Efficient Deobfuscation of Linear Mixed Boolean-Arithmetic Expressions

Benjamin Reichenwallner & Peter Meerwald-Stadler

November 2022

Authors

Benjamin Reichenwallner
Denuvo GmbH, Austria
benjamin.reichenwallner@denuvo.com

Peter Meerwald-Stadler
Denuvo GmbH, Austria
peter.meerwald@denuvo.com

- Denuvo provides anti-piracy 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?

Motivation

- Mixed Boolean-arithmetic expressions (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

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

- Hard to simplify due to the incompatibility of arithmetic and bitwise operations

Motivation

- Mixed Boolean-arithmetic expressions (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

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

- Hard to simplify due to the incompatibility of arithmetic and bitwise operations
- There is a variety of techniques for simplification
  - pattern matching, neural networks, bit-blasting, stochastic program synthesis ...
- Most existing tools fail on simplifying or even verifying those expressions
- In 2021, *MBA-Blast* and *MBA-Solver* simplify *linear* MBAs within fractions of a second
  - ... but there are some caveats to be resolved in order to make Liu et al.'s approach powerful in practice

Mixed Boolean-arithmetic expressions

- An MBA mixes bitwise ($\equiv$ logical) and arithmetic operations
- Introduced by Zhou et al. in 2006.
- Hard to simplify:
  - Arithmetic or logical simplification rules are not compatible
  - Established tools either concentrate on arithmetic or logical expressions
  - Most MBA simplifiers are not (yet) powerful enough

Mixed Boolean-arithmetic expressions

- An MBA mixes bitwise ($\equiv$ logical) and arithmetic operations
- Introduced by Zhou et al. in 2006.
- Hard to simplify:
  - Arithmetic or logical simplification rules are not compatible
  - Established tools either concentrate on arithmetic or logical expressions
  - Most MBA simplifiers are not (yet) powerful enough
- We concentrate on *linear* MBAs: functions $e : (B^n)^t \rightarrow B^n$ of the form

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

where $B = \{0, 1\}$, $n, t \in \mathbb{N}$, $I \subset \mathbb{N}$ is an index set, $a_i \in B^n$ are constants and e_i are bitwise expressions of $x_1, \ldots, x_t$ for $i \in I$.

Mixed Boolean-arithmetic expressions

- Examples:
  - $x + (x \& y) - 2 \cdot (x|y) + 42$ is a *linear* MBA
  - $y \cdot (x \wedge y) - (x \& y)^2 - 1$ is a *polynomial* MBA
  - $3 \cdot x^y + x + 17$ is *not polynomial*
  - $5 + (x|3) - (5 \& y)$ is *not polynomial*
- Methods for generation:
  - (Iterative) application of known rewriting rules (see, e.g., *LOKI*)
  - Solution of random linear equation systems (Zhou et al. 2007)
  - ...
- Zhou et al.: Linear MBAs are equivalent on B^n for any $n \in \mathbb{N}$ if they are equivalent on $B = \{0, 1\}$.
  - Bitwise expressions can be considered logical expressions.
  - MBAs with t variables identified by only 2^t (rather than 2^{nt}) values!

The myth of Zhou et al.'s reversed direction

- In more detail, Zhou et al. prove that an MBA $e \equiv 0$ if and only if the vector $Y_a = (a_1, \ldots, a_s)^T$ of its coefficients solves a linear equation system $AY = o$, where A is a *truth value matrix* of e's bitwise expressions.

The myth of Zhou et al.'s reversed direction

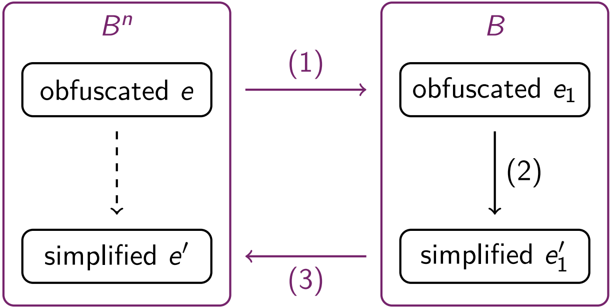
- In more detail, Zhou et al. prove that an MBA $e \equiv 0$ if and only if the vector $Y_a = (a_1, \ldots, a_s)^T$ of its coefficients solves a linear equation system $AY = o$, where A is a *truth value matrix* of e's bitwise expressions.
- But unfortunately, he formulates the theorem wrongly.

$A = (v_{i,j})_{2^t \times s}$, be the $\{0,1\}$-matrix of truth tables over $Z/(2^n)$. Then $e = 0$ if and only if the linear system $AY = 0$ has a solution over ring $Z/(2^n)$, where $Y_{s\times 1} = (y_0, \cdots, y_{s-1})^\mathsf{T}$ is a vector of s variables over $Z/(2^n)$.

