67%

Implementation

(in the QSynthesis utility)

QSynthesis



QSynthesis Framework

(developed in Python)

llvmlite

Used for reassembly features

(bit vector IR in ANF form)

Arybo

Triton

Dynamic Symbolic Execution framework

QBDI

Dynamic Binary Instrumentation Framework

Qtrace

Dynamic Tracing Framework & Time Travel Debugger (TTD)

dragonffi

For the JITTing of expression evaluation

(during table generation)

Level-DB

As database for table storage

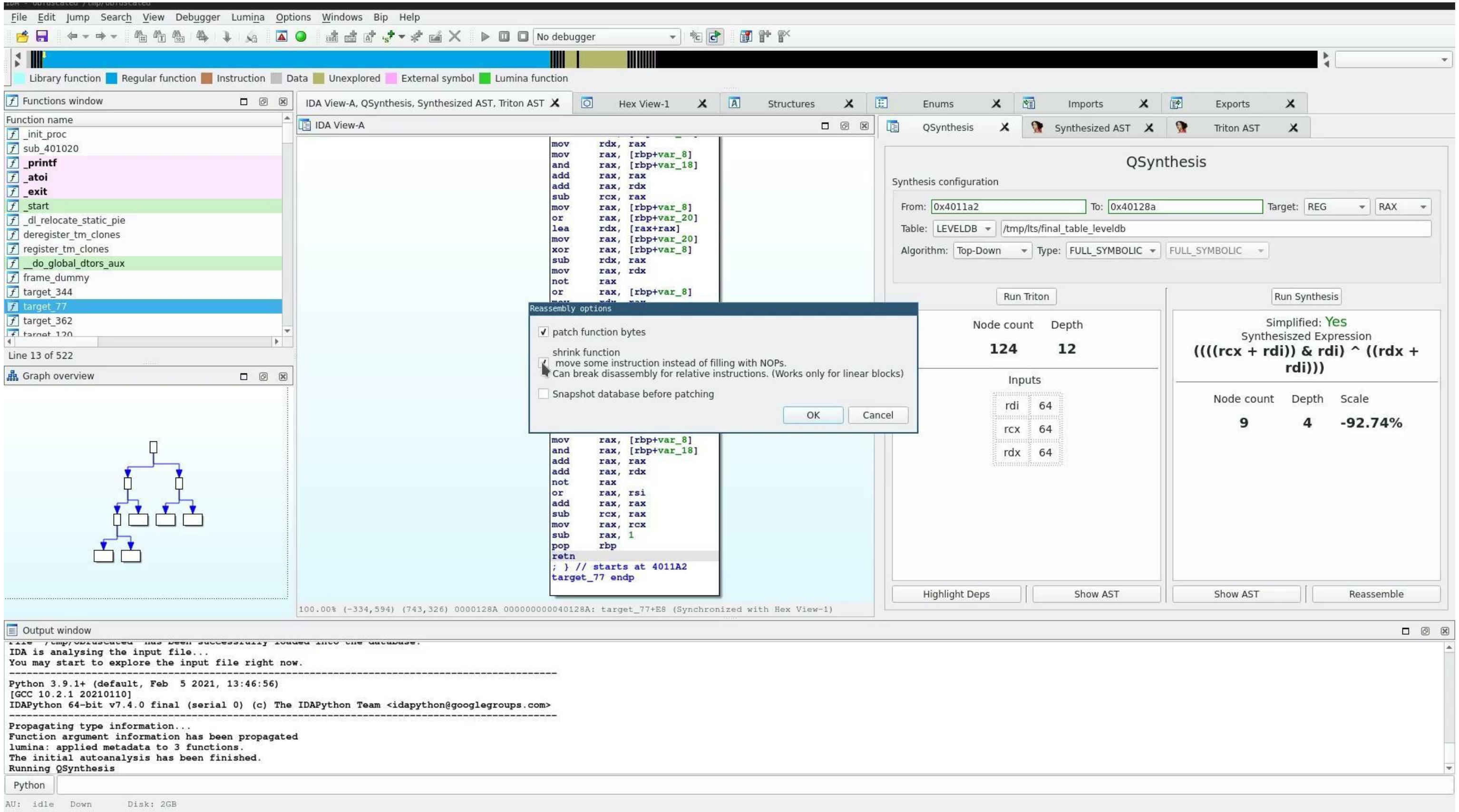
IDA Pro

Integrated as a plugin

FastAPI

To serve a table as a REST API

IDA Integration



Function name
- _init_proc
- sub_401020
- _printf
- _atoi
- _exit
- _start
- _dl_relocate_static_pie
- deregister_tm_clones
- register_tm_clones
- __do_global_dtors_aux
- frame_dummy
- target_344
- target_77
- target_362
- target_120

Line 13 of 522

```
mov     rdx, rax
mov     rax, [rbp+var_8]
and     rax, [rbp+var_18]
add     rax, rax
add     rax, rdx
sub     rcx, rax
mov     rax, [rbp+var_8]
or      rax, [rbp+var_20]
lea     rdx, [rax+rax]
mov     rax, [rbp+var_20]
xor     rax, [rbp+var_8]
sub     rdx, rax
mov     rax, rdx
not     rax
or      rax, [rbp+var_8]
```

Reassembly options

☑ patch function bytes

☑ shrink function
move some instruction instead of filling with NOPs.
Can break disassembly for relative instructions. (Works only for linear blocks)

☐ Snapshot database before patching

OK | Cancel

```
mov     rax, [rbp+var_8]
and     rax, [rbp+var_18]
add     rax, rax
add     rax, rdx
not     rax
or      rax, rsi
add     rax, rax
sub     rcx, rax
mov     rax, rcx
sub     rax, 1
pop     rbp
retn
; } // starts at 4011A2
target_77 endp
```

```
100.00% (-334,594) (743,326) 0000128A 000000000040128A: target_77+E8 (Synchronized with Hex View-1)
```

QSynthesis

