# Simplification of General Mixed Boolean-Arithmetic Expressions: GAMBA

Benjamin Reichenwallner & Peter Meerwald-Stadler

July 2023

# Motivation

- Denuvo provides anti-piracy and anti-cheat solutions for video games
- Interested in effective and efficient code obfuscation techniques
- We want small and fast MBAs that cannot be easily broken

# Motivation

- Denuvo provides anti-piracy and anti-cheat solutions for video games
- Interested in effective and efficient code obfuscation techniques
- We want small and fast MBAs that cannot be easily broken

- Are *linear* MBAs worth using?
  - No, they can be simplified too easily and fast – see *SiMBA*.

# Motivation

- Denuvo provides anti-piracy and anti-cheat solutions for video games
- Interested in effective and efficient code obfuscation techniques
- We want small and fast MBAs that cannot be easily broken

- Are *linear* MBAs worth using?
  - No, they can be simplified too easily and fast – see *SiMBA*.
- **What about nonlinear MBAs?**

# Obfuscation use cases

- Code transformations increase the cost of information extraction for an attacker
- Applications:
  - Hide secret information such as constants
  - Hide algorithms such as license checks or watermarks
  - Obscure the control flow (e.g., using *opaque predicates*)
  - Make pattern matching harder via expression diversity
  - ...

# Obfuscation use cases

- Code transformations increase the cost of information extraction for an attacker
- Applications:
  - Hide secret information such as constants
  - Hide algorithms such as license checks or watermarks
  - Obscure the control flow (e.g., using *opaque predicates*)
  - Make pattern matching harder via expression diversity
  - ...
- Also applicable to malware: make analysis harder
  - Malware rarely relies on state-of-the-art self-protection, often only multi-level encryption and packing

# Example

```
...

if (input == 2271560481) {
  doSomething();
} else {
  ...
}
...
```

# Example

```
...

if (input == 2271560481) {
  doSomething();
} else {
  ...
}
...
```

- Suppose we want to hide the information for which input values `doSomething()` is called.
- For obfuscation, introduce variables that have no effect on that.

# Example

```
...

int x = ..., y = ..., z = ...;
int a = 3689348816559158708*~(z|(x&y)) + 3689348814287598227*~(z|(x&~y))
+ 3689348814287598227*~((x&z)^x^(y&z)) + 14757395259421953389*~(y&(~x|z))
+ 11068046442862794681*~(z^(~x&(y|z))) + 7378697628575196454*(z^(x|(y&~z)))
+ 18446744071437991135*~(z^(x&(y|z)));


if (input == a) {
  doSomething();
} else {
  ...
}
...
```

# Example

```
...

int x = ..., y = ..., z = ...;
int a = 36893488165591587O8*~(z|(x&y)) + 36893488142875982 27*~(z|(x&~y))
+ 36893488142875982 27*~((x&z)^x^(y&z)) + 14757395259421953389*~(y&(~x|z))
+ 11068046442862794681*~(z^(~x&(y|z))) + 7378697628575196454*(z^(x|(y&~z)))
+ 18446744071437991135*~(z^(x&(y|z)));


if (input == a) {
  doSomething();
} else {
  ...
}
...
```

- The constant is gone; the condition does actually **not** depend on $x, y, z$
- Can be made arbitrarily complex
  - More variables
  - More operations
  - ...

# Mixed Boolean-arithmetic expressions

- A mixed Boolean-arithmetic expression (MBA) mixes bitwise ($\equiv$ logical) and arithmetic operations
- Introduced by Zhou et al. in 2006.
- MBAs are a common ingredient for obfuscation
  - Hide secret information or code via introduction of exaggerated complexity
- E.g., $x + y$ can be written as

$$2\,((x\,\&\,y)\,|\,(\sim x\,\&\,\sim y)) - 2\,(\sim x\,\&\,y) + 3\,((\sim x\,\&\,y)\,|\,(x\,\&\,\sim y)) - 2\cdot\sim y$$

- Assumed to be hard to simplify due to the incompatibility of arithmetic and bitwise operations

# Possible attacks

- Computer algebra systems
  - e.g., `Mathematica`, `Maple`, `Matlab` ...
- SMT solvers
  - e.g., `Z3`, `STP`, `Boolector` ...

