emproof

Breaking Mixed Boolean-Arithmetic Obfuscation in Real-World Applications

Tim Blazytko

@mr_phrazer
synthesis.to
tim@blazytko.to

Nicolò Altamura

@nicolodev
nicolo.dev
seekbytes@protonmail.com

About Us

- Tim Blazytko
  - Chief Scientist & Head of Engineering, co-founder of emproof
  - designs software protections for embedded devices
  - trainer for (de)obfuscation and reverse engineering techniques

- Nicolò Altamura
  - CS master student at University of Verona
  - security engineer at emproof
  - code deobfuscation by night

Setting the Scene

- Obfuscated code, MBAs
- Attacks on MBAs
- State of Binary Analysis Tooling

Motivation

~~Prevent~~ **Complicate** reverse engineering attempts.

- intellectual property
- malicious payloads
- Digital Rights Management

MBAs

Mixed Boolean-Arithmetic

What does this expression compute?

$$(x \oplus y) + 2 \cdot (x \wedge y)$$

Mixed Boolean-Arithmetic

What does this expression compute?

$$(x \oplus y) + 2 \cdot (x \wedge y)$$
$$= \mathsf{x + y}$$

Mixed Boolean-Arithmetic

What does this expression compute?

$$(((x \oplus y) + ((x \wedge y) \ll 1)) \vee z) + (((x \oplus y) + ((x \wedge y) \ll 1)) \wedge z)$$

Mixed Boolean-Arithmetic

What does this expression compute?

$$(((x \oplus y) + ((x \wedge y) \ll 1)) \vee z) + (((x \oplus y) + ((x \wedge y) \ll 1)) \wedge z)$$
$$= \mathbf{x + y + z}$$

- Boolean identities?
- Arithmetic identities?
- Karnaugh-Veitch maps?

$$A \cdot 0 = 0$$
$$A + B = \overline{\overline{A} \cdot \overline{B}}$$
$$x^2 - y^2 = (x + y)(x - y)$$

CD \ AB	00	01	11	10
10	0	0	1	1
11	0	0	1	1
01	0	0	0	1
00	0	1	1	1

Mixed Boolean-Arithmetic

Boolean-arithmetic algebra BA[n]

$$(\mathrm{B}^n, \wedge, \vee, \oplus, \neg, \leq, \geq, >, <, \leq^s, \geq^s, >^s, <^s, \neq, =, \gg^s, \gg, \ll, +, -, \cdot)$$

is a Boolean-arithmetic algebra BA[n], for $n > 0$, $\mathrm{B} = \{0, 1\}$.

BA[n] includes, amongst others, both:

- Boolean algebra $(\mathrm{B}^n, \wedge, \vee, \neg)$,
- Integer modular ring $\mathbb{Z}/(2^n)$.

No techniques to simplify such expressions easily!

Boolean-arithmetic algebra BA[n]

$$(\mathrm{B}^n, \wedge, \vee, \oplus, \neg, \leq, \geq, >, <, \leq^s, \geq^s, >^s, <^s, \neq, =, \gg^s, \gg, \ll, +, -, \cdot)$$

is a Boolean-arithmetic algebra BA[n], for $n > 0$, $\mathrm{B} = \{0, 1\}$.

Commonly found in gaming, advanced DRM solutions and malware

BA[n] includes, amongst others, both:

- Boolean algebra $(\mathrm{B}^n, \wedge, \vee, \neg)$,
- Integer modular ring $\mathbb{Z}/(2^n)$.

No techniques to simplify such expressions easily!

Obfuscation Primitive

Hiding Computations

```
uint8_t hidden_computation(uint8_t x, uint8_t y){
   int k1 = (x & (((x & y) + (x & y)) + (x ^ y)));
   int k2 = ((x ^ y) + (x & y) << 1) - y;
   int k3 = (x ^ (((x & y) + (x & y)) + (x ^ y)));

   return k1 + k2 + k3;
}
```

```
uint8_t hidden_computation(uint8_t x, uint8_t y){
   int k1 = (x & (((x & y) + (x & y)) + (x ^ y)));
   int k2 = ((x ^ y) + (x & y) << 1) - y;
   int k3 =                                    + (x ^ y)));

   return k1 + k2 + k3;
}
```

2 * x - y

Hiding Constants

```
uint8_t constant(uint8_t x, uint8_t y) {
    uint8_t k1 = 202 + 231 * (y | x) + 244 * (x | y) + 7 * (y & y);
    uint8_t k2 = 180 * y + 85 * (y ^ y);
    uint8_t k3 = 155 * (x | x) + 139 * (x ^ x) + 206 * (x & y);
    uint8_t k4 = 174 * (y | y) + 65 * (x & x);
    uint8_t k5 = 115 * x + 67 * ~x + 93 * ~y;
    uint8_t k6 = 35 * (y ^ x) + 246 * (x ^ y) + 63 * (y & x);

    return k1 + k2 + k3 + k4 + k5 + k6;
}
```

```
uint8_t constant(uint8_t x, uint8_t y) {
    uint8_t k1 = 202 + 231 * (y | x) + 244 * (x | y) + 7 * (y & y);
    uint8_t k2 = 180 * y + 85 * (y ^ y);
    uint8_t k3 = 155 * (x | x) + 139 * (x ^ x) + 206 * (x & y);
    uint8_t k4 = 174 * (y | y) + 65 * (x & x);
    uint8_t k5 = 115 * x + 67 * [illegible] 93 * ~y;
    uint8_t k6 = 35 * (y ^ x) + 246 * (x ^ y) + 63 * (y & x);

    return k1 + k2 + k3 + k4 + k5 + k6;
}
```

42

Opaque Predicates

```
uint8_t opaque_predicate(uint8_t x, uint8_t y) {
   uint8_t k1 = 96 + 159 * (y ^ x) + 160 * y + 194 * (y | x) + 96 * ~x;
   uint8_t k2 = 193 + 64 * ~y + 65 * (y & y) + 130 * x + 129 * ~x;

   if (k1 != k2) {
      // dead code
      return x - y;
   }

   return x + y;
}
```

Opaque Predicates

```
uint8_t opaque_predicate(uint8_t x, uint8_t y) {
    uint8_t k1 = 96 + 159 * (y ^ x) + 160 * y + 194 * (y | x) + 96 * ~x;
    uint8_t k2 = 193 + 64 * ~y + 65 * (y & y) + 130 * x + 129 * ~x;

    if (k1 != k2) {
        // dead code
        return x - y;
    }

    return x + y;
}
```

Opaque Predicates

```
uint8_t opaque_predicate(uint8_t x, uint8_t y) {
    uint8_t k1 = 96 + 159 * (y ^ x) + 160 * y + 194 * (y | x) + 96 * ~x;
    uint8_t k2 = 193 + 64 * ~y + 65 * (y & y) + 130 * x + 129 * ~x;

    if (k1 != k2) {
        // dead c
        return x
    }

    return x + y;
}
```

$x + y \mathrel{!}= x + y$

MBA Construction

Linear MBAs

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

Linear MBAs

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

- $(x \oplus y) + 2 \cdot (x \wedge y)$
- $(((x \oplus y) + ((x \wedge y) \ll 1)) \vee z) + (((x \oplus y) + ((x \wedge y) \ll 1)) \wedge z)$

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

No multiplication between variables

Polynomial MBAs

$$\sum_{i \in I} a_i \left(\prod_{j \in J_i} e_{i,j}(x_1, \ldots, x_t) \right)$$

Polynomial MBAs

$$\sum_{i \in I} a_i \left(\prod_{j \in J_i} e_{i,j}(x_1, \ldots, x_t) \right)$$

- $85 \cdot (x \vee y \wedge z)^3 + (xy \wedge x) + (xz)^2$
- $xy + 2 \cdot (x \wedge y) + 3 \cdot (x \wedge \neg y)(x \vee y) - 5$

Polynomial MBAs

$$\sum_{i \in I} a_i \left(\prod_{j \in J_i} e_{i,j}(x_1, \ldots, x_t) \right)$$

Multiplication between variables

- $85 \cdot (x \vee y \wedge z)^3 + (xy \wedge x) + (xz)^2$
- $xy + 2 \cdot (x \wedge y) + 3 \cdot (x \wedge \neg y)(x \vee y) - 5$

Recursive MBA Rewriting

$$x - y \cdot (x + y)$$

Rewriting rules:

1) $x + y \rightarrow (x \oplus y) + 2 \cdot (x \wedge y)$
2) $x \oplus y \rightarrow (x \vee y) - (x \wedge y)$

$\ldots$

47) $x \wedge y \rightarrow (\neg x \vee y) - \neg x$

Recursive MBA Rewriting

$$x - y \cdot (x + y)$$

Rewriting rules:

1) $x + y \rightarrow (x \oplus y) + 2 \cdot (x \wedge y)$
2) $x \oplus y \rightarrow (x \vee y) - (x \wedge y)$

$\ldots$

47) $x \wedge y \rightarrow (\neg x \vee y) - \neg x$

Recursive MBA Rewriting

$$x - y \cdot (x + y)$$

Rewriting rules:

1) $x + y \rightarrow (x \oplus y) + 2 \cdot (x \wedge y)$
2) $x \oplus y \rightarrow (x \vee y) - (x \wedge y)$

$\dots$

47) $x \wedge y \rightarrow (\neg x \vee y) - \neg x$

Recursive MBA Rewriting

$$x - y \cdot (x + y)$$

Rewriting rules:

1) $x + y \rightarrow (x \oplus y) + 2 \cdot (x \wedge y)$
2) $x \oplus y \rightarrow (x \vee y) - (x \wedge y)$

$\ldots$

47) $x \wedge y \rightarrow (\neg x \vee y) - \neg x$

$$x - y \cdot ((x \oplus y) + 2 \cdot (x \wedge y))$$

Recursive MBA Rewriting

$$x - y \cdot (x + y)$$

$$\Bigg\Updownarrow \textit{equal}$$

$$x - y \cdot ((x \oplus y) + 2 \cdot (x \land y))$$

Rewriting rules:

1) $x + y \rightarrow (x \oplus y) + 2 \cdot (x \land y)$
2) $x \oplus y \rightarrow (x \lor y) - (x \land y)$

$\dots$

47) $x \land y \rightarrow (\neg x \lor y) - \neg x$

Recursive MBA Rewriting

$$x - y \cdot (x + y)$$

Rewriting rules:

1) $x + y \rightarrow (x \oplus y) + 2 \cdot (x \wedge y)$
2) $x \oplus y \rightarrow (x \vee y) - (x \wedge y)$

$\ldots$

47) $x \wedge y \rightarrow (\neg x \vee y) - \neg x$

$$\boxed{x - y \cdot ((x \oplus y) + 2 \cdot (x \wedge y))}$$

final expression

Traditional Approach

Recursive MBA Rewriting

$$x - y \cdot (x + y)$$

Rewriting rules:

1) $x + y \rightarrow (x \oplus y) + 2 \cdot (x \wedge y)$
2) $x \oplus y \rightarrow (x \vee y) - (x \wedge y)$

$\dots$

47) $x \wedge y \rightarrow (\neg x \vee y) - \neg x$

$$x - y \cdot ((x \oplus y) + 2 \cdot (x \wedge y))$$

final expression

Recursive MBA Rewriting

$$x - y \cdot (x + y)$$

Rewriting rules:

1) $x + y \rightarrow (x \oplus y) + 2 \cdot (x \wedge y)$

2) $x \oplus y \rightarrow (x \vee y) - (x \wedge y)$

$\dots$

847,000) $x \wedge y \rightarrow (\neg x \vee y) - \neg x$

$$x - y \cdot ((x \oplus y) + 2 \cdot (x \wedge y))$$

final expression

$$x - y \cdot (x + y)$$

Rewriting rules:

1) $x + y \rightarrow (x \oplus y) + 2 \cdot (x \wedge y)$
2) $x \oplus y \rightarrow (x \vee y) - (x \wedge y)$

Lookup table w/ *all* identities

$$x - y \cdot ((x \oplus y) + 2 \cdot (x \wedge y))$$

final expression

Recursive MBA Rewriting

$$x - y \cdot (x + y)$$

Rewriting rules:

$$\begin{array}{rl} 1) & x + y \rightarrow (x \oplus y) + 2 \cdot (x \wedge y) \\ 2) & x \oplus y \rightarrow (x \vee y) - (x \wedge y) \\ & \dots \\ 847{,}000) & x \wedge y \rightarrow (\neg x \vee y) - \neg x \end{array}$$

$$x - y \cdot ((x \oplus y) + 2 \cdot (x \wedge y))$$

~~*final expression*~~

Recursive MBA Rewriting

$$x - y \cdot (x + y)$$

Rewriting rules:

