ソフトウェア工学III

ソフトウェアプロテクション（2）

～ソフトウェアの盗用抑止技術～

ソフトウェア工学講座
門田暁人
akito-m@is.naist.jp
B303室，内線5311

先週の続き．．．ソフトウェア特許の現状

- Eolas Technologies Inc.
  - カリフォルニア大学からスピンオフされた，1人の社員と多数の投資家
  - US Patent 5,838,906の権利を保持する．(1994年申請，1998年認可)
    - ハイパーテキストのタグによって外部アプリケーションを（ブラウザ画面内で）自動的に起動する．
    - サブマリン特許
- 1999年 IEが特許を侵害しているという理由で，Microsoftを訴える．
- 2003年8月 Eolasが連邦裁判所で5億2100万ドルの損害賠償を勝ち取った．
- 2005年3月 連邦控訴裁判所は、下級審の判決を覆し、連邦地裁に差し戻した．
- 2005年9月 Microsoft 社の異議により再審査されていたEolasの特許が，米国特許商標庁により有効と認定された．
- 2005年10月 賠償金の金額が高すぎる（米国外の売り上げも入っている）という主張が却下される．Microsoftの主張「本社はマスターディスクを国外のOEM 先に送付しただけであり、「製造」を担当したのは現地の企業だ」
- 現在，連邦地裁に差し戻された裁判の判決はまだ出ていない．

ソフトウェア盗用の事例

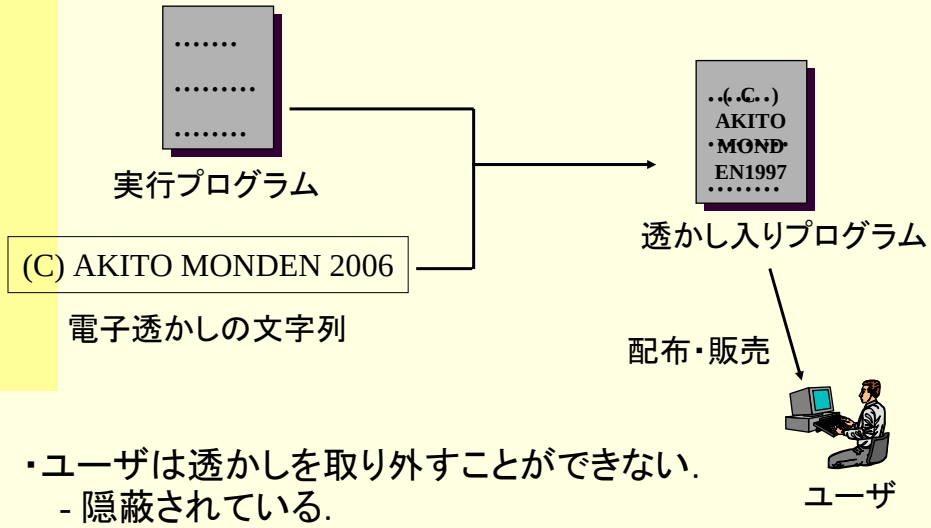
- 複数のOSS → Divx コンバータ [1]
  - プロジー が販売している「Divx コンバータ」を調べると，DeCSS, DVD2avi, bbMPEG, Lame のソースコードと同じファイル名が確認された．これらは GPL に準拠しているにも関わらず，Divx コンバータはソースコードを公開しておらず，GPL に違反していた．
- PearPC → Cherry OS [2]
  - チェリー OSとペアーPCの関数名や変数名を比較したところ，全てが一致したと報告されている
- 盗用が行われた場合…
  - 盗用の疑いのあるソフトウェアを発見することが困難
  - 盗用の立証が困難

[1]http://slashdot.jp/article.pl?sid=03/02/13/099215
[2]http://www.wired.com/news/mac/0,2125,66847,00.html

