# ソフトウェアに対する意図的バースマーク

M2007MM025 志村敏彦
指導教員：真野芳久

## 1　はじめに

現在プログラムの盗用が大きな問題となっている。プログラムの盗用防止技術として電子透かし、バースマーク、難読化など様々な技術がある。

本研究では、その中からバースマークの弱点である著作権主張力の弱さの改善を目標に、意図的バースマークを提案する。

## 2　バースマーク

バースマークとは、自然に書かれたプログラムが持つ特徴のことであり、盗用が疑われるプログラムと作成元のプログラムのバースマークを取り出し比較することで盗用証明の 1 つの手掛かりにできる。プログラムのどのような情報に注目するか、その情報をどう扱うかによって様々なバースマークの種類がある。

バースマークの利点として、電子透かしと異なりあらかじめ情報を埋め込んでおく必要がない点がある。新たに提案されたバースマークを過去に配布されたプログラムに対して適用することが可能である。

バースマークは以下のように定義される。$p$, $q$ を与えられたプログラムとする。$f(p)$ を $p$ からある方法 $f$ により抽出された特徴の集合とする。このとき、以下の条件を満たすとき、$f(p)$ を $p$ のバースマークであるという。

- $f(p)$ はプログラム p のみから得られる。
- $q$ が $p$ のコピーならば，$f(p) = f(q)$ である。

バースマークが満たすべき性質として、弁別性と保存性がある。

**弁別性** プログラム $p$ から得られるバースマークとプログラム $p$ から等価変換されて得られたプログラム $p'$ から得られるバースマークが同一の性質を持つこと。

**保存性** 同一の仕様を持つ独立に書かれたプログラム $p$ と $q$ から取り出されたバースマークが異なる性質を持つこと。

図 1 にバースマークの取り出し、比較の流れの図を示す。

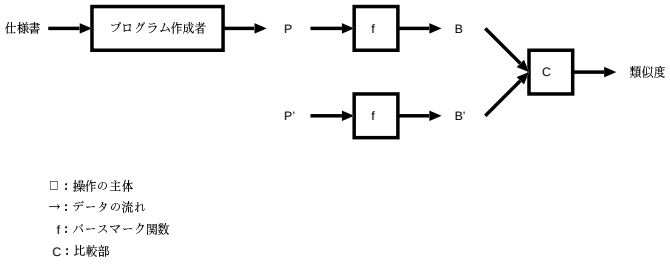


図 1　バースマークの取り出し、比較の流れの図

人は仕様書に従って、プログラム $P$ を作成する。次に、そのプログラムからバースマークの定義者によって定義されたバースマーク $B$ をバースマーク関数を使って取り出す。盗用が疑われるプログラム $P'$ からも同様にバースマーク $B'$ を取り出す。そして、$B$ と $B'$ を比較し、その類似度を出す。類似度がある程度大きな値となれば、プログラムの盗用が疑われる。

## 3　意図的バースマーク

### 3.1　バースマークの弱点

バースマークはそれだけで、ソフトウェアプロテクションとしての有効性が確立されている技術である。しかし、バースマークは、プログラムから取り出した特徴を比較する技術であり、明らかな著作権情報を出力する技術ではないので、著作権主張力は弱い。

### 3.2　意図的バースマークの概要

3.1 節の弱点に対して本研究では、意図的バースマークを提案する。意図的バースマークとは、バースマークの一種である。バースマークは自然に作成されたプログラムから得られるプログラムの特徴であるのに対して、意図的バースマークは、プログラムに何らかの特徴を持つように作成されたプログラムから得られるプログラムの特徴である。

#### 3.2.1　意図的ガイドライン

バースマークに特徴をつけるためのガイドラインである。意図的バースマーク考案者によって作成される。その方法に従ってプログラムを作成することで、プログラム作成者はプログラムに特徴をつけることができる。

#### 3.2.2　意図的バースマークの流れ

意図的バースマークを埋め込み、取り出し、比較する際の流れを示す。

1. まず、意図的バースマークの手法の考案者が意図的ガイドラインを作成する。
2. プログラムの作成者は埋め込みたい特徴を意図的ガイドラインに従って、プログラム中に埋め込む。
3. 盗用が疑われるプログラムが出てきたときは、まず意図的に埋め込んだバースマークを取り出し、盗用が疑われるプログラムからもそのバースマークを取り出す。そして、それらを比較し、類似度を出力する。
4. 類似度が一定値以上の場合は、盗用が疑われる。