1) $x + y \rightarrow (x \oplus y) + 2 \cdot (x \wedge y)$
2) $x \oplus y \rightarrow (x \vee y) - (x \wedge y)$

$\dots$

847,000) $x \wedge y \rightarrow (\neg x \vee y) - \neg x$

$$x - y \cdot ((x \oplus y) + 2 \cdot (x \wedge y))$$

~~*final expression*~~

Recursive Approach

$$x - y \cdot (x + y)$$

$$x - y \cdot ((x \oplus y) + 2 \cdot (x \wedge y))$$

Rewriting rules:

1) $x + y \rightarrow (x \oplus y) + 2 \cdot (x \wedge y)$
2) $x \oplus y \rightarrow (x \vee y) - (x \wedge y)$

$\dots$

847,000) $x \wedge y \rightarrow (\neg x \vee y) - \neg x$

Recursive Approach

Recursive MBA Rewriting

$$x - y \cdot ((x \oplus y) + 2 \cdot (x \land y))$$

Rewriting rules:

$$
\begin{array}{rl}
1) & x + y \rightarrow (x \oplus y) + 2 \cdot (x \land y) \\
2) & x \oplus y \rightarrow (x \lor y) - (x \land y) \\
 & \ldots \\
847{,}000) & x \land y \rightarrow (\neg x \lor y) - \neg x
\end{array}
$$

$$x - y \cdot ((x \oplus y) + 2 \cdot (x \land y))$$

Recursive Approach

$$x - y \cdot ((x \oplus y) + 2 \cdot (x \wedge y))$$

Rewriting rules:

1) $x + y \rightarrow (x \oplus y) + 2 \cdot (x \wedge y)$
2) $x \oplus y \rightarrow (x \vee y) - (x \wedge y)$

$\ldots$

847,000) $x \wedge y \rightarrow (\neg x \vee y) - \neg x$

$$x - y \cdot (((x \vee y) - (x \wedge y)) + 2 \cdot (x \wedge y))$$

Recursive Approach

$$x - y \cdot ((x \oplus y) + 2 \cdot (x \wedge y))$$

Rewriting rules:

$$\begin{array}{rl} 1) & x + y \rightarrow (x \oplus y) + 2 \cdot (x \wedge y) \\ 2) & x \oplus y \rightarrow (x \vee y) - (x \wedge y) \\ & \dots \\ 847{,}000) & x \wedge y \rightarrow (\neg x \vee y) - \neg x \end{array}$$

$$x - y \cdot (((x \vee y) - (x \wedge y)) + 2 \cdot (x \wedge y))$$

Recursive Approach

$$x - y \cdot ((x \oplus y) + 2 \cdot (x \wedge y))$$

$$\Updownarrow \; \textit{equal}$$

$$x - y \cdot (((x \vee y) - (x \wedge y)) + 2 \cdot (x \wedge y))$$

Rewriting rules:

1) $x + y \rightarrow (x \oplus y) + 2 \cdot (x \wedge y)$
2) $x \oplus y \rightarrow (x \vee y) - (x \wedge y)$

...

847,000) $x \wedge y \rightarrow (\neg x \vee y) - \neg x$

Recursive Approach

Recursive MBA Rewriting

$$x - y \cdot (((x \vee y) - (x \wedge y)) + 2 \cdot (x \wedge y))$$

Rewriting rules:

1)	$x + y \rightarrow (x \oplus y) + 2 \cdot (x \wedge y)$
2)	$x \oplus y \rightarrow (x \vee y) - (x \wedge y)$
	$\ldots$
847,000)	$x \wedge y \rightarrow (\neg x \vee y) - \neg x$

$$x - y \cdot (((x \vee y) - (x \wedge y)) + 2 \cdot (x \wedge y))$$

Recursive Approach

$$x - y \cdot (((x \vee y) - (x \wedge y)) + 2 \cdot (x \wedge y))$$

Rewriting rules:

$$\begin{array}{rl} 1) & x + y \to (x \oplus y) + 2 \cdot (x \wedge y) \\ 2) & x \oplus y \to (x \vee y) - (x \wedge y) \\ & \ldots \\ 847{,}000) & x \wedge y \to (\neg x \vee y) - \neg x \end{array}$$

$$x - y \cdot (((x \vee y) - (x \wedge y)) + 2 \cdot (x \wedge y))$$

$$x - y \cdot (((x \vee y) - (x \wedge y)) + 2 \cdot (x \wedge y))$$

Rewriting rules:

$$\begin{array}{rl}
1) & x + y \rightarrow (x \oplus y) + 2 \cdot (x \wedge y) \\
2) & x \oplus y \rightarrow (x \vee y) - (x \wedge y) \\
 & \ldots \\
847{,}000) & x \wedge y \rightarrow (\neg x \vee y) - \neg x
\end{array}$$

$$x - y \cdot (((x \vee y) - ((\neg x \vee y) - \neg x)) + 2 \cdot (x \wedge y))$$

Recursive MBA Rewriting

$$x - y \cdot (((x \vee y) - (x \wedge y)) + 2 \cdot (x \wedge y))$$

Rewriting rules:

$$\begin{aligned} 1) &\quad x + y \rightarrow (x \oplus y) + 2 \cdot (x \wedge y) \\ 2) &\quad x \oplus y \rightarrow (x \vee y) - (x \wedge y) \\ &\quad \dots \\ 847{,}000) &\quad x \wedge y \rightarrow (\neg x \vee y) - \neg x \end{aligned}$$

$$\boxed{x - y \cdot (((x \vee y) - ((\neg x \vee y) - \neg x)) + 2 \cdot (x \wedge y))}$$

final expression

$$x - y \cdot (((x \vee y) - (x \wedge y)) + 2 \cdot (x \wedge y))$$

Rewriting rules:

1) $x + y \rightarrow (x \oplus y) + 2 \cdot (x \wedge y)$
2) $x \oplus y \rightarrow (x \vee y) - (x \wedge y)$

Recursive Rewriting

$(\neg x \vee y) - \neg x$

$$x - y \cdot (((x \vee y) - ((\neg x \vee y) - \neg x)) + 2 \cdot (x \wedge y))$$

Insertion of Identities

$$x - y \cdot (x + y)$$

Rewrite as:

$$expr \equiv h^{-1}(h(expr))$$

Insertion of Identities

$$x - y \cdot (x + y)$$

Rewrite as:

$$expr \equiv h^{-1}(h(expr))$$

Insertion of Identities

$$x - y \cdot (x + y)$$

Rewrite as:

$$expr \equiv h^{-1}(h(expr))$$

Insertion of Identities

$$x - y \cdot (x + y)$$

Rewrite as:

$$expr \equiv h^{-1}(h(expr))$$

Invertible function on 1 byte:

$$\begin{aligned} h: &\quad a \mapsto 39a + 23 \\ h^{-1}: &\quad a \mapsto 151a + 111 \end{aligned}$$

Insertion of Identities

$$x - y \cdot (x + y)$$

Rewrite as:

$$expr \equiv h^{-1}(h(expr))$$

Invertible function on 1 byte:

$$\begin{aligned} h: &\quad a \mapsto 39a + 23 \\ h^{-1}: &\quad a \mapsto 151a + 111 \end{aligned}$$

$$\implies expr \equiv h^{-1}(h(expr)) \mod 2^8$$

Insertion of Identities

$$x - y \cdot (x + y)$$

Rewrite as:

$$expr \equiv h^{-1}(h(expr))$$

Invertible function on 1 byte:

$$\begin{aligned} h: &\quad a \mapsto 39a + 23 \\ h^{-1}: &\quad a \mapsto 151a + 111 \end{aligned}$$

$$\implies expr \equiv h^{-1}(h(expr)) \mod 2^8$$

Insertion of Identities

$$x - y \cdot (x + y)$$

$$x - y \cdot (h^{-1}(h(x + y)))$$

Rewrite as:

$$expr \equiv h^{-1}(h(expr))$$

Invertible function on 1 byte:

$$h : \quad a \mapsto 39a + 23$$
$$h^{-1} : \quad a \mapsto 151a + 111$$

$$\implies expr \equiv h^{-1}(h(expr)) \mod 2^8$$

Insertion of Identities

$$x - y \cdot (x + y)$$

$$x - y \cdot (h^{-1}(h(x + y)))$$

Rewrite as:

$$expr \equiv h^{-1}(h(expr))$$

Invertible function on 1 byte:

$$h: \quad a \mapsto 39a + 23$$
$$h^{-1}: \quad a \mapsto 151a + 111$$

$$\implies expr \equiv h^{-1}(h(expr)) \mod 2^8$$

Insertion of Identities

$$x - y \cdot (x + y)$$

$$x - y \cdot (h^{-1}(h(x + y)))$$

$$x - y \cdot (h^{-1}(39 \cdot (x + y) + 23))$$

Rewrite as:

$$expr \equiv h^{-1}(h(expr))$$

Invertible function on 1 byte:

$$h: \quad a \mapsto 39a + 23$$
$$h^{-1}: \quad a \mapsto 151a + 111$$

$$\implies expr \equiv h^{-1}(h(expr)) \mod 2^8$$

Insertion of Identities

$$x - y \cdot (x + y)$$

$$x - y \cdot (h^{-1}(h(x + y)))$$

$$x - y \cdot (h^{-1}(39 \cdot (x + y) + 23))$$

Rewrite as:

$$expr \equiv h^{-1}(h(expr))$$

Invertible function on 1 byte:

$$\begin{aligned} h: \quad & a \mapsto 39a + 23 \\ h^{-1}: \quad & a \mapsto 151a + 111 \end{aligned}$$

$$\implies expr \equiv h^{-1}(h(expr)) \mod 2^8$$

Insertion of Identities

$$x - y \cdot (x + y)$$

$$x - y \cdot (h^{-1}(h(x + y)))$$

$$x - y \cdot (h^{-1}(39 \cdot (x + y) + 23))$$

$$x - y \cdot (151 \cdot (39 \cdot (x + y) + 23) + 111)$$

Rewrite as:

$$expr \equiv h^{-1}(h(expr))$$

Invertible function on 1 byte:

$$\begin{aligned} h: &\quad a \mapsto 39a + 23 \\ h^{-1}: &\quad a \mapsto 151a + 111 \end{aligned}$$

$$\Longrightarrow \quad expr \equiv h^{-1}(h(expr)) \mod 2^8$$

Insertion of Identities

$$x - y \cdot (x + y)$$

$$\Updownarrow \; \textit{equal}$$

$$x - y \cdot (151 \cdot (39 \cdot (x + y) + 23) + 111)$$

Rewrite as:

$$expr \equiv h^{-1}(h(expr))$$

Invertible function on 1 byte:

$$h: \quad a \mapsto 39a + 23$$
$$h^{-1}: \quad a \mapsto 151a + 111$$

$$\Longrightarrow \quad expr \equiv h^{-1}(h(expr)) \mod 2^8$$

Binary Permutation Polynomial Inversion and Application to Obfuscation Techniques

Lucas Barthelemy[abd]
lbarthelemy@quarkslab.com

Ninon Eyrolles[a]
neyrolles@quarkslab.com

Guenaël Renault[bce]
guenael.renault@upmc.fr

Raphaël Roblin[bd]
raph.roblin@gmail.com

[a]Quarkslab, Paris, France
[b]Sorbonne Universités, UPMC Univ Paris 06, F-75005, Paris, France
[c]CNRS, UMR 7606, LIP6, F-75005, Paris, France
[d]UPMC Computer Science Master Department, SFPN Course
[e]Inria, Paris Center, PolSys Project

How to attack MBAs?