ソフトウェア盗用の抑止技術

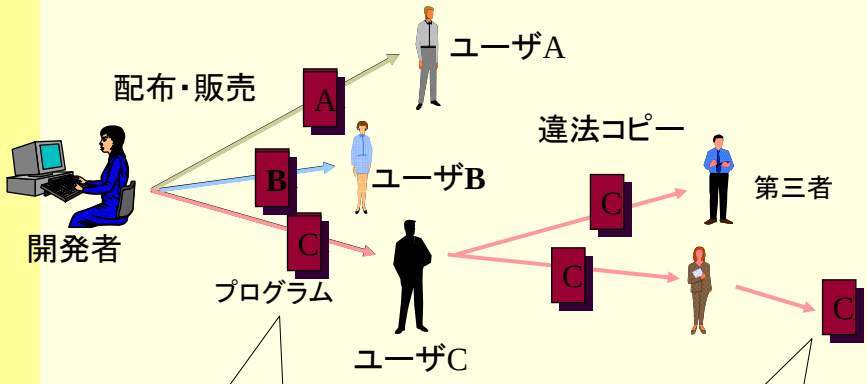
- 電子透かし
  - ソフトウェアに埋め込む「サイン」
- バースマーク
  - 各ソフトウェア固有の特徴量
- （電子署名）
  - 電子透かしと似ているが，盗用抑止にはならない．
  - 署名を見て出どころが確認できるが，署名そのものは取り外せる．

ソフトウェア電子透かし



・ユーザは透かしを取り外すことができない.
  - 隠蔽されている.
・開発者は透かしを取り出せる.

電子透かしが有用となる場合
1) 違法コピープログラム流通の抑止

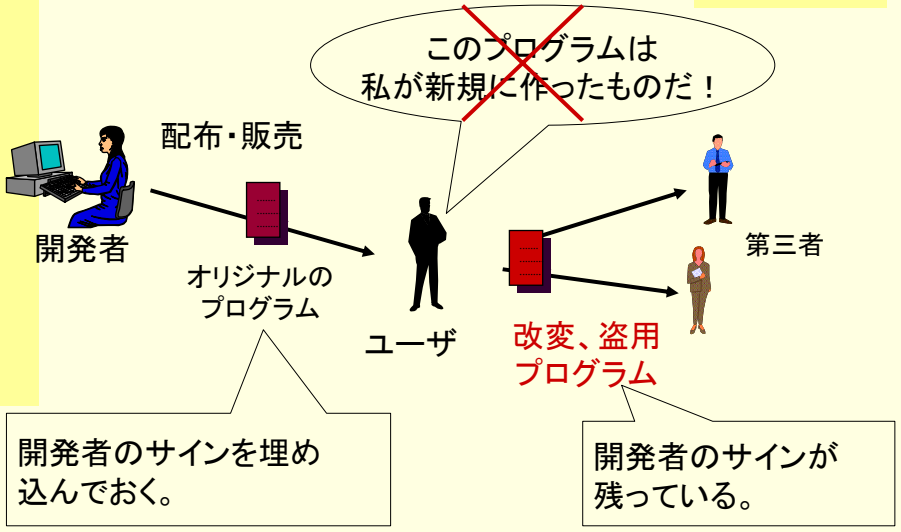


ユーザIDをプログラム中に埋め込んでおく

コピー元のユーザが特定できる。

電子透かしにより、違法コピーをばらまいた者を特定できる。

2) 改変・盗用プログラムの流通の抑止



電子透かしによって、著作権の侵害を立証することができる。

ソフトウェア電子透かし　ーより形式的な説明

- 透かし *w* は，プログラム開発者の著作権等を表す文字列，もしくは，プログラムのユーザ *Ui* のID（識別子）
- プログラム *P* の開発者（配布者）は，*P* をリリースする前に，透かし埋め込みツール *E* により透かし *w* を予め埋め込んでおく．
- 透かしが埋め込まれたプログラム $P' = E(P, w)$ をユーザ *Ui* に配布する．
- P' の配布者は，透かし取り出しツール *D* を用いることで，P' から透かし $w = D(P')$ を取り出すことができる．
- 望ましい*D* の性質として，P'がたとえ改ざんされた後でも *w* を取り出せることが挙げられる．
- *D* の正当性を証明する（ありもしない透かしを表示するでっちあげのツールでないことを示す）ことが別途必要となる．一つの方法は，*D* の実装を予め世間に公開しておくことである．

