2J-04

難読化されたプログラムの自動解析への耐性に関する考察

西 陽太 † 神崎 雄一郎 ‡ 門田 暁人 *

† 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
* 岡山大学大学院 自然科学研究科

1 はじめに

プログラムの難読化は，ソフトウェアに対する不正な解析・改ざん行為を困難にする手段として知られ，従来様々な方法が提案されている [1]．逆アセンブラやデバッガなどによって攻撃者がプログラムを理解しながら手動で解析する攻撃に対しては，既存の難読化方法の有効性が数多く議論されている一方で，近年見られるようになったシンボリック実行などに基づく自動解析を伴う攻撃に対しては，どのような難読化方法が有効であるかの議論は乏しい．

そこで本研究では，プログラムの難読化によって保護されたソフトウェアが，自動解析を用いた攻撃に対してどの程度耐性を持つか (解析が困難であるか) を実験を通して考察する．具体的には，既存の難読化方法で変形された実験用のライセンスチェックのプログラムに対して，シンボリック実行とブルートフォースの 2 種類の攻撃を行い，パスワード (アクティベーションコード) の特定に要する時間を計測する．その計測時間に基づき，各攻撃に耐性のある難読化方法について議論する．先行研究として，Banescu らはシンボリック実行に対する静的難読化方法の耐性について述べている [2]．本研究では，静的難読化方法に加え，動的難読化方法 (自己書換えを用いた難読化方法) の耐性についても検討する．また，プログラム構造に注目して解析を行うシンボリック実行に加えて，入出力関係のみに注目して行われるブルートフォース攻撃に対する耐性についても検討する．

2 対象とする解析方法

本研究で対象とする解析方法であるシンボリック実行およびブルートフォースについて述べる．シンボリック実行は，実行可能なプログラムの各部分に至るパスと，各パスが実行されるための入力条件を求めるプログラム解析方法である．近年，KLEE や angr などのシンボリック実行のツールが公開されており，ソフトウェアテストやマルウェア解析に利用される一方で，実行させたいプログラム部分へのパスを自動的に求める手段として不正なソフトウェア解析にも利用され得る．ブルートフォースは，入出力関係のみに注目して行う総当たり攻撃である．攻撃の性質上，自動化することが容易であり，単純なライセンスチェック機構などに対しては脅威となり得る．本研究では，シンボリック実行のツールや自動化されたブルートフォースを自動解析の手段とみなし，実験の対象とする．

3 ケーススタディ

既存の難読化方法が，自動解析を用いた攻撃に対してどの程度耐性を持つかを実験を通して検討するケーススタディを行う．解析対象は，簡単なライセンスチェックのプログラムを既存の難読化方法で難読化したものである．ライセンスチェックのプログラム (C 言語) の一部を図 1 に示す．このプログラムでは，ライセンスの有効期限を確認し，期限が切れている場合はアクティベーションコードの入力を求める．図 1 ではアクティベーションコードを整数で表現しているが，文字列で表現したものも実装した．これらを難読化するために用いた方法は，演算表現の複雑化，制御フローの平滑化，独自の命令セットによる仮想化，opaque predicate の挿入といった 4 種類の静的難読化方法，および，命令のカムフラージュ，JIT による動的書換えといった 2

```
int checklicense(void) {
  time_t current_time;
  char code[256];
  time(&current_time); /* set current time */
  if (current_time > mktime(&EXPIRE_TIME)) {
    printf("Your license is expired.\n");
    while(1) {
      printf("Enter activation code: ");
      fgets(code, sizeof(code), stdin);
      if(atoi(code) == ACTIVATION_CODE) {
        renewlicense();
        return 0;   /* success */
      } else {
        failure();  /* failure */
      }
    }
  }
  return 0;  /* success */
}
```

図 1: ライセンスチェックのプログラムの一部

A Study on the Effectiveness of Code Obfuscation Techniques Against Automated Attacks
Yota Nishi†, Yuichiro Kanzaki‡, Akito Monden*
†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
*Graduate School of Natural Science and Technology, Okayama University

表 1: 解析に要した時間 (単位は秒)