Synthesis configuration

From: 0x4011a2 To: 0x40128a Target: REG RAX

Table: LEVELDB /tmp/lts/final_table_leveldb

Algorithm: Top-Down Type: FULL_SYMBOLIC FULL_SYMBOLIC

Run Triton | Run Synthesis

Node count	Depth
124	12

Inputs

rdi	64
rcx	64
rdx	64

Simplified: Yes

Synthesized Expression

((((rcx + rdi)) & rdi) ^ ((rdx + rdi)))

Node count	Depth	Scale
9	4	-92.74%

Highlight Deps | Show AST | Show AST | Reassemble

```
IDA is analysing the input file...
You may start to explore the input file right now.
---------------------------------------------------------------------------------------
Python 3.9.1+ (default, Feb  5 2021, 13:46:56)
[GCC 10.2.1 20210110]
IDAPython 64-bit v7.4.0 final (serial 0) (c) The IDAPython Team <idapython@googlegroups.com>
---------------------------------------------------------------------------------------
Propagating type information...
Function argument information has been propagated
lumina: applied metadata to 3 functions.
The initial autoanalysis has been finished.
Running QSynthesis
```

```
AU:  idle   Down      Disk: 2GB
```

https://youtu.be/AwZs56YajJw

Use-Cases

(getting our hands dirty!)

Attacking YANSOllvm

Transforms:

- **VM:** transforms basic operators (+, ⊕..) with function calls
- **Merge**: merges all internal linkage functions in a single one
- **Flattening**: CFG flattening
- **Connect**: splits basic blocks and uses switch to add false branches
- **ObfCon**: obfuscates constants with MBAs
- **BB2func**: splits & extracts basic blocks in new functions
- **ObfCall**: changes internal linkage function calling convention

YANSOllvm

Yet Another Not So Obfuscated LLVM

LLVM Version

Based on the release version 9.0.1. Other version might work as well, but one has to merge/rebase the X86 related code.

