拡張 opaque 述語の応用
− 難読化ツールの実装の検討と拡張 opaque 多値述語 −

2004MT013 福岡 俊和　　2004MT036 伊藤 雄一　　2004MT087 坂戸 雄輝
指導教員　真野 芳久

1　はじめに

現在、ソースコードの盗用、ソフトウェアの解析や改ざんといった様々な海賊行為が問題となっている。ソフトウェアを保護するためにソフトウェアプロテクションの必要性が高まっている。本研究では、難読化ツールによってソフトウェアを保護することを目的とした拡張 opaque 述語を用いた難読化ツールの作成の検討を行った。さらに、拡張 opaque 多値述語を用いた新しい難読化方法について提案を行う。拡張 opaque 多値述語と拡張 opaque 述語を組み合わせることで得られる難読化効果は非常に大きいと考えられる。

2　ソフトウェアプロテクション

2.1　ソフトウェアプロテクション

ソフトウェア盗用などからソフトウェアの著作権を保護するためにソフトウェアプロテクションという技術が存在する。代表的な例として、電子透かし、難読化、指紋 (fingerprint)、バースマークといった技術が存在する。

2.2　難読化

攻撃者にプログラムを解読されないように、プログラムを意図的に読みにくくする技術のことである。これにより、プログラム中のアルゴリズムやデータ構造が知られるのを防ぐことができる。大規模プログラムになるほどプログラム全体を読みにくくする余地があり、攻撃者が解析するコストをプログラムの価値と比べて高くすることで事実上解析を不可能にする。

2.3　難読化の分類

難読化には、対象とする情報の種類によって、レイアウト難読化、データ難読化、制御難読化に分類される [2]。制御難読化には、制御構造を隠す方法と制御構造を複雑にする方法があり、opaque 述語が後者の例である。opaque 述語とは、プログラム作成者側はあらかじめ真理値を判定できるが、攻撃者から見た場合、一見どのような真理値をとるかわからない述語である。本研究で扱う拡張 opaque 述語、拡張 opaque 多値述語を用いた難読化は、プログラムの制御構造を複雑にする制御難読化に分類される。

3　拡張 opaque 述語

3.1　拡張 opaque 述語の定義

述語 P が拡張 opaque 述語であるとは、実行開始から i 回目の評価値 P_i が作成者側にあらかじめ知られているものをいう。

難読化に対しての利用しやすさを考えて、T(真)、F(偽) の有限列の繰返しの形で記述できる列のみを考える。

例えば、T、F、T、F、…と変化する拡張 opaque 述語は $P^{(TF)*}$、T、T、F、T、T、F、…と変化する拡張 opaque 述語は $P^{(TTF)*}$ と記述する [1]。

3.2　拡張 opaque 述語の特徴

拡張 opaque 述語の特徴は、真理値である T、F は常に一定ではないが値のとりかたの規則を作成者側は知っているということである。また、1 つの真理値だけを利用する opaque 述語と比べて、拡張 opaque 述語は真理値の T、F を組み合わせて使うので、バリエーションが豊富で種類は無数に存在する。

3.3　拡張 opaque 述語の作成例

拡張 opaque 述語の構成方法を [1] から引用する。L_i を T または F の論理値とし、拡張 opaque 述語 $P^{(L_0L_1\cdots L_{n-1})*}$ を構成すると考える。利用する変数または変数群を v とする。v のとる値集合 V の互いに素な部分集合 V_0、V_1、…、V_{n-1} があり関数 $f : V \to \{T, F\}$ と関数 $g : V \to V$ が次の条件を満たすとする。

- $\forall i\ \forall x \in V_i\ f(x) = L_i$
- $\forall i\ \forall x \in V_i\ g(x) \in V_{(i+1)\ mod\ n}$

この条件を満たす場合、v の初期設定 $v \leftarrow x$ (ただし $x \in V_{n-1}$) が実行開始直後に一度だけ実行されるとして、拡張 opaque 述語 $P^{(L_0L_1\cdots L_{n-1})*}$ は “ $v \leftarrow g(v)$ を実行した後の $f(v)$ ” として書くことができる。

