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
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...

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

\[a + b\]
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 ..)

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 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
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**.

---

**A Synthesizing Superoptimizer**

Raimondas Simonaitis
ISE Engineering
raimondas@ise.it.uu.se

Yang Chen
Nvidia, Inc.
yangchen@nvidia.com

Graeme Logie
Microsoft, Inc.
graeme.logie@microsoft.com

John Regan
University of Dublin
reganj@i.tcd.ie

Peter Collinge
Google, Inc.
peter.collinge@google.com

Joakim Taneja
University of Utah
johannes@ece.utah.edu

---

**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 is in some cases expert handwritten code.

**Publications**

**STOKE** has appeared in a number of publications:

- [Stochastic Superoptimization – ASPL '13](#)
- [Data-Driven Equivalence Checking – DOPLA '13](#)
- [Stochastic Optimization of Floating-Point Programs with Tunable Precision – PLDI '14](#)
- [Conditionally Correct Superoptimization – DOPLA '15](#)
- [Stochastic Program Optimization – CACM '16](#)
- [Stratified Synthesis: Automatically Learning the x86-64 Instruction Set – PLDI '16](#)
- [Sound Loop Superoptimizer for Google Native Client – ASPL '17](#)

---

**STOKE**: stochastic superoptimizer at assembly level (x86_64)
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**
  - **Syntia**
    - Monte-Carlo Search Tree based approach

- **2017**
  - **SSPAM**
    - Approach based on pattern matching rewriting rules and arithmetic simplification *(not synthesis per se)*
  - **F. Biondi et al.**
    - SMT based approach to defeat MBAs

- **2020**
  - **QSynth**
    - Offline enumerative search based approach *(our approach)*

- **2021**
  - **LOKI**
    - (obfuscation oriented) discuss how to defeat synthesis approaches
  - **Xyntia**
    - Search-based approach using S-Metaheuristics *(expected CCS 2021)*
  - **MSynth**
    - Implementation of QSynth algorithm with MIASM framework
Greybox Synthesis

*(design & principles of our algorithm)*
Our algorithm is based on an enumerative approach backed by symbolic execution and a synthesis (itself based on two sub-components).
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`
- `add     rax, rdi`
- `retn`

**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`

---

**AST**
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.

**Whitebox component**

**Blackbox component**
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/output behavior

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

**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

((((((a ∧ ¬b) + b) << 1) ∧ ¬((a ∨ b) − (a ∧ b)))) << 1) − (((a ∧ ¬b) + b) << 1) ⊕ ((a ∨ b) − (a ∧ b))))
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:

\[
V_{out} \mapsto expr
\]

<table>
<thead>
<tr>
<th>( V_{out} )</th>
<th>expr</th>
</tr>
</thead>
<tbody>
<tr>
<td>(&lt;1, 2, 5&gt;)</td>
<td>A + B</td>
</tr>
<tr>
<td>(&lt;-1, -4, 3&gt;)</td>
<td>A - B</td>
</tr>
<tr>
<td>(&lt;1, -1, 5&gt;)</td>
<td>A</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

- **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?
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)

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

https://youtu.be/ID_PEVseeC1
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).*

**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/VQRg3LHC6Lw

https://youtu.be/G1lB0qmwLaI
Algorithm Visualization

https://youtu.be/Nz8KC1HtgI
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 memoized 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)*.

↓

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
Tables are limited by the enumerative approach, combining some variables \((a, b, c..)\) with some operators \((+, -, & \ldots)\). 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.

<table>
<thead>
<tr>
<th>Original</th>
<th>Linearized</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a - (c - a))</td>
<td>(2a - c)</td>
</tr>
<tr>
<td>((a-b) - (a + a))</td>
<td>(-a - b)</td>
</tr>
<tr>
<td>(a + (b * b))</td>
<td>(b^2 + a)</td>
</tr>
<tr>
<td>\ldots</td>
<td>\ldots</td>
</tr>
</tbody>
</table>

### Pros/Cons:

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

- **Cons:**
  - we thus do not use it in practice
Expression Learning

**Problem**

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

⇒ We also now introduce simple constants in our table generation process.
Benchmarks
Paper benchmarks