Algebraic Attacks

Efficient Deobfuscation of Linear Mixed Boolean-Arithmetic Expressions

Benjamin Reichenwallner & Peter Meerwald-Stadler
Denuvo GmbH
Salzburg, Austria

https://github.com/DenuvoSoftwareSolutions/SiMBA

Efficient Deobfuscation of Linear Mixed Boolean-Arithmetic Expressions

Powerful attack without binary analysis tooling

Denuvo GmbH
Salzburg, Austria

https://github.com/DenuvoSoftwareSolutions/SiMBA

Simplification of General Mixed Boolean-Arithmetic Expressions: GAMBA

Benjamin Reichenwallner & Peter Meerwald-Stadler
Denuvo GmbH
Salzburg, Austria

https://arxiv.org/pdf/2305.06763.pdf

Simplification of General Mixed Boolean-Arithmetic Expressions: GAMBA

Limited success on unseen MBA patterns

Denuvo GmbH
Salzburg, Austria

---

https://arxiv.org/pdf/2305.06763.pdf

Code Analysis Attacks

Inspecting Compiler Optimizations on Mixed Boolean Arithmetic Obfuscation

Rachael Little
University of New Hampshire
rachael.little@unh.edu

Dongpeng Xu
University of New Hampshire
dongpeng.xu@unh.edu

https://www.ndss-symposium.org/wp-content/uploads/bar2025-final7.pdf

Inspecting Compiler Optimizations on Mixed Boolean-Arithmetic Obfuscation

Limited simplification capabilities

Rachael Little
University of New Hampshire
rachael.little@unh.edu

Dongpeng Xu
University of New Hampshire
dongpeng.xu@unh.edu

https://www.ndss-symposium.org/wp-content/uploads/bar2025-final7.pdf

Symbolic Execution

```
int mixed_boolean(int A, int B, int C) {
    int result;

    result = (((1438524315 + ((((1438524315 + C) + 1438524315 * ((2956783114 - -1478456685 * C) |
    (-1478456685 * (1668620215 - A) - 2956783115))) + A) - 1553572265)) + 1438524315 * ((2956783114 -
    -1478456685 * ((((1438524315 + C) + 1438524315 * ((2956783114 - -1478456685 * C) | (-1478456685 *
    (1668620215 - A) - 2956783115))) + A) - 1553572265)) | (-1478456685 * (1668620215 - B) -
    2956783115))) - ((1438524315 + (1668620215 - ((((1438524315 + C) + 1438524315 * ((2956783114 -
    -1478456685 * C) | (-1478456685 * (1668620215 - A) - 2956783115))) + A) - 1553572265))) +
    1438524315 * ((2956783114 - -1478456685 * (1668620215 - ((((1438524315 + C) + 1438524315 *
    ((2956783114 - -1478456685 * C) | (-1478456685 * (1668620215 - A) - 2956783115))) + A) -
    1553572265))) | (-1478456685 * B - 2956783115)))) + 1553572265;

    return -1478456685 * result - 2956783115;
}
```