- This might have delayed powerful algebraic simplification tools for years!
- Liu et al. claim to be the first to prove the reverse direction in their MBA-Blast paper in 2021.

MBA-Blast & MBA-Solver

- Very similar simplifiers written in `Python`.
- Both transform a linear MBA from B^n to B and simplify it there.



MBA-Blast & MBA-Solver

- Each bitwise expression (and hence each MBA) with t variables can be written as a linear combination of 2^t base expressions.
  - E.g., for $t = 2$, $\{1, x, y, x\&y\}$
  - ... or $\{\sim(x \mid y), \sim(x \mid \sim y), x \,\&\, \sim y, x \,\&\, y\}$
- Simple potential solutions can be found via a lookup table.
  - E.g., $x + y - (x\&y) \rightarrow x|y$

MBA-Blast & MBA-Solver

- Each bitwise expression (and hence each MBA) with t variables can be written as a linear combination of 2^t base expressions.
  - E.g., for $t = 2$, $\{1, x, y, x\&y\}$
  - ... or $\{\sim(x \mid y), \sim(x \mid \sim y), x \,\&\, \sim y, x \,\&\, y\}$
- Simple potential solutions can be found via a lookup table.
  - E.g., $x + y - (x\&y) \rightarrow x|y$
- Main differences:
  - MBA-Solver transforms whole MBAs while MBA-Blast transforms bitwise expressions and combines the results.
  - MBA-Solver uses a more complex basis in order to solve equation systems without usage of `NumPy`.
  - MBA-Solver is significantly faster.
  - MBA-Solver misses simple solutions.

SiMBA (*Simple MBA Simplifier*)

- We adapt the approach and resolve the existing tools' main sufferings.
- We follow the following principles:
  - We use generic code that works for *arbitrary* variable count.
  - We want to be independent of lookup tables (4 variables: $2^{2^4} = 65\,536$ entries).
  - We do not use any library such as `SymPy` or `NumPy` for simplification or equation system solving.
  - We find simplest solutions whenever the simplified expression has few enough variables.
  - We validate input expressions.
  - Our algorithm is invariant to an expression's concrete representation.

Success factors

- We evaluate input expressions **directly in** B^n for variables in B.
  - We prove and use a generalization of Zhou et al.'s theorem.
  - Hence there is no need for any simplification library.

Success factors

- We evaluate input expressions **directly in** B^n for variables in B.
  - We prove and use a generalization of Zhou et al.'s theorem.
  - Hence there is no need for any simplification library.
- We use a **generic** basis for every $t \in \mathbb{N}$:

$$\{1\} \cup \bigcup_{i=1}^{t} \{x_i\} \bigcup_{\substack{i_1, i_2 = 1 \\ i_1 < i_2}}^{t} \{x_{i_1} \,\&\, x_{i_2}\} \cup \bigcup_{\substack{i_1, i_2, i_3 = 1 \\ i_1 < i_2 < i_3}}^{t} \{x_{i_1} \,\&\, x_{i_2} \,\&\, x_{i_3}\} \cup \cdots \cup \{x_1 \,\&\, \cdots \,\&\, x_t\}$$

  - This gives us a nice truth table s.t. in each iteration of Gaussian elimination there is a row with only one 1.
  - We can generically determine the row in which a specific conjunction is eliminated.

Success factors

- We evaluate input expressions **directly in** B^n for variables in B.
  - We prove and use a generalization of Zhou et al.'s theorem.
  - Hence there is no need for any simplification library.
- We use a **generic** basis for every $t \in \mathbb{N}$:

$$\{1\} \cup \bigcup_{i=1}^{t} \{x_i\} \bigcup_{\substack{i_1, i_2 = 1 \\ i_1 < i_2}}^{t} \{x_{i_1} \,\&\, x_{i_2}\} \cup \bigcup_{\substack{i_1, i_2, i_3 = 1 \\ i_1 < i_2 < i_3}}^{t} \{x_{i_1} \,\&\, x_{i_2} \,\&\, x_{i_3}\} \cup \cdots \cup \{x_1 \,\&\, \cdots \,\&\, x_t\}$$

  - This gives us a nice truth table s.t. in each iteration of Gaussian elimination there is a row with only one 1.
  - We can generically determine the row in which a specific conjunction is eliminated.
- We find the **simplest** solution (in place of the resulting linear combination of conjunctions) for every input that does not depend on more than 3 variables.

Verification and comparison

- Experiments:
  - Verification via simplification of MBAs from Matteo Favaro's *NeuReduce* Github repository
  - Comparison via simplification of MBAs from *MBA-Solver*'s Github repository
  - Comparison via simplification of self-generated MBAs
  - Simplification of MBAs encoded via affine functions

- See the MBA-Blast and MBA-Solver papers for a comparison with *Arybo*, *SSPAM* and *Syntia*.