Comparison with Syntia

<table>
<thead>
<tr>
<th>Simplification</th>
<th>Mean expr. size</th>
<th>Simplification</th>
<th>Mean scale factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Orig</td>
<td>Obf₅</td>
<td>Synt</td>
<td>Ø</td>
</tr>
<tr>
<td>Syntia</td>
<td>/</td>
<td>/</td>
<td>/</td>
</tr>
<tr>
<td>QSynth</td>
<td>3.97</td>
<td>203.19</td>
<td>3.71</td>
</tr>
</tbody>
</table>

Orig, Obf₅, Synt are rsp. original, obfuscated (source, binary level) and synthesized exprs

Accuracy & Speed

<table>
<thead>
<tr>
<th>Semantic</th>
<th>Time</th>
<th>Sym.Ex</th>
<th>Synthesis</th>
<th>Total</th>
<th>per fun.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntia</td>
<td>/</td>
<td>/</td>
<td>/</td>
<td>34 min</td>
<td>4.08s</td>
</tr>
<tr>
<td>QSynth</td>
<td>500</td>
<td>1m20s</td>
<td>15s</td>
<td>1m35s</td>
<td>0.19s</td>
</tr>
</tbody>
</table>

Against Tigress

<table>
<thead>
<tr>
<th>Simplification</th>
<th>Mean expr. size</th>
<th>Simplification</th>
<th>Mean Scale factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Orig</td>
<td>Obf₅</td>
<td>Synt</td>
<td>Ø</td>
</tr>
<tr>
<td>Dataset 2</td>
<td>EA</td>
<td>13.5</td>
<td>245.81</td>
</tr>
<tr>
<td>Dataset 3</td>
<td>VR-EA</td>
<td>13.5</td>
<td>443.64</td>
</tr>
<tr>
<td>Dataset 4</td>
<td>EA-ED</td>
<td>13.5</td>
<td>9223.46</td>
</tr>
</tbody>
</table>

Orig, Obf₅, Synt are respectively original, obfuscated (source, binary level) and synthesized expressions

Accuracy & Speed

<table>
<thead>
<tr>
<th>Semantic</th>
<th>Time</th>
<th>Sym.Ex</th>
<th>Synthesis</th>
<th>Total</th>
<th>per fun.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dataset 2</td>
<td>EA</td>
<td>OK: 413</td>
<td>1m7s</td>
<td>1m42s</td>
<td>2m49s</td>
</tr>
<tr>
<td>Dataset 3</td>
<td>VR-EA</td>
<td>OK: 401</td>
<td>17m10s</td>
<td>2m46s</td>
<td>19m56s</td>
</tr>
<tr>
<td>Dataset 4</td>
<td>EA-ED</td>
<td>-</td>
<td>13m18s</td>
<td>2h7m</td>
<td>2h21m</td>
</tr>
</tbody>
</table>