3.3.1　拡張 opaque 述語 $P^{(TF)*}$ の作成例

整数の基本的な性質を利用する単純な例を [1] から引用して述べる。V を整数集合、使用変数を n とし、n の初期値は任意の偶数とする。

- g を “ n に任意の奇数を加える ”
- f を “ n が奇数 (偶数) であれば真 (偽) ”

とすることで、拡張 opaque 述語 $P^{(TF)*}$ を作成することができる。

この作成例は単純ではあるが、実行効率への影響は少なく整数を多用するプログラムにおいては自然な演算だけからなるといえる。

3.3.2　拡張 opaque 述語 $P^{(TTF)*}$ の作成例

拡張 opaque 述語 $P^{(TTF)*}$ の作成例を以下に示す。m と n の 2 つの整数型変数を用い、初期値の設定は $m = 2$、$n = -1$ とする。

- g を “ m を 2 倍して n^2 で引く、n に 1 を加える ”
- f を “ m の 3 の剰余は 0 ”

とする。

この拡張 opaque 述語 $P^{(TTF)*}$ の成立を見破るにはかなりの労力が必要とされると思われる。しかし、計算

に乗除の計算が組み込まれているので、計算効率の点に不安が考えられる。

またこの拡張 opaque 述語 $P^{(TTF)*}$ の作成によって得られた数値を利用して更なる計算をすると拡張 opaque 述語 $Q^{(TFF)*}$ が得られる。整数型変数 a と、前述の m と n の差を使用する。

- g を“ a を $m-n$ ”
- f を“ a の 3 の剰余は 0 以外 ”

とする。これによって拡張 opaque 述語 $Q^{(TFF)*}$ が作成される。複雑な計算結果を利用しているので簡単には見破ることはできず、2 つの拡張 opaque 述語の組合せによって得られる難読化の効果は大きいだろう。

4 拡張 opaque 述語を用いた難読化

4.1 難読化のルールの定義

本研究で扱う難読化ルールとは、拡張 opaque 述語を挿入する際に定める制御フローのことである。以降では、S_i は文または文の列を示し、S を同等でより複雑な制御フローに変換する。

難読化ルールは、与えられた分岐であるどの真理値 T、F も必ず一度は選択をし、フローチャート上の進路を必ず一度は通過をし、S を一回のみ実行するという点が挙げられる。そして最終的にフローチャートの与えられた全分岐を通過し、全進路を通過までに規定の S を実行したあとでフローチャートの過程を終了する。再びフローチャートを進行するにあたり動作性を崩さないものであるということが条件である。

難読化ツールに拡張 opaque 述語を組み込んでいくときに、この難読化ルールを元に作業を進めていくので大変重要な役割をもっている。

4.2 拡張 opaque 述語を用いた難読化ルールの例

図 1 は、拡張 opaque 述語 $P^{(TF)*}$ を使った最も簡単な難読化ルールである。

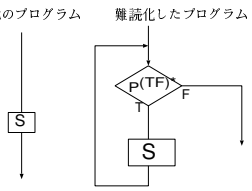

図 1　拡張 opaque 述語 $P^{(TF)*}$ を使った難読化ルール

難読化ルールの定義で挙げた通り、全ての分岐を進行する。ここでは T と F を一回ずつ選択し、S を一度のみ実行してフローチャートの進行を終了している。

次に、図 1 の拡張 opaque 述語 $P^{(TF)*}$ を複数個組み合わせた難読化ルールを示す。拡張 opaque 述語 $P^{(TF)*}$ をどれだけ組み合わせても難読化ルールが作成でき、さらに別の拡張 opaque 述語からの分岐から出現する進路が入れ子の形となりスパゲティプログラムとなるような拡張 opaque 述語の作成ができた。

図 2 は拡張 opaque 述語 $P^{(TF)*}$ を 3 個組み合わせたものである。この拡張 opaque 述語の組合せには S_1 から S_5 まで埋め込むことが可能となる。

我々が発見したこの種の難読化ルールは拡張 opaque 述語の数を増やしていき検証していった結果、拡張 opaque 述語 $P^{(TF)*}$ の数を n とすると、S_1 から $S_{(n\times 2-1)}$ 個まで挿入することが可能である。