Symbolic Execution

```
RAX = {((((- ((((- (((@32[(RBP_init+0xFFFFFFFFFFFFFFEC)][31:32]?(({@32[(RBP_init+0xFFFFFFFFFFFFFFEC)] 0 32, 0xFFFFFFFF 32 64}*0x55BE239B),({@32[(RBP_init+0xFFFFFFFFFFFFFFEC)] 0 32, 0x0 32 64}*0x55BE239B)))[0:32]+(@32[(RBP_init
+0xFFFFFFFFFFFFFFFC)][31:32]?(({@32[(RBP_init+0xFFFFFFFFFFFFFFFC)] 0 32, 0xFFFFFFFF 32 64}*0x55BE239B),({@32[(RBP_init+0xFFFFFFFFFFFFFFFC)] 0 32, 0x0 32 64}*0x55BE239B)))[0:32]+((((- (((@32[(RBP_init+0xFFFFFFFFFFFFFFEC)][31:32]?(({@32
[(RBP_init+0xFFFFFFFFFFFFFFEC)] 0 32, 0xFFFFFFFF 32 64}*0x55BE239B),({@32[(RBP_init+0xFFFFFFFFFFFFFFEC)] 0 32, 0x0 32 64}*0x55BE239B)))[0:32]+0x5C99A1A9)[31:32]?(({(((@32[(RBP_init+0xFFFFFFFFFFFFFFEC)][31:32]?(({@32[(RBP_init
+0xFFFFFFFFFFFFFFEC)] 0 32, 0xFFFFFFFF 32 64}*0x55BE239B),({@32[(RBP_init+0xFFFFFFFFFFFFFFEC)] 0 32, 0x0 32 64}*0x55BE239B)))[0:32]+0x5C99A1A9) 0 32, 0xFFFFFFFF 32 64}*0xFFFFFFFFA7E08A93),({(((@32[(RBP_init+0xFFFFFFFFFFFFFFEC)]
[31:32]?(({@32[(RBP_init+0xFFFFFFFFFFFFFFEC)] 0 32, 0xFFFFFFFF 32 64}*0x55BE239B),({@32[(RBP_init+0xFFFFFFFFFFFFFFEC)] 0 32, 0x0 32 64}*0x55BE239B)))[0:32]+0x5C99A1A9) 0 32, 0x0 32 64}*0xFFFFFFFFA7E08A93)))[0:32])+0xB03CEE0A)|
((((@32[(RBP_init+0xFFFFFFFFFFFFFFFC)][31:32]?(({@32[(RBP_init+0xFFFFFFFFFFFFFFFC)] 0 32, 0xFFFFFFFF 32 64}*0x55BE239B),({@32[(RBP_init+0xFFFFFFFFFFFFFFFC)] 0 32, 0x0 32 64}*0x55BE239B)))[0:32]+0x5C99A1A9)[31:32]?(({(((@32[(RBP_init
+0xFFFFFFFFFFFFFFFC)][31:32]?(({@32[(RBP_init+0xFFFFFFFFFFFFFFFC)] 0 32, 0xFFFFFFFF 32 64}*0x55BE239B),({@32[(RBP_init+0xFFFFFFFFFFFFFFFC)] 0 32, 0x0 32 64}*0x55BE239B)))[0:32]+0x5C99A1A9) 0 32, 0xFFFFFFFF 32 64}*0x581F756D),({(((@32
[(RBP_init+0xFFFFFFFFFFFFFFFC)][31:32]?(({@32[(RBP_init+0xFFFFFFFFFFFFFFFC)] 0 32, 0xFFFFFFFF 32 64}*0x55BE239B),({@32[(RBP_init+0xFFFFFFFFFFFFFFFC)] 0 32, 0x0 32 64}*0x55BE239B)))[0:32]+0x5C99A1A9) 0 32, 0x0 32 64}*0x581F756D)))
[0:32]+0xB03CEE0A))[31:32]?(({((((- (((@32[(RBP_init+0xFFFFFFFFFFFFFFEC)][31:32]?(({@32[(RBP_init+0xFFFFFFFFFFFFFFEC)] 0 32, 0xFFFFFFFF 32 64}*0x55BE239B),({@32[(RBP_init+0xFFFFFFFFFFFFFFEC)] 0 32, 0x0 32 64}*0x55BE239B)))[0:32]
+0x5C99A1A9)[31:32]?(({(((@32[(RBP_init+0xFFFFFFFFFFFFFFEC)][31:32]?(({@32[(RBP_init+0xFFFFFFFFFFFFFFEC)] 0 32, 0xFFFFFFFF 32 64}*0x55BE239B),({@32[(RBP_init+0xFFFFFFFFFFFFFFEC)] 0 32, 0x0 32 64}*0x55BE239B)))[0:32]+0x5C99A1A9) 0 32,
0xFFFFFFFF 32 64}*0xFFFFFFFFA7E08A93),({(((@32[(RBP_init+0xFFFFFFFFFFFFFFEC)][31:32]?(({@32[(RBP_init+0xFFFFFFFFFFFFFFEC)] 0 32, 0xFFFFFFFF 32 64}*0x55BE239B),({@32[(RBP_init+0xFFFFFFFFFFFFFFEC)] 0 32, 0x0 32 64}*0x55BE239B)))[0:32]
+0x5C99A1A9) 0 32, 0x0 32 64}*0xFFFFFFFFA7E08A93)))[0:32])+0xB03CEE0A)|((((@32[(RBP_init+0xFFFFFFFFFFFFFFFC)][31:32]?(({@32[(RBP_init+0xFFFFFFFFFFFFFFFC)] 0 32, 0xFFFFFFFF 32 64}*0x55BE239B),({@32[(RBP_init+0xFFFFFFFFFFFFFFFC)] 0 32,
0x0 32 64}*0x55BE239B)))[0:32]+0x5C99A1A9)[31:32]?(({(((@32[(RBP_init+0xFFFFFFFFFFFFFFFC)][31:32]?(({@32[(RBP_init+0xFFFFFFFFFFFFFFFC)] 0 32, 0xFFFFFFFF 32 64}*0x55BE239B),({@32[(RBP_init+0xFFFFFFFFFFFFFFFC)] 0 32, 0x0 32 64}
*0x55BE239B)))[0:32]+0x5C99A1A9) 0 32, 0xFFFFFFFF 32 64}*0x581F756D),({(((@32[(RBP_init+0xFFFFFFFFFFFFFFFC)][31:32]?(({@32[(RBP_init+0xFFFFFFFFFFFFFFFC)] 0 32, 0xFFFFFFFF 32 64}*0x55BE239B),({@32[(RBP_init+0xFFFFFFFFFFFFFFFC)] 0 32,
0x0 32 64}*0x55BE239B)))[0:32]+0x5C99A1A9) 0 32, 0x0 32 64}*0x581F756D)))[0:32]+0xB03CEE0A)) 0 32, 0xFFFFFFFF 32 64}*0x55BE239B),({((((- (((@32[(RBP_init+0xFFFFFFFFFFFFFFEC)][31:32]?(({@32[(RBP_init+0xFFFFFFFFFFFFFFEC)] 0 32,
0xFFFFFFFF 32 64}*0x55BE239B),({@32[(RBP_init+0xFFFFFFFFFFFFFFEC)] 0 32, 0x0 32 64}*0x55BE239B)))[0:32]+0x5C99A1A9)[31:32]?(({(((@32[(RBP_init+0xFFFFFFFFFFFFFFEC)][31:32]?(({@32[(RBP_init+0xFFFFFFFFFFFFFFEC)] 0 32, 0xFFFFFFFF 32 64}
*0x55BE239B),({@32[(RBP_init+0xFFFFFFFFFFFFFFEC)] 0 32, 0x0 32 64}*0x55BE239B)))[0:32]+0x5C99A1A9) 0 32, 0xFFFFFFFF 32 64}*0xFFFFFFFFA7E08A93),({(((@32[(RBP_init+0xFFFFFFFFFFFFFFEC)][31:32]?(({@32[(RBP_init+0xFFFFFFFFFFFFFFEC)] 0 32,
0xFFFFFFFF 32 64}*0x55BE239B),({@32[(RBP_init+0xFFFFFFFFFFFFFFEC)] 0 32, 0x0 32 64}*0x55BE239B)))[0:32]+0x5C99A1A9) 0 32, 0x0 32 64}*0xFFFFFFFFA7E08A93)))[0:32])+0xB03CEE0A)|((((@32[(RBP_init+0xFFFFFFFFFFFFFFFC)][31:32]?(({@32
[(RBP_init+0xFFFFFFFFFFFFFFFC)] 0 32, 0xFFFFFFFF 32 64}*0x55BE239B),({@32[(RBP_init+0xFFFFFFFFFFFFFFFC)] 0 32, 0x0 32 64}*0x55BE239B)))[0:32]+0x5C99A1A9)[31:32]?(({(((@32[(RBP_init+0xFFFFFFFFFFFFFFFC)][31:32]?(({@32[(RBP_init
+0xFFFFFFFFFFFFFFFC)] 0 32, 0xFFFFFFFF 32 64}*0x55BE239B),({@32[(RBP_init+0xFFFFFFFFFFFFFFFC)] 0 32, 0x0 32 64}*0x55BE239B)))[0:32]+0x5C99A1A9) 0 32, 0xFFFFFFFF 32 64}*0x581F756D),({(((@32[(RBP_init+0xFFFFFFFFFFFFFFFC)][31:32]?(({@32
[(RBP_init+0xFFFFFFFFFFFFFFFC)] 0 32, 0xFFFFFFFF 32 64}*0x55BE239B),({@32[(RBP_init+0xFFFFFFFFFFFFFFFC)] 0 32, 0x0 32 64}*0x55BE239B)))[0:32]+0x5C99A1A9) 0 32, 0x0 32 64}*0x581F756D)))[0:32]+0xB03CEE0A)) 0 32, 0x0 32 64}
*0x55BE239B)))[0:32]+0xB9334352)[31:32]?(({(((@32[(RBP_init+0xFFFFFFFFFFFFFFEC)][31:32]?(({@32[(RBP_init+0xFFFFFFFFFFFFFFEC)] 0 32, 0xFFFFFFFF 32 64}*0x55BE239B),({@32[(RBP_init+0xFFFFFFFFFFFFFFEC)] 0 32, 0x0 32 64}*0x55BE239B)))
[0:32]+(@32[(RBP_init+0xFFFFFFFFFFFFFFFC)][31:32]?(({@32[(RBP_init+0xFFFFFFFFFFFFFFFC)] 0 32, 0xFFFFFFFF 32 64}*0x55BE239B),({@32[(RBP_init+0xFFFFFFFFFFFFFFFC)] 0 32, 0x0 32 64}*0x55BE239B)))[0:32]+((((- (((@32[(RBP_init
+0xFFFFFFFFFFFFFFEC)][31:32]?(({@32[(RBP_init+0xFFFFFFFFFFFFFFEC)] 0 32, 0xFFFFFFFF 32 64}*0x55BE239B),({@32[(RBP_init+0xFFFFFFFFFFFFFFEC)] 0 32, 0x0 32 64}*0x55BE239B)))[0:32]+0x5C99A1A9)[31:32]?(({(((@32[(RBP_init
+0xFFFFFFFFFFFFFFEC)][31:32]?(({@32[(RBP_init+0xFFFFFFFFFFFFFFEC)] 0 32, 0xFFFFFFFF 32 64}*0x55BE239B),({@32[(RBP_init+0xFFFFFFFFFFFFFFEC)] 0 32, 0x0 32 64}*0x55BE239B)))[0:32]+0x5C99A1A9) 0 32, 0xFFFFFFFF 32 64}
*0xFFFFFFFFA7E08A93),({(((@32[(RBP_init+0xFFFFFFFFFFFFFFEC)][31:32]?(({@32[(RBP_init+0xFFFFFFFFFFFFFFEC)] 0 32, 0xFFFFFFFF 32 64}*0x55BE239B),({@32[(RBP_init+0xFFFFFFFFFFFFFFEC)] 0 32, 0x0 32 64}*0x55BE239B)))[0:32]+0x5C99A1A9) 0 32,
0x0 32 64}*0xFFFFFFFFA7E08A93)))[0:32])+0xB03CEE0A)|((((@32[(RBP_init+0xFFFFFFFFFFFFFFFC)][31:32]?(({@32[(RBP_init+0xFFFFFFFFFFFFFFFC)] 0 32, 0xFFFFFFFF 32 64}*0x55BE239B),({@32[(RBP_init+0xFFFFFFFFFFFFFFFC)] 0 32, 0x0 32 64}
*0x55BE239B)))[0:32]+0x5C99A1A9)[31:32]?(({(((@32[(RBP_init+0xFFFFFFFFFFFFFFFC)][31:32]?(({@32[(RBP_init+0xFFFFFFFFFFFFFFFC)] 0 32, 0xFFFFFFFF 32 64}*0x55BE239B),({@32[(RBP_init+0xFFFFFFFFFFFFFFFC)] 0 32, 0x0 32 64}*0x55BE239B)))
[0:32]+0x5C99A1A9) 0 32, 0xFFFFFFFF 32 64}*0x581F756D),({(((@32[(RBP_init+0xFFFFFFFFFFFFFFFC)][31:32]?(({@32[(RBP_init+0xFFFFFFFFFFFFFFFC)] 0 32, 0xFFFFFFFF 32 64}*0x55BE239B),({@32[(RBP_init+0xFFFFFFFFFFFFFFFC)] 0 32, 0x0 32 64}
*0x55BE239B)))[0:32]+0x5C99A1A9) 0 32, 0x0 32 64}*0x581F756D)))[0:32]+0xB03CEE0A))[31:32]?(({((((- (((@32[(RBP_init+0xFFFFFFFFFFFFFFEC)][31:32]?(({@32[(RBP_init+0xFFFFFFFFFFFFFFEC)] 0 32, 0xFFFFFFFF 32 64}*0x55BE239B),({@32[(RBP_init
+0xFFFFFFFFFFFFFFEC)] 0 32, 0x0 32 64}*0x55BE239B)))[0:32]+0x5C99A1A9)[31:32]?(({(((@32[(RBP_init+0xFFFFFFFFFFFFFFEC)][31:32]?(({@32[(RBP_init+0xFFFFFFFFFFFFFFEC)] 0 32, 0xFFFFFFFF 32 64}*0x55BE239B),({@32[(RBP_init
+0xFFFFFFFFFFFFFFEC)] 0 32, 0x0 32 64}*0x55BE239B)))[0:32]+0x5C99A1A9) 0 32, 0xFFFFFFFF 32 64}*0xFFFFFFFFA7E08A93),({(((@32[(RBP_init+0xFFFFFFFFFFFFFFEC)][31:32]?(({@32[(RBP_init+0xFFFFFFFFFFFFFFEC)] 0 32, 0xFFFFFFFF 32 64}
*0x55BE239B),({@32[(RBP_init+0xFFFFFFFFFFFFFFEC)] 0 32, 0x0 32 64}*0x55BE239B)))[0:32]+0x5C99A1A9) 0 32, 0x0 32 64}*0xFFFFFFFFA7E08A93)))[0:32])+0xB03CEE0A)|((((@32[(RBP_init+0xFFFFFFFFFFFFFFFC)][31:32]?(({@32[(RBP_init
+0xFFFFFFFFFFFFFFFC)] 0 32, 0xFFFFFFFF 32 64}*0x55BE239B),({@32[(RBP_init+0xFFFFFFFFFFFFFFFC)] 0 32, 0x0 32 64}*0x55BE239B)))[0:32]+0x5C99A1A9)[31:32]?(({(((@32[(RBP_init+0xFFFFFFFFFFFFFFFC)][31:32]?(({@32[(RBP_init
+0xFFFFFFFFFFFFFFFC)] 0 32, 0xFFFFFFFF 32 64}*0x55BE239B),({@32[(RBP_init+0xFFFFFFFFFFFFFFFC)] 0 32, 0x0 32 64}*0x55BE239B)))[0:32]+0x5C99A1A9) 0 32, 0xFFFFFFFF 32 64}*0x581F756D),({(((@32[(RBP_init+0xFFFFFFFFFFFFFFFC)][31:32]?(({@32
[(RBP_init+0xFFFFFFFFFFFFFFFC)] 0 32, 0xFFFFFFFF 32 64}*0x55BE239B),({@32[(RBP_init+0xFFFFFFFFFFFFFFFC)] 0 32, 0x0 32 64}*0x55BE239B)))[0:32]+0x5C99A1A9) 0 32, 0x0 32 64}*0x581F756D)))[0:32]+0xB03CEE0A)) 0 32, 0xFFFFFFFF 32 64}
*0x55BE239B),({((((- (((@32[(RBP_init+0xFFFFFFFFFFFFFFEC)][31:32]?(({@32[(RBP_init+0xFFFFFFFFFFFFFFEC)] 0 32, 0xFFFFFFFF 32 64}*0x55BE239B),({@32[(RBP_init+0xFFFFFFFFFFFFFFEC)] 0 32, 0x0 32 64}*0x55BE239B)))[0:32]+0x5C99A1A9)[31:32]?
(({(((@32[(RBP_init+0xFFFFFFFFFFFFFFEC)][31:32]?(({@32[(RBP_init+0xFFFFFFFFFFFFFFEC)] 0 32, 0xFFFFFFFF 32 64}*0x55BE239B),({@32[(RBP_init+0xFFFFFFFFFFFFFFEC)] 0 32, 0x0 32 64}*0x55BE239B)))[0:32]+0x5C99A1A9) 0 32, 0xFFFFFFFF 32 64}
*0xFFFFFFFFA7E08A93),({(((@32[(RBP_init+0xFFFFFFFFFFFFFFEC)][31:32]?(({@32[(RBP_init+0xFFFFFFFFFFFFFFEC)] 0 32, 0xFFFFFFFF 32 64}*0x55BE239B),({@32[(RBP_init+0xFFFFFFFFFFFFFFEC)] 0 32, 0x0 32 64}*0x55BE239B)))[0:32]+0x5C99A1A9) 0 32,
0x0 32 64}*0xFFFFFFFFA7E08A93)))[0:32])+0xB03CEE0A)|((((@32[(RBP_init+0xFFFFFFFFFFFFFFFC)][31:32]?(({@32[(RBP_init+0xFFFFFFFFFFFFFFFC)] 0 32, 0xFFFFFFFF 32 64}*0x55BE239B),({@32[(RBP_init+0xFFFFFFFFFFFFFFFC)] 0 32, 0x0 32 64}
*0x55BE239B)))[0:32]+0x5C99A1A9)[31:32]?(({(((@32[(RBP_init+0xFFFFFFFFFFFFFFFC)][31:32]?(({@32[(RBP_init+0xFFFFFFFFFFFFFFFC)] 0 32, 0xFFFFFFFF 32 64}*0x55BE239B),({@32[(RBP_init+0xFFFFFFFFFFFFFFFC)] 0 32, 0x0 32 64}*0x55BE239B)))
[0:32]+0x5C99A1A9) 0 32, 0xFFFFFFFF 32 64}*0x581F756D),({(((@32[(RBP_init+0xFFFFFFFFFFFFFFFC)][31:32]?(({@32[(RBP_init+0xFFFFFFFFFFFFFFFC)] 0 32, 0xFFFFFFFF 32 64}*0x55BE239B),({@32[(RBP_init+0xFFFFFFFFFFFFFFFC)] 0 32, 0x0 32 64}
*0x55BE239B)))[0:32]+0x5C99A1A9) 0 32, 0x0 32 64}*0x581F756D)))[0:32]+0xB03CEE0A)) 0 32, 0x0 32 64}*0x55BE239B)))[0:32]+0xB9334352) 0 32, 0xFFFFFFFF 32 64}*0x581F756D),({(((@32[(RBP_init+0xFFFFFFFFFFFFFFEC)][31:32]?(({@32[(RBP_init
+0xFFFFFFFFFFFFFFEC)] 0 32, 0xFFFFFFFF 32 64}*0x55BE239B),({@32[(RBP_init+0xFFFFFFFFFFFFFFEC)] 0 32, 0x0 32 64}*0x55BE239B)))[0:32]+(@32[(RBP_init+0xFFFFFFFFFFFFFFFC)][31:32]?(({@32[(RBP_init+0xFFFFFFFFFFFFFFFC)] 0 32, 0xFFFFFFFF 32
64}*0x55BE239B),({@32[(RBP_init+0xFFFFFFFFFFFFFFFC)] 0 32, 0x0 32 64}*0x55BE239B)))[0:32]+((((- (((@32[(RBP_init+0xFFFFFFFFFFFFFFEC)][31:32]?(({@32[(RBP_init+0xFFFFFFFFFFFFFFEC)] 0 32, 0xFFFFFFFF 32 64}*0x55BE239B),({@32[(RBP_init
+0xFFFFFFFFFFFFFFEC)] 0 32, 0x0 32 64}*0x55BE239B)))[0:32]+0x5C99A1A9)[31:32]?(({(((@32[(RBP_init+0xFFFFFFFFFFFFFFEC)][31:32]?(({@32[(RBP_init+0xFFFFFFFFFFFFFFEC)] 0 32, 0xFFFFFFFF 32 64}*0x55BE239B),({@32[(RBP_init
+0xFFFFFFFFFFFFFFEC)] 0 32, 0x0 32 64}*0x55BE239B)))[0:32]+0x5C99A1A9) 0 32, 0xFFFFFFFF 32 64}*0xFFFFFFFFA7E08A93),({(((@32[(RBP_init+0xFFFFFFFFFFFFFFEC)][31:32]?(({@32[(RBP_init+0xFFFFFFFFFFFFFFEC)] 0 32, 0xFFFFFFFF 32 64}
*0x55BE239B),({@32[(RBP_init+0xFFFFFFFFFFFFFFEC)] 0 32, 0x0 32 64}*0x55BE239B)))[0:32]+0x5C99A1A9) 0 32, 0x0 32 64}*0xFFFFFFFFA7E08A93)))[0:32])+0xB03CEE0A)|((((@32[(RBP_init+0xFFFFFFFFFFFFFFFC)][31:32]?(({@32[(RBP_init
+0xFFFFFFFFFFFFFFFC)] 0 32, 0xFFFFFFFF 32 64}*0x55BE239B),({@32[(RBP_init+0xFFFFFFFFFFFFFFFC)] 0 32, 0x0 32 64}*0x55BE239B)))[0:32]+0x5C99A1A9)[31:32]?(({(((@32[(RBP_init+0xFFFFFFFFFFFFFFFC)][31:32]?(({@32[(RBP_init
+0xFFFFFFFFFFFFFFFC)] 0 32, 0xFFFFFFFF 32 64}*0x55BE239B),({@32[(RBP_init+0xFFFFFFFFFFFFFFFC)] 0 32, 0x0 32 64}*0x55BE239B)))[0:32]+0x5C99A1A9) 0 32, 0xFFFFFFFF 32 64}*0x581F756D),({(((@32[(RBP_init+0xFFFFFFFFFFFFFFFC)][31:32]?(({@32
[(RBP_init+0xFFFFFFFFFFFFFFFC)] 0 32, 0xFFFFFFFF 32 64}*0x55BE239B),({@32[(RBP_init+0xFFFFFFFFFFFFFFFC)] 0 32, 0x0 32 64}*0x55BE239B)))[0:32]+0x5C99A1A9) 0 32, 0x0 32 64}*0x581F756D)))[0:32]+0xB03CEE0A))[31:32]?(({((((- (((@32[(RBP_init
+0xFFFFFFFFFFFFFFEC)][31:32]?(({@32[(RBP_init+0xFFFFFFFFFFFFFFEC)] 0 32, 0xFFFFFFFF 32 64}*0x55BE239B),({@32[(RBP_init+0xFFFFFFFFFFFFFFEC)] 0 32, 0x0 32 64}*0x55BE239B)))[0:32]+0x5C99A1A9)[31:32]?(({(((@32[(RBP_init
+0xFFFFFFFFFFFFFFEC)][31:32]?(({@32[(RBP_init+0xFFFFFFFFFFFFFFEC)] 0 32, 0xFFFFFFFF 32 64}*0x55BE239B),({@32[(RBP_init+0xFFFFFFFFFFFFFFEC)] 0 32, 0x0 32 64}*0x55BE239B)))[0:32]+0x5C99A1A9) 0 32, 0xFFFFFFFF 32 64}
*0xFFFFFFFFA7E08A93),({(((@32[(RBP_init+0xFFFFFFFFFFFFFFEC)][31:32]?(({@32[(RBP_init+0xFFFFFFFFFFFFFFEC)] 0 32, 0xFFFFFFFF 32 64}*0x55BE239B),({@32[(RBP_init+0xFFFFFFFFFFFFFFEC)] 0 32, 0x0 32 64}*0x55BE239B)))[0:32]+0x5C99A1A9) 0 32,
0x0 32 64}*0xFFFFFFFFA7E08A93)))[0:32])+0xB03CEE0A)|((((@32[(RBP_init+0xFFFFFFFFFFFFFFFC)][31:32]?(({@32[(RBP_init+0xFFFFFFFFFFFFFFFC)] 0 32, 0xFFFFFFFF 32 64}*0x55BE239B),({@32[(RBP_init+0xFFFFFFFFFFFFFFFC)] 0 32, 0x0 32 64}
*0x55BE239B)))[0:32]+0x5C99A1A9)[31:32]?(({(((@32[(RBP_init+0xFFFFFFFFFFFFFFFC)][31:32]?(({@32[(RBP_init+0xFFFFFFFFFFFFFFFC)] 0 32, 0xFFFFFFFF 32 64}*0x55BE239B),({@32[(RBP_init+0xFFFFFFFFFFFFFFFC)] 0 32, 0x0 32 64}*0x55BE239B)))
[0:32]+0x5C99A1A9) 0 32, 0xFFFFFFFF 32 64}*0x581F756D),({(((@32[(RBP_init+0xFFFFFFFFFFFFFFFC)][31:32]?(({@32[(RBP_init+0xFFFFFFFFFFFFFFFC)] 0 32, 0xFFFFFFFF 32 64}*0x55BE239B),({@32[(RBP_init+0xFFFFFFFFFFFFFFFC)] 0 32, 0x0 32 64}
*0x55BE239B)))[0:32]+0x5C99A1A9) 0 32, 0x0 32 64}*0x581F756D)))[0:32]+0xB03CEE0A)) 0 32, 0xFFFFFFFF 32 64}*0x55BE239B),({((((- (((@32[(RBP_init+0xFFFFFFFFFFFFFFEC)][31:32]?(({@32[(RBP_init+0xFFFFFFFFFFFFFFEC)] 0 32, 0xFFFFFFFF 32 64}
*0x55BE239B),({@32[(RBP_init+0xFFFFFFFFFFFFFFEC)] 0 32, 0x0 32 64}*0x55BE239B)))[0:32]+0x5C99A1A9)[31:32]?(({(((@32[(RBP_init+0xFFFFFFFFFFFFFFEC)][31:32]?(({@32[(RBP_init+0xFFFFFFFFFFFFFFEC)] 0 32, 0xFFFFFFFF 32 64}*0x55BE239B),
```