⇒ Results were promising!
<table>
<thead>
<tr>
<th>Algorithm Evolution</th>
<th>Mean size</th>
<th>Simplification</th>
<th>Mean Scale factor</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Synt Expr.</td>
<td>@</td>
<td>Partial</td>
<td>Full</td>
</tr>
<tr>
<td><strong>Dataset 1</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Syntia</td>
<td>Paper</td>
<td>3.71</td>
<td>0</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>New</td>
<td>3.71</td>
<td>0</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>Mul</td>
<td>3.71</td>
<td>0</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>Concat</td>
<td>3.71</td>
<td>0</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>LDB</td>
<td><strong>3.71</strong></td>
<td>0</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>370M</td>
<td>3.85</td>
<td>0</td>
<td>500</td>
</tr>
<tr>
<td><strong>Dataset 2</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EA</td>
<td>Paper</td>
<td>21.92</td>
<td>0</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>New</td>
<td>19.93</td>
<td>0</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>Mul</td>
<td>19.48</td>
<td>1</td>
<td>499</td>
</tr>
<tr>
<td></td>
<td>Concat</td>
<td>19.48</td>
<td>1</td>
<td>499</td>
</tr>
<tr>
<td></td>
<td>LDB</td>
<td>19.48</td>
<td>1</td>
<td>499</td>
</tr>
<tr>
<td></td>
<td>370M</td>
<td><strong>17.37</strong></td>
<td>2</td>
<td>498</td>
</tr>
<tr>
<td><strong>Dataset 3</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VR-EA</td>
<td>Paper</td>
<td>25.42</td>
<td>0</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>New</td>
<td>75.14</td>
<td>14</td>
<td>486</td>
</tr>
<tr>
<td></td>
<td>Mul</td>
<td>73.98</td>
<td>18</td>
<td>482</td>
</tr>
<tr>
<td></td>
<td>Concat</td>
<td>21.50</td>
<td>0</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>LDB</td>
<td>21.52</td>
<td>0</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>370M</td>
<td><strong>19.07</strong></td>
<td>0</td>
<td>500</td>
</tr>
<tr>
<td><strong>Dataset 4</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EA-ED</td>
<td>Paper</td>
<td>3812.84</td>
<td>5</td>
<td>234</td>
</tr>
<tr>
<td></td>
<td>New</td>
<td>483.26</td>
<td>0</td>
<td>239</td>
</tr>
<tr>
<td></td>
<td>Mul</td>
<td>375.36</td>
<td>0</td>
<td>239</td>
</tr>
<tr>
<td></td>
<td>Concat</td>
<td>375.36</td>
<td>0</td>
<td>239</td>
</tr>
<tr>
<td></td>
<td>LDB</td>
<td>375.45</td>
<td>0</td>
<td>239</td>
</tr>
<tr>
<td></td>
<td>370M</td>
<td><strong>315.01</strong></td>
<td>0</td>
<td>239</td>
</tr>
</tbody>
</table>
### Benchmarks improvements

<table>
<thead>
<tr>
<th>Algorithm Evolution</th>
<th>Mean size</th>
<th>Simplification</th>
<th>Mean Scale factor</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dataset 1 Syntia</td>
<td>Paper</td>
<td>3.71</td>
<td>0</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>New</td>
<td>3.71</td>
<td>0</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>Mul</td>
<td>3.71</td>
<td>0</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>Concat</td>
<td>3.71</td>
<td>0</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>LDB</td>
<td>3.71</td>
<td>0</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>370M</td>
<td>3.85</td>
<td>0</td>
<td>500</td>
</tr>
<tr>
<td>Dataset 2 EA</td>
<td>Paper</td>
<td>21.92</td>
<td>0</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>New</td>
<td>19.93</td>
<td>0</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>Mul</td>
<td>19.48</td>
<td>1</td>
<td>499</td>
</tr>
<tr>
<td></td>
<td>Concat</td>
<td>19.48</td>
<td>1</td>
<td>499</td>
</tr>
<tr>
<td></td>
<td>LDB</td>
<td>19.48</td>
<td>1</td>
<td>499</td>
</tr>
<tr>
<td></td>
<td>370M</td>
<td><strong>17.37</strong></td>
<td>2</td>
<td>498</td>
</tr>
<tr>
<td>Dataset 3 VR-EA</td>
<td>Paper</td>
<td>25.42</td>
<td>0</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>New</td>
<td>75.14</td>
<td>14</td>
<td>486</td>
</tr>
<tr>
<td></td>
<td>Mul</td>
<td>73.98</td>
<td>18</td>
<td>482</td>
</tr>
<tr>
<td></td>
<td>Concat</td>
<td>21.50</td>
<td>0</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>LDB</td>
<td>21.52</td>
<td>0</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>370M</td>
<td><strong>19.07</strong></td>
<td>0</td>
<td>500</td>
</tr>
<tr>
<td>Dataset 4 EA-ED</td>
<td>Paper</td>
<td>3812.84</td>
<td>5</td>
<td>234</td>
</tr>
<tr>
<td></td>
<td>New</td>
<td>483.26</td>
<td>0</td>
<td>239</td>
</tr>
<tr>
<td></td>
<td>Mul</td>
<td>375.36</td>
<td>0</td>
<td>239</td>
</tr>
<tr>
<td></td>
<td>Concat</td>
<td>375.36</td>
<td>0</td>
<td>239</td>
</tr>
<tr>
<td></td>
<td>LDB</td>
<td>375.45</td>
<td>0</td>
<td>239</td>
</tr>
<tr>
<td></td>
<td>370M</td>
<td><strong>315.01</strong></td>
<td>0</td>
<td>239</td>
</tr>
</tbody>
</table>