電子透かしの分類

- 静的電子透かし
  - プログラム中に文字列として埋め込む
  - プログラムを動作させなくても透かしの取り出しが可能
- 動的電子透かし
  - (1) Easter Egg透かし（特殊動作により実行結果に現れる）
  - (2) 実行時の状態（メモリ）に現れる透かし
  - (3) 実行系列に現れる透かし
  - プログラムを動作させないと透かしが取り出せない.
  - 静的透かしよりも耐タンパー性を高めやすい.

単純な電子透かしの例

```
public class Fibonacci {
   String watermark = “(C) AKITO MONDEN”;
   public int fibonacci ( int n ) {
        if ( false )
              println ( “(C) AKITO MONDEN” );
        if ( n <= 2 ) {
              return 1;
        } else {
              return fib ( n - 1 ) + fib ( n - 2 );
        }
   ...
```

単純な電子透かしの例

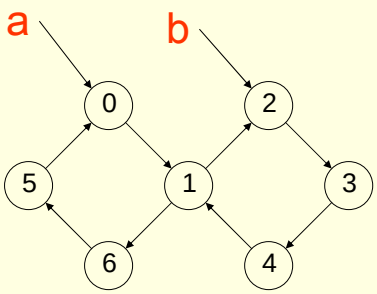
```
public class Fibonacci {

   public int fibonacci ( int n ) {
        if ( opaque predicate )
                println ( “(C) AKITO MONDEN” );
        if ( n <= 2 ) {
                return 1;
        } else {
                return fib ( n - 1 ) + fib ( n - 2 );
        }
   ...
```

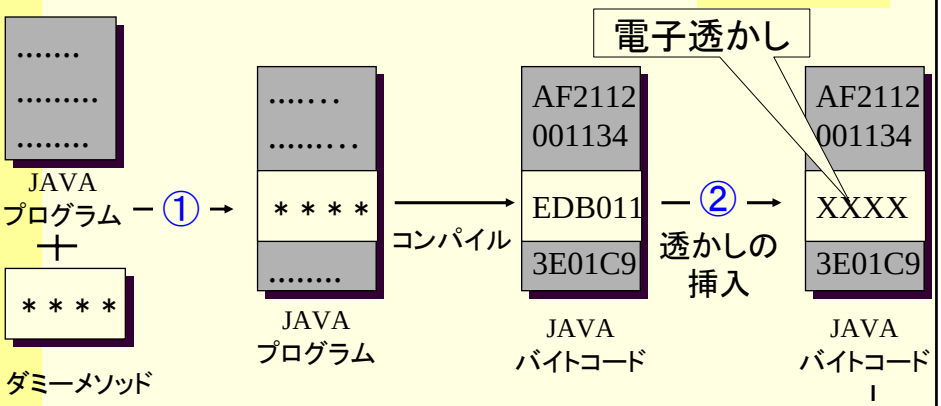
Opaque Predicates

- Opaque predicate は
  - 恒真，または，恒偽となる式である.
  - 式の値を推測することは困難である.