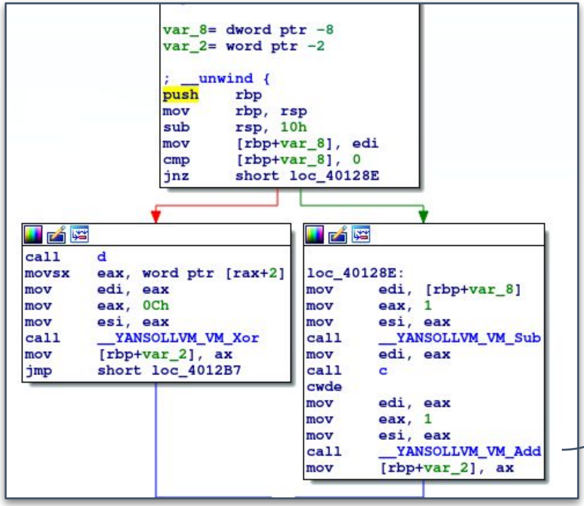
Build

```
wget https://github.com/llvm/llvm-project/releases/download/llvmorg-9.0.1/llvm-9.0.1.src.tar.xz
tar xf llvm-9.0.1.src.tar.xz && cd llvm-9.0.1.src
git init
git remote add origin https://github.com/emc2314/YANSOllvm.git
```

https://github.com/emc2314/YANSOllvm

⇒ There are plenty of other Obfuscator-LLVM derivatives used in the wild

YANSOllvm: VM obfuscation

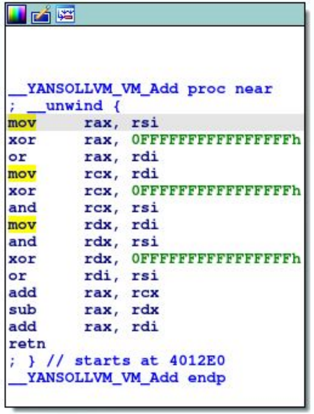


```
var_8= dword ptr -8
var_2= word ptr -2

; __unwind {
push    rbp
mov     rbp, rsp
sub     rsp, 10h
mov     [rbp+var_8], edi
cmp     [rbp+var_8], 0
jnz     short loc_40128E
```

```
call    d
movsx   eax, word ptr [rax+2]
mov     edi, eax
mov     eax, 0Ch
mov     esi, eax
call    __YANSOLLVM_VM_Xor
mov     [rbp+var_2], ax
jmp     short loc_4012B7
```

```
loc_40128E:
mov     edi, [rbp+var_8]
mov     eax, 1
mov     esi, eax
call    __YANSOLLVM_VM_Sub
mov     edi, eax
call    c
cwde
mov     edi, eax
mov     eax, 1
mov     esi, eax
call    __YANSOLLVM_VM_Add
mov     [rbp+var_2], ax
```



```
__YANSOLLVM_VM_Add proc near
; __unwind {
mov     rax, rsi
xor     rax, 0FFFFFFFFFFFFFFFFh
or      rax, rdi
mov     rcx, rdi
xor     rcx, 0FFFFFFFFFFFFFFFFh
and     rcx, rsi
mov     rdx, rdi
and     rdx, rsi
xor     rdx, 0FFFFFFFFFFFFFFFFh
or      rdi, rsi
add     rax, rcx
sub     rax, rdx
add     rax, rdi
retn
; } // starts at 4012E0
__YANSOLLVM_VM_Add endp
```

Synthesized and reassembled to

```
lea rax, [rsi+rdi]
ret
```

⇒ We then could go further by removing calls and replacing them by the operation directly

YANSOllvm: MBA used

OpaqueConstant

MBAs

- `((~x | 0x7AFAFA69) & 0xA061440) + ((x & 0x1050504) | 0x1010104) == 185013572`
- `p1*(x|any)**2 != p2*(y|any)**2`
- `x + y = x^y + 2*(x & y)`
- `x ^ y = (x|~y) - 3*(~(x|y)) + 2*(~x) - y`

