SCIS 2003 The 2003 Symposium on Cryptography and Information Security
Hamamatsu,Japan, Jan.26-29,2003
The Institute of Electronics, Information and Communication Engineers

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
Obfuscating of Embedding Position of the Software Fingerprinting

福島　和英 *　　櫻井　幸一 †
Kazuhide FUKUSHIMA　　Kouichi SAKURAI

あらまし Java プログラムの盗用を防ぐための方法として，購入者の個人識別情報をプログラム中に埋め込むことが考えられる．門田らによって，Java プログラム中に任意の文字列を挿入する方法が提案されているが，すべてのプログラムに同一の情報を埋め込むことを前提としている．そのためこの手法を用いて購入者ごとに異なる個人情報を埋め込んだ場合，複数のプログラムを比較することで埋め込み部分が直ちに判明してしまう．本研究では Java の難読化を利用することでこの問題を解決し，さらに，クラスファイルのセキュリティを高めることを試みる．

キーワード ソフトウェア透かし，個人識別情報，難読化，耐タンパーソフトウェア，Java

1　はじめに

Java のコードはマシンやオペレーティングシステムに依存しないクラスファイルにコンパイルされ，Java 仮想マシン (JVM) 又は Java インタプリタなどのソフトウェアを通じて実行される．そのため，他のプラットフォームで実行を行うときにも移植やリコンパイルなどの作業は一切不要である．しかし，この性質を実現するためにクラスファイル中には，ソースコード中のほとんどのデータが残ってしまう．そのためクラスファイルの可読性は高く，現在までに逆コンパイラをはじめとするクラスファイルの解析ツールが数多く開発されてきた．結果として，ソフトウェアは悪意ある第三者のリバースエンジニアリング (逆行解析) によりプログラム中のデータ構造やアルゴリズムを盗用されてしまう危険にさらされてしまう．この問題に対処するために難読化という手法がある [6, 7, 8, 9, 10, 11, 15]．難読化とはプログラムの機能を保ったまま，ソースコードを解析されにくいものに変換することである．難読化を用いると，プログラムの盗用，解析を防ぐことは可能である．しかし，盗用や不正コピーを行った個人を特定するのは不可能である．

また，クラスファイルは再利用性が非常に高い．このため，第三者が Java プログラム中の一部のクラスファイルを盗用し新たなプログラムを作成してしまう可能性もある．この問題に対処するための手法としてプログラムの中に開発者の署名などを透かしとして埋め込む方法がある [12, 13, 14]．仮に第三者がクラスファイルを盗用し，そのクラスファイルを用いたプログラムを配布した場合にも，そのプログラム中から電子透かしを抽出することによって，盗用の事実を立証することが可能である．しかしこの場合でも，埋め込まれる情報は開発者の識別情報であるので，盗用や不正コピーを行った個人は特定できない．

本論文では，プログラム中に個人識別情報を埋め込む手法を提案する．電子透かしを埋め込む方法としては門田らによる手法 [13, 14] を用いる．透かしとして個人識別情報を埋め込む場合は複数のプログラムを比較して透かしの挿入位置が判明してしまうという問題がある．この問題を解決するためにメソッドに着目した Java の難読化変換を行う [15]．このことにより，複数の購入者が結託した場合の透かしの位置の特定が難しくなる．さらに単一のプログラムでも透かしの位置の特定が困難となる．

2　関連研究

これまで，電子透かしといえば画像，音声に対するものだけを指す風潮があった．しかし，ソフトウェアに対する透かし方法も提案されている．Java を対象とする透かしの提案は [12, 13, 14] である．

北川ら [12] は，Java のプログラムに対して任意の数値列を透かしとして埋め込む手法を提案している．この手法では透かしを格納するための変数を用意し，その変数に透かし情報を表す数列を格納する．透かしの取り出しには特定のクラスファイルを透かし取り出し用のクラ

* 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, Graduate School of Information Science and Electrical Engineering, Kyushu University, 6-10-1 Hakozaki, Higashi-ku, Fukuoka City, Japan, fukusima@tcslab.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, Faculty of Information Science and Electrical Engineering, Kyushu University, 6-10-1 Hakozaki, Higashi-ku, Fukuoka City, Japan, sakurai@csce.kyushu-u.ac.jp

スファイルに変換する．しかしこの手法では透かしを検出しようとする際に取り出し用のクラスファイルに置き換えるべきクラスファイルが存在しない場合がある．このときは検出不可能になる．