ポインタを使った例



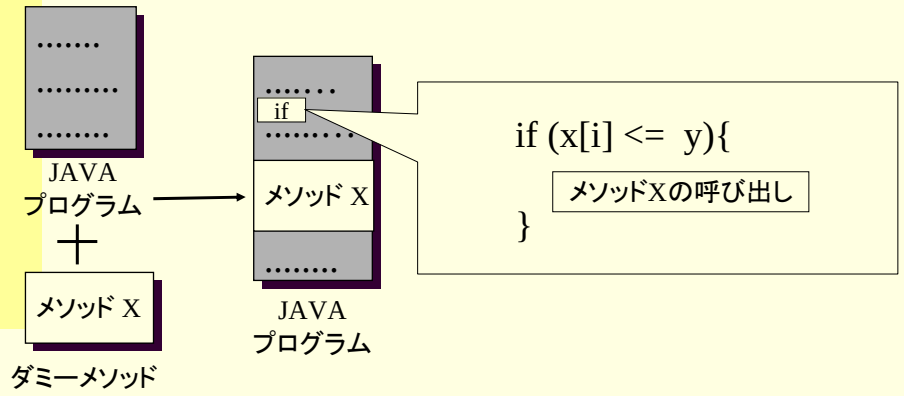
Javaを対象とする電子透かし法の例



① ソースプログラムにダミーメソッドを追加してコンパイルする。
② ダミーメソッドに電子透かしを入れる。
③ 透かしを取り出す。

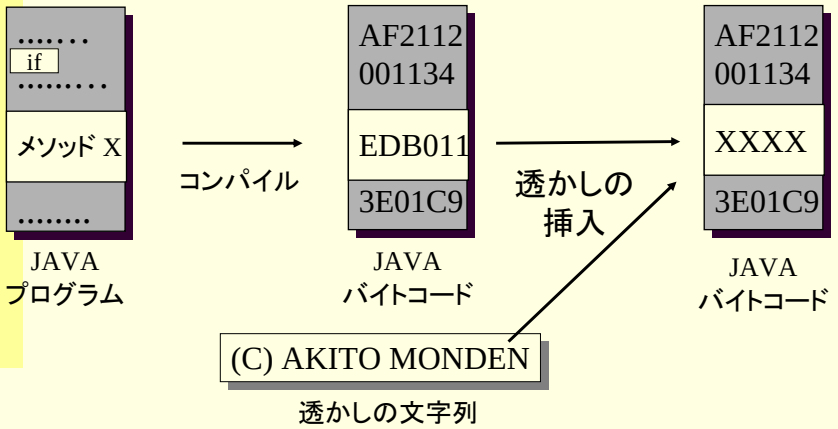
ダミーメソッドの隠蔽

恒偽のopaque predicateを用いる.