# Possible attacks

- Computer algebra systems
  - e.g., `Mathematica`, `Maple`, `Matlab` ...
- SMT solvers
  - e.g., `Z3`, `STP`, `Boolector` ...
- Dedicated MBA solvers using various techniques
  - pattern matching (e.g., *SSPAM*)
  - neural networks (e.g., *NeuReduce*)
  - bit-blasting (e.g., *Arybo*)
  - stochastic program synthesis (e.g., *Stoke*, *Syntia*, *Xyntia*)
  - synthesis-based expression simplification (e.g., *QSynth*, *msynth*)
  - ...

# Possible attacks

- Computer algebra systems
  - e.g., `Mathematica`, `Maple`, `Matlab` ...
- SMT solvers
  - e.g., `Z3`, `STP`, `Boolector` ...
- Dedicated MBA solvers using various techniques
  - pattern matching (e.g., *SSPAM*)
  - neural networks (e.g., *NeuReduce*)
  - bit-blasting (e.g., *Arybo*)
  - stochastic program synthesis (e.g., *Stoke*, *Syntia*, *Xyntia*)
  - synthesis-based expression simplification (e.g., *QSynth*, *msynth*)
  - ...
- Simplification vs. verification
  - Checking against some hypothesis is much easier!

# MBA types

- Each MBA is a function $e : (B^n)^t \to B^n$ for $B = \{0, 1\}$ and $n, t \in \mathbb{N}$.

# MBA types

- Each MBA is a function $e : (B^n)^t \to B^n$ for $B = \{0, 1\}$ and $n, t \in \mathbb{N}$.

  - ***Linear* MBA:**

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

where $n, t \in \mathbb{N}$, $I \subset \mathbb{N}$, $a_i \in B^n$ and $e_i$ bitwise expressions for $i \in I$

*Example:* $x + (x \& y) - 2 \cdot (x|y) + 42$

# MBA types

- Each MBA is a function $e : (B^n)^t \to B^n$ for $B = \{0, 1\}$ and $n, t \in \mathbb{N}$.

  - ***Linear*** **MBA:**

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

    where $n, t \in \mathbb{N}$, $I \subset \mathbb{N}$, $a_i \in B^n$ and $e_i$ bitwise expressions for $i \in I$

    *Example:* $x + (x\&y) - 2 \cdot (x|y) + 42$

  - ***Polynomial*** **MBA:**

    $$e(x_1, \ldots, x_t) = \sum_{i \in I} a_i \prod_{j \in J_i} e_{ij} (x_1, \ldots, x_t),$$

    where $n, t \in \mathbb{N}$, $I, J_i \subset \mathbb{N}$, $a_i \in B^n$ and $e_{ij}$ bitwise expressions for $j \in J_i$ and $i \in I$

    *Example:* $y \cdot (x \wedge y) - (x\&y)^2 - 1$

# MBA types

- Reasons for an MBA being nonpolynomial:

# MBA types

- Reasons for an MBA being nonpolynomial:
  - Powers with nonconstant MBAs in their exponents $\Rightarrow$ ***exponential* MBA**

    *Example:* $3x^y + x + 17$

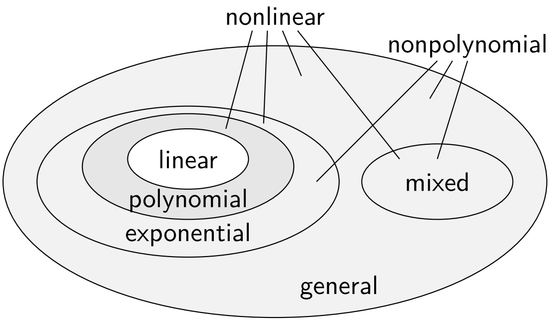
# MBA types

- Reasons for an MBA being nonpolynomial:
  - Powers with nonconstant MBAs in their exponents $\Rightarrow$ ***exponential* MBA**