門田ら [13, 14] は，Java のプログラムの数値オペランド部分，オペコード部分に透かしを埋め込む手法を提案している．この手法ではソースコードに対してダミーのメソッドを追加することによって透かしを埋め込むための領域を確保する．コンパイル後のクラスファイル中の数値オペランド，オペコード部分に透かし情報を埋め込む．この手法では一つのクラスファイルが盗用された場合でも透かし挿入部分を調べることにより簡単に透かしが復元できる．この手法に基づいたツール [5] も公開されている．

しかし，これらの手法は開発者の識別情報を埋め込む対象としており，購入者の個人識別情報をそのまま埋め込むことはできない．

一方で難読化に対する研究も数多く行われている．Don Quixote [1]，ZelixKlassMaster [4] などのツールが開発されており，Collberg [7]・Douglas [8] らはこれらのコンテスト，難読化ツールで用いられた手法をレイアウト難読化，データ難読化，制御フロー難読化などに分類した．しかし，これらの手法は安全性に対する理論的な根拠を持っていなかった．その後，Wang ら [9] はポインタを含むプログラムの解析の困難さを利用し，安全性の理論的根拠をもつ難読化手法を提案した．Wang らの手法では，手続き内の制御フローのみを難読化の対象としていたが，小木曽ら [10] によって手続き間の制御フローにも適用できることが示された．また，Wang らの手法はポインタを利用しており，そのままの形で Java に適用することは不可能である．そこで刑部ら [11] は，ポインタの代わりにメソッドオーバロードやインタフェースを用いることにより Wang らの手法 [9] を Java に適用することに成功した．

現在までにクラスファイルを解析するツールは数多く開発されている．特に，JAD [2] や Mocha [3] などを代表とするコンパイラの発達は 著しい．刑部らの手法 [11] は安全性の計算理論的な根拠を持っているがクラスファイル内の解析のみで解読することが可能であり，現実的な大きさのプログラムでは総当たり的な方法で解析される可能性がある．そこで，福島らはメソッドに着目しクラスファイル構造を変換を行う難読化手法 [15] を提案した．

本研究では透かし埋め込み手法として門田らの手法 [13, 14]，難読化手法として福島らの手法 [15] を適用した．

3　個人識別情報埋め込み手法

3.1　門田らによる透かし手法

個人識別情報の埋め込みには門田ら [13, 14] の手法を用いる．Java ではプログラムの実行前に検証器によるチェックが行われる．そのため透かし情報の埋め込みには注意を要する．

3.1.1　個人識別情報の挿入

手順 1　透かし挿入部の追加

コンパイル前のソースコードに対して実際には実行されないダミーのメソッドを追加する．ダミーメソッドに含まれるプログラムコードは，透かしの文字列を書き込むための領域となる．ダミーメソッドは任意のものでよいが，透かし情報を埋め込むために十分なサイズを用いる必要がある．

手順 2　コンパイル

ダミーメソッドを追加した Java ソースコードをコンパイルし，クラスファイルを作成する．

手順 3　透かしの埋め込み

クラスファイル中のダミーメソッドに対応する部分に，電子透かしとなる文字列の書き込みを行う．Java の実行形式の一つである Java アップレットが実行される前には，バイトコード検証器が，プログラムの文法や型のチェックを行う．このため，ダミーメソッド中のプログラムコードの文法，型が等しくなるようにするため，2 つの方法を用いて透かしの埋め込みを行う．

1. 数値オペランドの書き換え
   スタックに値を push するオペコードの数値オペランド部分，スタック上の値に演算を施すオペコードの数値オペランド部分は任意の数値に置き換えても文法，型の整合性が保たれる．

2. オペコードの書き換え
   スタック内の要素の足し算を行うオペコード *iadd* は，他の演算を行う他のオペコードに置き換えても，文法の整合性は保たれる．(*isub*,*imul*,*idiv*,*irem* ,*iand*,*ior*,*ixor*) に置き換えても文法，型の整合性は保たれる．よって，これらの数値演算を行う数値演算のオペコードが出現した場合は，可換な 8 つのオペコードのうちいずれかに置き換えることで 3 ビットの情報を埋め込むことができる．例えば，*iadd* を 000, *isub* を 001, *imul* を 010 に，$\cdots$,*ixor* を 111 に割り当てることによって，000∼ 111 のデータを表現できる．