メソッドXが実行されないことは、ユーザに知られにくい。

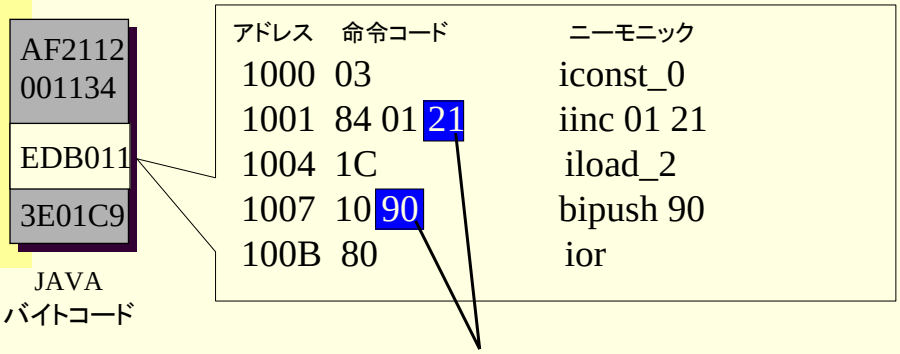
バイトコードベリファイアの問題



透かしとして挿入したい文字列をそのまま上書きすると、
バイトコードベリファイアのチェックに通らなくなる。

電子透かしの挿入 1/5

値を変更してもよい数値オペランドに情報を
上書きする。

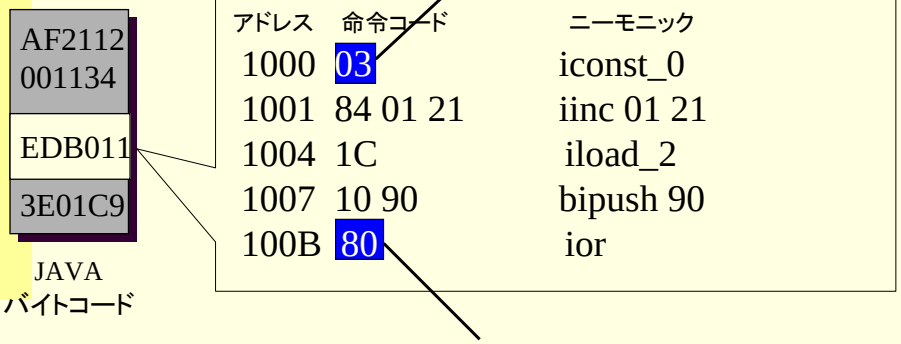


値を変更しても、バイトコード検証器の
チェックに影響を与えない。

電子透かしの挿入 2/5

命令の置き換えを行う。

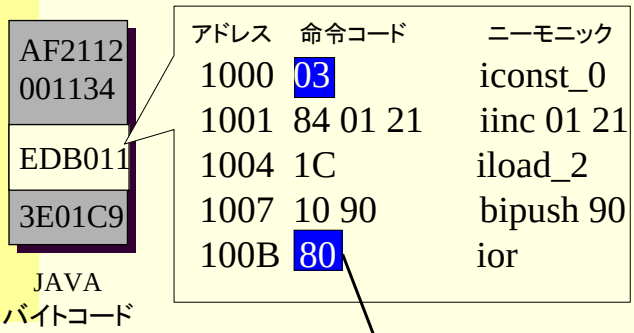
02, 04, 05に置き換え可能



60, 64, 68, 6C, 70, 7E, 82に置き換え可能

電子透かしの挿入 3/5

命令コードに情報を割り当てる。

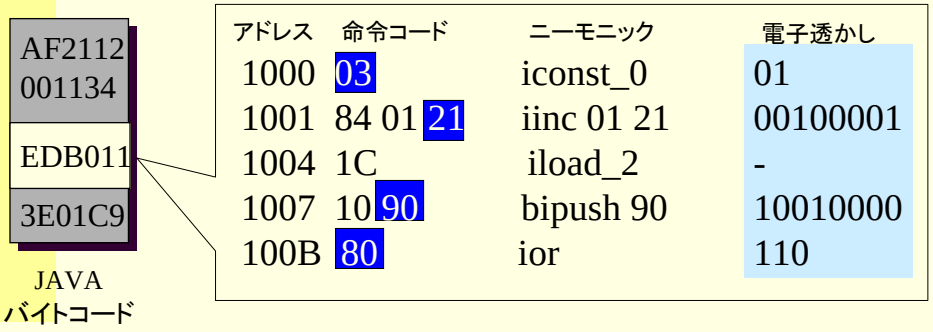


60, 64, 68, 6C, 70, 7E, 82に
置き換え可能

命令コード	情報
60 iadd	… 000
64 isub	… 001
68 imul	… 010
6C idiv	… 011
70 irem	… 100
7E iand	… 101
80 ior	… 110
82 ixor	… 111