難読化種別	難読化方法	シンボリック実行 整数	シンボリック実行 文字列 (4 桁)	シンボリック実行 文字列 (8 桁)	ブルートフォース 整数	ブルートフォース 文字列 (4 桁)	ブルートフォース 文字列 (8 桁)
難読化なし	—	8.48	3.02	2.98	1.54×10^4	7.53	TO
静的難読化	演算表現の複雑化	8.08	3.09	3.16	1.42×10^4	7.50	TO
	制御フローの平滑化	9.86	4.10	4.06	1.43×10^4	7.25	TO
	プログラムの仮想化	13.9	8.33	9.01	1.51×10^4	7.65	TO
	opaque predicate の挿入	9.66	3.65	3.57	1.42×10^4	7.56	TO
動的難読化	命令のカムフラージュ	WA	WA	WA	1.41×10^4	7.68	TO
	JIT による動的書換え	WA	WA	WA	1.40×10^4	7.56	TO

種類の動的難読化方法である．命令のカムフラージュについてはアセンブリ言語レベルにおいて手動で難読化の変形を行ったが，その他の難読化は，難読化ツールの Tigress[1] を用いて C 言語のレベルで自動的に難読化の変形を行った．今回対象とした難読化方法については，文献 [1] や Tigress の Web サイト [1] に説明がある．紙面の都合上，難読化結果の詳細は省略するが，静的難読化によって，比較演算の表現が複雑になる，条件分岐の階層の深さが変わるなどの変化がプログラムに生じ，動的難読化によって，比較演算の式や制御の分岐点にあたる命令が隠されるなどの変化がプログラムに生じる．難読化された各プログラムは，難読化前と同じように正常に動作することを確認した．

本実験での解析のゴールは，難読化された各プログラムからアクティベーションコードを特定する (図 1 中の `renewlicense` 関数に到達する入力条件を求める) ことである．ここでは，解析の成功に要する時間を解析への耐性の強さとみなす．本実験では，ライセンスの有効期限が切れておりアクティベーションコードの入力が求められるという前提で，解析対象のプログラムに対してシンボリック実行とブルートフォースの解析を 1 回ずつ行い，解析に要した時間を求めた．プログラムのアクティベーションコードは，整数で表現したものは 4 バイト符号なし整数の範囲内で，文字列で表現したものは英数字からなる 4 文字，および，8 文字でそれぞれランダムに設定した．なお，シンボリック実行による解析は angr[2] を，ブルートフォースによる解析は Python で独自に実装したものを使用した．解析を行った環境は，OS が Ubuntu 14.04，CPU が Intel Core i5-2320 (3.00GHz)，メインメモリが 2GB である．

表 1 に実験結果を示す．表中の `TO` は 12 時間以内で解析が終了しなかったこと，`WA` は解析が終了したが正しい解析結果が得られなかったことを意味する．シンボリック実行による解析では，プログラムの仮想化が比較的長い時間を要しているものの，静的難読化が適用されたものは 15 秒以内で解析が成功した．一方，動的難読化が適用されたものは，正しいアクティベーションコードを求められず，図 1 のような簡単なライセンスチェック機構に対しても，今回の実験の範囲内では解析成功に至らなかった．シンボリック実行は，変数をシンボルとして扱い，到達目標である関数までのパスとその入力条件を求める．動的難読化は実行時に実行内容を書き換えるが，実行前のプログラムから得られる情報からでは目的のパスを正しく求めることができず，解析に失敗した可能性がある．さらなる実験が必要であるが，動的難読化は静的難読化よりもシンボリック実行に対する耐性が強いことを示唆する結果となった．ブルートフォースによる解析では，アクティベーションコードが文字列 8 桁で表される設定では，いずれも 12 時間以内に解析できなかったが，整数や文字列 4 桁の場合は，解析に成功する場合があった．また，シンボリック実行では解析に失敗した動的難読化が適用されたプログラムに対しても，解析が成功する場合があった．ブルートフォースの性質上，解析時間は設定されたアクティベーションコードの内容や総当たり攻撃の手順に関係する一方で，適用されている難読化方法には大きく依存しないことが結果から窺える．

[1]Tigress: `http://tigress.cs.arizona.edu/`
[2]angr: `http://angr.io/`

4 おわりに

本研究では，難読化によって保護されたプログラムが，自動解析を用いた攻撃に対してどの程度耐性を持つかを実験を通して考察した．実験結果から，一部の動的難読化方法は静的難読化方法と比較してシンボリック実行による解析攻撃に強い耐性を持つ可能性があることがわかった．自動解析の方法や，解析対象となるプログラムの種類を増やして実験を行うことが，今後の課題である．

参考文献

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

[2] S. Banescu, C. Collberg, V. Ganesh, Z. Newsham and A. Pretschner, “Code Obfuscation Against Symbolic Execution Attacks,” *Proc. 32nd Annual Computer Security Applications Conference*, pp.189-200, 2016.