コンピュータセキュリティ 3−5
（1998. 11. 27）

耐タンパー技術のためのアルゴリズム難読化の一提案

坂本憲昭　　岡本栄司
sakamoto@jaist.ac.jp　　okamoto@jaist.ac.jp

北陸先端科学技術大学院大学
情報科学研究科
〒 923-1292 石川県能美郡辰口町旭台 1-1

あらまし　システムなどから秘密情報を得る方法の１つに、実行プログラムを解析する方法がある。攻撃者からの不正な解析からシステムを保護するため、あらかじめ、解析されにくいようにプログラムを作成しておく必要がある。これらの手法は耐タンパーソフトウエアと呼ばれ、ソフトウエアの機密保持や不正利用を阻止する技術として注目されている。本稿では、プログラム解析について考察し、どのようなプログラムが解析するのが難しいか考えてみた。そこで、条件分岐を利用することにより複雑にプログラムを構成する方法を提案する。条件分岐のようなプログラムの制御構造に着目することにより、難読化を達成したい。
キーワード　情報セキュリティ、耐タンパーソフトウエア、難読化プログラム、リバースエンジニアリング

Scrambling Programs for Tamper Resistant Software

Noriaki SAKAMOTO　　Eiji OKAMOTO
sakamoto@jaist.ac.jp　　okamoto@jaist.ac.jp

School of Information Science
Japan Advanced Institute of Science and Technology

1-1 Asahidai, Tatsunokuchi, Nomi, Ishikawa, 923-1292, JAPAN
Tel.: 0761-51-1699 ext.1387

Abstract　One of the methods which get secret information from computer/network systems is to analyze executive programs. We need to make programs which are hard to analyze in order to protect system from the illegal analysis of attackers. This technique is called tamper resistant software, it is expected to prevent illegal uses and protect secrets of the software from the illegal analysis of attackers. In this paper, we discuss how to make a program hard to understand, and propose a method which composes a complicated program using a conditional branch. By taking advantage of control structure of the program, the conditional branch is effective measures against reverse engineering.
Keywords　information security, tamper resistance, software scrambling, reverse engineering

1 はじめに

暗号の高度な発展により、攻撃者は暗号そのものを対象とする他に、ソフトウエアやハードウエアなどのシステムを攻撃することで、秘密情報を奪取しようとしている。耐タンパー技術とは、このような不正な攻撃からソフトウエアやハードウエアなどのシステムを保護する技術である。このような解析が困難であるハードウエアやソフトウエアを耐タンパーモジュールと呼び、ソフトウエアによる耐タンパー技術にアルゴリズム難読化がある。これは、ソフトウエアの不正な解析が問題になってきているため、ソフトウエアを攻撃者による解析から保護する技術である。

2 解析とプログラム

解析者が実行コードなどから解析するときには、リバースエンジニアリングの技術を使う。この技術は、実行コードから解析者にとって、有益な情報を抜き出すことである。このときに使用されるツールとしては、実行コードからアセンブラに変換するツール[1]や、実行しながらプログラム解析ができるトレーサーなどがある。また、解析法には静的解析法と、動的解析法があり、前者は、逆アセンブルした情報をもとに解析するのに対し、後者は、トレーサーなどのツールで実際に実行されているコードを追跡しながら解析する。以上のようにして得られた解析の情報から、プログラムがどのようなものであるかを調べる。

3 プログラム解析

3.1 変数名の情報について

プログラムの解析時において障害になるのは、変数名や関数名がアドレス情報に変換されており、何ら意味を持たないことである。プログラム作成時において変数名や関数名は、作成者が理解しやすく、プログラムの内容が想像できる名前の付け方をおこなう。例えば、消費税を加算した支払金額を計算するプログラムは、次のように書ける。

SYOHIZEI = TEIKA * 0.05 ;

SIHARAI = TEIKA + SYOHIZEI ;

以上は、消費税を算出し、それに定価を加算することで支払金額を計算している。このプログラムは、この2行からどういう処理をおこなうものかが理解しやすい。しかし、プログラムをコンパイルすると、SYOHIZEI、TEIKA、SIHARAIなどは、アドレス情報に変換されるので、実行コードを解析すると次のような情報が得られるだろう。

[160] = [164] * 0.05 ;

[168] = [164] + [160] ;