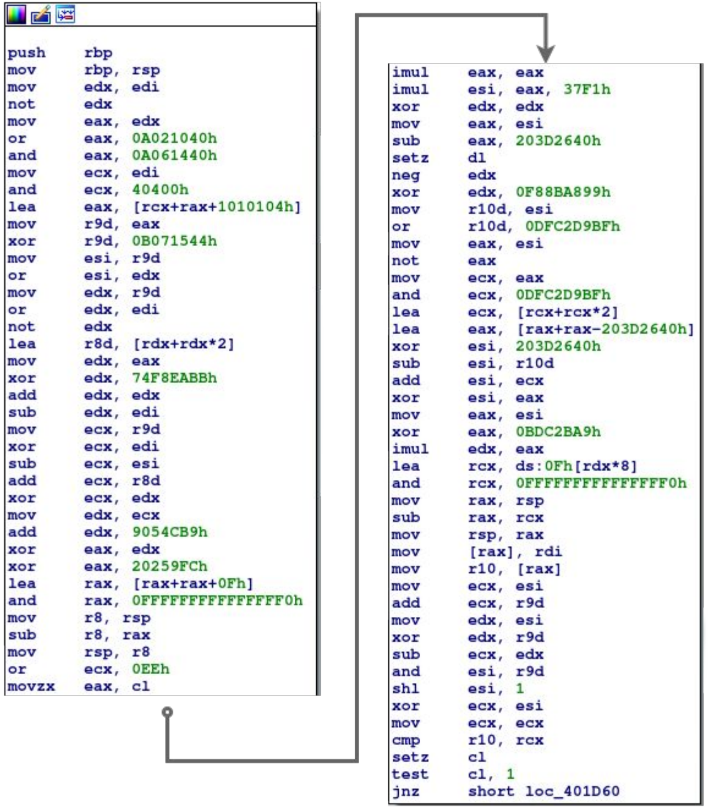
`x + y`	`(x|~y)+(~x&y)-(~(x&y))+(x|y)`
`x - y`	`x + ~y + 1`
`x << y`	`/`
`x >a y`	`/`
`x >l y`	`/`
`x & y`	`-(~(x&y)) + (~x|y) + (x&~y)`
`x | y`	`(x^y) + y - (~x&y)`
`x ^ y`	`x + y - ((x&y) << 1)`

About MBA & constants:

expression using constants: `a & 0xdeadbeef` ⇒ ✖ tables do not contains constants

constants: `0xd00dfeed` ⇒ ✔ can synthesize it !

Example: Opaque Constant



```
push    rbp
mov     rbp, rsp
mov     edx, edi
not     edx
mov     eax, edx
or      eax, 0A021040h
and     eax, 0A061440h
mov     ecx, edi
and     ecx, 40400h
lea     eax, [rcx+rax+1010104h]
mov     r9d, eax
xor     r9d, 0B071544h
mov     esi, r9d
or      esi, edx
mov     edx, r9d
or      edx, edi
not     edx
lea     r8d, [rdx+rdx*2]
mov     edx, eax
xor     edx, 74F8EABBh
add     edx, edx
sub     edx, edi
mov     ecx, r9d
xor     ecx, edi
sub     ecx, esi
add     ecx, r8d
xor     ecx, edx
mov     edx, ecx
add     edx, 9054CB9h
xor     eax, edx
xor     eax, 20259FCh
lea     rax, [rax+rax+0Fh]
and     rax, 0FFFFFFFFFFFFFFF0h
mov     r8, rsp
sub     r8, rax
mov     rsp, r8
or      ecx, 0EEh
movzx   eax, cl
```

```
imul    eax, eax
imul    esi, eax, 37F1h
xor     edx, edx
mov     eax, esi
sub     eax, 203D2640h
setz    dl
neg     edx
xor     edx, 0F88BA899h
mov     r10d, esi
or      r10d, 0DFC2D9BFh
mov     eax, esi
not     eax
mov     ecx, eax
and     ecx, 0DFC2D9BFh
lea     ecx, [rcx+rcx*2]
lea     eax, [rax+rax-203D2640h]
xor     esi, 203D2640h
sub     esi, r10d
add     esi, ecx
xor     esi, eax
mov     eax, esi
xor     eax, 0BDC2BA9h
imul    edx, eax
lea     rcx, ds:0Fh[rdx*8]
and     rcx, 0FFFFFFFFFFFFFFF0h
mov     rax, rsp
sub     rax, rcx
mov     rsp, rax
mov     [rax], rdi
mov     r10, [rax]
mov     ecx, esi
add     ecx, r9d
mov     edx, esi
xor     edx, r9d
sub     ecx, edx
and     esi, r9d
shl     esi, 1
xor     ecx, esi
mov     ecx, ecx
cmp     r10, rcx
setz    cl
test    cl, 1
jnz     short loc_401D60
```