   これ以外にも，互いに可換なオペコード群が存在

するので可換なオペコードの数に対応するビット数を割り当てることができる．

3.1.2 個人識別情報の抽出

透かしの取り出しにおいては，バイトコードと透かしのビット列の対応，ビット列と文字の対応を知っていなければならない．クラスファイルの各メソッドに含まれるメソッドオペランドとオペコードを対応するビット列に置き換え，さらに，ビット列を文字列へと置き換える．すると，ダミーメソッドに対応する部分に透かしが現れる．

以上の電子透かしの埋め込み，抽出を自動で行うツール jmark [5] が公開されている．

3.2 問題点

門田らの手法を個人識別情報の埋め込みに応用するにあたり以下の 2 つ問題点がある．

はじめに，透かしそのものの脆弱性があげられる．彼らの手法を用いた場合，攻撃者が透かしを挿入しているダミーメソッドを特定できれば，そのメソッドを除去したり，同じツールを用いてそのメソッドの透かしを上書きできる．攻撃者は総当たり的にメソッドを調べることによって透かし挿入部を特定することができる．この攻撃に要する時間はクラスファイル内のメソッド数に比例するが，一般的な Java のプログラムでは 1 つのクラスファイル中にあるメソッドは数個から多くとも数十個程度であることを考慮すれば，この攻撃は有効である．実際に透かしを埋め込んだクラスファイルからダミーメソッドを特定し，彼らのツールを用いて透かし情報をを他のものに改竄することは可能であった．クラスファイル埋め込まれた個人識別情報は不正者の特定に用いられるので上書きによる改竄は特に大きな問題になる．

次に，複数の購入者の結託攻撃に対する脆弱性である．開発者の識別情報を透かしとして埋め込む場合は，各購入者に配布されるプログラム (クラスファイル) はすべて同一である．しかし，購入者の個人識別情報を透かしとして埋め込む場合は透かし挿入部 (ダミーメソッド部分) が各購入者ごとに異なってしまう．このとき，複数の購入者が結託して，各クラスファイルを比較することにより，透かし挿入部分が特定できる．実際に異なる透かしが埋め込まれた 2 つのクラスファイルの差分をとり (UNIX の **cmp** コマンドを利用)，透かし挿入部分を特定することは可能であった．

これら 2 つの問題を解決するために，次章で紹介する難読化手法を重ねて適用する．

4 難読化手法

4.1 難読化手法

Java では，データ構造 (フィールド) とそれに対する操作 (メソッド) をひとまとめにして定義する (カプセル化) ことにより，クラスの外部からは内部のメソッドが行っている処理を考慮する必要がなくなる．このことを Java における抽象性と呼ぶ．提案手法ではメソッドを分散させ，抽象性を破壊することにより，クラス間の関連を分かりにくくすることができる．

以下で示す手順 1 から手順 3 までの操作によって，メソッドをクラス間に分散させるための準備を行う．さらにメソッドを分散した後には，手順 4 によって呼び出し側の修整を行う必要がある．

手順 1　カプセル化の破壊

1. フィールドの公開
   すべてのクラスのすべてのフィールドの修飾子を **public** に変更する．このことにより，すべてのプログラムのサイトからすべてのフィールドにアクセスできるようになる．

2. メソッドの公開
   すべてのクラスのすべてのメソッドの修飾子を **public static** に変更する．このことにより，すべてのメソッドはクラスメソッドとしてすべてのクラスのサイトから (クラス名).(メソッド名) の形で呼び出されるようになる．

手順 2　メソッドの変更

手順 1 のカプセル化の破壊の際に，もともとはインスタンスメソッドであったものが，クラスメソッドに変更されてしまう場合がある．そのため，メソッドの引数としてインスタンスを新たに追加する必要がある．

以下のコード 1，コード 2 に例を示す．

```
class A{
    private int a;
    public void set(int x){
        a = x;
    }
}
```

コード 1　変更前

```
class A{
    public int a;
    public static void set(int x, A Obj){
       Obj.a = x;
    }
}
```

コード 2　変更後

変更後にはメソッド set はクラス A のインスタンスである Obj を新たに引数としてとるようにする必要が

ある．更に，フィールド a への代入命令もインスタンス Obj のフィールド a への代入であることを明示的に示さなければならなくなる．

手順 3　クラス情報の除去

手順 2 のメソッド変更により，もともとインスタンスメソッドであったものは，そのクラスのインスタンスを引数として持つクラスメソッドに変換された．このままではメソッドをクラス間に分散させても，その引数を調べることによってもともと属していたクラスが直ちに分かってしまう．そこで以下の手順により，引数としてとるインスタンスの型を統一することができる．