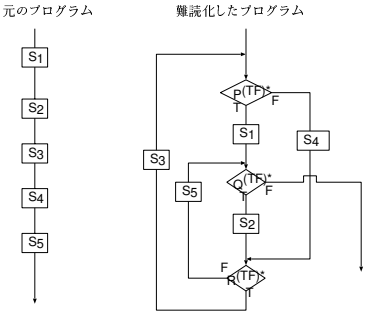

図 2　拡張 opaque 述語 $P^{(TF)*}$ を 3 個組み合わせた難読化ルール

5 拡張 opaque 多値述語

前章にて述べた拡張 opaque 述語をさらに拡張することが可能となる拡張 opaque 多値述語を提案する。

5.1 拡張 opaque 多値述語の定義

述語 P が拡張 opaque 多値述語であるとは、実行開始から i 回目の評価値 P_i が作成者側にあらかじめ知られているもので、定数の値が 3 つ以上のときを言う。その値は a、b、c、…とアルファベットで表す。

例えば、値が a、b、c、a、b、c、…と変化する拡張 opaque 多値述語は $P^{(abc)*}$、a、b、a、c、a、b、a、c、…と変化する拡張 opaque 述語は $P^{(abac)*}$ と記述する。

5.2 拡張 opaque 多値述語の特徴

拡張 opaque 多値述語の特徴は、値 a、b、c、…のとりかたは常に一定ではないが、値のとりかたの規則を作成者側は知っているということである。また、真理値 T、F だけを利用する拡張 opaque 述語と比べて、拡張 opaque 多値述語は定数の個数の上限は特に無いので、バリエーションが豊富で種類は無数に存在する。

5.3 拡張 opaque 多値述語の作成例

拡張 opaque 多値述語 $P^{(abcd)*}$ を作成するにあたって、整数型変数 m、n を使用する。初期値を $m=1$、$n=0$ とする。定数の対応付けは、m を 4 で割った時の剰余を利用して、剰余が 2 のとき値は a、剰余が 1 のとき値は b、剰余が 0 のとき値は c、剰余が 3 のとき値は d とする。

- g を“ $m=2m+n$ を計算して、n に 1 を加える。m が 1 万の値を越えたとき、m を 2、n を 1 として剰余を求める。”
- f を“ m を 4 で割ったときの剰余。”

とする。これによって拡張 opaque 多値述語 $P^{(abcd)*}$ が作成できる。

5.4 拡張 opaque 多値述語を用いた難読化

図 3 は拡張 opaque 多値述語 $P^{(abcd)*}$ によって作成した最も単純な難読化ルールである。1 つの拡張 opaque 多値述語を挿入することによって、このような難読化ルールを得られることは拡張 opaque 述語からの大きな進化と言えるだろう。5.3 節の拡張 opaque 多値述語と図 3 の難読化ルールを用いて C プログラムを難読化した例を例 1 に示す。例 1(a) は角谷問題と呼ばれるプログラムであり、難読化した結果が例 1(b) である。機械的な攻撃の耐性については、拡張 opaque 多値述語の値の取り方の規則を判断しなければならなく、解析が困難になったといえる。また、人間の攻撃の耐性については、条件分岐が増えたことで、複雑になっているので混乱させることができる。

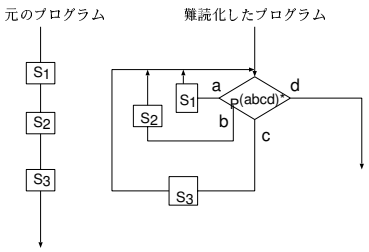

図 3　拡張 opaque 多値述語 $P^{(abcd)*}$ を使った難読化ルール

```
(a)  main(){
       int x;
       for(;;){
         printf("x ?");
         scanf ("%d",&x);
         if(x<=1)
           break;
         while(x>1){
           if(x%2)
             x = 3*x + 1;            /* S1 */
           printf("x1=%d\n",x);      /* S2 */
           x /= 2;                   /* S3 */
           printf("x2=%d\n",x);
         }
       }
     }
(b)  main(){
       int a, x, m=1, n=0;
       for( ; ; ){
         printf("x ?");
         scanf("%d",&x);
         if(x<=1)
           break;
         while(x > 1){
           for( ; ; ){
             m = 2*m + n;
             n=n++;
             a = m % 4;
             if(a==2){
               if(x%2)
                 x = 3*x + 1;        /* S1 */
             }
             else if(a==0){
               x /= 2;               /* S3 */
               printf("x2=%d\n",x);
             else if(a==1)
               printf("x1=%d\n",x)   /* S2 */
             else
               break;
           }
         }
       }
     }
```

