社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2003－CSEC－23 (15)
2003／12／19

Javaクラスファイルの等価変換に耐性を持つプログラム指紋抽出法

福島 和英† 田端 利宏‡ 櫻井 幸一‡

†九州大学大学院システム情報科学府
812-8581 福岡市東区箱崎 6-10-1
fukusima@itslab.csce.kyushu-u.ac.jp

‡九州大学大学院システム情報科学研究院
812-8581 福岡市東区箱崎 6-10-1
{tabata,sakurai}@csce.kyushu-u.ac.jp

Javaクラスファイルの盗用を検知することを目的として，Javaクラスファイルからプログラム指紋を抽出する手法について検討を行う．玉田らによって，クラスファイルの初期値代入部分，メソッド呼び出し部分，および，継承関係を透かしとして抽出する手法が提案されている．本論文では，初期値代入部分と継承関係によるプログラム指紋は容易に改ざんできることを示す．さらに，Javaクラスファイルの各メソッドをオートマトンにより抽象的に表現し，プログラム指紋として抽出する方法を提案する．

Program Birthmark Scheme with Tolerance to Equivalent Conversion of Java Classfiles

Kazuhide FUKUSHIMA† Toshihiro TABATA ‡ Kouichi SAKURAI‡

†Graduate School of Information Science and Electrical Engineering, Kyushu University
6-10-1 Hakozaki, Higashi-ku, Fukuoka, 812-8581 Japan
fukusima@itslab.csce.kyushu-u.ac.jp

‡Faculty of Information Science and Electrical Engineering, Kyushu University
6-10-1 Hakozaki, Higashi-ku, Fukuoka, 812-8581 Japan
{tabata,sakurai}@csce.kyushu-u.ac.jp

Abstract This paper examines program birthmark scheme for Java in order to detect illegal thefts of Java classfile. Tamada et al. proposed birthmark scheme based on initial value assignments, the sequence of method call, and the inheritance structure. We show that the birthmarks based on initial value assignments and the inheritance structure can be modified easily. In addition, we propose a birthmark scheme using automata.

1 はじめに

近年，Javaが広く普及している．携帯電話や小型情報端末上で実行できるという高い移植性やインターネットへの高い親和性が大きな要因である．しかし，Javaは著作権保護を特に必要とする．Javaプログラムの一般的な流通形態であるクラスファイルは，ソフトウェアの構造を知る手がかりとなる多くの情報を含む．このため，攻撃者は逆コンパイラなどのツールにより，クラスファイルから元のソースコードに近いソースコードを復元できる．また，クラスファイルの盗用も大きな問題である．一般に，クラスファイルはソフトウェアにおける特定の機能を実現している．このため，クラスファイルは他のプログラムの構成部品として利用できる．攻撃者は盗用したクラスファイルを利用した新しいプログラムを作成し，そのプログラムの所有権を主張することがあり得る．

本論文ではプログラムの盗用を検知することを目的として，Javaクラスファイルからプログラム指紋を抽出し，識別を行う方法を提案する．提案手法は，クラスファイル内の各メソッドを静的に解析し，抽象度の高いオートマトンをプログラム指紋として抽出する．

2 関連研究

ソフトウェアの著作権を保護するための手法として，現在までにさまざまな手法が提案されている．

ソフトウェア難読化[1, 2, 5]は，ソフトウェアを機能を保ちつつ，解読が難しいものに変換することである．難読化によって，攻撃者が，ソフトウェアから鍵となるアルゴリズムや秘密のデータを盗むことを，より難しくすることは可能である．しかし，完全にソフトウェアの解析を防ぐことは困難であり，盗用の事実を立証することも難しい．

プログラムに情報を埋め込んでおき，プログラムの盗用を立証する手法も提案されている．電子透かし[3, 4]は，ソフトウェアの作成者の識別情報をソフトウェアに埋め込む手法である．ソフトウェアが盗用された場合にも，電子透かしを抽出することができれば，作成者はその所有権を主張することができる．一方，フィンガープリンティング[6]は，ソフトウェアの各ユーザの識別情報を埋め込む手法である．ソフトウェアのコピーが不正に配布された場合にも，フィンガープリンティングを抽出することができれば，不正を行ったユーザを追跡できる．しかし，これらの手法は埋め込まれた情報が除去，あるいは改ざんされてしまう可能性がある．

情報の埋め込みを行わずに，ソフトウェアの識別を行う方法がある．この手法はプログラム指紋[7]と呼ばれる．ここで，プログラム指紋とは，ソフトウェアに含まれる固有の特徴を指す．難読化や最適化などのプログラムの等価変換に耐性を持つプログラム指紋を抽出できれば，ソフトウェアの識別を行うことができる．