Symbolic Execution

RAX = {((((- ((((- (((@32[(RBP_init+0xFFFFFFFFFFFFFFEC)][31:32]?(({@32[(RBP_init+0xFFFFFFFFFFFFFFEC)] 0 32, 0xFFFFFFFF 32 64}*0x55BE239B),({@32[(RBP_init+0xFFFFFFFFFFFFFFEC)] 0 32, 0x0 32 64}*0x55BE239B)))[0:32]+(@32[(RBP_init+0xFFFFFFFFFFFFFFFC)][31:32]?(({@32[(RBP_init+0xFFFFFFFFFFFFFFFC)] 0 32, 0xFFFFFFFF 32 64}*0x55BE239B),({@32[(RBP_init+0xFFFFFFFFFFFFFFFC)] 0 32, 0x0 32 64}*0x55BE239B)))[0:32]+((((- (((@32[(RBP_init+0xFFFFFFFFFFFFFFEC)][31:32]?(({@32[(RBP_init+0xFFFFFFFFFFFFFFEC)] 0 32, 0xFFFFFFFF 32 64}*0x55BE239B),({@32[(RBP_init+0xFFFFFFFFFFFFFFEC)] 0 32, 0x0 32 64}*0x55BE239B)))[0:32]+0x5C99A1A9)[31:32]?((((@32[(RBP_init+0xFFFFFFFFFFFFFFEC)][31:32]?(({@32[(RBP_init+0xFFFFFFFFFFFFFFEC)] 0 32, 0xFFFFFFFF 32 64}*0x55BE239B),({@32[(RBP_init+0xFFFFFFFFFFFFFFEC)] 0 32, 0x0 32 64}*0x55BE239B)))[0:32]+0x5C99A1A9) 0 32, 0xFFFFFFFF 32 64}*0xFFFFFFFFA7E08A93),({((@32[(RBP_init+0xFFFFFFFFFFFFFFEC)][31:32]?(({@32[(RBP_init+0xFFFFFFFFFFFFFFEC)] 0 32, 0xFFFFFFFF 32 64}*0x55BE239B),({@32[(RBP_init+0xFFFFFFFFFFFFFFEC)] 0 32, 0x0 32 64}*0x55BE239B)))[0:32]+0x5C99A1A9) 0 32, 0x0 32 64}*0xFFFFFFFFA7E08A93)))[0:32])+0xB03CEE0A)] …

Useful only in combination with other attacks

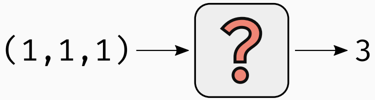
Synthesis-based Attacks

We use f as a black-box:

$$f(x, y, z) := (((x \oplus y) + ((x \wedge y) \cdot 2)) \vee z) + (((x \oplus y) + ((x \wedge y) \cdot 2)) \wedge z)$$

We use f as a black-box:

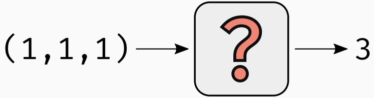
$$f(x,y,z) := (((x \oplus y) + ((x \wedge y) \cdot 2)) \vee z) + (((x \oplus y) + ((x \wedge y) \cdot 2)) \wedge z)$$



Syntia & Xyntia: Stochastic Program Synthesis

We use f as a black-box:

$$f(x, y, z) := (((x \oplus y) + ((x \wedge y) \cdot 2)) \vee z) + (((x \oplus y) + ((x \wedge y) \cdot 2)) \wedge z)$$

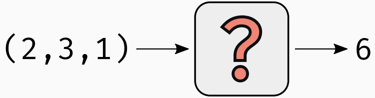


$(1, 1, 1) \to 3$

Syntia & Xyntia: Stochastic Program Synthesis

We use f as a black-box:

$$f(x,y,z) := (((x \oplus y) + ((x \wedge y) \cdot 2)) \vee z) + (((x \oplus y) + ((x \wedge y) \cdot 2)) \wedge z)$$

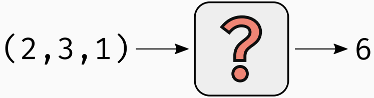


$(1,1,1) \rightarrow 3$

Syntia & Xyntia: Stochastic Program Synthesis

We use f as a black-box:

$$f(x,y,z) := (((x \oplus y) + ((x \wedge y) \cdot 2)) \vee z) + (((x \oplus y) + ((x \wedge y) \cdot 2)) \wedge z)$$



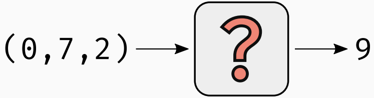
$(1,1,1) \rightarrow 3$

$(2,3,1) \rightarrow 6$

Syntia & Xyntia: Stochastic Program Synthesis

We use f as a black-box:

$$f(x, y, z) := (((x \oplus y) + ((x \land y) \cdot 2)) \lor z) + (((x \oplus y) + ((x \land y) \cdot 2)) \land z)$$

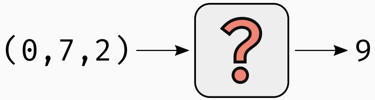


$(1, 1, 1) \rightarrow 3$

$(2, 3, 1) \rightarrow 6$

We use f as a black-box:

$$f(x,y,z) := (((x \oplus y) + ((x \wedge y) \cdot 2)) \vee z) + (((x \oplus y) + ((x \wedge y) \cdot 2)) \wedge z)$$



$(1,1,1) \rightarrow 3$

$(2,3,1) \rightarrow 6$

$(0,7,2) \rightarrow 9$

We use f as a black-box:

$$f(x, y, z) := (((x \oplus y) + ((x \wedge y) \cdot 2)) \vee z) + (((x \oplus y) + ((x \wedge y) \cdot 2)) \wedge z)$$

$$(1, 1, 1) \rightarrow 3$$
$$(2, 3, 1) \rightarrow 6$$
$$(0, 7, 2) \rightarrow 9$$

We **learn** a function h that has the same I/O behavior.

We use f as a black-box:

$$f(x, y, z) := (((x \oplus y) + ((x \wedge y) \cdot 2)) \vee z) + (((x \oplus y) + ((x \wedge y) \cdot 2)) \wedge z)$$

$$\boxed{h(x, y, z) := x + y + z}$$

$\rightarrow 3$

$(2, 3, 1) \rightarrow 6$

$(0, 7, 2) \rightarrow 9$

We **learn** a function h that has the same I/O behavior.

We use f as a black-box:

$$f(x, y, z) := (((x \oplus y) + ((x \wedge y) \cdot 2)) \vee z) + (((x \oplus y) + ((x \wedge y) \cdot 2)) \wedge z)$$

Successful on semantically simple expressions

$$(2, 3, 1) \rightarrow 6$$

$$(0, 7, 2) \rightarrow 9$$

We **learn** a function h that has the same I/O behavior.

QSynthesis: Simplification via Precomputed Lookup Tables

$$((x \oplus y) + (2 \cdot (x \wedge y))) + ((x \oplus z) + (2 \cdot (x \wedge z)))$$

QSynthesis: Simplification via Precomputed Lookup Tables

$$((x \oplus y) + (2 \cdot (x \wedge y))) + ((x \oplus z) + (2 \cdot (x \wedge z)))$$

QSynthesis: Simplification via Precomputed Lookup Tables

$$((x \oplus y) + (2 \cdot (x \wedge y))) + ((x \oplus z) + (2 \cdot (x \wedge z)))$$

$(1, 1, 1) \rightarrow 3$

QSynthesis: Simplification via Precomputed Lookup Tables

$$((x \oplus y) + (2 \cdot (x \wedge y))) + ((x \oplus z) + (2 \cdot (x \wedge z)))$$

$(1, 1, 1) \rightarrow 3$

$(2, 3, 1) \rightarrow 6$

QSynthesis: Simplification via Precomputed Lookup Tables

$$((x \oplus y) + (2 \cdot (x \wedge y))) + ((x \oplus z) + (2 \cdot (x \wedge z)))$$

$(1, 1, 1) \rightarrow 3$

$(2, 3, 1) \rightarrow 6$

$(0, 7, 2) \rightarrow 9$

QSynthesis: Simplification via Precomputed Lookup Tables

$$((x \oplus y) + (2 \cdot (x \wedge y))) + ((x \oplus z) + (2 \cdot (x \wedge z)))$$

$(1, 1, 1) \rightarrow 3$

$(2, 3, 1) \rightarrow 6$

$(0, 7, 2) \rightarrow 9$

not in database

QSynthesis: Simplification via Precomputed Lookup Tables

$$((x \oplus y) + (2 \cdot (x \wedge y))) + ((x \oplus z) + (2 \cdot (x \wedge z)))$$

QSynthesis: Simplification via Precomputed Lookup Tables

$$((x \oplus y) + (2 \cdot (x \wedge y))) + ((x \oplus z) + (2 \cdot (x \wedge z)))$$

$(1, 1, 1) \rightarrow 2$

QSynthesis: Simplification via Precomputed Lookup Tables

$$((x \oplus y) + (2 \cdot (x \wedge y))) + ((x \oplus z) + (2 \cdot (x \wedge z)))$$