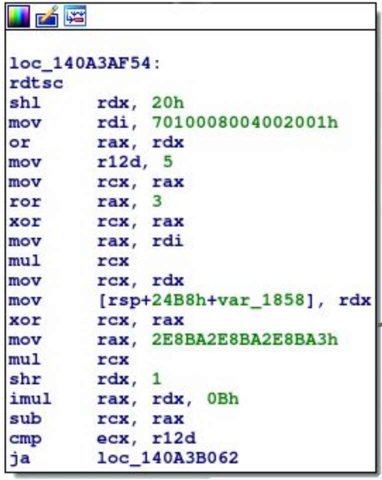
Value
synthesized

⇒ 0x0

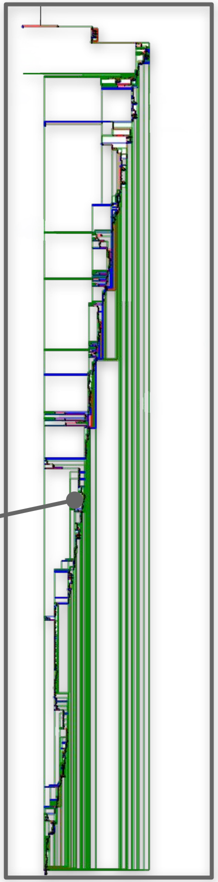
blackbox I/O optimization
If the evaluation of all inputs produces the same output, thus the expression encodes a constant.

Windows Warbird

⇒ Part of the Windows kernel is known to be obfuscated with a framework called **Warbird**. More specifically **PatchGuard** features are obfuscated. We gave a very quick look at the `PatchGuardInit` function.

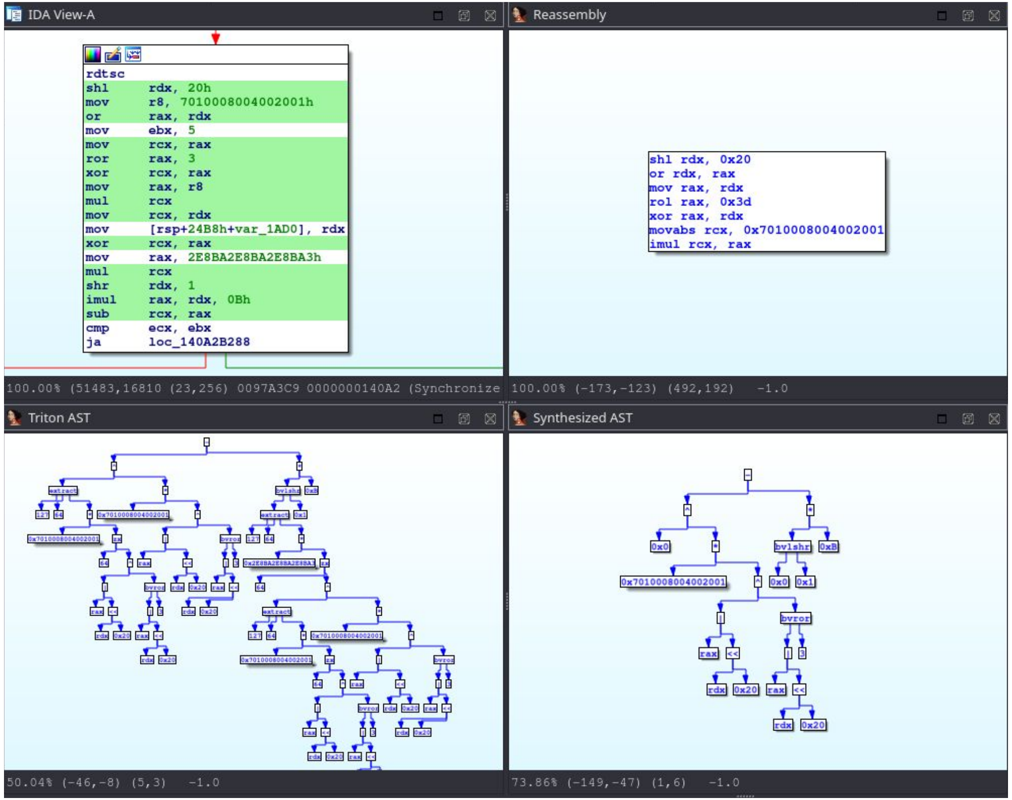


```
loc_140A3AF54:
rdtsc
shl     rdx, 20h
mov     rdi, 7010008004002001h
or      rax, rdx
mov     r12d, 5
mov     rcx, rax
ror     rax, 3
xor     rcx, rax
mov     rax, rdi
mul     rcx
mov     rcx, rdx
mov     [rsp+24B8h+var_1858], rdx
xor     rcx, rax
mov     rax, 2E8BA2E8BA2E8BA3h
mul     rcx
shr     rdx, 1
imul    rax, rdx, 0Bh
sub     rcx, rax
cmp     ecx, r12d
ja      loc_140A3B062
```