Verification

- Verification of SiMBA on the dataset of linear MBAs provided by Matteo Favaro's Github repository:

	Total	Solved	Runtime
2 variables	4 000	4 000	0.00024 s
3 variables	4 560	4 560	0.00069 s
4 variables	441	441	0.00084 s
5 variables	999	999	0.00147 s

Comparison

- Comparison on the dataset of linear MBAs provided by MBA-Solver's Github repository:

	2 variables (551 expr.)	**3 variables** (350 expr.)	**4 variables** (107 expr.)
MBA-Blast	0.02501 s	0.06726 s	—
MBA-Solver	0.00047 s	0.00121 s	0.14362 s
MBA-Flatten*	0.01872 s	0.08455 s	0.85104 s
SiMBA	0.00024 s	0.00116 s	0.00257 s

* published in September 2022 by Liu et al.

Comparison

- Comparison on 1 000 self-generated expressions for 5 target functions:
  - $e_1(x, y) = x + y$
  - $e_2 = 49\,374$
  - $e_3(x) = 3\,735\,936\,685\,x + 49\,374$
  - $e_4(x, y) = 3\,735\,936\,685\,(x \wedge y) + 49\,374$
  - $e_5(x) = 3\,735\,936\,685 \cdot {\sim}x$

MBA-Solver

	2 variables	3 variables	4 variables
e_1	0.00074 s	0.00091 s	0.12761 s
e_2	0.00052 s	0.00092 s	0.12352 s
e_3	0.00026 s	0.00150 s	0.12802 s
e_4	0.00041 s	0.00160 s	0.12683 s
e_5	0.00072 s	0.00134 s	0.12871 s

SiMBA

	2 variables	3 variables	4 variables
e_1	0.00019 s	0.00117 s	0.00550 s
e_2	0.00010 s	0.00045 s	0.00528 s
e_3	0.00020 s	0.00109 s	0.00544 s
e_4	0.00024 s	0.00100 s	0.00564 s
e_5	0.00022 s	0.00108 s	0.00543 s

Encoded expressions

- Runtime of SiMBA on 1 000 MBAs with output encoding generated for five expressions:
  - $e_i' = ae_i + b$ for random $a, b \in B^{64}$
  - Not supported by MBA-Solver and MBA-Blast

	2 variables	3 variables	4 variables
e_1'	0.00020 s	0.00118 s	0.00484 s
e_2'	0.00010 s	0.00029 s	0.00481 s
e_3'	0.00021 s	0.00125 s	0.00460 s
e_4'	0.00020 s	0.00098 s	0.00516 s
e_5'	0.00015 s	0.00110 s	0.00547 s

Outlook: Nonlinear MBAs

- Possible extension to polynomial and nonpolynomial MBAs via identifying all linear subexpressions of a nonlinear MBA and additional tricks
- Comparison on the datasets of linear, polynomial and nonpolynomial MBAs provided by MBA-Solver's Github repository:

Category	Total	MBA-Solver*		MBA-Flatten*		SiMBA	
		Solved	Runtime	Solved	Runtime	Solved	Runtime
Linear	1 008	1 008	0.01546 *s*	1 008	0.12374 *s*	1 008	0.00250 *s*
Polynom.	1 008	1 008	0.02271 *s*	737	0.18124 *s*	1 008	0.00326 *s*
Nonpolyn.	899	109	0.07965 *s*	88	0.22775 *s*	899	0.00957 *s*

* MBA-Solver and MBA-Flatten, as available to us, require additional knowledge about the input.

Conclusion

- Zhou et al.'s transformation $B^n \leftrightarrow B$ is a powerful basis for deobfuscation
- Linear MBAs can be broken easily and fast
  - Hypothesis: This is also true for all nonlinear MBAs that can be reduced to linear ones.
  - However, increasing the number of variables makes it exponentially harder.

- *SiMBA*'s source code is available here: github.com/DenuvoSoftwareSolutions/SiMBA

Conclusion

- Zhou et al.'s transformation $B^n \leftrightarrow B$ is a powerful basis for deobfuscation
- Linear MBAs can be broken easily and fast
  - Hypothesis: This is also true for all nonlinear MBAs that can be reduced to linear ones.
  - However, increasing the number of variables makes it exponentially harder.
- *SiMBA*'s source code is available here: github.com/DenuvoSoftwareSolutions/SiMBA

THANK YOU!

References

FAVARO, Matteo:
NeuReduce.
https://github.com/fvrmatteo/NeuReduce

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

SCHLOEGEL, Moritz ; BLAZYTKO, Tim ; CONTAG, Moritz ; ASCHERMANN, Cornelius ; BASLER, Julius ; HOLZ, Thorsten ; ABBASI, Ali:
LOKI. Hardening code obfuscation against automated attacks.
In: *31st USENIX Security Symposium*.
Boston, MA : USENIX Association, August 2022 (USENIX 2022), 3055–3073

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

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