図 2 に意図的バースマークの埋め込み、取り出し、比較の流れを示す。

### 3.3　意図的バースマークとバースマークの比較

意図的バースマークとバースマークの比較を行う。意図的バースマークでは、プログラム作成者の特徴がプログラム中に残るようにしてプログラムの作成を行うので、

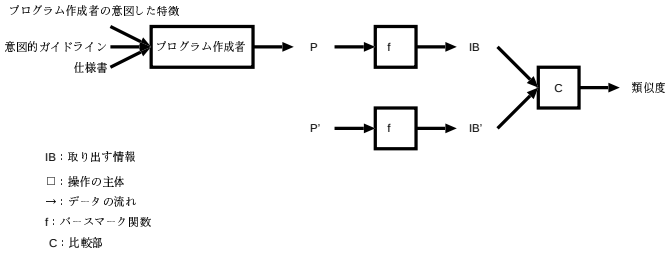


図 2 意図的バースマークの流れの図

バースマークと比較して弁別性、保存性が良くなり、その結果著作権主張力は強力となることが期待される。

また、コストに関しても、プログラム作成者の癖として考えられるものを制御するだけなので、それほど増加はしないと思われる。表 1 でバースマーク、意図的バースマークの比較を行う。

| | バースマーク | 意図的バースマーク |
|---|---|---|
| コスト | ◎ | ○ |
| 著作権主張力 | △ | ○ |

◎：非常に良い○：良い△：あまり良くない

表 1 バースマーク、意図的バースマークの比較

# 4 意図的バースマークの具体例

## 4.1 ローカル変数名に対する意図的バースマーク

繰り返し回数を制御するローカル変数は、プログラム中に多数出現する。本来プログラム内に現れる変数名は、命名法に従いその変数の使用目的にあわせた名前を付けることが、プログラム保守の観点から良いとされている。しかし、繰り返し回数を制御するローカル変数は、繰り返し回数の制御以外の役割を果たしていないため、例えば $x$ や $i$ など特定の意味を持たない名前がつけられている場合も多い。

そこで繰り返し回数を制御するローカル変数の変数名に対してプログラム作成者の意図を付けることが可能なのではないかと考えた。その一例として、プログラム中に出てくる全ての繰り返しを制御するローカル変数名を特定の正規表現から作り出される変数名にすることで、プログラムに作成者の特徴を付けることが可能である。

例えば $(x|y)+$という正規表現をプログラム作成者の特徴として付ける場合、そのプログラム内で使われる繰り返し構造を制御するためのローカル変数名はその正規表現から作りだされる $x$,$y$,$xy$,$xx$,$yy$,$\cdots$ などになる。ただし、$a|b$ は $a$ または $b$、$x+$ は $x$ の 1 回以上の繰り返しとする。

### 4.1.1 定義

ローカル変数名に対するバースマークの定義を示す。

$p$ プログラムのソースファイル

$m_1$,$m_2$,$\cdots$,$m_n$ プログラムのソースファイルから取り出された繰り返し構造を制御するローカル変数名の列

列 $m_1$,$m_2$,$\cdots$,$m_n$ を繰り返し回数を制御するローカル変数に対する意図的バースマークとする。

プログラムの盗用が疑われる場合には、作成元のプログラムから取り出した、バースマークと、盗用が疑われるプログラムから取り出したバースマークを比較し、その両方が特定の正規表現から作り出されることを示すことで盗用を疑う。

### 4.1.2 検討

この手法のコストに関する検討を行う。まず人的コストについて議論を行う。変数名を意図的に変化させるために必要なプログラム作成者の労力は微々たるものであり、プログラム作成コストは、それほど増加しないと考えられる。保守管理者は、プログラム作成者本人とは限らず、第三者がプログラムを保守管理する際に、プログラムの読みにくさの観点から保守管理コストは増加すると考えられる。

次に計算機コストについて議論する。また繰り返し回数の制御を行うことは、同じ整数型の変数であるので、その変数を宣言することで使用するメモリの量は同じである従ってメモリコストは掛からないと考えられる。また、変数名を変更した時でも、CPU が行う計算の量は同じであるため、掛かる CPU コストは変化しないと考えられる。

