5J-05

# 逆難読化に向けた適用された難読化手法の特定

匂坂 勇仁　　玉田 春昭
京都産業大学

## 1 はじめに

近年、不正解析など違法行為が増加している。その一対策手法に難読化と呼ばれる保護手法がある。難読化とは、プログラムの解析を困難にする手法である。一方、難読化されたプログラムを難読化前のプログラムに戻す逆難読化がある。逆難読化を行うには、どの難読化が適用されているか特定する必要がある。異なる方法での逆変換はプログラムを壊すためである。このような逆難読化は保護を無効にするため、対策が必要である。そこで本稿では保護手法の特定のしやすさを測定し、保護手法のみつけやすさを計測する。具体的には、難読化されたプログラムを解析し、難読化手法の特徴を探る。我々は、神崎らの不自然さ評価法 [1] を利用した測定を行い、Allatori という難読化ツールの特定の可能性を示した [2]。本稿では、この研究をさらに推し進め、より多くの保護手法の特定を試みる。

## 2 提案手法

### 2.1 キーアイデア

図 1 に提案手法の模式図を示す。図 1 のようにプログラム $p$ を何らかの保護手法 $f_1$ と $f_2$ で保護し、$p_1 = f_1(p)$, $p_2 = f_2(p)$ を得る。そして、特徴抽出関数 $C$ を用いて $p$, $p_1$, $p_2$ から何らかの特徴 $C(p)$, $C(p_1)$, $C(p_2)$ を得る。その後 $C(p)$ と $C(p_1)$, $C(p_2)$ がどれくらい似ているか、また、$C(p)$ と $C(q)$ がどのくらい違うかを評価する。

その後、新たに不明な難読化手法 $f_x$ で難読化されたプログラム $f_x(p)$ を用意し、提案手法と同様に解析を行う。その結果、どの難読化手法が似ているかを検証する。

### 2.2 特徴抽出法

神崎らが提案した不自然さ評価法に着目する [1]。この手法は、オペコードの $n$-gram の珍しさを利用するものである。保護手法に現れる命令列の珍しさが保護手法の特徴であると言えるためである。珍しさを算出するために幾つかの手法が提案されている [1, 3]。本稿では、珍しさの算出にパープレキシティを利用する。パープレキシティとは平均分岐数として知られ、言語としての複雑性を表す指標である。この値が小さいほど、次に来る語が予測しやすいことを表す。そのため、値が小さいほど自然であると言える。

一方で、保護手法の中ではコンパイラが出力した本来の命令を重複させる場合もある。本来の命令は使用頻度の違いはあるものの、珍しいとは言い難い。そこで、$n$-gram の頻度にも着目する。このように、プログラムから命令の $n$-gram を抽出し、その珍しさ (パープレキシティ) と頻度を保護手法の特徴とする。

表 1: 利用した Jar ファイル一覧

| Product | Version | Product | Version | Products | Version |
|---|---|---|---|---|---|
| ASM | 3.3.1 | FakeHack | 1.0 | JCalendar | 1.3.3 |
| Jhstop | 0.0.1 | Jwhich | 1.0 | Robocode-setup | 1.6.0.1 |

表 2: 利用した難読化ツール

| Tools | Abbr. | Overview |
|---|---|---|
| Allatori Java Obfuscator | ALL | 商用の難読化ツール |
| ProGurad | PG | OSS の難読化ツール |
| Sandmark | | 研究用の難読化ツール |
| Duplication registers | DR | 代入を重複させる. |
| Merge local integers | MLI | 2 つの int 変数を long に収める. |
| Irreducibility | IRR | 制御フローを複雑にする. |

## 3 評価実験

### 3.1 概要

本稿では、提案手法により、保護手法ごとの特徴の検出を目的とする。そこで、Java を対象とした難読化ツールを用いて、実際に jar ファイルを難読化する。抽出する特徴は、第 2.2 節で述べたようにオペコードの

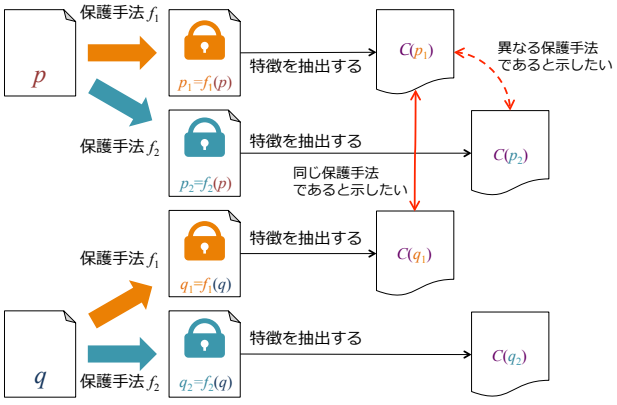


図 1: 提案手法の模式図

Identifying the applied obfuscation method for de-obfuscation
Hayato Sagisaka　Haruaki TAMADA
Kyoto Sangyo University