$(1, 1, 1) \rightarrow 2$

$(2, 3, 1) \rightarrow 5$

QSynthesis: Simplification via Precomputed Lookup Tables

$$((x \oplus y) + (2 \cdot (x \wedge y))) + ((x \oplus z) + (2 \cdot (x \wedge z)))$$

$(1, 1, 1) \rightarrow 2$

$(2, 3, 1) \rightarrow 5$

$(0, 7, 2) \rightarrow 7$

QSynthesis: Simplification via Precomputed Lookup Tables

$$((x \oplus y) + (2 \cdot (x \wedge y))) + ((x \oplus z) + (2 \cdot (x \wedge z)))$$

$(1, 1, 1) \to 2$

$(2, 3, 1) \to 5$

$(0, 7, 2) \to 7$

$r_0 : x + y$

QSynthesis: Simplification via Precomputed Lookup Tables

$$r_0 + ((x \oplus z) + (2 \cdot (x \wedge z)))$$

$$(1, 1, 1) \rightarrow 2$$
$$(2, 3, 1) \rightarrow 5$$
$$(0, 7, 2) \rightarrow 7$$

$$r_0 : x + y$$

QSynthesis: Simplification via Precomputed Lookup Tables

$$r_0 + ((x \oplus z) + (2 \cdot (x \land z)))$$

$$r_0 : x + y$$

QSynthesis: Simplification via Precomputed Lookup Tables

$$r_0 + ((x \oplus z) + (2 \cdot (x \wedge z)))$$

$(1, 1, 1) \rightarrow 2$

$r_0 : x + y$

QSynthesis: Simplification via Precomputed Lookup Tables

$$r_0 + ((x \oplus z) + (2 \cdot (x \land z)))$$

$$(1, 1, 1) \rightarrow 2$$

$$(2, 3, 1) \rightarrow 3$$

$$r_0 : x + y$$

QSynthesis: Simplification via Precomputed Lookup Tables

$$r_0 + ((x \oplus z) + (2 \cdot (x \wedge z)))$$

$(1, 1, 1) \rightarrow 2$

$(2, 3, 1) \rightarrow 3$

$(0, 7, 2) \rightarrow 2$

$r_0 : x + y$

QSynthesis: Simplification via Precomputed Lookup Tables

$$r_0 + ((x \oplus z) + (2 \cdot (x \wedge z)))$$

$(1, 1, 1) \rightarrow 2$

$(2, 3, 1) \rightarrow 3$

$(0, 7, 2) \rightarrow 2$

$r_0 : x + y$

$r_1 : x + z$

QSynthesis: Simplification via Precomputed Lookup Tables

$$r_0 + r_1$$

$(1, 1, 1) \rightarrow 2$

$(2, 3, 1) \rightarrow 3$

$(0, 7, 2) \rightarrow 2$

$r_0 : x + y$

$r_1 : x + z$

QSynthesis: Simplification via Precomputed Lookup Tables

$$r_0 + r_1$$

$$r_0 : x + y$$

$$r_1 : x + z$$

QSynthesis: Simplification via Precomputed Lookup Tables

$$r_0 + r_1$$

$(1, 1, 1) \rightarrow 2$

$r_0 : x + y$

$r_1 : x + z$

QSynthesis: Simplification via Precomputed Lookup Tables

$$r_0 + r_1$$

$(1, 1, 1) \rightarrow 2$

$(2, 3, 1) \rightarrow 5$

$r_0 : x + y$

$r_1 : x + z$

QSynthesis: Simplification via Precomputed Lookup Tables

$$r_0 + r_1$$

$(1, 1, 1) \to 2$

$(2, 3, 1) \to 5$

$(0, 7, 2) \to 7$

$r_0 : x + y$

$r_1 : x + z$

QSynthesis: Simplification via Precomputed Lookup Tables

$$r_0 + r_1$$

$(1, 1, 1) \rightarrow 2$

$(2, 3, 1) \rightarrow 5$

$(0, 7, 2) \rightarrow 7$

$r_0 : x + y$

$r_1 : x + z$

$r_2 : r_0 + r_1$

QSynthesis: Simplification via Precomputed Lookup Tables

r_2

$r_0 : x + y$

$r_1 : x + z$

$r_2 : r_0 + r_1$

QSynthesis: Simplification via Precomputed Lookup Tables

r_2

$r_0 : x + y$

$r_1 : x + z$

$r_2 : r_0 + r_1$

QSynthesis: Simplification via Precomputed Lookup Tables

$$((x \oplus y) + (2 \cdot (x \wedge y))) + ((x \oplus z) + (2 \cdot (x \wedge z)))$$

$$r_0 : x + y$$

$$r_1 : x + z$$

$$r_2 : r_0 + r_1$$

QSynthesis: Simplification via Precomputed Lookup Tables

$$((x \oplus y) + (2 \cdot (x \wedge y))) + ((x \oplus z) + (2 \cdot (x \wedge z)))$$

$r_0 : x + y$

$r_1 : x + z$

$r_2 : r_0 + r_1$

r_2

QSynthesis: Simplification via Precomputed Lookup Tables

$$((x \oplus y) + (2 \cdot (x \wedge y))) + ((x \oplus z) + (2 \cdot (x \wedge z)))$$

$r_0 : x + y$

$r_1 : x + z$

$r_2 : r_0 + r_1$

$$r_2 \to r_0 + r_1$$

QSynthesis: Simplification via Precomputed Lookup Tables

$$((x \oplus y) + (2 \cdot (x \wedge y))) + ((x \oplus z) + (2 \cdot (x \wedge z)))$$

$r_0 : x + y$

$r_1 : x + z$

$r_2 : r_0 + r_1$

$$r_2 \rightarrow r_0 + r_1 \rightarrow (x + y) + (x + z)$$

QSynthesis: Simplification via Precomputed Lookup Tables

$$((x \oplus y) + (2 \cdot (x \wedge y))) + ((x \oplus z) + (2 \cdot (x \wedge z)))$$

Most flexible and powerful attack

$r_1 : x + z$

$r_2 : r_0 + r_1$

$$r_2 \rightarrow r_0 + r_1 \rightarrow (x + y) + (x + z)$$

Dealing with MBAs in Binaries

Challenges

- **extraction of MBAs** (symbolic execution, decompiler, ...)

Challenges

- **extraction of MBAs** (symbolic execution, decompiler, ...)
- **selection** of **meaningful** MBA boundaries:

  $x \oplus y \oplus y$ vs. $x \oplus y$

Challenges

- **extraction of MBAs** (symbolic execution, decompiler, ...)
- **selection** of **meaningful** MBA boundaries:
  $x \oplus y \oplus y$ vs. $x \oplus y$
- type casts `rdi[0:32] + rsi[0:32]`

Challenges

- **extraction of MBAs** (symbolic execution, decompiler, ...)
- **selection** of **meaningful** MBA boundaries:
  $x \oplus y \oplus y$ vs. $x \oplus y$
- type casts `rdi[0:32] + rsi[0:32]`
- memory access `@64[rax + 0x20] + rbx`

Challenges

- **extraction of MBAs** (symbolic execution, decompiler, ...)
- **selection** of **meaningful** MBA boundaries:
  $x \oplus y \oplus y$ vs. $x \oplus y$

Challenges can only be partially solved

- type casts `rdi[0:32] + rsi[0:32]`
- memory access `@64[rax + 0x20] + rbx`

msynth

msynth

Author: **Tim Blazytko** and **Moritz Schloegel**

msynth is a code deobfuscation framework to simplify Mixed Boolean-Arithmetic (MBA) expressions. Given a pre-computed simplification oracle, it walks over a complex expression represented as an abstract syntax tree (AST) and tries to simplify subtrees based on oracle lookups. Alternatively, it tries to simplify expressions via stochastic program synthesis.

msynth is built on top of Miasm and inspired by the papers

- "QSynth: A Program Synthesis based Approach for Binary Code Deobfuscation" by Robin David, Luigi Coniglio and Mariano Ceccato (NDSS, BAR 2020),
- "Syntia: Synthesizing the Semantics of Obfuscated Code" by Tim Blazytko, Moritz Contag, Cornelius Aschermann and Thorsten Holz (USENIX Security 2017) and
- "Search-Based Local Blackbox Deobfuscation: Understand, Improve and Mitigate" by Grégoire Menguy, Sébastien Bardin, Richard Bonichon and Cauim de Souza de Lima (CCS 2021).

It can be used in combination with Miasm's symbolic execution engine to simplify complex expressions in obfuscated code or as a standalone tool to play around with MBA simplification.

```
original: {(((((((RSI[0:32] ^ 0xFFFFFFFF) & RDX[0:32]) + RSI[0:32]) ^ 0xFFFFFFFF) & RDX[0:3

simplified: {(-RDX[0:32] + ((RDI[0:32] + RDX[0:32] + RSI[0:32]) << 0x1)) * 0x2 0 32, 0x0 32
```

https://github.com/mrphrazer/msynth

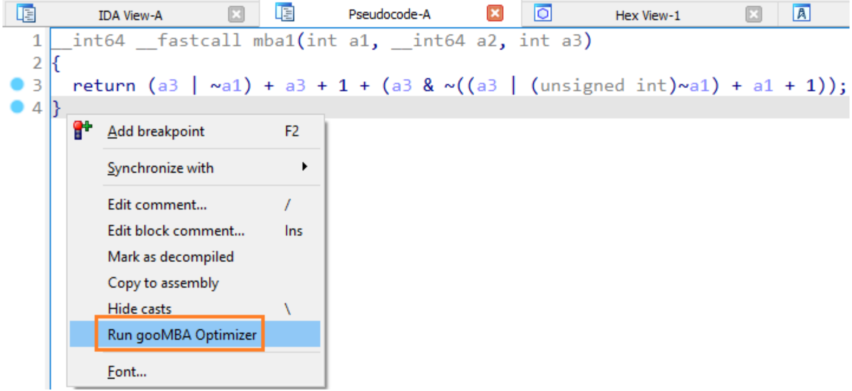
msynth: MBA Deobfuscation Framework

- inspired by QSynthesis
- synthesis-based simplification via precomputed lookup tables
- based on Miasm’s intermediate representation
- MBAs can be extracted via symbolic execution

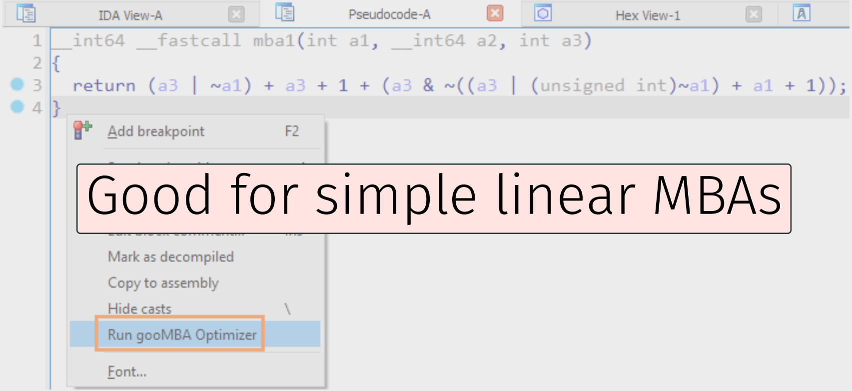
- inspired by QSynthesis
- synthesis-based simplification via precomputed lookup tables
- Powerful framework, requires scripting
- MBAs can be extracted via symbolic execution

Goomba

Hex-Rays Decompiler Plugin



https://github.com/HexRaysSA/goomba



Good for simple linear MBAs

https://github.com/HexRaysSA/goomba

Tool Shortcomings

Tool Shortcomings

Existing tools for dealing with MBAs on the binary level:

- are usable only via manual scripting (msynth)
- are only stable for simple linear MBAs (Goomba)
- only implement a subset of promising techniques

Tool Shortcomings

Existing tools for dealing with MBAs on the binary level:

- are usable only via manual scripting (msynth)
- are only stable for simple linear MBAs (Goomba)
- only implement a subset of promising techniques

There's a lot of room for improvement!

Combine Best of Both Worlds!