例 1 角谷の問題とその難読化

6 拡張 opaque 述語を用いた難読化ツールの設計

6.1 難読化ツールの機能

難読化ツールに期待される機能は、新しい拡張 opaque 述語による難読化が容易に追加できるということである。難読化ルールと拡張 opaque 述語を利用者が変更し、難読化ツールへ加えることで新しい拡張 opaque 述語が利用できるという難読化ツールの設計を行う。難読化の複雑さの基準を設けることは困難であるので、ユーザ指定にすることで解決した。

以降、『プログラム』と記述した場合は、難読化の対象となるプログラムのことを表わし、『難読化ツールのプログラム』は難読化ツールを実現するプログラムとする。

6.2 難読化ツールの内部構造

6.2.1 データベース

難読化ツールで拡張 opaque 述語を挿入するときに難読化ルールと拡張 opaque 述語が必要になる。それぞれのデータベースを用意し必要な難読化ルールと拡張 opaque 述語を選択できるようにする。

難読化ルールのデータベースは、使用する S の長さで分類を行なう。S の長さが多いほど複雑なルールを使用するものに分類される。また、拡張 opaque 述語のデータベースは、難読化するプログラムに最も適した拡張 opaque 述語を挿入するために、整数プログラム、実数プログラム、テキストプログラム、システムプログラムに分類をし、さらに拡張 opaque 述語の種類によって分類をする。

6.2.2 内部構造

難読化ツールの構成要素を図 4 に示す。始めに、入力されたプログラムの管理をクラスファイルの管理部分で行なう。次に、クラスファイルの解析部分では、プログラム開発者の S の長さの入力により、難読化を挿入する部分を解析し決定する。拡張 opaque 述語挿入部分では、クラスファイルの解析部分によって得られた情報により難読化ルールデータベースからルールを取り出す。取り出した拡張 opaque 述語を挿入し、難読化が完了する。

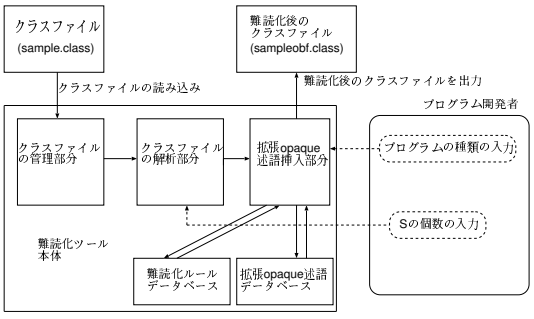

図 4　難読化ツールの内部構造

6.2.3 クラスファイル解析部分

入力された難読化対象プログラムの解析を行ない拡張 opaque 述語を挿入する部分を決定する。難読化の前提として拡張 opaque 述語を用いた難読化の挿入は各メソッドごとに 1 つとする。

入力されたプログラムをメソッド単位に分割して解析を行なう。さらに取り出したメソッドを 1 つの命令の単位まで分割してリストを作成し、リストの中の命令に対して入力された長さに対応する S を選択する。

S にする文を選択する条件として、プログラムの実行効率を極端に落さないために二重以上のループの内側の命令は S に用いないなどの制限を行なう。

6.2.4 拡張 opaque 述語挿入部分

拡張 opaque 述語をプログラムへ挿入するために、データベースからの取り出しを行なう。難読化ルールの呼び出しでは、S の長さごとに分類したルールの中からランダムに選択を行なう。また、拡張 opaque 述語のデータベースからの取り出しでは、入力されたプログラムの種類、選択されたルールの情報より対応した拡張 opaque 述語が選択される。取り出しの様子を図 5 に示す。