    *Example:* $3x^y + x + 17$
  - Nontrivial constants or arithmetic operations in bitwise operations $\Rightarrow$ ***mixed* MBA**

    *Example:* $5 + (x|3) - (5\&y)$

# MBA types


nonlinear
nonpolynomial
linear
polynomial
exponential
mixed
general

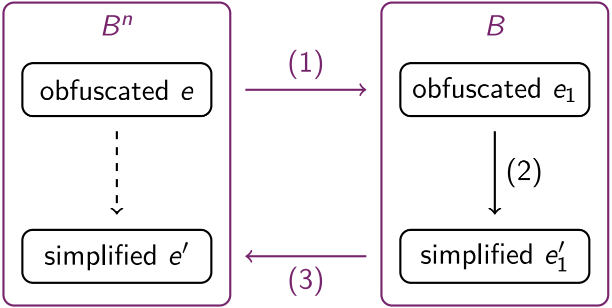
# Algebraic simplifiers for linear MBAs

- Most existing tools fail on simplifying or even verifying (linear) MBAs
- Since 2021, *algebraic* simplifiers solve *linear* MBAs within fractions of a second
  - *MBA-Blast* (2021)$^\star$
  - *MBA-Solver* (2021)$^\star$
  - *MBA-Flatten* (2022)
  - ***SiMBA*** (2022)$^\star$

$^\star$ using a transformation into the 1-bit space

# Algebraic simplification

- Zhou et al.: Linear MBAs are equivalent on $B^n$ for any $n \in \mathbb{N}$ if they are equivalent on $B = \{0, 1\}$.
  - Transform a linear MBA from $B^n$ to $B$ and simplify it there.



# *SiMBA*

- *Simple **MBA** simplifier*
- Simplifies **all** linear MBAs with **arbitrary complexity and variable count**, and **independently of their representations** in shortest time
  - In fact, simplifies all MBAs that are **reducible to linear ones**

# SiMBA

- *Simple **MBA** simplifier*
- Simplifies **all** linear MBAs with **arbitrary complexity and variable count**, and **independently of their representations** in shortest time
  - In fact, simplifies all MBAs that are **reducible to linear ones**
- Presented at *CheckMATE* 2022
- Source code available: https://github.com/DenuvoSoftwareSolutions/SiMBA

# SiMBA

- *Simple **MBA** simplifier*
- Simplifies **all** linear MBAs with **arbitrary complexity and variable count**, and **independently of their representations** in shortest time
  - In fact, simplifies all MBAs that are **reducible to linear ones**

- Presented at *CheckMATE* 2022
- Source code available: https://github.com/DenuvoSoftwareSolutions/SiMBA

- Ported to `C/C++` by Peter Garba
- Partly incorporated into IDA plugin *gooMBA* released by *Hexrays*

# SiMBA

- *Simple* ***MBA*** *simplifier*
- Simplifies **all** linear MBAs with **arbitrary complexity and variable count**, and **independently of their representations** in shortest time
  - In fact, simplifies all MBAs that are **reducible to linear ones**

- Presented at *CheckMATE* 2022
- Source code available: https://github.com/DenuvoSoftwareSolutions/SiMBA

- Ported to `C/C++` by Peter Garba
- Partly incorporated into IDA plugin *gooMBA* released by *Hexrays*
- Tim Blazytko: *"Linear MBAs are dead"* (referring to *SiMBA*)

(Tim Blazytko and Moritz Schloegel, *The Next Generation of Virtualization-based Obfuscators*, HITBSecConf2023)

# Simplification of general MBAs

- We know well how to simplify linear MBAs
- Peer tools claim to be able to simplify polynomial as well as nonpolynomial MBAs, but they can't in general
  - Too many assumptions on input expressions
  - They require knowledge about the MBAs to simplify

# Simplification of general MBAs

- We know well how to simplify linear MBAs
- Peer tools claim to be able to simplify polynomial as well as nonpolynomial MBAs, but they can't in general
  - Too many assumptions on input expressions
  - They require knowledge about the MBAs to simplify
- **Idea:** Extend *SiMBA* to simplify nonlinear MBAs with some additional effort
  - Iteratively isolate as many linear subexpressions as possible

# Simplification of general MBAs

- We know well how to simplify linear MBAs
- Peer tools claim to be able to simplify polynomial as well as nonpolynomial MBAs, but they can't in general
  - Too many assumptions on input expressions
  - They require knowledge about the MBAs to simplify
- **Idea:** Extend *SiMBA* to simplify nonlinear MBAs with some additional effort
  - Iteratively isolate as many linear subexpressions as possible
- **Disclaimer:** Unlike with linear MBAs, we can never claim to be able to simplify every input expression optimally.
  - But the result should be correct, though.