Idea

1. Take statements already recovered by RE tools.
2. Feed them into dedicated MBA solvers.
3. Profit!

Simplification of Decompiler Output

$$
\begin{aligned}
a &:= x \oplus y \\
b &:= x \wedge y \\
b &:= b \ll 1 \\
c &:= a + b \\
d &:= c \vee z \\
b &:= c \wedge z \\
a &:= d + b
\end{aligned}
$$

$$
\begin{aligned}
a &:= x \oplus y \\
b &:= x \wedge y \\
b &:= b \ll 1 \\
c &:= a + b \\
d &:= c \vee z \\
a &:= a + b
\end{aligned}
$$

Static Single Assignment

Simplification of Decompiler Output

$$
\begin{aligned}
a &:= x \oplus y \\
b &:= x \wedge y \\
b &:= b \ll 1 \\
c &:= a + b \\
d &:= c \vee z \\
b &:= c \wedge z \\
a &:= d + b
\end{aligned}
$$

Simplification of Decompiler Output

$$a_1 := x \oplus y$$
$$b_1 := x \wedge y$$
$$b_2 := b_1 \ll 1$$
$$c_1 := a_1 + b_2$$
$$d_1 := c_1 \vee z$$
$$b_3 := c_1 \wedge z$$
$$a_2 := d_1 + b_3$$

$$a_1 := x \oplus y$$
$$b_1 := x \wedge y$$
$$b_2 := b_1 \ll 1$$
$$c_1 := a_1 + b_2$$
$$d_1 := c_1 \vee z$$

Backward Slicing

$$a_2 := a_1 + b_3$$

Simplification of Decompiler Output

$$
\begin{aligned}
a_1 &:= x \oplus y \\
b_1 &:= x \wedge y \\
b_2 &:= b_1 \ll 1 \\
c_1 &:= a_1 + b_2 \\
d_1 &:= c_1 \vee z \\
b_3 &:= c_1 \wedge z \\
a_2 &:= d_1 + b_3
\end{aligned}
$$

$$a_2$$

Simplification of Decompiler Output

$$
\begin{aligned}
a_1 &:= x \oplus y \\
b_1 &:= x \wedge y \\
b_2 &:= b_1 \ll 1 \\
c_1 &:= a_1 + b_2 \\
d_1 &:= c_1 \vee z \\
b_3 &:= c_1 \wedge z \\
a_2 &:= d_1 + b_3
\end{aligned}
$$

$$d_1 + b_3$$

Simplification of Decompiler Output

$$
\begin{aligned}
a_1 &:= x \oplus y \\
b_1 &:= x \wedge y \\
b_2 &:= b_1 \ll 1 \\
c_1 &:= a_1 + b_2 \\
d_1 &:= c_1 \vee z \\
b_3 &:= c_1 \wedge z \\
a_2 &:= d_1 + b_3
\end{aligned}
$$

$$d_1 + b_3$$

Simplification of Decompiler Output

$$
\begin{aligned}
a_1 &:= x \oplus y \\
b_1 &:= x \wedge y \\
b_2 &:= b_1 \ll 1 \\
c_1 &:= a_1 + b_2 \\
d_1 &:= c_1 \vee z \\
b_3 &:= c_1 \wedge z \\
a_2 &:= d_1 + b_3
\end{aligned}
$$

$$(c_1 \vee z) + (c_1 \wedge z)$$

Simplification of Decompiler Output

$$
\begin{aligned}
a_1 &:= x \oplus y \\
b_1 &:= x \wedge y \\
b_2 &:= b_1 \ll 1 \\
c_1 &:= a_1 + b_2 \\
d_1 &:= c_1 \vee z \\
b_3 &:= c_1 \wedge z \\
a_2 &:= d_1 + b_3
\end{aligned}
$$

$$(c_1 \vee z) + (c_1 \wedge z)$$

Simplification of Decompiler Output

$$
\begin{aligned}
a_1 &:= x \oplus y \\
b_1 &:= x \wedge y \\
b_2 &:= b_1 \ll 1 \\
c_1 &:= a_1 + b_2 \\
d_1 &:= c_1 \vee z \\
b_3 &:= c_1 \wedge z \\
a_2 &:= d_1 + b_3
\end{aligned}
$$

$$((a_1 + b_2) \vee z) + ((a_1 + b_2) \wedge z)$$

Simplification of Decompiler Output

$$
\begin{aligned}
a_1 &:= x \oplus y \\
b_1 &:= x \wedge y \\
b_2 &:= b_1 \ll 1 \\
c_1 &:= a_1 + b_2 \\
d_1 &:= c_1 \vee z \\
b_3 &:= c_1 \wedge z \\
a_2 &:= d_1 + b_3
\end{aligned}
$$

$$((a_1 + b_2) \vee z) + ((a_1 + b_2) \wedge z)$$

Simplification of Decompiler Output

$$
\begin{aligned}
a_1 &:= x \oplus y \\
b_1 &:= x \wedge y \\
b_2 &:= b_1 \ll 1 \\
c_1 &:= a_1 + b_2 \\
d_1 &:= c_1 \vee z \\
b_3 &:= c_1 \wedge z \\
a_2 &:= d_1 + b_3
\end{aligned}
$$

$$(((x \oplus y) + (b_1 \ll 1)) \vee z) + (((x \oplus y) + (b_1 \ll 1)) \wedge z)$$

Simplification of Decompiler Output

$$
\begin{aligned}
a_1 &:= x \oplus y \\
b_1 &:= x \wedge y \\
b_2 &:= b_1 \ll 1 \\
c_1 &:= a_1 + b_2 \\
d_1 &:= c_1 \vee z \\
b_3 &:= c_1 \wedge z \\
a_2 &:= d_1 + b_3
\end{aligned}
$$

$$(((x \oplus y) + (b_1 \ll 1)) \vee z) + (((x \oplus y) + (b_1 \ll 1)) \wedge z)$$

Simplification of Decompiler Output

$$
\begin{aligned}
a_1 &:= x \oplus y \\
b_1 &:= x \wedge y \\
b_2 &:= b_1 \ll 1 \\
c_1 &:= a_1 + b_2 \\
d_1 &:= c_1 \vee z \\
b_3 &:= c_1 \wedge z \\
a_2 &:= d_1 + b_3
\end{aligned}
$$

$$(((x \oplus y) + ((x \wedge y) \ll 1)) \vee z) + (((x \oplus y) + ((x \wedge y) \ll 1)) \wedge z)$$

Simplification of Decompiler Output

$$
\begin{aligned}
a_1 &:= x \oplus y \\
b_1 &:= x \wedge y \\
b_2 &:= b_1 \ll 1 \\
c_1 &:= a_1 + b_2 \\
d_1 &:= c_1 \vee z \\
b_3 &:= c_1 \wedge z \\
a_2 &:= d_1 + b_3
\end{aligned}
$$

$$(((x \oplus y) + ((x \wedge y) \ll 1)) \vee z) + (((x \oplus y) + ((x \wedge y) \ll 1)) \wedge z)$$

Simplification of Decompiler Output

$$
\begin{aligned}
a_1 &:= x \oplus y \\
b_1 &:= x \land y \\
b_2 &:= b_1 \ll 1 \\
c_1 &:= a_1 + b_2 \\
d_1 &:= c_1 \lor z
\end{aligned}
$$

Simplification

$$d_2 := d_1 + b_3$$

$$(((x \oplus y) + ((x \land y) \ll 1)) \lor z) + (((x \oplus y) + ((x \land y) \ll 1)) \land z)$$

Simplification of Decompiler Output

$$
\begin{aligned}
a_1 &:= x \oplus y \\
b_1 &:= x \wedge y \\
b_2 &:= b_1 \ll 1 \\
c_1 &:= a_1 + b_2 \\
d_1 &:= c_1 \vee z \\
b_3 &:= c_1 \wedge z \\
a_2 &:= d_1 + b_3
\end{aligned}
$$

$$(((x \oplus y) + ((x \wedge y) \ll 1)) \vee z) + (((x \oplus y) + ((x \wedge y) \ll 1)) \wedge z)$$

Simplification of Decompiler Output

$$
\begin{aligned}
a_1 &:= x \oplus y \\
b_1 &:= x \wedge y \\
b_2 &:= b_1 \ll 1 \\
c_1 &:= a_1 + b_2 \\
d_1 &:= c_1 \vee z \\
b_3 &:= c_1 \wedge z \\
a_2 &:= d_1 + b_3
\end{aligned}
$$

$$x + y + z$$

Good news:
Some decompilers already support backward slicing!

$\Rightarrow$ We retrieve the unfolded expression

But how to simplify?

Simplification

We already have msynth..

..but it relies on Miasm's intermediate representation!

Simplification

$\Rightarrow$ IL-to-IL translation

Simplification

Traverse the AST tree of the SSA statement and translate every node to a Miasm construct

HLIL to Miasm IL Translation

$$(((x \oplus y) + ((x \wedge y) \ll 1)) \vee z) + (((x \oplus y) + ((x \wedge y) \ll 1)) \wedge z)$$

Binary Ninja HLIL	Miasm IL
HLIL_VAR	ExprId
HLIL_CONST	ExprInt
HLIL_XOR, HLIL_OR, HLIL_SHL, ...	ExprOp

HLIL to Miasm IL Translation

$$(((x \oplus y) + ((x \wedge y) \ll 1)) \vee z) + (((x \oplus y) + ((x \wedge y) \ll 1)) \wedge z)$$

Binary Ninja HLIL	Miasm IL
HLIL_VAR	ExprId
HLIL_CONST	ExprInt
HLIL_XOR, HLIL_OR, HLIL_SHL, ...	ExprOp

$$(((x \oplus y) + ((x \wedge y) \ll 1)) \vee z) + (((x \oplus y) + ((x \wedge y) \ll 1)) \wedge z)$$

Binary Ninja HLIL	Miasm IL
HLIL_VAR	ExprId
HLIL_CONST	ExprInt
HLIL_XOR, HLIL_OR, HLIL_SHL, ...	ExprOp

HLIL to Miasm IL Translation

$$(((x \oplus y) + ((x \wedge y) \ll 1)) \vee z) + (((x \oplus y) + ((x \wedge y) \ll 1)) \wedge z)$$

Binary Ninja HLIL	Miasm IL
HLIL_VAR	ExprId
HLIL_CONST	ExprInt
HLIL_XOR, HLIL_OR, HLIL_SHL, ...	ExprOp

$$(((x \oplus y) + ((x \wedge y) \ll 1)) \vee z) + (((x \oplus y) + ((x \wedge y) \ll 1)) \wedge z)$$

```
ExprOp("+", ExprOp("+", ExprOp("+", ExprVar("z", 64), …), …,),…)
```

HLIL_VAR	ExprId
HLIL_CONST	ExprInt
HLIL_XOR, HLIL_OR, HLIL_SHL, …	ExprOp

Showcase

```
mba5:
 0 @ 004011b1  int32_t rcx = arg2
 1 @ 004011bc  int32_t rbp = 0xfffffffe - arg2
 2 @ 004011be  int32_t r13 = arg2 + 1
 3 @ 004011c4  int32_t rdx_1 = rbp | r13
 4 @ 004011ca  int32_t r11_1 = arg2 - rdx_1
 5 @ 004011d0  int32_t r10_1 = arg2 & 1
 6 @ 004011d9  arg2.b = r10_1 == 0
 7 @ 004011dd  uint32_t rsi = zx.d(arg2.b)
 8 @ 004011e4  int32_t r14_1 = r13 | rsi
 9 @ 004011e7  int32_t r8 = rcx + arg1
10 @ 004011f8  int32_t r12_1 = not.d(r8)
11 @ 00401203  int32_t rdx_5 = (rdx_1 - rcx - 1) | r11_1 | 1
12 @ 0040120b  int32_t rdx_7 = rcx & 0xfffffffe
13 @ 00401211  int32_t var_34 = rcx
14 @ 00401215  int32_t r9_1 = rcx | rdx_7
15 @ 0040121b  int32_t r11_3 = (r11_1 + arg1) & rdx_5
16 @ 0040122a  int32_t rdi_2 = (0xfffffffe - r8) & (r8 + 1) & rcx
17 @ 00401266  int32_t rdx_16 = (not.d(((((rdx_7 + 2 - rsi) | (arg1 - 2)) & r13) + arg1) | (0xfffffffe - r8))
17 @ 00401266      & rcx) - ((r8 - ((r14_1 - 1 - ((not.d(rsi) & rbp) | r14_1 | 2)) | rbp)) & rdx_5)
18 @ 004012a5  int32_t rbx_8 = (((r9_1 - rcx) ^ ((rdi_2 - r11_3) & r12_1)) - 1)
18 @ 004012a5      | ((((((not.d((0xfffffffe - r8) ^ (r8 + 1)) & rcx) - r11_3) & r12_1) ^ (r9_1 - rcx)) - 1) & 1)
19 @ 004012ef  int32_t rdi_5 = ((rdi_2 -
19 @ 004012ef      ((r8 - (((((rcx * 2 + 1) & not.d(rcx) & ((not.d(rcx) * 2) | rcx)) ^ rcx) + 1) | rbp)) & rdx_5))
19 @ 004012ef      & r12_1) ^ (r9_1 + 1 - ((not.d(r10_1) | rcx) ^ rbp))
20 @ 00401300  return zx.q(arg3 - 1 - ((((rdx_16 & r12_1) ^ (rcx - 1 - r9_1)) + rbx_8 + 1) & rdi_5))
```

```
20 @ 00401300  // zeroExt_64(arg1 + arg2 + arg3)
20 @ 00401300  return zx.q(arg3 - 1 - ((((rdx_16 & r12_1) ^ (rcx - 1 - r9_1)) + rbx_8 + 1) & rdi_5))
```

