4L-07

命令列の自動生成機構を用いたLLVM IRコードの難読化の試み

光本 智洋 †　　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 電子情報システム工学系

1 はじめに

従来，ソフトウェアに対するMan-At-The-End攻撃，すなわち，実行可能コードを所有するエンドユーザによるソフトウェアの解析･改ざん行為が，ソフトウェア市場にとっての懸念事項となっている．近年の攻撃手段としては，逆コンパイラを用いて行う解析や，シンボリック実行を用いた自動解析攻撃 [1] などがあり，これらを困難にするための方法が求められる．本研究では，シンボリック実行を用いた自動解析攻撃からソフトウェアを保護する方法として，LLVM IR（コンパイラ基盤であるLLVMの中間表現）のコードを対象にした難読化方法について検討する．提案方法は，LLVM IRのレベルにおいて，コード中の単純な命令（例えばadd命令）を，命令の意味が保たれた複雑な表現を持つ命令列に置換することで，コードの難読化を試みる．複雑な表現を持つ命令列は，元来の命令（置換対象の命令）の入出力例などをもとに，SMTソルバを用いて自動生成する．

プログラムコード内の単純な命令をSMTソルバによって自動生成された複雑な命令列に置き換える，という難読化方法については，著者らの先行研究 [2] において，GAS (GNU Assembler) を対象にしたものが検討されている．本研究では，この方法をLLVM IRレベルでのコードの難読化に応用することを試みる．LLVMのIRレベルで難読化を行う仕組みを構築することで，多くのプログラミング言語やアーキテクチャに対応できるという利点が生じる．LLVMのIRを対象にした既存の難読化システムとして，Obfuscator-LLVM [3] などが提案されている．本研究では，SMTソルバによって複雑なLLVM IRの命令列を生成するいう新たなアプローチに基づく難読化システムを試作し，それによって難読化された実行可能コードについて，実行結果の正しさや，シンボリック実行を用いた自動解析に要する時間などを確認する．

Obfuscating LLVM IR Code Using a Code Fragment Generation Technique

Tomohiro Mitsumoto†, Yuichiro Kanzaki‡
†Electronics and Information Systems Engineering Advanced Course, National Institute of Technology, Kumamoto College
‡Faculty of Electronics and Information Systems Engineering, National Institute of Technology, Kumamoto College

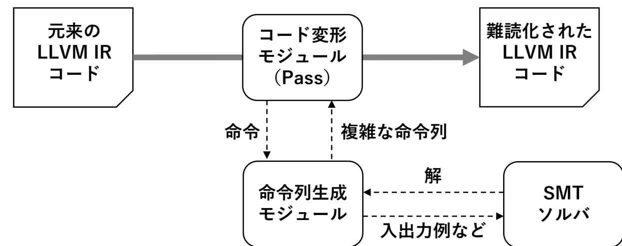

図 1　提案システムの概略

2 提案方法

提案する難読化システムの概略を，図1に示す．提案システムは，LLVM IRのレベルでコードを難読化する．具体的には，LLVMのPassとして実装された「コード変形モジュール」が，入力として与えられたLLVM IRコード中の特定の命令（本稿では，add命令など5命令）を複雑な命令列に置き換えることで，難読化されたLLVM IRコードを出力する．ここで「複雑な命令列」とは，元来の命令と同様の意味を持つ，2命令以上で構成される命令列（コードの断片）を指す．複雑な命令列は,「命令列生成モジュール」が，文献 [2] や [4] において検討されているアセンブリ命令列の自動生成のアイデアに基づき生成する．具体的には，元来の命令の入出力例，命令列を構成する候補となる命令の挙動の情報，命令列長などをSMTソルバに制約として与え，得られた解をもとに命令列を生成する．

図2に示すのは，add命令（加算）と同様の意味を持つ長さ3（3命令）の命令列を生成する流れである．命令列の生成には，対象命令の動作に応じた複数の入出力例を用いるが，ここでは簡単のため，1組の入出力例から命令列を生成する例を示す．まず，5個のレジスタ (%0〜%4）が定義されており，最初の2つのレジスタ%0と%1に入力値1と5が，最後のレジスタ%4に出力値6が設定されている．各命令（$inst_1$〜$inst_3$）は，直前の状態のレジスタに対して特定の操作を行い，新たなレジスタに結果を出力する．$inst_1$〜$inst_3$にどのような命令をあてはめれば，入力値1と5から最終的に6を出力できるかをSMTソルバに解かせることで，命令列を生成する．この例では，結果として，and，or，addの3命令が割り当てられており，これら3命令を順につなぐことで，add命令と同様の意味を持つ，

Copyright ©2022 Information Processing Society of Japan.
All Rights Reserved.