# GAMBA (*General Advanced MBA Simplifier*)

Key ingredients:

- Apply a variety of **transformations** in order to standardize and isolate linear subexpressions
- Apply ***SiMBA*** to linear subexpressions

# GAMBA (*General Advanced MBA Simplifier*)

Key ingredients:

- Apply a variety of **transformations** in order to standardize and isolate linear subexpressions
- Apply ***SiMBA*** to linear subexpressions
- **Refactor** expressions in order not to miss opportunities

# GAMBA (*General Advanced MBA Simplifier*)

Key ingredients:

- Apply a variety of **transformations** in order to standardize and isolate linear subexpressions
- Apply ***SiMBA*** to linear subexpressions
- **Refactor** expressions in order not to miss opportunities
- **Substitution logic** in order to resolve mixed MBAs

# Examples

$$
\begin{aligned}
e_1 &= ((-c-1)\&a)((-c-1)|a) + ((-c-1)\&{\sim}a)({\sim}(-c-1)\&a) \\
\tilde{e}_1 &= ({\sim}c\&a)({\sim}c|a) + ({\sim}c\&{\sim}a)(c\&a) \\
&= -a-(a\&c)
\end{aligned}
$$

# Examples

$$
\begin{aligned}
e_1 &= ((-c-1)\&a)((-c-1)|a) + ((-c-1)\&{\sim}a)({\sim}(-c-1)\&a) \\
\tilde{e}_1 &= ({\sim}c\&a)({\sim}c|a) + ({\sim}c\&{\sim}a)(c\&a) \\
&= -a - (a\&c)
\end{aligned}
$$

$$
\begin{aligned}
e_2 &= (({\sim}d+1) \wedge e) - (({\sim}({\sim}d+1)\&e) + ({\sim}({\sim}d+1)\&e)) \\
\tilde{e}_2 &= ((-d) \wedge e) - (({\sim}(-d)\&e) + ({\sim}(-d)\&e)) \\
&= (f \wedge e) - (({\sim}f\&e) + ({\sim}f\&e)) && [f := -d] \\
&= f - e \\
&= -d - e
\end{aligned}
$$