1. **継承関係を持たないクラス同士の場合**

   1. 他のクラスとの継承関係を持たない $C_1, \cdots, C_n$ を選び出す．
   2. 新たなクラス C_0 を宣言する．
   3. クラス C_0 のフィールドは，クラス $C_1, \cdots C_n$ に含まれるすべてのフィールドを集めたものとする．
   4. クラス C_0 のメソッドは必要はないが，後の操作で分散したメソッドを置くことは可能である．
   5. クラス $C_1, \cdots, C_n$ をクラス C_0 のサブクラスにする．すなわち，クラス $C_1, \cdots, C_n$ の宣言の後に **extends** C_0 を付け加える．
   6. クラス $C_1, \cdots, C_n$ の中のメソッドの引数のインスタンスをすべて C_0 に変更する．

図 1: 継承関係を持たない場合

クラス構造は図 1 のようになる．クラス C_0 に全てのデータ構造が集められ，クラス $C_1, C_2, \ldots, C_n$ はクラス C_0 のサブクラスになる．

2. **継承関係を持つクラス同士の場合**

2 つのクラスの場合，上位クラスを C とし，サブクラスを C′ とする．この場合はクラス C，クラス C’ のメソッドの引数のインスタンスを C′ に統一することができる．クラスが複数の場合にも同様にして，インスタンスを最も下位のクラスに統一することができる．

手順 1 から手順 3 までの操作によってすべてのメソッドはどこからでも呼び出すことが可能なクラスメソッドに変更される．これにより，全てのメソッドは任意のクラスに配置可能である．

手順 4　呼び出し側の修整

手順 1 から手順 3 までのメソッドの変更および，メソッドをクラス間に分散したことにより，メソッドの呼び出し側でも修整を行う必要がある．

例えば，クラス A のインスタンスメソッド setA(int) はクラス A のインスタンス obj_A を介して，obj_A.setA(3) の形式で呼び出されていたはずである．

しかし，難読化を適用することによって，引数としてクラスのインスタンスが加わり，他のクラス (例えばクラス B) のクラスメソッドに変更される．ゆえに，B.set(3,Obj_A) のように呼び出しを修整する必要がある．ここで，B はメソッド分散時に setA(int) を配置したクラスを指し，2 つ目の引数である Obj_A はメソッド setA を適用する対象を示している．

図 2: 難読化前のクラス構造

図 3: 難読化後のクラス構造

図 4: 提案手法

5 提案手法

5.1 手順

以下の手順に従い，Java のプログラムに個人情報を埋め込む．

1. ソースコードのいずれかのクラス内にダミーのメソッドを挿入する．
2. ソースコードをコンパイルし，クラスファイルを得る．
3. ダミーのメソッドが存在するクラスに対して 3 章の手法 [13, 14] を用いて個人識別情報を挿入する．
4. 全てのクラスファイルに対して，4 章のクラスファイル変換による難読化手法 [15] を施す．
5. 得られたクラスファイルを購入者に配布する．

プログラムから透かしを抽出する場合は以下のようにする．

1. 透かしを挿入したダミーメソッドを含むクラスファイルを探す．
2. 見つかったクラスファイルに対して透かし復元の作業を行う．

5.2 提案手法に対する考察

提案手法で個人識別情報を埋め込んだプログラムに対する考察を行う．

個人識別情報を埋め込むダミーのメソッドは実行されないので実行時間，に影響を与えない．また，重ねて行う難読化手法もメソッドの配置する位置のみを変換する手法なので実行時間はほとんど増大しないと考えられる．

