7K-05

SMT ソルバを用いた演算表現の難読化方法の検討

熊井 優樹 † 神崎 雄一郎 ‡

† 熊本高等専門学校 電子情報システム工学専攻 ‡ 熊本高等専門学校 人間情報システム工学科

1 はじめに

プログラムコードの難読化は，ソフトウェアに対する不正な解析を妨げる方法として知られている．難読化の方法として，制御フローの平滑化やプログラムの仮想化など多数提案されており [1]，C 言語のソースコードレベルで難読化を行う Tigress [2] など，難読化を行うためのツールも公開されている．

本研究では，難読化方法の新たなアプローチとして，SMT ソルバ (述語論理式の充足可能性を判定するツール) を用いてコードの演算の表現を複雑にする方法について検討する．提案方法は，アセンブリコードのレベルにおいて，演算命令 (例えば `ADD` 命令) を，命令の意味を保ったまま複雑な表現を持つコード (複数の命令の組み合わせ) に変換することで，演算の表現を不明瞭にする．このようなコードは，変換対象となる演算命令の処理内容や，コードを構成する候補となる命令の振舞いに関する条件を SMT ソルバに与えることで生成する．Yurichev による文献 [3] において，SMT ソルバを用いて擬似的なアセンブリコードを生成するアイデアが述べられており，本研究では，そのアイデアをコードの難読化に応用することを試みる．ケーススタディでは，提案方法に基づいて難読化されたプログラムについて，コンパイラの最適化による逆変換 (簡単化) やシンボリック実行による解析に対する耐性について評価を行い，提案方法の有効性を考察する．

2 基本アイデア

まず，提案方法の基本となる，Yurichev によるコードの生成方法のアイデア [3] について述べる．この方法では，演算の入出力に関する条件を記述し，条件を満たす命令列を SMT ソルバによって求めることで，その演算を行うコードを生成する．図 1 は，演算の一連の操作を表すのに用いるチェーン (chain) の概略である．チェーンを構成する各ブロック (B_1，B_2，...，B_n) は，それぞれレジスタ群の状態を受け取り，割り当てられた命令 (i 番目のブロックであれば $inst_i$) に従って新たなレジスタ群の状態を生成する．チェーン全体では，入力の集合 I を含むレジスタ群の初期状態 RS_0 から，n 個のブロックによるレジスタの操作を経て，出力 o を含むレジスタ群の最終状態 RS_n を得る．演算内容に応じて，具体的な入力値と出力値を与えたチェーンを 1 つ以上記述し，すべてのチェーンの入出力関係が成立するように各ブロックの命令の割当てを SMT ソルバによって求めることで，コードを構成する命令列を生成する．このとき，割り当てられる候補となる命令群の振舞いについての制約もソルバに与える．

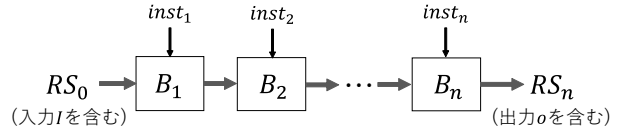

図 1: 演算内容を表すチェーンの概略

```
r1 = SHL r0, 3
r2 = ADD r1, r0
r3 = SHL r2, 1
rf = ADD r3, r0
```

図 2: レジスタ `r0` の値を 19 倍するコードの例

例えば，定数の乗算 (入力値の定数倍の出力値を得る演算) を行うコードを生成したい場合は，その演算の性質から，入力が 0 のとき出力が 0 になるチェーンと，入力が 1 のとき出力が乗数 (定数) になる 2 つのチェーンを記述する [3]．この条件下で生成した，入力値を 19 倍した値を出力するコードを図 2 に示す．ここでは，レジスタ群として `r0`〜`r3` および `rf` が存在し，入力値 `r0` に対する出力が `rf` に格納される．この演算は，`ADD` (加算) および `SHL` (左シフト) をオペコードに持つ 4 つの命令によって記述されている．なお，コード中の各命令は，計算結果が左辺のレジスタに代入される擬似的なアセンブリ表現で示されている (本稿では，以降もこの表記を用いる)．

本研究では，特定の演算処理を行うコードを，指定した命令群によって任意の長さで生成できるこのような仕組みを，コードを複雑にし意味を不明瞭にすることが目的のコードの難読化に応用することを試みる．

Obfuscating Operational Expressions Using an SMT Solver

Yuki Kumai†, Yuichiro Kanzaki‡

†Advanced Course of Electronics and Information Systems Engineering, National Institute of Technology, Kumamoto College

‡Department of Human-Oriented Information Systems Engineering, National Institute of Technology, Kumamoto College

Copyright ©2020 Information Processing Society of Japan.
All Rights Reserved.