## 4.2 WFD 意図的バースマーク

構造化されたプログラムの中には、数多くの繰り返し構造が出現する。プログラミング言語として Java 言語を用いる場合、繰り返し構造の表現方法には、for,while,do-while の 3 つの表現方法がある。この 3 つのどの表現を使うかは、プログラム作成者にとって比較的自由に選択できる。従ってプログラムの中で出現する繰り返し構造の表現方法の系列の中にプログラム作成者の意図を埋め込むことを考える。

プログラム作成者は自分の意図として、系列を生成するための正規表現を用意する。そしてその正規表現に従って、プログラムの中に出現する繰り返し構造の表現方法を制御する。$F$ を for、$W$ を while、$D$ を do-while とすると、例えば $(FW)+$ という系列をプログラム作成者が考えたときには繰り返し構造の出現順にその表現方法として for,while,for,while,$\cdots$ となる。

ただ、for,while,do-while のそれぞれの表現には、以下のような場合に使われることが多くそれらの暗黙の了解を無視して使うことができる状況であるかを判断する必要はある。

**for** 始めから何回繰り返すかという情報が分かりやすい場合に使われることが多い。

**while** 特定の条件が満たされたときや、満たされなくなったとき、ループを抜けるなど始めから何回実行するか分かりにくい場合に、使われることが多い。

**do-while** while と同じように、始めから何回実行するか分かりにくく、かつ繰り返しが 1 回以上は実行される場合に用いられる。

またそのプログラムが複数のファイルに分けられているときには、それぞれのファイルにおいて、繰り返し構造の表現の系列は、同じ正規表現から作りだされる系列

とする。このようにすることで、意図的バースマークの破壊を目的としたプログラムの改変攻撃に対しても、ある程度の耐性ができるものと考えられる。

#### 4.2.1 定義

WFD 意図的バースマークについての定義を示す。

$p$ プログラムのソースファイル

$(r_1,r_2,\cdots,r_n)$ プログラム中に出現する for,while,do-while の列

$(r_1,r_2,\cdots,r_n)$ の列を WFD 意図的バースマークと定義する。

#### 4.2.2 検討

WFD 意図的バースマークにおいては、2 系列の一致率によってその比較を行う。

2 系列 $A = < a_1, ..., a_m >$,$B = < b_1, ..., b_n >$ の最長共通部分系列の長さ $L(m, n)$ は次で与えられるとする。

$$L(m,n) = \begin{cases} L(m-1,n-1)+1 & (a_m = b_n) \\ max\{L(m,n-1), L(m-1,n)\} & (a_m \neq b_n) \end{cases}$$

2 系列 A と B の一致率 $SR(A, B)$ を次で定める。

$$SR(A,B) = \frac{2 \cdot L(m,n)}{m+n}$$

得られた一致率を類似度とする。[2]

### 4.3 変数の宣言順序に対する意図的バースマーク

クラス内において、宣言されているフィールド変数の間にもし依存関係はない場合、その並び順は自由に変更することが可能である。ただし、依存関係とは宣言されている変数 $x$ にその変数より後に宣言されている $y$ を用いて変数宣言時代入がされている場合の変数 $x$ と $y$ の間の関係のことである。

そのため、フィールド変数の宣言順序に対してその並び順を意図的に制御することを考えた。Java 言語の場合、変数はプリミティブ型とクラス型の変数にわけることができる。従ってフィールド変数の宣言を行う場合にプリミティブ型の変数の宣言を行うかクラス型の変数宣言を先に行うかによってプログラム作成者の特徴をつけることが可能である。さらに、プリミティブ型には int,short,long,byte,float,double,char,boolean の 8 種類の型がある。従って、その型の並び方によってプログラム作成者の意図をつけることが可能である。

さらに、public,protected,private の 3 種類の変数の可視性についても同様にどの順序で並べるかについてプログラム作成者の特徴を付けることが可能である。同じようにクラス変数を示す static を先に宣言するかどうかや final 定数を先に宣言するかによってもプログラムに作成者特徴を付けることが可能である。