表 3: ツールごとの特徴

| ツール | 特徴 |
|---|---|
| ALL | オリジナルの命令列を swap 命令で入れ替える |
| PG | オリジナルと変化なし |
| DR | istore 命令を 2 回続けて実行する |
| MLI | dup2x2 lxor を連続して実行する |
| IRR | nop 命令がある |

表 4: JUnit (5-gram) の命令列

| 命令列 | 頻度 | PPL |
|---|---|---|
| dup2x2 lxor lconst_1 lneg bipush | 16 | 11.30E+05 |
| lload dup2x2 lxor lconst_1 lneg | 16 | 5.37E+05 |
| bipush lushr land lxor lstore | 9 | 0.52E+05 |
| lconst_1 leng bipush lushr land | 9 | 0.96E+05 |
| lneg bipush lushr land lxor | 9 | 1.37E+05 |

*n*-gram を基本とし、そのパープレキシティと頻度を用いて比較を行う。対象とする Jar ファイルは表 1 に、難読化ツールは表 2 に記す。

その解析結果をもとに、不明な難読化手法の解析を行う。初めに、未知のプロダクトをケーススタディで扱った 5 つの難読化手法 (表 2) をランダムに選択し、難読化する。その後、どの難読化手法が使われているかを検証する。次に、既知のプロダクト (表 1) を未知の難読化手法で難読化する。その後、どの難読化手法が使われているかを検証する。未知のプロダクトとして JUnit 3.7 を使用した。また、未知の難読化手法として Sandmark の Simple Opaque Predicates (SOP) を使用した。

### 3.2 *n*-gram の分析

ここでは、それぞれの難読化手法を頻度とパープレキシティを参考に *n*-gram の命令列を解析した。解析結果は、難読化手法の保護前後で違った命令列がみられた。また、追加された命令列はパープレキシティの値が高く、珍しい命令列であることがわかった。これらのデータから追加された命令列を難読化手法の特徴とし、その特徴を表 3 に記す。

### 3.3 既知の難読化手法の特定

ここでは、未知のプロダクトを表 2 のいずれかを用いて難読化し、どの難読化手法が使われているかを検証する。難読化適用前の *n*-gram に出現していない *n*-gram を調査した。その結果のうち、高頻度の *n*-gram を 5 つ、表 4 に記す。

表 4 をみると、命令列には、dup2x2 lxor といったパープレキシティの値が高い命令列を呼び出している。その命令列を特徴とし、5 つのツールの特徴 (表 3) と比較する。その結果、適用されている難読化手法の特徴と MLI 手法の特徴が当てはまる。

表 5: Jhstop (5-gram) の命令列

| 命令列 | 頻度 | PPL |
|---|---|---|
| irem iconst_0 if_icmpne aload getfield | 15 | 0.15E+05 |
| dup dup dup imul imul | 14 | 42.40E+05 |
| dup dup imul imul isub | 14 | 8.74E+05 |
| dup imul imul isub iconst_3 | 14 | 3.49E+05 |
| iload dup dup dup imul | 14 | 101.00E+05 |

### 3.4 未知の難読化手法の特定

ここでは、既知のプロダクトを未知の難読化手法で難読化を行い、どの難読化手法が使われているかを検証する。難読化適用前の *n*-gram に出現していない *n*-gram を調査した。その結果のうち、高頻度の *n*-gram を 5 つ、表 5 に記す。

表 5 をみると、パープレキシティの高い命令列であり、dup, imul 命令を連続で呼び出している。その命令列を特徴とし、5 つのツールの特徴 (表 3) と比較する。その結果、未知の難読化手法の特徴は、解析したどの特徴にも当てはまらなかった。次に、異なるプロダクトを未知の難読化手法で難読化し、それぞれの特徴を比較する。その結果、同様の特徴が検出されたため、未知の難読化手法が SOP 手法だとわかった。

## 4 まとめ

本稿では不自然さ評価に着目して、適用された保護手法の特定に取り組んだ。実際のプログラムに、5 種類の保護手法を適用し、*n*-gram の命令列を頻度順に確認した。その結果をもとに不明な難読化の解析を行った。今回の評価実験では、既知の難読化は特定しやすいが、未知の難読化の特定は難しい。この解決策として、難読化手法の特徴を増やすことである。特徴を増やすことで未知の難読化手法に対応できる。

## 参考文献

[1] 神崎, 尾上, 門田. コードの「不自然さ」に基づくソフトウェア保護機構のステルシネス評価. 情報処理学会論文誌, Vol. 55, No. 2, pp. 1005–1015, 2014.

[2] 匂坂, 玉田. 適用保護手法特定の試み ー不自然さ評価方法を用いてー. 信学技報, SS2015-21, pp. 63–68, 2015.

[3] 大滝, 大堂, 玉田, 神崎, 門田. Java バイトコード命令のオペコード、オペランドを用いた難読化手法のステルシネス評価. 2014 年暗号と情報セキュリティシンポジウム予稿集 (SCIS2014), 2D2–2, 2014.