# Overview

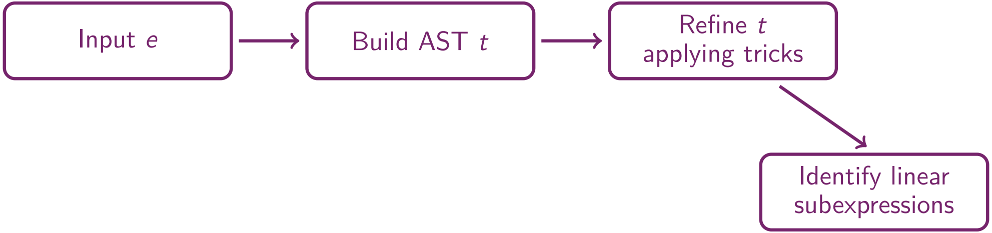
Input $e$

# Overview

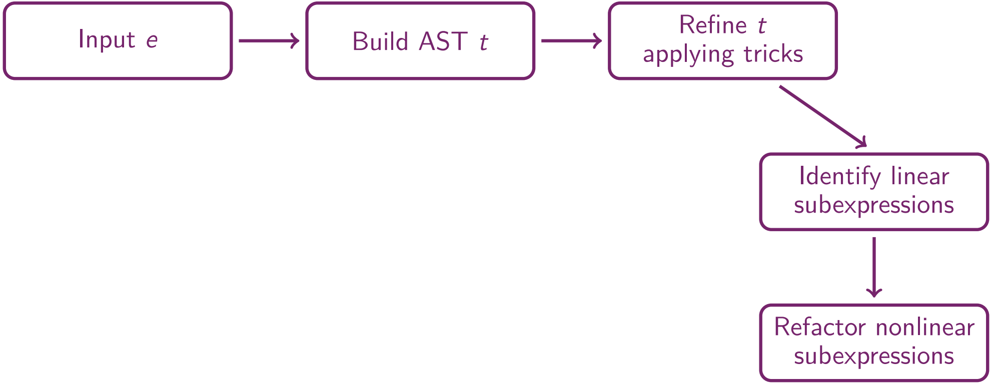
Input $e$ $\longrightarrow$ Build AST $t$

# Overview

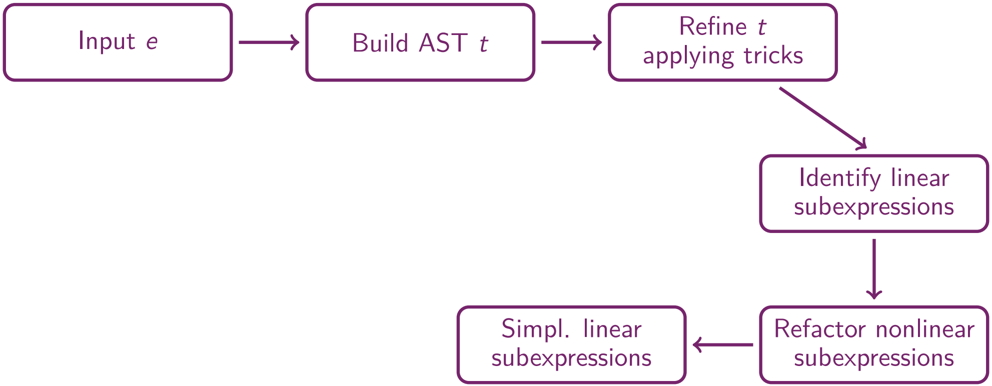
Input $e$ → Build AST $t$ → Refine $t$ applying tricks


Input $e$
Build AST $t$
Refine $t$ applying tricks
Identify linear subexpressions

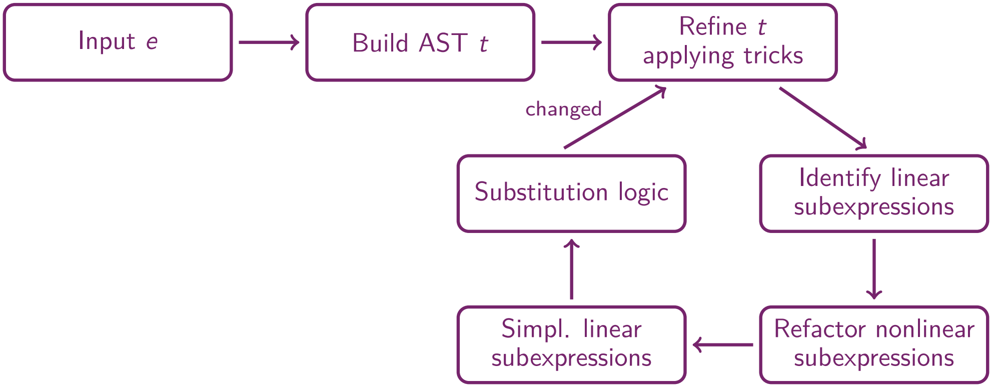
# Overview


Input $e$
Build AST $t$
Refine $t$ applying tricks
Identify linear subexpressions
Refactor nonlinear subexpressions