3 従来の手法の問題点

玉田ら[7]は，Javaクラスファイルからプログラム指紋を抽出する方法を提案している．提案手法は，プログラム内の特徴的な箇所である初期値代入部分，継承関係，およびメソッド呼び出し部分を抽出し，プログラム指紋として用いる．しかし，初期値代入部分，および継承関係によるプログラム指紋に対しては，容易に攻撃を行うことができる．ここで，攻撃とは，プログラムの機能を保ったまま，プログラム指紋が異なる別のプログラムに変換することを指す．以下に，各プログラム指紋の抽出方法について説明し，攻撃に対する耐性について考察を行う．

3.1 初期値代入部分

3.1.1 プログラム指紋の抽出方法

クラスのフィールドへの初期値代入部分をプログラム指紋として抽出する．プログラム指紋の取り出し対象となるフィールドはプリミティブ型とjava.lang.string型に制限する．2つのクラスファイルの初期値代入において，その型と値が一致すればプログラム指紋として一致するものとする．また，初期値が代入されないフィールドや，コンストラクタの引数によって初期値が決定されるフィールドに関しては，初期値なしと見なす．初期値が代入されているかどうかはstaticイニシャライザ，コンストラクタ，または，クラスファイルに含まれるConstantValue属性によって判定する．

3.1.2 攻撃に対する耐性

初期値代入部分を改ざんすることによって，このプログラム指紋に対しての攻撃が可能である．以下のクラスへの具体的な攻撃方法を示す．

```
public class SampleClass{
  public static int value1;
  SampleClass(){
    value1 = 3;
  }
}
```

このクラスを以下の等価なクラスに変換する．

```
public class SampleClass{
  public static int value1;
  SampleClass(){
    value1 = 1;
    value1 += 2;
  }
}
```

コンストラクタが完了した時点では，value1には，共に整数3が格納される．しかし，初期値として代入される値はことなる．前者のクラスでは，コンストラクタ内でvalue1に代入される初期値は3である．一方，後者のクラスでは，value1に代入される初期値は1である．このた

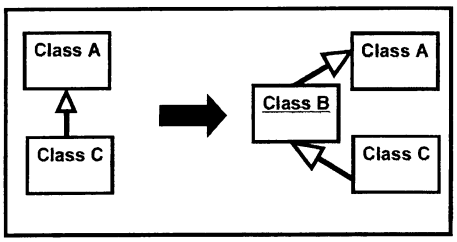

図 1: 継承関係の改ざん

め，初期値代入部分によるプログラム指紋を改ざんしたことになる．

3.2 継承関係

3.2.1 プログラム指紋の抽出方法

対象のクラスのスーパクラスを順にたどっていき，ルートまでの数とたどっていったスーパクラスがユーザ定義でないクラス名をプログラム指紋として抽出する．

3.2.2 攻撃に対する耐性

継承関係によるプログラム指紋に対する，具体的な攻撃方法を図 1 に示す．スーパクラス A を持つ，クラス C を仮定し，以下の手順でクラス C の継承関係を改ざんする．

(1) 空のクラス B を定義する．ここで，空のクラスとはフィールド，およびメソッドが存在しないクラスのことである．

(2) クラス C のスーパクラスをクラス B に設定する．

(3) クラス B のクラスをクラス A に設定する．

クラス C は元々，クラス A を直接継承していた．しかし，上記の処理により，クラス B がクラス A を継承し，さらに，クラス C がクラス B を継承することになった．このことにより，ルートからクラス C までの深さは 1 増えることになり，クラスの継承関係によるプログラム指紋を改ざんできる．

3.3 メソッド呼び出し部分

3.3.1 プログラム指紋の抽出方法

ユーザ定義でないクラスのメソッド呼び出しとその順番をプログラム指紋として抽出する．一般的に，クラスには複数のメソッドが存在するので，このプログラム指紋は全てのメソッドから抽出する．

3.3.2 攻撃に対する耐性

ユーザが定義していないメソッドを変更するためには，標準 API クラス，および，標準拡張 API クラスを直接変更する必要がある．これらのクラスの中には，Java が動くために必要となるクラスや，各ユーザが必要に応じて，個別に入手しているクラスがある．このため，これらのクラスをユーザが直接変更すると，プログラム自体が動かなくなる可能性がある．このため，ユーザ定義でないメソッドのメソッド呼び出しはプログラム指紋として適切であると考えられる．