メソッドの分散によるクラスファイルの変換を行うことには以下の 5 つの利点がある．

1. クラスファイルの盗用防止
   難読化変換を行う前のクラスファイルにはデータ構造とそれを操作するアルゴリズムが含まれる．このため，他の購入者は盗用したクラスファイルを別のプログラムの部品として用いることができる．しかし，クラスファイルの変換を実行すると，データとアルゴリズムが分離されるため，各クラスファイルを部品として用いることができなくなる．ゆえに提案手法はクラスファイルの盗用に対しても有効な対策であると考えられる．
2. クラスファイルの可読性の低下
   変換によりデータ構造とアルゴリズムが分離されているので，プログラムの可読性が著しく低下する．そのため悪意あるユーザの逆行解析を防ぐことができる．
3. ダミーメソッドの削減
   変換前は 1 つのクラスファイル毎に 1 つのダミーメソッドを埋め込む必要があった．しかしクラスファイル毎の盗用を防止することができるので，プログラム毎に 1 つの透かしを挿入すれば十分になる．
4. 透かしの位置が特定が困難に
   門田らの手法では各クラスファイルに 1 つの透かし挿入部 (ダミーメソッド) があった．しかし変換後には，上記のようにプログラム毎に 1 つの透かしで十分になった．そのため，全てのクラスファイルにある全てのメソッドから 1 つのダミーメソッドを見つける必要がある．
5. 複数のクラスファイルの比較による攻撃が困難に
   変換を行う前は，複数の異なる購入者のクラスファイルを比較することによってダミーのメソッド (透かし挿入部) の位置を特定することができた．しか

表 1: 門田らの手法を用いて個人識別情報を埋め込んだ場合 (難読化前) と難読化を重ねて適用した場合 (提案手法) の比較

	透かし位置特定のため調べるべきメソッド数	購入者の結託による攻撃	必要な透かし数	クラスファイルの解析	クラスファイルの盗用
難読化前	1 つのクラス中のメソッド	可能	クラスファイル数	容易	可能
難読化後	全クラス中のメソッド	不可能	1 つ	困難	不可能

し，変換後はメソッドが複数のクラスファイル間に分散することによって，比較による透かしの位置特定が困難になる．

3.2 章で提起した 2 つの問題，透かしそのものの脆弱性と複数の購入者の結託に対する脆弱性はそれぞれ上記の 4，5 の性質により解決することが可能である．以上の結果を表 1 にまとめる．

6 まとめ

Java プログラムに対して，購入者の個人識別情報を挿入する方法を提案した．開発者の識別情報を挿入する場合と異なり，複数の購入者の結託による攻撃の可能性のため既存の電子透かしの手法をそのまま適用することは困難であった．しかし，クラスファイル変換による難読化を重ねて適用することによりこの問題を解決できただけでなく，さらにクラスファイルのセキュリティを高めることができた．

今後の課題は，提案手法の実装，埋め込んだ透かしに対する耐性を高めることである．

謝辞

本研究は一部，21 世紀 COE プログラム “システム情報科学での社会基盤システム形成” の支援を受けている．

参考文献

[1] Don quixote
. http://www.oikaze.com/tamada/Products/DonQuiote.

[2] Jad - the fast java decompiler
. http://kpdus.tripod.com/jad.html.

[3] Mocha, the java decompiler
. http://www.brouhaha.com/eric/computers/mocha.html.

[4] Zelix klassmaster
. http://www.zeli.com/klassmaster.

[5] jmark: A lightweight tool for watermarking Java class files
http://se.aist-nara.ac.jp/jmark/

[6] R. Chatterjee, B. G. Ryder, and W. A. Landi. “Complexity of point-to analysis of java in the presence of exceptions”. *IEEE Transactions on Software Engineering*, No. 27, pp. 481–512, 2001.

[7] Christian Collberg, Clark Thomborson, and Douglas Low. “A taxonomy of obfuscating transformations”. Technical Report 48, Department of Computer Science, University of Auckland, 1997.

[8] Douglas Low. “Java control flow obfuscation”. Master’s thesis, University of Auckland.

[9] Chenxi Wang, Jonathan Hill, and Jack Davidson. ‘Software tampler resistance: Obstructing static analysis of programs”. Technical report, Department of Computer Science.

[10] 小木曽俊夫, 刑部裕介, 双紙正和, 宮地充子. “手続き間の呼出関係に着目した難読化手法の提案とその評価”. 2002 年暗号と情報セキュリティシンポジウム, SCIS2002.

[11] 刑部裕介, 双紙正和, 宮地充子. “オブジェクト指向言語の難読化の提案”. Technical report, 電子情報通信学会, 2002.

[12] 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

[13] 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

[14] 門田暁人，飯田元，松本健一，鳥居宏次 “Java プログラムを対象とする電子透かし法”. 日本ソフトウェア科学会第 16 回大会論文集, pp.253-256, Sep. 1999

[15] 福島和英，櫻井幸一 “メソッド分散による Java 言語の難読化手法の提案”. コンピュータセキュリティシンポジウム 2002, Sep. 2002