- **Paper**: Original results
- **Syntia**: ED + EA (very simple)
- **EA**: EncodeArithmetic ⇒ MBA
- **VR-EA**: Virtualization + EA
- **EA-ED**: EA + EncodeData
### Benchmarks improvements

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

**Speed improvement ranging from 31% to 67%**

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Mean size</th>
<th>Simplification</th>
<th>Mean Scale factor</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Syntax</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Paper</td>
<td>3.71</td>
<td>0</td>
<td>x35.03</td>
</tr>
<tr>
<td></td>
<td>New</td>
<td>3.71</td>
<td>0</td>
<td>x35.03</td>
</tr>
<tr>
<td></td>
<td>Mul</td>
<td>3.71</td>
<td>0</td>
<td>x35.03</td>
</tr>
<tr>
<td></td>
<td>Concat</td>
<td>3.71</td>
<td>0</td>
<td>x35.03</td>
</tr>
<tr>
<td></td>
<td>LDB</td>
<td>3.71</td>
<td>0</td>
<td>x35.03</td>
</tr>
<tr>
<td></td>
<td>370M</td>
<td>3.85</td>
<td>0</td>
<td>x35.03</td>
</tr>
<tr>
<td>Dataset 1</td>
<td>Syntia</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Paper</td>
<td>21.92</td>
<td>0</td>
<td>x18.34</td>
</tr>
<tr>
<td></td>
<td>New</td>
<td>19.93</td>
<td>0</td>
<td>x18.34</td>
</tr>
<tr>
<td></td>
<td>Mul</td>
<td>19.48</td>
<td>1</td>
<td>x18.34</td>
</tr>
<tr>
<td></td>
<td>Concat</td>
<td>19.48</td>
<td>1</td>
<td>x18.34</td>
</tr>
<tr>
<td></td>
<td>LDB</td>
<td>17.37</td>
<td>2</td>
<td>x18.34</td>
</tr>
<tr>
<td></td>
<td>370M</td>
<td>19.07</td>
<td>0</td>
<td>x18.34</td>
</tr>
<tr>
<td>Dataset 2</td>
<td>EA-ED</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Paper</td>
<td>25.42</td>
<td>0</td>
<td>x18.34</td>
</tr>
<tr>
<td></td>
<td>New</td>
<td>75.14</td>
<td>14</td>
<td>x18.34</td>
</tr>
<tr>
<td></td>
<td>Mul</td>
<td>73.98</td>
<td>18</td>
<td>x18.34</td>
</tr>
<tr>
<td></td>
<td>Concat</td>
<td>21.50</td>
<td>0</td>
<td>x18.34</td>
</tr>
<tr>
<td></td>
<td>LDB</td>
<td>21.52</td>
<td>0</td>
<td>x18.34</td>
</tr>
<tr>
<td></td>
<td>370M</td>
<td>19.07</td>
<td>0</td>
<td>x18.34</td>
</tr>
<tr>
<td>Dataset 4</td>
<td>EA-ED</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Paper</td>
<td>[3812.81]</td>
<td>5</td>
<td>x405.25</td>
</tr>
<tr>
<td></td>
<td>New</td>
<td>483.26</td>
<td>0</td>
<td>x458.47</td>
</tr>
<tr>
<td></td>
<td>Mul</td>
<td>375.36</td>
<td>0</td>
<td>x458.47</td>
</tr>
<tr>
<td></td>
<td>Concat</td>
<td>375.36</td>
<td>0</td>
<td>x458.47</td>
</tr>
<tr>
<td></td>
<td>LDB</td>
<td>375.45</td>
<td>0</td>
<td>x458.47</td>
</tr>
<tr>
<td></td>
<td>370M</td>
<td>315.01</td>
<td>0</td>
<td>x458.47</td>
</tr>
</tbody>
</table>
Implementation

(in the QSynthesis utility)
QSynthesis

Triton
Dynamic Symbolic Execution framework

Arybo
Used for reassembly features (bit vector IR in ANF form)