*thanks Damien for pinpointing me that function

Windows Warbird



```
rdtsc
shl     rdx, 20h
mov     r8, 7010008004002001h
or      rax, rdx
mov     ebx, 5
mov     rcx, rax
ror     rax, 3
xor     rcx, rax
mov     rax, r8
mul     rcx
mov     rcx, rdx
mov     [rsp+24B8h+var_1AD0], rdx
xor     rcx, rax
mov     rax, 2E8BA2E8BA2E8BA3h
mul     rcx
shr     rdx, 1
imul    rax, rdx, 0Bh
sub     rcx, rax
cmp     ecx, ebx
ja      loc_140A2B288
```

```
shl rdx, 0x20
or rdx, rax
mov rax, rdx
rol rax, 0x3d
xor rax, rdx
movabs rcx, 0x7010008004002001
imul rcx, rax
```

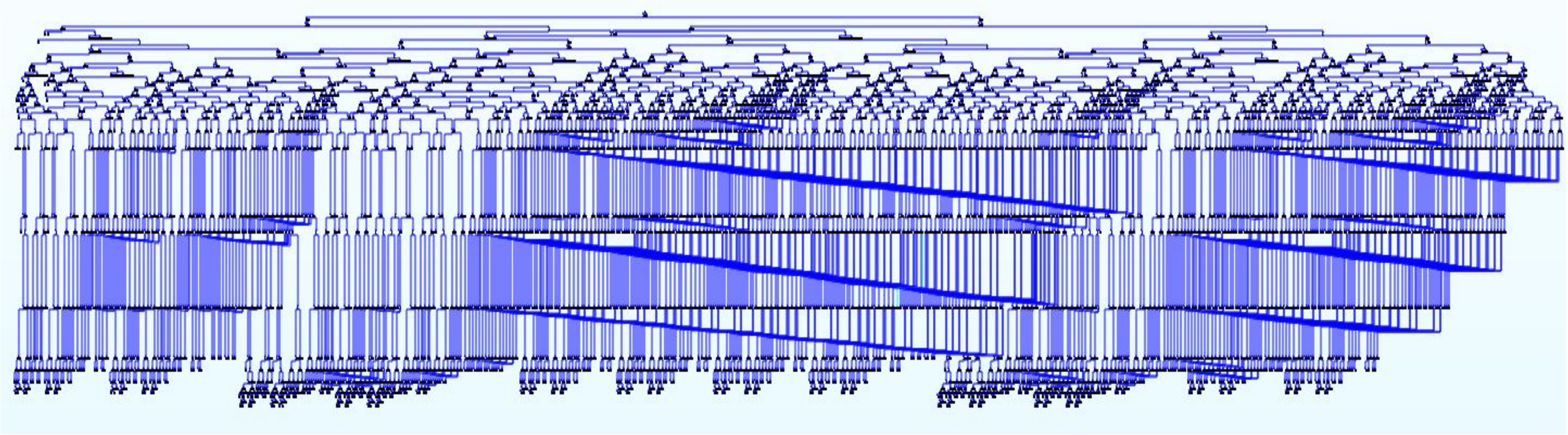
⇒ Deobfuscating it would require a deeper understanding of the function **and more time!**