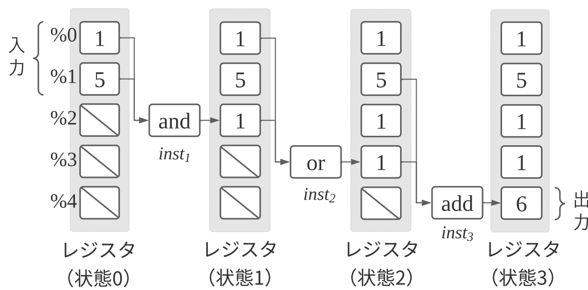

図 2　長さ 3 の命令列生成の流れ

```
%4 = xor i32 %1, %1
%5 = and i32 %0, %4
%6 = xor i32 %5, %0
%7 = add i32 %1, %6
%8 = or  i32 %4, %7
```

(a) 長さ 5 の命令列

```
%2  = xor i32 %1, %1
%3  = and i32 %0, %2
%4  = sub i32 %3, %1
%5  = add i32 %0, %4
%6  = xor i32 %5, %2
%7  = add i32 %0, %6
%8  = sub i32 %7, %0
%9  = sub i32 %8, %0
%10 = and i32 %2, %9
%11 = or  i32 %10, %4
%12 = add i32 %2, %11
%13 = sub i32 %0, %12
```

(b) 長さ 12 の命令列

図 3　「add %0，%1」を置換できる命令列の例

長さ 3 の命令列が得られることになる．実際に提案システムで得られた，命令「add %0，%1」を置換できる長さ 5 および長さ 12 の命令列を，図 3 に示す．

3　実験

3.1　概要

提案方法によって難読化されたプログラムについて，実行結果の正しさやシンボリック実行による自動解析に要する時間を，実験を通して確認する．実験対象は，Banescu によって公開されている文字列のハッシュ値を計算する 3 種類のプログラム[1] P_{djb} (djbhash.c)，P_{elf} (elfhash.c) および P_{pjw} (pjwhash.c) とする．ここでは，C 言語のソースコードを Clang（バージョン 13.0.0）によって LLVM IR に変換し，提案システムで難読化を行った後に実行可能コードを生成する．置換対象の命令は，add，sub，and，or，xor の 5 つであり，生成する命令列を構成する候補となる命令も，これら 5 命令とする．本実験では，1 回の難読化処理において，プログラムに含まれるすべての置換対象命令を長さ 5 の命令列で置き換える．実験は CentOS 7（CPU は Intel Core i9-9900KF）の環境で行い，命令列生成のための SMT ソルバは Z3[2]を，シンボリック実行解析のツールは angr[3]を用いる．

[1]obfuscation-benchmarks : https://github.com/tum-i4/obfuscation-benchmarks/
[2]Z3: https://github.com/Z3Prover/z3/
[3]angr: http://angr.io/

表 1　angr を用いた自動解析に要した時間

対象	難読化前	難読化後		
		1 回	2 回	3 回
P_{djb}	3 秒	3 秒	8 秒	Error
P_{elf}	4 秒	5 秒	21 秒	Timeout
P_{pjw}	4 秒	5 秒	7 秒	237 秒

3.2　結果と考察

まず，難読化後の各プログラムが，難読化前と同様に正常に動作することを確認した．また，難読化するごとに，IR のコードの命令数が増加していることを確認した．各プログラムのシンボリック実行解析に要した時間を表 1 に示す．解析時間は，特定の出力に到達する入力値を得るのに要した時間を 3 回計測し，その平均値を求めたものである．Timeout は解析時間が 1 時間を超えたことを，Error は angr が解析時にエラーで終了したことを示す．なお，1 回目の計測で Timeout となった場合は，2 回目以降の計測を行っていない．結果から，いずれのプログラムも難読化するごとに解析時間が増加する傾向があることがわかる．特に 3 回難読化した後のプログラムは，解析時のエラーや解析時間の大幅な増加が認められ，難読化によってシンボリック実行解析への耐性が高くなる傾向を確認できた．

4　おわりに

本研究では，SMT ソルバによる命令列の自動生成機構を用いた LLVM IR コードの難読化方法について検討した．実験では，難読化されたプログラムが正常に動作することや，難読化によってシンボリック実行解析への耐性が高くなる場合があることを確認した．今後の課題として，より多くの種類の命令を置換対象にできるよう提案システムを改善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] S. Banescu, C. Collberg, V. Ganesh, Z. Newsham, and A. Pretschner, “Code obfuscation against symbolic execution attacks,” Proceedings of the 32nd Annual Conference on Computer Security Applications, pp.189–200, 2016.

[2] 光本智洋，神崎雄一郎，“SMT ソルバによる命令列生成を用いたアセンブリプログラムの難読化，” 情報処理学会第 83 回全国大会講演論文集 (講演番号 2K-04)，March 2021.

[3] P. Junod, J. Rinaldini, J. Wehrli, and J. Michielin, “Obfuscator-LLVM – software protection for the masses,” Proceedings of the IEEE/ACM 1st International Workshop on Software Protection (SPRO’15), pp.3–9, 2015.

[4] D. Yurichev, “SAT/SMT by example,” https://sat-smt.codes/.

Copyright ©2022 Information Processing Society of Japan.
All Rights Reserved.