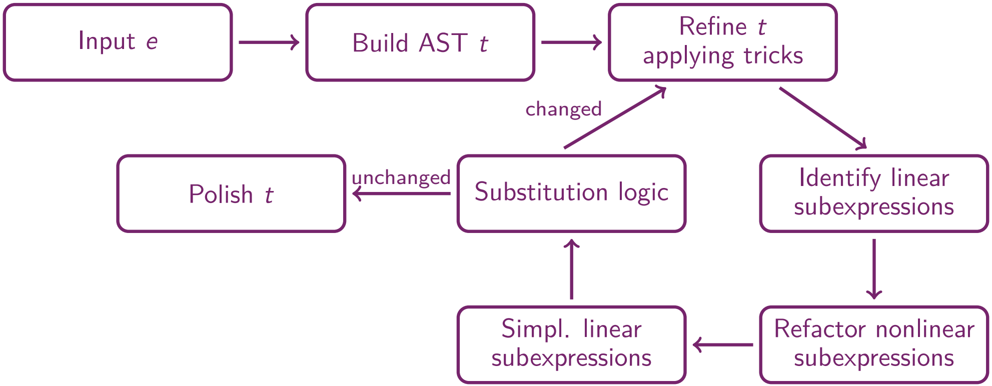
# Overview


Input $e$
Build AST $t$
Refine $t$ applying tricks
Identify linear subexpressions
Refactor nonlinear subexpressions
Simpl. linear subexpressions

# Overview

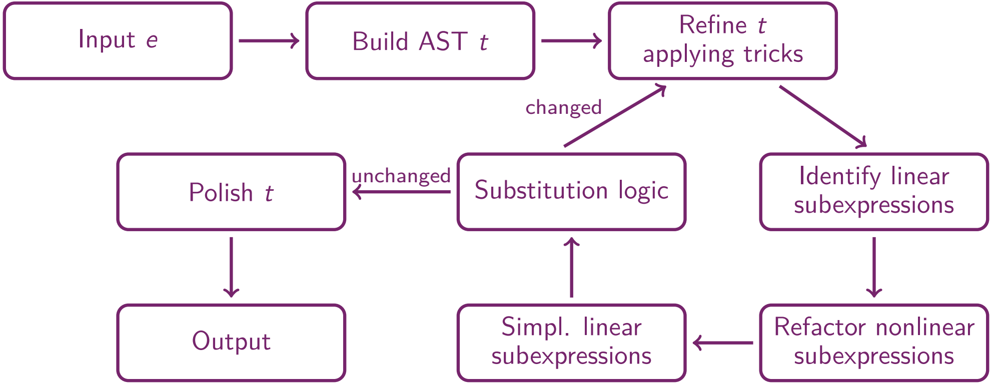

Input $e$
Build AST $t$
Refine $t$ applying tricks
changed
Substitution logic
Identify linear subexpressions
Simpl. linear subexpressions
Refactor nonlinear subexpressions

# Overview



Input $e$
Build AST $t$
Refine $t$ applying tricks
changed
Identify linear subexpressions
Refactor nonlinear subexpressions
Simpl. linear subexpressions
Substitution logic
unchanged
Polish $t$

# Overview



Input $e$
Build AST $t$
Refine $t$ applying tricks
changed
Polish $t$
unchanged
Substitution logic
Identify linear subexpressions
Output
Simpl. linear subexpressions
Refactor nonlinear subexpressions

# Experiments

- *Syntia dataset*: 500 linear MBAs with 1 to 3 variables
  - 182 linear, 51 polynomial, 267 nonpolynomial
  - $\varnothing$ MBA alternation: 1.6 (linear), 3.6 (poly), 6.9 (nonpoly)
  - $\varnothing$ node count: 9.4 (linear), 17.0 (poly), 27.6 (nonpoly)

| Tool | $\equiv$ | $\approx$ | $\times$ | Timeout | % |
|---|---|---|---|---|---|
| SSPAM | N/A | 332 | 168 | 0 | 66.4 |
| Syntia | N/A | 369 | 131 | 0 | 73.8 |
| QSynth | N/A | 500 | 0 | 0 | 100.0 |
| MBA-Blast | N/A | 416 | 0 | 84 | 83.2 |
| MBA-Solver | N/A | 454 | 0 | 46 | 90.8 |
| *MBA-Flatten* | 302 | 198 | 0 | 0 | 100.0 |
| *SiMBA* | 317 | 0 | 183 | 0 | 63.4 |
| *GAMBA* | 500 | 0 | 0 | 0 | 100.0 |