ここで、[160] などは、160番地のアドレスの中身とする。

最初のプログラムは、変数名が「どのような用途に変数を使用するか」を表現しているので、プログラムの意味が理解しやすい。つまり、解析者は「このプログラムは、定価に0.05をかけることにより、消費税を計算し、定価を足すことで支払金額を求めている。」と、読みとるであろう。このことから解析者は、意識的にせよ、そうでないにせよ、このプログラムは金銭などを計算するプログラムではないかと、とらえることができる。このようにプログラムの情報や動作そのものからでなく、解析者の知識を、変数名の情報により引き出すことができる。このようにして引き出された情報を、プログラムの解析に使用することで効率的に作業が進めることができる。

しかし、2番目のプログラムは「[164] に0.05をかけて、[160] に格納し、[164] と [160] とを足し合わせている」となるので、これだけでは、何をしているのかが理解できない。解析者は、このプログラムは何のために作成されたかを知らないので、もしかしたら、「何らかの科学技術計算の式を表している」と受け止めるかも知れないし、また「他の用途での計算に使う式」と受け止めるかも知れない。どちらにしろ、変数名（ここではアドレス情報）から、どのような性質を保持して

[1] 逆アセンブラや逆コンパイラといわれるツール

いるデータなのかを推測できないので、ただ変数を「足して掛ける」だけでは、何をしようとしているのか理解し難い。解析者が、このコードが何のために書かれたものかを知るためには、外部からの情報を得るか、プログラムの解析を進めて、このコードが何のために書かれたかを知る必要がある。

解析者は、変数名などに頼るのではなく、データがどのように操作されているかにおいてプログラムの意味を把握しようとするだろう。解析の過程において、どのようにデータが変化するか、どのような演算が使用されているかを調べ、そこから得られた情報から解析者がどのような用途で使用されているかを考えるだろう。

3.2 命令・制御の情報について

プログラム解析では、「変数名、関数名が持つ意味」、そして「命令・制御の持つ意味」を考える必要がある。「命令の持つ意味」とは、「その命令が何を表しているか」である。つまり、アセンブラの命令にしても、1命令で複雑な命令を実行できるものもあれば、そうでないものもある。

例えば、jsr 命令は1命令でサブルーチンジャンプという複雑な処理ができるのに対し、add 命令は、足し算という簡単な処理しかできない。

このことは、解析時において複雑な処理をもつ命令ほど、命令の意味がとりやすいと考えられる。もし、このような命令を複数の簡単な命令で構成したとすると、解析者は簡単な命令の列から、これらの命令が何を処理しようとしているのかを考えなければならない。1行あたりの処理情報の濃度が低いからである。このことを利用して、意図的に1命令で表現できる処理を複数の命令で表すと、難読化がかかる。jsr 命令は複数の簡単な命令で構成できるので、簡単な命令の列の方が解析は難しくなる。このように複雑な命令を単純な命令の列に書き換えたりする方法は、文献 [1] に提案されている。

また、ループを複雑に変換して解析し難くする方法が、文献 [2] で提案されている。この手法は、ループの等価変換するパターンをカタログとして整理しておき、これを用いて、難読化をかけたいプログラムに対して変換をおこなう。このとき、プログラムでは実行効率を落さずに変換が可能である。

3.3 難読化の評価

難読化についての評価は難しく、また、どの程度までかけるのかが問題である。評価に関しては、解析者に実際に解析してもらう方法などがあるが、これは解析者の技術力に左右されて不安定である。また、解析できないプログラムコードはないと考える解析者もいるだろうし、プログラムは解析するのは難しく、既存の難読化の技法でも効果があると考える解析者もいるだろう。

4 提案手法

難読化にあたって、「解析し難いプログラムとは何か」について考えなければならない。ここで考えたいことは、プログラムの条件分岐に関してである。一般的なプログラムは条件分岐を含んでいる。この条件分岐によりプログラムの流れが制御されているので、解析者は、条件分岐による複数の流れについて考えなければならない。つまり、その時点で解析者は流れの候補のうち一つを解析して、また、この地点にもどり、他の流れを解析しなければならないことである。このことを利用して、不必要に条件分岐文をプログラムに押し込むと、難読化の度合が高くなるのではないかと考える。このため、複数のプログラムコードを、難読化をかけたいプログラムに挿入することにより、難読化が達成できる。ここで難読化をかけたいプログラムを主プログラムといい、そのために挿入するプログラムをダミープログラムという。また、難読化プログラムは、難読化をかける前のプログラムと同じ動作をしなければならない。