3 ケーススタディ

3.1 概要

2章で述べたアイデアを用いてコードの難読化を行い，得られた結果について吟味する．ここで難読化とは，難読化対象となるアセンブリコードに含まれる演算命令について，その命令と同じ意味を持つ2命令以上から成る (複雑な) コードに変換する，という処理を1回以上適用することとする．ここでは，2つのレジスタの加算 (ADD)，減算 (SUB)，論理積 (AND)，論理和 (OR) および排他的論理和 (XOR) といった演算の変換方法を新たに実装し，変換の対象とした．また，生成するコードを構成する命令の候補として，ADD，SUB，AND，OR，XOR，SHL，ROR (右ローテート)，NOT (否定) の8つとした．コードの変換は，文献 [3] で示された例を参考に試作したツールを用いて行った．このツールでは，SMT ソルバとして Z3[1] を用いている．

3.2 コードの生成

上記の変換対象の各演算に対して，構成命令数を3から8まで1刻みに変化させてコード生成を行い，正しく動作すること確認した．図3に例を示す．図3 (a) では，2つのレジスタを加算する ADD の演算が，OR，AND，ADD の3命令で表現されている．これは，「$x + y$」から「$(x \,|\, y) + (x \,\&\, y)$」へと演算の表現の変換が行われたといえる．この変換のアイデアは，難読化ツール Tigress [2] の難読化変形でも参考にされている Warren の文献 [4] にも示されており，そのコード変形を自動生成できたといえる．また，図3 (b) では，論理和 OR の演算が，5命令で表現されたコードに変換されている．これらの変換は，直接的であった元の演算の表現が不明瞭になったという点で，難読化変形とみなすことができると考える．なお，例示したコードは擬似的なアセンブリコードで表現されているため，実際のプログラム内で使用するには，レジスタの扱いなどに関してコードの修正が必要となる．

3.3 評価

変換されたコードが，コンパイラによるコードの最適化によって，簡単化される，すなわち，変換前の命令に戻されたり，コードの一部が除去されたりする場合は，難読化されたコードとして有効とはいい難い．そこで，3.2 で生成した様々なコードに対して，GCC (バージョン 8.3) の最適化 (オプション-O2) を用いた場合のコード内容の変化を確認した．結果として，命令の組合せによっては，構成する命令の一部が除去されたり，複数の命令が1命令に置き換わる (そのまま逆変換される) ことがあった．一方，ADD などの四則演算の命令と，AND などの論理演算命令を組み合わせたコードは，簡単化されにくい傾向が見られた．なお，図3に示した2つの例は，どちらも簡単化されなかった．

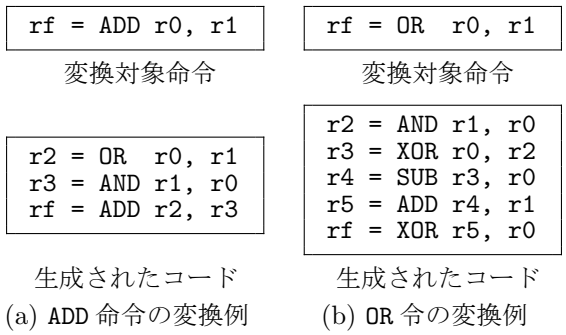

図 3: 命令の変換例

また，シンボリック実行による自動解析の困難さを評価する簡単な実験を行った．対象としたのは，入力値を素因数分解するコードを提案方法で難読化したものである．対象とした関数の命令数は，難読化によって 53 命令から 108 命令に増加した．シンボリック実行ツールの angr[2] を用いて，コード上の特定の位置に到達する入力条件の解析に要する時間を測定したところ，CPU が Intel Core i7-960 の環境における解析時間 (10 回の測定の平均) は，難読化前が 9.23 秒，難読化後が 10.0 秒であり，解析時間がわずかに増加する結果となった．自動解析に対する耐性を詳しく調査するために，今後別の条件で実験を行うことを検討している．

4 おわりに

本稿では，SMT ソルバによるコード生成を用いてアセンブリコードを難読化する方法を検討した．5種類の演算に対して，その命令と同じ意味を持つ複雑なコードを複数生成し，正しく動作することを確認できた．今後の課題として，演算以外のアセンブリ命令を対象にした変換方法について検討することや，既存の難読化方法と有効性を比較することが挙げられる．

謝辞 本研究は，JSPS 科研費 JP19K11916 の助成を受けたものである．

参考文献

[1] C. Collberg and J. Nagra, Surreptitious Software: Obfuscation, Watermarking, and Tamperproofing for Program Protection, Addison-Wesley Professional, 2009.

[2] C. Collberg, "The Tigress C diversifier/obfuscator," http://tigress.cs.arizona.edu/ (accessed: January 2020).

[3] D. Yurichev, "SAT/SMT by example," https://yurichev.com/SAT_SMT.html. (accessed: January 2020).

[4] H.S. Warren, Hacker ' s Delight, Second Edition, Addison-Wesley Professional, 2012.

[1] Z3: https://github.com/Z3Prover/z3

[2] angr: http://angr.io/

Copyright ©2020 Information Processing Society of Japan.
All Rights Reserved.