# Experiments

- *MBA-Solver dataset*: 1 000 linear, 1 000 polynomial and 1 000 nonpolynomial MBAs with 1 to 4 variables
  - ∅ MBA alternation: 9.1 (linear), 9.5 (poly), 57.4 (nonpoly)
  - ∅ node count: 71.6 (linear), 58.7 (poly), 306.1 (nonpoly)

| Tool | $\equiv$ | $\approx$ | $\times$ | Timeout | % |
|---|---|---|---|---|---|
| SSPAM | N/A | 705 | 320 | 1 975 | 34.2 |
| Syntia | N/A | 437 | 2 563 | 0 | 14.6 |
| MBA-Blast | N/A | 1 763 | 0 | 1 237 | 58.8 |
| MBA-Solver | N/A | 2 899 | 0 | 101 | 96.6 |
| *MBA-Flatten* | 2 500 | 443 | 0 | 57 | 98.1 |
| *SiMBA* | 1 757 | 87 | 1 156 | 0 | 61.5 |
| *GAMBA* | 2 998 | 2 | 0 | 0 | 100.0 |

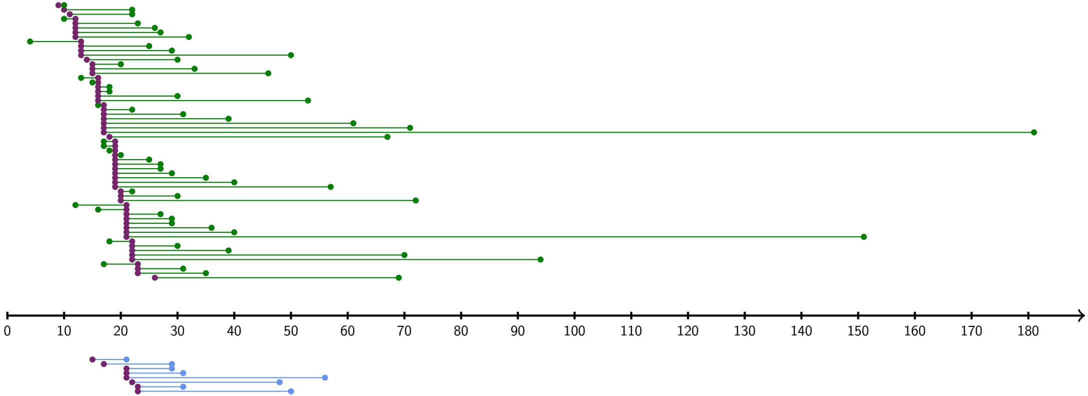
# Experiments

- *QSynth EA*: 500 polynomial and 1 000 nonpolynomial MBAs with 1 to 3 variables
  - ∅ MBA alternation: 77.6
  - ∅ node count: 281.7

| Tool | $\equiv$ | $\approx$ | $\times$ | Timeout | % |
|---|---|---|---|---|---|
| QSynth | N/A | 354 | 146 | 0 | 69.0 |
| *SiMBA* | 45 | 0 | 455 | 0 | 9.0 |
| *GAMBA* | 431 | 61 | 8 | 0 | 98.4 |