4.1 分岐命令の難読化

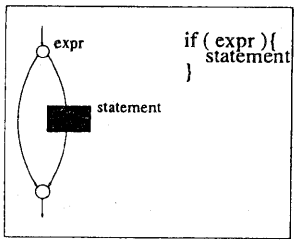
4.1.1 概要

この提案手法では、条件分岐命令を不必要に挿入する方法である。複数のプログラムを混在させたとき、変数名が重複していたり、他の混在するプログラムに影響を及ぼすような命令[2]を使用しない、などの点に注意する。そこで混在させるプログラムにはいくつかの制限をつけたい。

- それぞれのプログラムがもつ変数が、プログラムを混在させた時に、他のプログラムから不正なアクセスをされない。
- 他のプログラムに影響を及ぼすような命令は避ける。
- プログラムは、無限ループや時間がかかり過ぎる動作を避けるようにする。
- C言語でいう continue, break, return は使用しない。

4.1.2 難読化のかけかた

1. まず、プログラムを有向グラフに置き換える。置き換え方は、IF などの分岐命令と、その他の命令文をノードに変換し、そのノードをエッジでつなぐ。



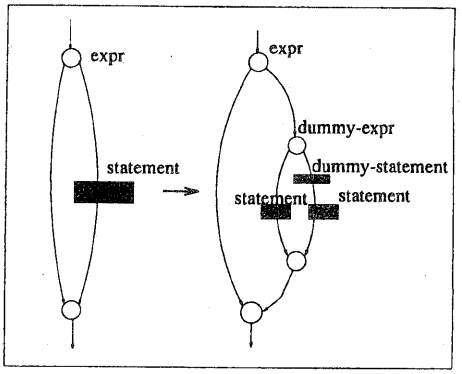
2. 次に、プログラムの変数を調べ、挿入するプログラムの変数と変数名が重複しないかどうかを調べる。もし重複するなら、その変数が属しているプログラムファイル名などの識別子をその変数名の先頭につける。

3. プログラム挿入にあたり、エッジ上で挿入したい区間をマークする。

4. マークした区間に、命令ノードがあれば、その命令ノードを分岐したあとでも実行できるようにした後、ダミープログラムを挿入する。

5. マークした区間に、制御ノードがあるとき、そのマークは取り消す。

6. マークした区間に、何もなければ、ダミープログラムを挿入する。



以上のことを繰り返す。

4.2 変数について

上記の方法で、条件分岐命令を組み合わせた。しかし、このままでは主プログラムとダミープログラムとの間の関係は弱い。これは、お互いのプログラムがお互いの変数を使用していないので、解析時に変数について解析されると、プログラムがそれぞれ分離しやすく、解析が容易である。難読化をかける方としては、解析者がプログラムの解析をするとき、多くのダミープログラムを解析させたい。どれだけ多くのダミープログラムを解

[2]Input/Output 命令など

析してもらうかが、難読化の評価基準のひとつでもある。

解析者は、ある命令を解析する時、その命令に使用されている変数がいままでにどういう使われ方をしていたのかも調べるであろう。その変数の使われ方を調べ、その命令がなぜ、そのように使われているかを調べるためである。

その解析過程において、より多くのダミープログラムを調べてもらうようにする。具体的には、異なるプログラムの変数に対して代入や参照をおこなう追加命令を意図的に挿入することである。これにより、プログラム間の連結は強まり解析しにくくなる。

4.2.1 追加命令を挿入する場所

追加命令は、どこにでも挿入すれば良いというものではない。挿入可能な区間がある。もし、この挿入可能な区間以外に追加命令がはいると、プログラムの動作が不安定になる可能性がある。これは、ある変数 V に、正しい値が代入されるはずなのに、難読化をかけるユーザやツールが不適切な命令を挿入し、その結果、不安定な値が V に代入されることによる。プログラムの配列の添字に使用される変数などは、適切な値を代入しなければ、不正なデータアクセスをおこして、プログラムの動作がおかしくなる。したがって、追加命令をどのように挿入するかは注意する必要がある。また、どのプログラムにも属していない、ダミー変数を使用しても構わない。これは、プログラムの実行について関与しない変数であり、追加命令に使用する変数として使える。