(more detailed analyses of Warbird here & here)

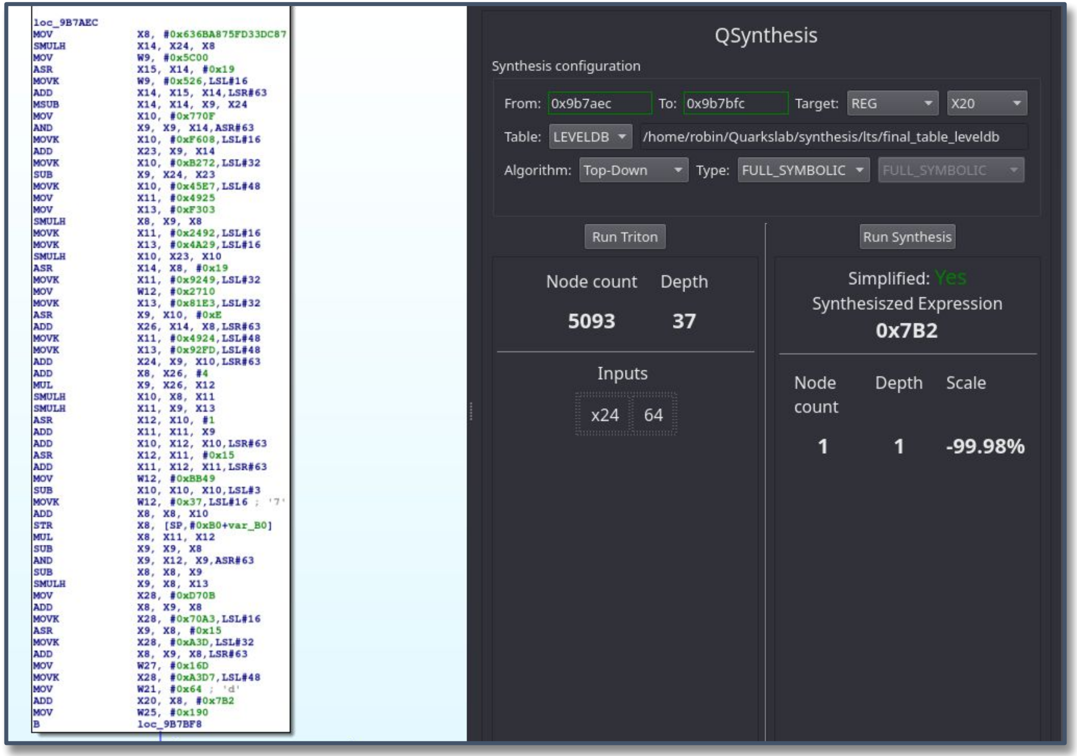
Messaging Application

Contains beautiful MBAs

⇓



Messaging Application



```
loc_9B7AEC
MOV             X8, #0x636BA875FD33DC87
SMULH           X14, X24, X8
MOV             W9, #0x5C00
ASR             X15, X14, #0x19
MOVK            W9, #0x526,LSL#16
ADD             X14, X15, X14,LSR#63
MSUB            X14, X14, X9, X24
MOV             X10, #0x770F
AND             X9, X9, X14,ASR#63
MOVK            X10, #0xF608,LSL#16
ADD             X23, X9, X14
MOVK            X10, #0xB272,LSL#32
SUB             X9, X24, X23
MOVK            X10, #0x45E7,LSL#48
MOV             X11, #0x4925
MOV             X13, #0xF303
SMULH           X8, X9, X8
MOVK            X11, #0x2492,LSL#16
MOVK            X13, #0x4A29,LSL#16
SMULH           X10, X23, X10
ASR             X14, X8, #0x19
MOVK            X11, #0x9249,LSL#32
MOV             W12, #0x2710
MOVK            X13, #0x81E3,LSL#32
ASR             X9, X10, #0xE
ADD             X26, X14, X8,LSR#63
MOVK            X11, #0x4924,LSL#48
MOVK            X13, #0x92FD,LSL#48
ADD             X24, X9, X10,LSR#63
ADD             X8, X26, #4
MUL             X9, X26, X12
SMULH           X10, X8, X11
SMULH           X11, X9, X13
ASR             X12, X10, #1
ADD             X11, X11, X9
ADD             X10, X12, X10,LSR#63
ASR             X12, X11, #0x15
ADD             X11, X12, X11,LSR#63
MOV             W12, #0xBB49
SUB             X10, X10, X10,LSL#3
MOVK            W12, #0x37,LSL#16 ; '7'
ADD             X8, X8, X10
STR             X8, [SP,#0xB0+var_B0]
MUL             X8, X11, X12
SUB             X9, X9, X8
AND             X9, X12, X9,ASR#63
SUB             X8, X8, X9
SMULH           X9, X8, X13
MOV             X28, #0xD70B
ADD             X8, X9, X8
MOVK            X28, #0x70A3,LSL#16
ASR             X9, X8, #0x15
MOVK            X28, #0xA3D,LSL#32
ADD             X8, X9, X8,LSR#63
MOV             W27, #0x16D
MOVK            X28, #0xA3D7,LSL#48
MOV             W21, #0x64 ; 'd'