4 提案手法

玉田らの手法のうち，初期値代入部分，および継承関係によるプログラム指紋に対する攻撃は簡単に行える．また，初期値代入部分や継承関係だけでは，本質的なプログラムの違いを指摘できない可能性がある．そこで，提案手法では，ユーザ定義の各メソッド内の命令を抽象的に表現し，プログラム指紋として抽出することを考える．

4.1 プログラム指紋の抽出方法

提案手法では，ユーザが定義したメソッドを抽象度の高い有限オートマトンとして表現する．このオートマトンがプログラム指紋となる．以下の手順に従い，メソッド内のバイトコードからオートマトンを構成する．

(1) 状態の作成
1 つのバイトコード命令に対し，その命令をラベルとして持つ頂点を作成する．この頂点はオートマトンにおける状態に対応している．このとき，何もしないバイトコード命令である nop 命令は無視する．

(2) 状態の統合 1（疑似命令の導入）
複数の命令を抽象度の高い 1 つ疑似命令に置き換える．このとき，複数の状態が 1 つの状態に統合される．ここでは，オペランドスタックに値が push され，演算が施され，演算の実行結果の値が pop されるまでの一連の命令列を 1 つの疑似命令として表す．

例えば，命令列（iload_3, istore_0）はオペランドスタックに整数 3 を積んだあ

と，局所変数 0 に格納する．このとき，2 つのバイトコードを命令を 1 つの代入命令として L0 ← 3 と表す．また，命令列（iload_3,iload_2, iadd, istore_1）はオペランドスタックに，2 と 3 が積んだ後，加算を行い，さらに，局所変数 1 に演算結果が格納する．この場合，4 つのバイトコード命令を 1 つの代入命令として L1 ← 2+3 と表す．また，2+3 は 5 であるので，L1 ← 5 と簡約化できる．

(3) 状態の統合 2（可換な命令のブロック化）
互いに可換である複数の命令を 1 つの状態に統合する．このとき，複数の状態が 1 つの状態に統合される．

例えば，命令列（L0 ← 1, L1 ← 2, L2 ← 3）は局所変数 0, 局所変数 1，局所変数 2 にそれぞれ 1, 2, 3 を代入する．この場合，3 つの命令はどの順番で実行されても，プログラムの実行に影響を与えない．このため，これらの命令は 1 つのブロックとして扱うことができる．

(4) 状態遷移の付加
各状態から次に実行される命令に対応する状態へ対する有向枝を加える．この有向枝は有限オートマトンにおける状態遷移を意味する．通常は，隣接する状態に遷移するが，分岐命令の場合，分岐先の命令に対応する状態に遷移する．条件分岐命令に対応する状態は，2 つの遷移先を持つ．そこで，この状態から外に出ている有向枝には，YES および NO のラベルを付加する．

(5) 無条件分岐の除去
無条件分岐（goto, goto_w）からなる状態を分岐先の状態と置き換える．無条件分岐の場合，分岐先の命令は一意に定まる．図 2 に変換の例を示す．この図には，goto 命令からなる状態があるが，変換後は分岐先の条件分岐命令 L5 ≦ 0?からなる状態に置き換わっている．

(6) 局所変数の汎用化
命令内で参照および代入が行われている局所変数について，その同一性にのみ着目し，汎用的に局所変数を表す記号 {A, B, C, ...} で置き換える．図 2 の例において，局所変数 3 を A，局所変数 5 を B で置き換えたものを図 3 に示す．

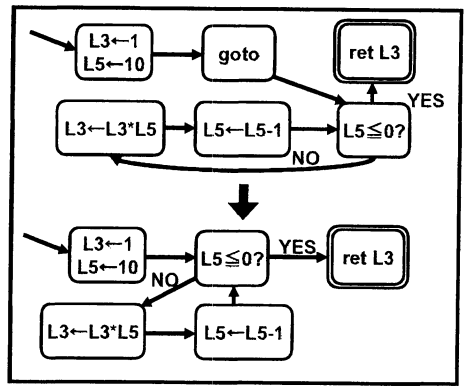

図 2: 無条件分岐の除去

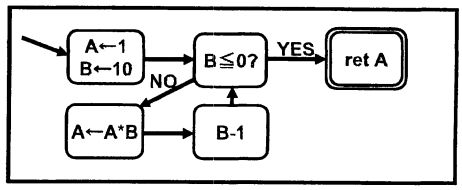

図 3: 局所変数の汎用化

4.2 プログラム指紋の抽出例

メソッドからプログラム指紋として，オートマトンを抽出する例を示す．ここでは，サンプルプログラムとして，与えられた引数 n に対し，フィボナッチ数列の第 n 項を返すメソッド long fib(long n) を対象とする．Java ソースコードによるこのメソッドの定義は図 4 の通りである．クラスをコンパイルしたクラスファイルを逆アセンブルして得られるバイトコードは図 5 のようになる．