Tools

Obfuscation Analysis (v1.1)

Authors: **Tim Blazytko & Nicolò Altamura**

Analyze and simplify obfuscated code

Description:

Obfuscation Analysis is a Binary Ninja plugin that takes the pain out of working with heavily protected binaries. It bundles a handful of focused helpers that let you

- simplify Mixed-Boolean Arithmetic (MBA) expressions in one click (based on msynth)
- locate and scrub functions with broken disassembly

`https://github.com/mrphrazer/obfuscation_analysis`

Obfuscation Analysis

- architecture agnostic Binary Ninja plugin
- simplifies complex MBA expressions
- uses msynth in the background for simplification

Real-World Examples

```
22 @ 180011737  r9_9.b = rcx_27 == 0
23 @ 18001173b  int32_t r10_3
24 @ 18001173b  r10_3.b = rcx_27 != 0
25 @ 180011743  rcx_27.b = r8 s> 9
26 @ 18001174a  bool r11 = r8 s< 0xa
27 @ 18001174e  char rax_30 = rcx_27.b
28 @ 180011750  char r8_1 = r10_3.b
29 @ 180011753  char rdx_14 = rcx_27.b
30 @ 180011755  rcx_27.b |= r10_3.b
31 @ 180011758  r10_3.b |= r11
32 @ 18001175b  rax_30 |= r9_9.b
33 @ 180011768  rax_30 = ((rax_30 | r10_3.b) ^ 1) | (r10_3.b ^ rax_30)
34 @ 180011770  rdx_14 = (rdx_14 & r9_9.b) | (r8_1 & r11)
35 @ 180011776  rcx_27.b ^= 1
36 @ 180011779  rcx_27.b |= r11 ^ r9_9.b
37 @ 18001177c  rcx_27.b ^= 1
38 @ 180011781  char rbx_1 = rdx_14 & rcx_27.b
39 @ 180011783  rcx_27.b ^= rdx_14
40 @ 180011785  rcx_27.b |= rbx_1
41 @ 180011789  char rdx_15 = rax_30 ^ rcx_27.b
42 @ 18001178f  rax_30 = ((rax_30 ^ 1) | rcx_27.b) ^ 1
43 @ 180011791  rdx_15 ^= 1
44 @ 1800117a1  if ((rax_30 & rdx_15 & 1) == 0 && ((rax_30 ^ rdx_15) & 1) == 0)
```

MBA-based Opaque Predicate

```
22 @ 180011737  r9_9.b = rcx_27 == 0
23 @ 18001173b  int32_t r10_3
24 @ 18001173b  r10_3.b = rcx_27 != 0
25 @ 180011743  rcx_27.b = r8 s> 9
26 @ 18001174a  bool r11 = r8 s< 0xa
27 @ 18001174e  char rax_30 = rcx_27.b
28 @ 180011750  char r8_1 = r10_3.b
29 @ 180011753  char rdx_14 = rcx_27.b
30 @ 180011755  rcx_27.b |= r10_3.b
31 @ 180011758  r10_3.b |= r11
32 @ 180
33 @ 180
34 @ 180
35 @ 180011776  rcx_27.b ^= 1
36 @ 180011779  rcx_27.b |= r11 ^ r9_9.b
37 @ 18001177c  rcx_27.b ^= 1
38 @ 180011781  char rbx_1 = rdx_14 & rcx_27.b
39 @ 180011783  rcx_27.b ^= rdx_14
40 @ 180011785  rcx_27.b |= rbx_1
41 @ 180011789  char rdx_15 = rax_30 ^ rcx_27.b
42 @ 18001178f  rax_30 = ((rax_30 ^ 1) | rcx_27.b) ^ 1
43 @ 180011791  rdx_15 ^= 1
44 @ 1800117a1  if ((rax_30 & rdx_15 & 1) == 0 && ((rax_30 ^ rdx_15) & 1) == 0)
```

```
@ 1800117a1  // 0x0
@ 1800117a1  if ((rax_30 & rdx_15 & 1) == 0 && ((rax_30 ^ rdx_15) & 1) == 0)
```

```
00882cc0    int32_t var_108 = 0
00882cca    int32_t var_104 = 0
00882cd4    int32_t var_1c = 0x978cbc69
00882d55    int32_t var_f8_3 =
00882d55        ((arg2 & 0xa8835f11) | 0x108392a4) + ((0x532880ee | not.d(arg2)) & 0xea004d11)
00882d55
00882d71    while (true)
00882d71        void var_498
00882d71        void var_488
00882d71        int64_t var_538
00882d71        void var_528
00882d71        void var_508
00882d71        void var_428
00882d71        void var_3e8
00882d71        void var_318
00882d71        void var_2e8
00882d71        void var_218
00882d71        void var_1f8
00882d71        void var_188
00882d71        void var_168
00882d71        void* var_18
00882d71        int32_t var_c
00882d71        void* rax_11
00882d71
00882d71        if (var_f8_3 + 0x57c206b u<= 0x66)
00882d9d            switch (var_f8_3)
00882f3c                case 0xfa83df95
00882f3c                    var_f8_3 = 0xfa83dfc0
00882f46                    continue
00882f52                case 0xfa83df96
00882f52                    var_18 = &var_528
00882f5d                    sub_868771(&var_508)
```

Control-Flow Flattening in a DRM System

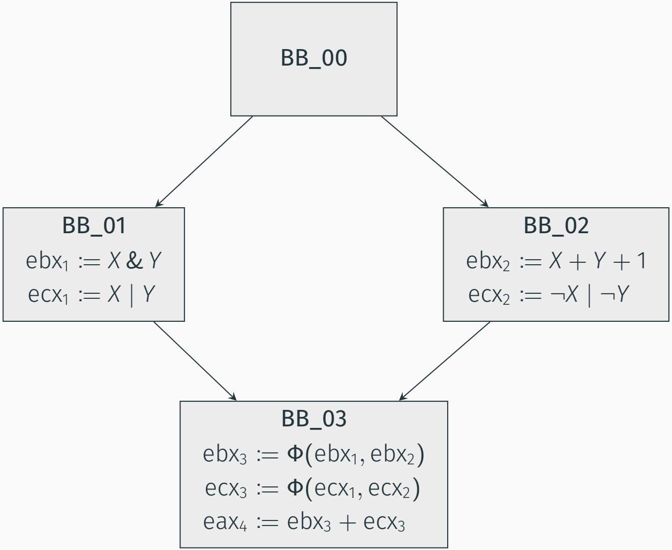
```
00882cc0    int32_t var_108 = 0
00882cca    int32_t var_104 = 0
00882cd4    int32_t var_1c = 0x978cbc69
00882d55    int32_t var_f8_3 =
00882d55        ((arg2 & 0xa8835f11) | 0x108392a4) + ((0x532880ee | not.d(arg2)) & 0xea004d11)
00882d55
00882d71    while (true)
00882d71        void var_498
00882d71        void var_488
00882d71        int64_t var_538
00882d71        void var_528
00882d71        void var_508
00882d71        void var_428
00882d71        void var_3e8

00882d71        void var_188
00882d71        void var_168
00882d71        void* var_18
00882d71        int32_t var_c
00882d71        void* rax_11
00882d71
00882d71        if (var_f8_3 + 0x57c206b u<= 0x66)
00882d9d            switch (var_f8_3)
00882f3c                case 0xfa83df95
00882f3c                    var_f8_3 = 0xfa83dfc0
00882f46                    continue
00882f52                case 0xfa83df96
00882f52                    var_18 = &var_528
00882f5d                    sub_868771(&var_508)
```

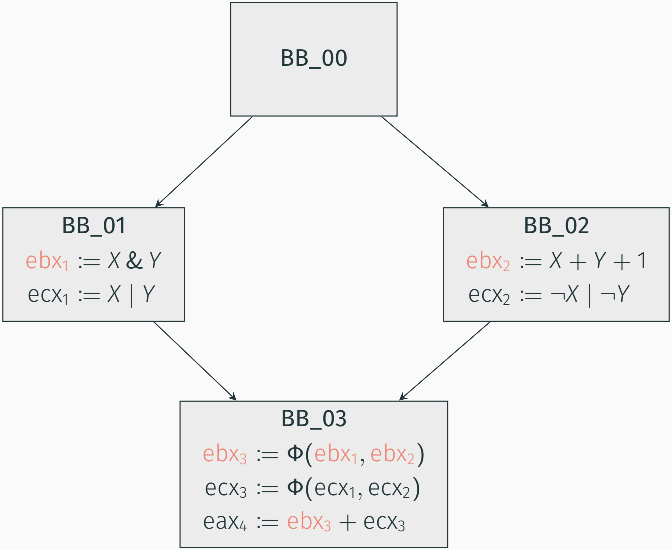
```
3 @ 00882d55  // 0xFA83DFB5
3 @ 00882d55  int32_t var_f8_3 = ((arg2 & 0xa8835f11) | 0x108392a4) + ((0x532880ee | not.d(arg2)) & 0xea004d11)
```

Can we do better?

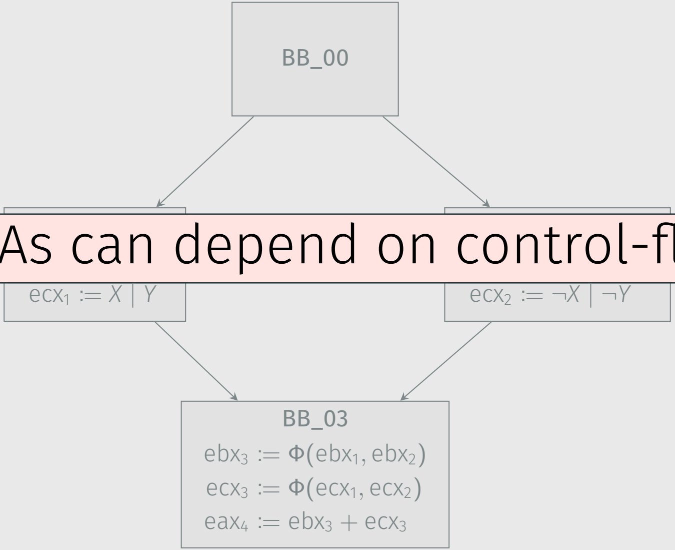
Backward slicing limited to basic block level



Backward slicing limited to basic block level



Backward slicing limited to basic block level



MBAs can depend on control-flow

Shortcomings: IL-to-IL translation

Unsupported translated nodes from Binary Ninja IL to Miasm

- floating point

Shortcomings: IL-to-IL translation

Unsupported translated nodes from Binary Ninja IL to Miasm

- floating point
- intrinsic operations

Shortcomings: IL-to-IL translation

Unsupported translated nodes from Binary Ninja IL to Miasm

- floating point
- intrinsic operations
- control flow

Future Trends

- **smooth integration** into tooling

Binary Analysis Tool Improvements

- **smooth integration** into tooling
- missing **stable** ways to **rewrite the IL**

Binary Analysis Tool Improvements

- **smooth integration** into tooling
- missing **stable** ways to **rewrite the IL**
- **interprocedural constructs** cannot be easily analyzed

Binary Analysis Tool Improvements

- **smooth integration** into tooling
- missing **stable** ways to **rewrite the IL**
- **interprocedural constructs** cannot be easily analyzed
- recent **attacks not adapted** to binary tooling

Binary Analysis Tool Improvements

- **smooth integration** into tooling
- missing **stable** ways to **rewrite the IL**
- **interprocedural constructs** cannot be easily analyzed
- recent **attacks not adapted** to binary tooling

Binary tools have to catch up!

Research Directions

- research on algebraic attacks continues
- **synergie of algebraic and synthesis-based attacks** amplifies simplification efficiency

Promising Approach: Equality Saturation

SECRET CLUB

Improving MBA Deobfuscation using Equality Saturation

fvrmatteo, mrphrazer
Aug 8, 2022

https://secret.club/2022/08/08/eqsat-oracle-synthesis.html

Conclusion

Takeaways

1. MBAs are an important **obfuscation primitive**.
2. Various attacks exist, **combined attacks** are promising.
3. Binary deobfuscation solutions exist, but are no panacea.
4. Many **attacks** still need **porting** into binary analysis tools.

Summary

- don’t be afraid of MBAs
- help us to support better binary analysis tooling

`https://github.com/mrphrazer/obfuscation_analysis`

Tim Blazytko

@mr_phrazer

https://synthesis.to

Nicolò Altamura

@nicolodev

https://nicolo.dev