ADD             X20, X8, #0x7B2
MOV             W25, #0x190
B               loc_9B7BF8
```

QSynthesis

Synthesis configuration

From: 0x9b7aec To: 0x9b7bfc Target: REG X20

Table: LEVELDB /home/robin/Quarkslab/synthesis/lts/final_table_leveldb

Algorithm: Top-Down Type: FULL_SYMBOLIC FULL_SYMBOLIC

Node count	Depth
5093	37

Inputs

x24	64

Simplified: Yes

Synthesiszed Expression

0x7B2

Node count	Depth	Scale
1	1	-99.98%

⇒ We managed to synthesize many MBAs *(but as usual it is mixed with other transformations and we do not really know what we are synthesizing)*

Conclusion

QSynthesis Conclusion

Greybox algorithm

The greybox algorithm strongly **reduces** the need for huge tables and enable opportunistically **synthesizing sub-expressions**

*(thus tables shall be more **representative** than exhaustive introducing constants etc)*

Next plans

- Breaking MBA using constants *(we have ideas on mechanisms that can be integrated within the synthesis algorithm but with some ad-hoc checks)*
- Restoring original simplification algorithm potency *(by fixing some complexity induced by Triton)*

Takeaways

- Breaking the obfuscation is crucial as it is the first step before further reversing

- Synthesis only help on a sub-part of the deobfuscation process:
  - it addresses PB#2: deobfuscating a data-flow expression
  - but **do not** addresses PB#1: locating the data to deobfuscate

- We do use these techniques to **assess** and continuously **improve** the strength of our own obfuscator *(Quarks AppShield)*

- *(As usual)* what makes obfuscation potent is carefully mixing obfuscation passes

Acknowledgement

- **Luigi Coniglio** how kickstarted that approach in our dynamic tracing framework Qtrace
- **Jonathan Salwan** that tweaked Triton to make it more efficient on this kind of use-cases
- My Quarkslab's colleagues, and people of the synthesis community with whom I had stimulating discussions

Thank you !
Q & A

✉ rdavid@quarkslab.com
@RobinDavid1

Quarkslab