まず、ある変数 V が代入されることを V_w と表し、参照されることを V_r と表す。追加命令を挿入する場所は、次の通りである。

1. 追加命令を挿入する前に、ある変数 V_w が、正しい命令で代入された場所をマーク１とする。
2. プログラムを実行方向とは逆にトレース[3]し、最初に参照される V_r をマーク２とする。
3. マーク１とマーク２の間に代入できる変数 V_w を使用した命令を挿入することができる。この命令の演算に使用する変数は、他のプログラム変数を使用する。命令の結果は変数に代入されるが、いずれ正しい命令により適切な値が代入されるので、演算は何でも良い。

以上のことは、追加命令の変数 V_w に代入がなされたあとに、正しい命令の適切な値が、この変数に代入されるので、プログラムは正しく動作する。

次の変数の集合があるとする。主プログラムの変数を V_1 V_2 V_3 とし、ダミープログラムの変数を X_1 X_2 及び、ダミー変数を D_1 とする。例えば、次のプログラムに、変数 V_1 に対して追加命令を挿入したいとき、以下のようにする。

- 挿入前

$V_1 = V_2 + V_3$
$V_2 = V_2 * V_3$
$V_3 = V_1 + V_3$
$V_1 = V_1 + V_2$

- 挿入後

$V_1 = V_2 + V_3$
$V_2 = V_2 * V_3$
$V_3 = V_1 + V_3$
$V_1 = X_1 + X_2$ 追加命令挿入
$D_1 = V_1 + X_2$ 追加命令挿入
$V_1 = V_1 + V_2$

変更後のプログラムのコードで

$V_1 = X_1 + X_2$
$D_1 = V_1 + X_2$

が挿入されており、この命令はダミープログラム

[3]実行の流れとは逆向きになる。

の変数である X_1,X_2 及び、ダミー変数である D_1 が使用されている。解析者は、この命令の理解を深めるために、これらの変数を用いた命令が、プログラムの中のどういうところで使用されているかを調べるはずである。そこで、変数がプログラムのどこで代入されているかを調べようとするだろう。代入されている場所がわかれば、そこの命令を調べ、どのような変数を参照して命令を構成しているかを調べる。参照されている変数があるならば、その変数を調べ、どこの命令で代入されているかを調べる。これらを繰り返すことにより、解析に必要な情報を得る。このような情報から、その変数がどのような役割を割り当てられているのかが、理解できるかも知れないからである。このことから、「ある変数に関して、その変数を調べるまでにどれだけの時間がかかるか」が、追加命令を利用しての難読化の強さになる。

4.2.2 追加命令を利用しての難読化

追加命令を使用すると、変数が媒介となり、主プログラムとダミープログラムの結び付きが強くなることがわかった。これを利用して、より多くの変数の追加命令を利用してプログラムを複雑にすることができる。

5 評価

提案方式では、条件分岐と追加命令を使用した難読化を提案した。難読化の評価は次のように考えられる。

- 条件分岐での難読化は、命令ノードがたくさんあるほど、プログラムは解析しにくくなる。
- 追加命令の難読化では、追加命令の個数が多いほど、プログラムを解析しにくくなる。
- あまり多くのプログラムを挿入させると、プログラムの性能が低下する。本提案手法に関しては、プログラムの性能と難読化はトレードオフの関係にある。

参考文献

[1] 村山 隆徳, 満保 雅浩, 岡本 栄司 ,"ソフトウエアの難読化について," 電子情報通信学会技術研究報告、ISEC95-25, pp.9-14, Nov,1995

[2] 門田 暁人, 高田 義広, 鳥居 宏次 ,"ループを含むプログラムを難読化する方法の提案," 電子情報通信学会論文誌 Vol.J80-D-I No.7 pp.644-652, Jul 1997

[3] 門田 暁人, 高田 義広, 鳥居 宏次 ,"プログラムの難読化法の提案," 情報処理学会第 51 回全国大会講演論文集 5G-7,4-263,1995

[4] 鴨志田 昭輝, 松本 勉, 井上 信吾, "耐タンパーソフトウェアの構成手法に関する考察,"ISEC97-59, pp.69-78,Dec 1997

[5] T. Murayama, M. Mambo and E. Okamoto, "A Tentative Approach to Constructing Tamper-Resistant Software," Proc. of New Security Paradigms'97, Sep,1997

[6] 下村 隆夫, "プログラムスライシング技術と応用," 共立出版,1995