#### 4.3.1 定義

変数の宣言順序に対する意図的バースマークの定義を行う。

**P** プログラムのクラスファイル

$m_1,m_2,\cdots,m_n$ プログラム p の中で出てくるフィールド変数の型の列

とするときの列 $m_1,m_2,\cdots,m_n$ を変数の宣言順序に対する意図的バースマークと定義する。

### 4.4 SMC に対する意図的バースマーク

#### 4.4.1 SMC に関して

静的バースマークの 1 つとして SMC がある。これは静的なメソッドの並びをバースマークとして利用するものである。

メソッドの並びを変更することは、メソッドの依存関係が破壊されてしまい、プログラムに重大な影響を及ぼす恐れがあるため攻撃されにくいという利点がある。またメソッドを他のメソッドに置き換える行為には大変な労力が掛かるため攻撃されにくいという利点もある。

**P** プログラムのクラスファイル

**C** well known class の集合

$m_1,m_2,\cdots,m_n$ プログラム p の中で出てくるメソッドの列ただし、$m_i$ は、$m_i \in C$ $(1 \leq i \leq n)$。[3]

#### 4.4.2 SMC に対する意図的バースマーク

メソッドの並び方を変更することは、メソッド間の依存関係を考慮しなければならないため、一般的に難しい。しかし、コスト、理解性を考慮に入れなければ、goto 文や次に実行する箇所を記憶しておく変数を導入することで、SMC を自由に変更することは可能である。これらの方法はコストが掛かりすぎ、同時に理解性が悪くなるため、意図的バースマークとして使うことは難しい。

ただ、条件文の箇所においては比較的コストを掛けずに SMC に特徴を付けることが可能である。その代表的な例として switch 文がある。

switch 文は、互いに全く等価な文を並列に並べるものであるので、その順序は自由に変更することが可能である。

ただ、プログラムの中に switch 文が現れることは少ない。したがって一般的に使われる if 文への応用を考える。

例えば、if(P) S1; else S2; は、if(!P) S2; else S1; と変換することができる。否定の演算は場合によっては、!P の方が計算量が少なくなる可能性も考えられることから P と!P の 2 つは、ほぼ同値であるとみなすことも可能である。このとき S1, S2 の中に 1 つ以上のメソッドが並んでいた場合、この変換を行うことで、コストをそれほど掛けることなく意図的に SMC を制御できたことになる。

### 4.5 CVFV に対する意図的バースマーク

#### 4.5.1 CVFV に関して

フィールド変数の初期値をバースマークとする。フィールド変数の初期値はそのオブジェクトの振る舞いを決定するため、その値を修正することは、プログラムの出力を変化させることにつながるため危険である。

現在定義されている CVFV バースマークの定義

$p$ プログラムのクラスファイル

$v_1$,$v_2$,$\cdots$,$v_n$　　$p$ の中のフィールド変数

$t_1$,$t_2$,$\cdots$,$t_n$　$v_1$,$v_2$,$\cdots$,$v_n$ の型

$a_1$,$a_2$,$\cdots$,$a_n$　　$v_1$,$v_2$,$\cdots$,$v_n$ の初期値

とするとき $(t_1,a_1)$,$(t_1,a_2)$,$\cdots$,$(t_n,a_n)$ の系列をプログラム $p$ の CVFV バースマークとして定義する。初期値がない場合は、null を初期値とする。[3]

#### 4.5.2　CVFV に対する意図的バースマーク

CVFV バースマークで定義されているフィールド変数のうち初期値代入がコンストラクタ内で行われるか、あるいはそれ以外の所で行われているかの情報を追加したバースマーク。初期値代入に関して、コンストラクタ内で行うかどうかは、比較的プログラム作成者の自由になることが多く、そこにプログラム作成者の特徴を入れることを考える。拡張版 CVFV バースマークの定義を示す。