Qtrace
Dynamic Tracing Framework & Time Travel Debugger (TTD)

QBDI
Dynamic Binary Instrumentation Framework

Ilvmlite

FastAPI
To serve a table as a REST API

ida pro
Integrated as a plugin

llvmlite
dragonffi
For the JITTing of expression evaluation (during table generation)

Level-DB
As database for table storage

QSynthesis Framework
(developed in Python)
IDA Integration

https://youtu.be/AwZs56YaJjw
Use-Cases

(getting our hands dirty!)
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

⇒ There are plenty of other Obfuscator-LLVM derivatives used in the wild
YANSOllvm: VM obfuscation

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

\[
\text{lea } \text{rax, [rsi+rdi]} \quad \text{ret}
\]
About MBA & constants:

- Expression using constants: `a & 0xdeadbeef`  ⇒  ❌ tables do not contains constants
- Constants: `0xd00dfeed`  ⇒  ✓ can synthesize it!

### OpaqueConstant

- `((~x | 0x7AFAP FA69) & 0xA0614A0) + ((x & 0x105050504) | 0x101010104) == 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`

### MBAs

<table>
<thead>
<tr>
<th>Expression</th>
<th>MBA Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>x + y</code></td>
<td>`(x</td>
</tr>
<tr>
<td><code>x - y</code></td>
<td><code>x + ~y + 1</code></td>
</tr>
<tr>
<td><code>x &lt;&lt; y</code></td>
<td><code>/</code></td>
</tr>
<tr>
<td><code>x &gt;a y</code></td>
<td><code>/</code></td>
</tr>
<tr>
<td><code>x &gt;l y</code></td>
<td><code>/</code></td>
</tr>
<tr>
<td><code>x &amp; y</code></td>
<td>`-(~(x&amp;y)) + (~x</td>
</tr>
<tr>
<td>`x</td>
<td>y`</td>
</tr>
<tr>
<td><code>x ^ y</code></td>
<td><code>x + y - ((x&amp;y) &lt;&lt; 1)</code></td>
</tr>
</tbody>
</table>
Example: Opaque Constant

If the evaluation of all inputs produces the same output, thus the expression encodes a constant.

```
push rbp
mov rbp, rsp
mov edx, edi
not edx
mov eax, edx
or eax, 0A021040h
and eax, 0A061440h
mov edx, edi
and edx, 40400h
lea eax, [rcx+rax*1010104h]
mov r9d, eax
xor r9d, OBO71544h
mov esi, r9d
or esi, edx
mov edx, r9d
or edx, edi
not edx
lea edx, [rdx+rdx*2]
mov edx, eax
xor edx, 74FBEAB8h
add edx, edx
sub edx, edx
mov eax, edx
xor eax, eax
add eax, edx
xor eax, edx
mov eax, ecx
add eax, 90544BC9h
xor eax, edx
xor eax, 2035FCH
lea eax, [rax+rax*0Fh]
and eax, 0FFFFF0FF0FF0FF0h
mov r8, r8
sub r8, rax
mov r8, r8
or ecx, 00EH
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, 0F888888899h
mov r10d, esi
or r10d, 0DFC2D98Fh
mov eax, esi
not eax
mov eax, eax
and eax, 0DFC2D98Fh
lea eax, [rcx+rax*2]
lea eax, [rax+rax-203D2640h]
xor esi, 203D2640h
sub esi, r10d
add esi, eax
xor eax, eax
mov eax, esi
xor eax, 0BD52B49h
imul edx, eax
lea rcx, dx:0Fh[rdx*8]
and rcx, 0FFFFFFFFFFF0FF0h
movrax, rcx
sub rax, rcx
mov r8p, rax
mov [rax], rdl
mov r10, [rax]
mov ecx, esi
add ecx, r9d
mov edx, esi
xor edx, r9d
sub ecx, edx
and esi, r9d
shr esi, 1
xor eax, rsi
mov ecx, ecx
cmp r10, rcx
setz ol
test cl, 1
jnz short loc_401D60
```

⇒ 0x0
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.

*thanks Damien for pinpointing me that function*
⇒ 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

↓
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