このバイトコードからプログラム指紋として抽出されるオートマトンは図 6 に示す．局所変数 3, 局所変数 5, 局所変数 9，および，局所変数 7 をそれぞれ A, B, C, D で表している．なお，メソッドの引数は局所変数 9 に格納されている．8 バイト目には無条件分岐があるが，状態遷移図の中では，分岐先の 27 バイト目から 31 バイト目にかけて行われる条件判定命令に置き換えられている．この命令からの分岐先は 35 バイ

```
long fib(long n){
  long a,b,dummy;
  int k;
  a=1L; b=1L;
  for (k=3; k<=n; k++){
    dummy=b;
    b=a+b;
    a=dummy;
  }
  return b;
}
```

図 4: メソッド fib の定義

```
long fib(long);
 Code:
  0:  lconst_1
  1:  lstore_3
  2:  lconst_1
  3:  lstore 5
  5:  iconst_3
  6:  istore 9
  8:  goto 27
 11:  lload 5
 13:  lstore 7
 15:  lload_3
 16:  lload 5
 18:  ladd
 19:  lstore 5
 21:  lload 7
 23:  lstore_3
 24:  iinc 9, 1
 27:  iload 9
 29:  i2l
 30:  lload_1
 31:  lcmp
 32:  ifle 11
 35:  lload 5
 37:  lreturn
```

図 5: メソッド fib を構成するバイトコード

ト目，または 11 バイト目の 2 つがあるので，この状態からは YES, NO のラベルが付いた 2 つの有向枝が出ている．

5 攻撃に対する考察

プログラム指紋を用いてソフトウェアの著作権保護を行う場合，攻撃者はソフトウェアを変換することによって，プログラム指紋を改ざんしようとする．難読化は，ソフトウェアを機能を保ちつつ，別のソフトウェアに変換する手法である．このため，難読化はプログラム指紋に対する攻撃として有効であると考えられる．ここでは，

- ダミーコードの挿入
- 可換命令の入れ替え
- 無条件分岐の導入
- 命令の分解
- 使用する局所変数の入れ替え

の 5 つの難読化手法を対象とし，これらの難読化手法による攻撃に対するプログラム指紋の耐性について考察を行う．

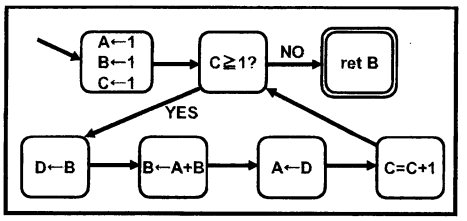

図 6: プログラム指紋として構成されたオートマトン

5.1 ダミーコードの挿入

ダミーコードの挿入については，2 つの手法が考えられる．1 つは，何もしないバイトコード命令である nop 命令を挿入する手法である．もう 1 つは，実行されることがないブロックを挿入する手法である．例えば，恒偽となる if 文を用意しておき，その中にダミー命令を記述する方法がある．

プログラム指紋の抽出方法の手順 (1) で nop 命令を無視している．このため，前者の攻撃は提案手法にとって無効である．後者の攻撃は，条件分岐の真偽を確かめる必要があり，静的解析では不可能な場合もある．

5.2 可換命令の入れ替え

可換な命令を入れ替えることにより，プログラムの機能を保つ変換が可能である．具体例としては，独立している 2 つ以上の局所変数への代入命令の入れ替えがある．命令列（iload_1, istore_0, iload_2, istore_1）を実行すると，局所変数 0 に整数 1 を，局所変数 1 に整数 2 が代入される．この命令列を（iload_2, iload_1, istore_0, istore_1）と変換しても同等な結果が得られる．

プログラム指紋の抽出方法の手順 (2) で抽象度の高い疑似命令の導入，手順 (3) で可換な命令のブロック化を行っている．上記の 2 つの命令列はともに，疑似命令を用いて {L0 ← 1,L1 ← 2} と表され，1 つの状態となる．このため，この手法によるプログラム指紋の攻撃は無効である．

5.3 無条件分岐の導入

無条件分岐を導入することによって，プログラムの制御構造を複雑にすることが可能である．具体的には，逐次実行される命令列を任意のブロックに分け，各ブロックを並び替える．さら