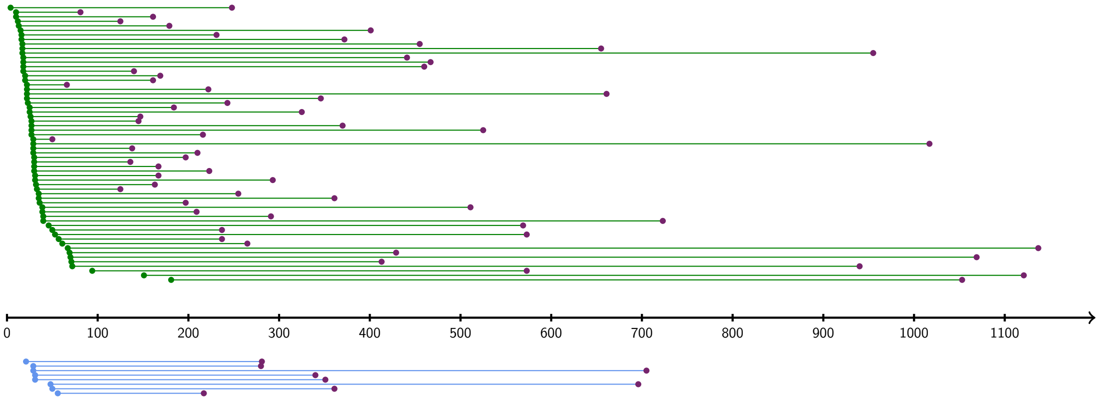
# Complexity comparison

*QSynth EA*: Numbers of nodes in groundtruths and results ($\approx$ and $\times$)



# Complexity comparison

*QSynth EA*: Numbers of nodes in input expressions and results (≈ and ×)



# Conclusion

- Linear MBAs can be broken easily and fast.
- Nonlinear MBAs can be simplified too, but in a less straightforward and non-unique way.
- Hence, choose and use MBAs wisely!
- Better designed datasets of nonlinear MBAs needed

- *GAMBA*'s source code will become available here: https://github.com/DenuvoSoftwareSolutions/GAMBA

# Conclusion

- Linear MBAs can be broken easily and fast.
- Nonlinear MBAs can be simplified too, but in a less straightforward and non-unique way.
- Hence, choose and use MBAs wisely!
- Better designed datasets of nonlinear MBAs needed

- *GAMBA*'s source code will become available here: https://github.com/DenuvoSoftwareSolutions/GAMBA

THANK YOU!

peter.meerwald@denuvo.com
benjamin.reichenwallner@denuvo.com

# References

DAVID, Robin ; CONIGLIO, Luigi ; CECCATO, Mariano:
QSynth. A program synthesis based approach for binary code deobfuscation.
In: *Workshop on Binary Analysis Research (BAR), Network and Distributed Systems Security (NDSS) Symposium 2020*. San Diego, CA, USA, February 2020

LIU, Binbin ; SHEN, Junfu ; MING, Jiang ; ZHENG, Qilong ; LI, Jing ; XU, Dongpeng:
MBA-Blast. Unveiling and simplifying mixed Boolean-arithmetic obfuscation.
In: *Proceedings of the 30th USENIX Security Symposium*, USENIX Association, August 2021 (USENIX 2021), 1701–1718

LIU, Binbin ; ZHENG, Qilong ; LI, Jing ; XU, Dongpeng:
An In-Place Simplification on Mixed Boolean-Arithmetic Expressions.
In: *Security and Communication Networks*, Hindawi, September 2022 (2022), 1–14

REICHENWALLNER, Benjamin ; MEERWALD-STADLER, Peter:
Efficient deobfuscation of linear mixed Boolean-arithmetic expressions.
In: *Proc. 2022 ACM Workshop on Research on offensive and defensive techniques in the context of Man-At-The-End attacks*. Los Angeles, November 2022 (CheckMATE'22), 19–28

UNH SOFTSEC GROUP:
*MBA-Solver Code and Dataset*.
`https://github.com/softsec-unh/MBA-Solver`

XU, Dongpeng ; LIU, Binbin ; FENG, Weijie ; MING, Jiang ; ZHENG, Qilong ; YU, Qiaoyan:
Boosting SMT solver performance on mixed-bitwise-arithmetic expressions.
In: *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*. Virtual, Canada : Association for Computing Machinery, June 2021 (PLDI 2021), 651–664

# References II


ZHOU, Yongxin ; MAIN, Alec:
Diversity via code transformations. A solution for NGNA renewable security.
In: *The NCTA Technical Papers 2006*.
Atlanta : The National Cable and Telecommunications Association Show, 2006, 173–182

ZHOU, Yongxin ; MAIN, Alec ; GU, Yuan X. ; JOHNSON, Harold:
Information hiding in software with mixed Boolean-arithmetic transforms.
In: *Proceedings of the 8th International Conference on Information Security Applications*.
Berlin, Heidelberg : Springer-Verlag, August 2007 (WISA 2007), 61–75