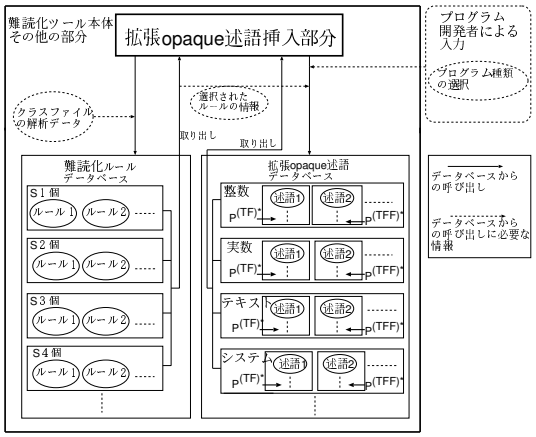

図 5　データベースからの取り出しの様子

6.3 データベースでの記述形式

難読化ルールで用いるフローチャートの流れを難読化ツールに理解させることは困難である。そこで難読化ルールを文字列で記述することでツールに理解させる必要がある。実際のデータベース内ではバイトコードにして拡張 opaque 述語挿入部分で使用する。

7 SandMark への実装の検討

本章では、前章で述べた難読化ツールの SandMark[3] への実装について検討する。拡張 opaque 述語を用いた難読化アルゴリズムを実装するための主なメソッドとして、eopObfuscate、insertEOP、ruleDataBase、eopDataBase を用意する。

図 6 は上記のメソッドの関係を表わしたものである。eopObfuscate は対象プログラムをメソッドごとに分析し、拡張 opaque 述語の挿入を指示するメソッドである。またユーザとのインタフェースの役割も含んでいて、挿入するルールや拡張 opaque 述語をユーザに指定させる。insertEOP は eopObfuscate によって呼び出され、実際に拡張 opaque 述語を挿入するメソッドである。BasicBlock(bb) と valueType を引数として valueType に従い拡張 opaque 述語を挿入した BasicBlock(bbeop) を eopObfuscate に返す。ruleDataBase、eopDataBase はそれぞれルール、拡張 opaque 述語を管理するメソッドで、insertEOP によって呼び出される。ruleDataBase は bb、valueType を引数とし、rule を insertEOP に返す。eopDataBase は valueType、rule を引数とし、eop が返され拡張 opaque 述語が決定される。

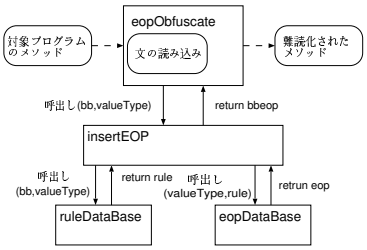

図 6　難読化アルゴリズムの概略図

上記のメソッドを補助する多くのメソッドが必要であるが、主に 4 つのメソッドを用意することで SandMark へ拡張 opaque 述語を用いた難読化アルゴリズムを実装することができると考えられる。

8 おわりに

難読化ツール作成に必要な個々の拡張 opaque 述語を用意することで、種々のプログラムに対応できるツールの設計を行なうことができた。SandMark の構造の解析は、難読化ツールの設計を検討する上で役に立った。また、提案した拡張 opaque 多値述語によって、1 つの述語でより大きな難読化効果を得ることができた。

今後の課題は、プログラムの分類をより厳密にし、対応した拡張 opaque 述語のデータベースを用意することと、難読化ツールの実装を実現することが考えられる。

参考文献

[1] 真野：拡張 opaque 述語：その応用と構成法, 通信学会論文誌 D, Vol.J90-D, No.3, pp.971-974 (2007.3).

[2] C.Collberg et.al：A Taxonomy of Obfuscating Transformations, TR 148, Department of CS, Univ of Auckland (July 1997).

[3] C.Collberg et.al：SandMark - A Tool for Software Protection Research, IEEE Security and Privacy Vol.1, No.4 (Aug.2003).

[4] C.Collberg et.al：Manufacturing Cheap, Resilient, and Stealthy Opaque Constructs, ACM POPL'98 pp.184-196 (Jan.1998).