に，各ブロックの最後に，次に実行される命令への無条件分岐命令を付け加えることによって，プログラムの構造を複雑化できる．

プログラム指紋の抽出手法の手順(5)により，無条件分岐を除去することが可能である．このため，提案手法は，無条件分岐の導入による攻撃に対して耐性を持つと考えられる．

5.4 命令の分解

バイトコード命令を分解することにより，プログラムを等価変換することができる．具体例として，命令列（iconst_3, istore_0）は，局所変数0に整数3を代入している．この命令列を命令列（iconst_1, iconst_2, iadd, istore_0）に変換する．この命令列は，整数1と整数2の加算を行い，加算結果を局所変数0に格納しているが，結果的には局所変数0には整数3が代入されている．

プログラム指紋の抽出方法の手順(2)により，複数のバイトコード命令は，抽象的な疑似命令で置き換えられる．上記の場合は．疑似命令を用いると，上記の命令列はともに，L0←3と表すことができ，抽出されるプログラム指紋は同一のものとなる．

局所変数を介した命令の分解も考えられる．参照されていない局所変数1を用いて，命令列（iconst_1, istore_1, iload_1, iconst_2, iadd 2, istore_0）に変換する．この命令列は，局所変数1に1を代入した後，局所変数1の値と整数2の加算結果を局所変数0に格納している．このため，最終的には局所変数0には整数3が代入され，この命令列は上記の2つの命令列と等価である．しかし，この命令列を疑似命令で表すと，(L1 ← 1, L0 ← L1+2）となる．この場合は，L1=2であることは容易に分かるが，静的解析によって，ある時点での変数の値を解析することは一般的に困難である．

5.5 使用する局所変数の入れ替え

使用する変数を入れ替えることによって，クラスファイルの等価変換を行うことが可能である．命令列（iconst_1, istore_0, iconst_2, istore_1, iload_0, iload_1, isub, istore_2）は，局所変数0に整数1を，局所変数1に2を格納した後，局所変数2に，局所変数0と局所変数1の和（整数3）を格納しているが，この命令列を命令列（iconst_1,istore_1, iconst_2, istore_0, iload_1, iload_0, isub, istore_2）に変更する．この命令列は，元の命令列と比較して，局所変数0と局所変数1の役割を入れ替わっているが，局所変数2には，同じ値である整数3が代入される．

プログラム指紋の抽出方法の手順(6)により，メソッド内で用いられる局所変数は，汎用的に局所変数を表す記号で置き換えられる．上記の命令列は，(A ← 1, B ← 2, C ← A-B）と表現される．このため，抽出されるプログラム指紋は同一のものとなる．

6 まとめ

本論文では，玉田らによるプログラム指紋の抽出方法において，初期値代入部分および継承関係によるプログラム指紋を簡単に攻撃でき，メソッド呼び出しによるプログラム指紋は有効であることを示した．また，クラスファイルから抽象度の高いオートマトンをプログラム指紋として抽出する手法を提案した．さらに，難読化を用いた攻撃に対し，耐性があることを示した．

参考文献

[1] C. Collberg, C. Thomborson and D. Low. "A taxonomy of obfuscating transformation," Technical Report of Department. of Computer Science, University of Auckland, No. 148, 1997.

[2] C. Wang, J.Hill, J. Knight and D.Davidson. "Software tamper resistance: obfuscating static analysis of programs," Technical Report SC-2000-12, Department of Computer Science, University of Virginia, 2000.

[3] 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，" 信学論 D-I, Vol.J80-D-I, No.7, pp.644–652, 1997.

[4] 北川隆，楫勇一，嵩忠雄．"JAVAで記述されたプログラムに対する電子透かし法，" 1998年暗号と情報セキュリティシンポジウム (SCIS'98), 1998.

[5] 門田暁人，飯田元，松本健一，鳥居宏次，一杉裕志．"プログラムに電子透かしを挿入する一手法，" 1998年暗号と情報セキュリティシンポジウム (SCIS'98), 1998.

[6] 福島和英，櫻井幸一．"ソフトウェア透かしにおける個人識別情報埋め込み位置の難読化，" 2003年暗号と情報セキュリティシンポジウム (SCIS2003), 2003.

[7] 玉田春昭，神崎雄一郎，中村匡秀，門田暁人，松本健一．"Javaクラスファイルからプログラム指紋を抽出する方法，" 信学技報，Vol. 103, No.95, pp.127–133, 2003.