$P$　プログラムのソースファイル

$v_1$,$v_2$,$\cdots$,$v_n$　$p$ の中において宣言されている変数

$t_1$,$t_2$,$\cdots$,$t_n$　$v_1$,$v_2$,$\cdots$,$v_n$ の型

$a1$,$a2$,$\cdots$,$a_n$　$v_1$,$v_2$,$\cdots$,$v_n$ の初期値

$d_1$,$d_2$,$\cdots$,$d_n$　コンストラクタ内において初期値代入が行われるか否かを決める boolean の値

とするとき $(t_1,a_1,d_1)$,$(t_2,a_2,d_2)$,$\cdots$, $(t_n,a_n,d_n)$ の系列をプログラム $p$ の拡張版 CVFV バースマークとして定義する。対象とする言語は Java 言語でソースファイルとする。また、$v_i$ は出現順に並べる。$d_i$ に関して、1 つ以上のコンストラクタにおいて初期値代入が行われていた場合 boolean の値は true とする。ただし初期値代入とは、宣言された変数にソーステキスト上ではじめて値が代入されることとし、初期値とはその際の値のこととする。$a_i$ に関してオーバーロードされたいくつかのコンストラクタにおいて、初期値代入が行われている場合は、最初に書かれたコンストラクタにおいて代入された値とする。

## 5　実験に関して

意図的バースマークが有効な手法であるかを確認する実験を行う。プログラムの盗用者は盗用したプログラムに対して攻撃を仕掛けるものと考えられる。本研究では難読化攻撃が仕掛けられたと仮定し、保存性の確認を行う。まず、1 つのプログラムを本研究で提案した 5 つの意図的バースマークを埋め込むように作成した。そして、そのプログラムに対して、難読化変換を行った。難読化には、Sandmark に用意されている 39 の難読化アルゴリズムを適用した。作成元のプログラムからバースマークを取り出し、難読化した後のプログラムからも、同様にバースマークを取り出し、そのバースマークの間で変化があったかどうか確認した。図 3 に実験の概要を、表 2 に実験結果を示す。

その結果、ローカル変数名に対する意図的バースマーク、変数の宣言順序に対する意図的バースマーク、SMC に対する意図的バースマーク、CVFV に対する意図的バースマークの 4 種類のバースマークに関しては、39 全ての難読化に対して意図的バースマークは壊されることなく、

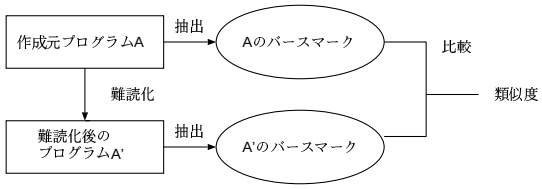


図 3　行う実験の概要

| | 壊されなかった | 壊された |
|---|---|---|
| ローカル変数 | 39 | 0 |
| WFD | 0 | 39 |
| 変数の宣言順序 | 39 | 0 |
| 拡張版 SMC | 39 | 0 |
| 拡張版 CVFV | 39 | 0 |

表 2　実験結果

残った。一方、WFD 意図的バースマークに関しては、39 全ての難読化に対して意図的バースマークは壊されるという両極端なものとなった。

## 6　今後の課題

本研究では、ソフトウェアの盗用の現状に対して、意図的バースマークを提案した。意図的バースマークは、プログラムにプログラム作成者の特徴を意図的につける技術である。まず、簡単なバースマークとしてローカル変数名に対する意図的バースマーク、WFD 意図的バースマーク、変数の宣言順序に対する意図的バースマークを提案した。そして、現在提案されている 2 つの静的バースマークを意図的に制御する方法を検討した。

今後の課題として、さらに大掛かりな実験を行うこと、攻撃耐性の検討を行うことなどが残されている。

## 参考文献

[1] C.Collberg, C.Thomborson: “Software Watermarking:Modelsand Dynamic Embedding” , POPL’99, pp.311-324 (Jan.1999).

[2] Ginger Myles et al.: “Detecting Software Theft via Whole Program Path Birthmarks”, ISC2004 in LNCS 3325, pp.404-415 (2004).

[3] 林ら:“特徴抽出と抽象化による動的バースマークの構成とその検証”, 情報処理学会論文誌 D, VolJ89-D, No.8, pp.1751-1763 (Aug.2006).

[4] Haruaki Tamada et al.: “Java Birthmarks – Detecting the Software Theft–”, Journal ofIEICE Transactions on Information and Systems, Vol.E88-D, No.9, pp.2148–2158 (Sept.2005).

[5] 岡本ら:“API 呼出しを用いた動的バースマーク”, 電子情報通信学会論文誌 D, Vol.J89-D, No.8, pp.1751-1763 (Aug.2006).

[6] 森山ら:“API 関数呼出履歴によるソフトウェア動的バースマークの一方式”, 電子情報通信学会研究技術報告 ISEC2006-82, pp.77-84 (Sep.2006).