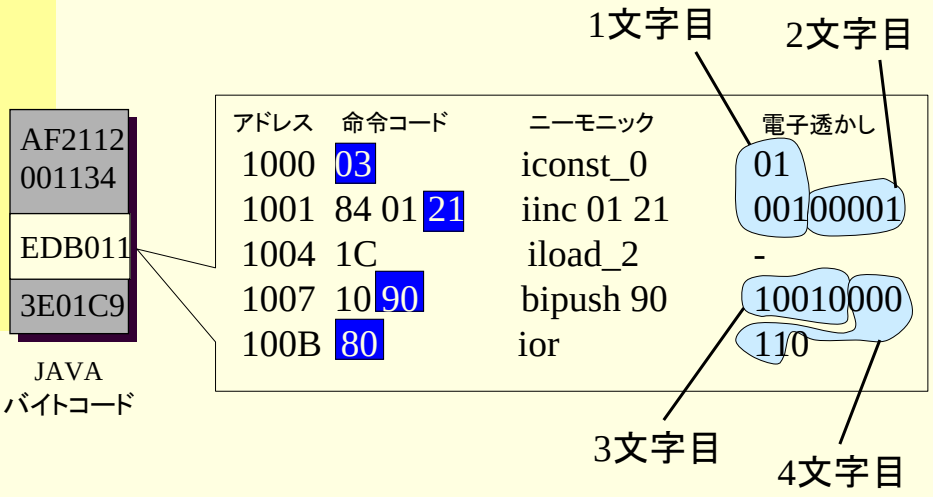
電子透かしの挿入 4/5

電子透かしの表現



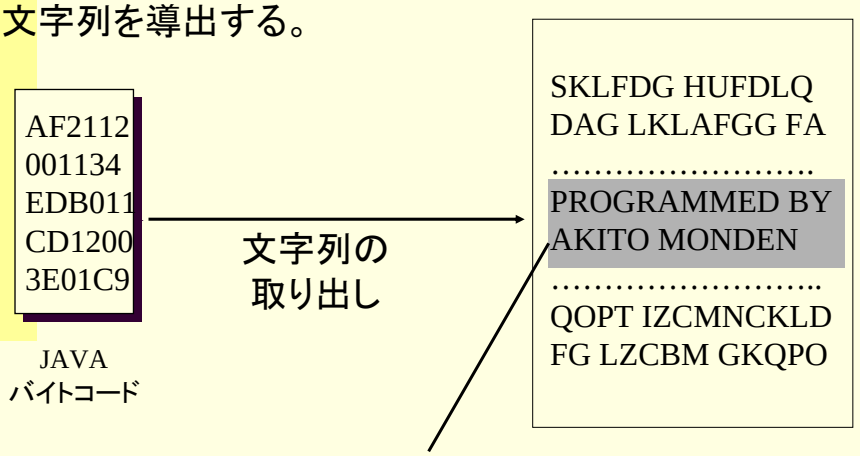
電子透かしの挿入 5/5

文字列の表現 (1文字/ 5bit)



電子透かしの取り出し

プログラム全体に透かしが入っているとみなして
文字列を導出する。



一部分に透かしの文字列が現れる。

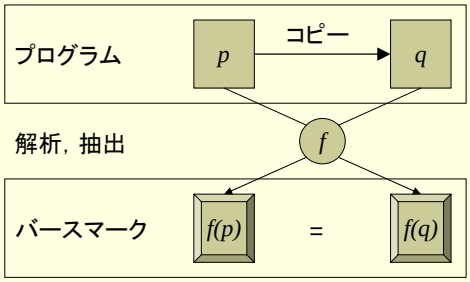
http://se.naist.jp/jmark/

バースマーク

- ソフトウェアが動作するために必要不可欠なソフトウェア固有の特徴の集合
  - ソースコードを対象とするもの
  - 実行バイナリを対象とするもの
- バースマークを利用してソフトウェアを識別することができる
  - 盗用の発見や立証ができる
  - 電子透かしと異なり，あらかじめ情報を埋め込む必要がない
- バースマークの種類
  - 静的バースマーク
  - 動的バースマーク

バースマークの定義

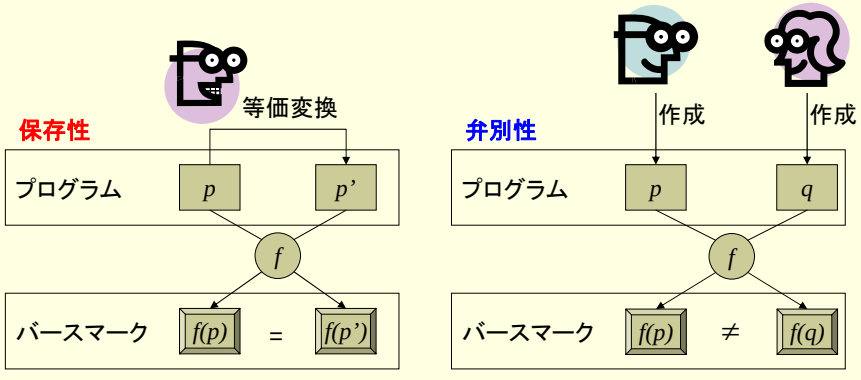
- プログラム固有の特徴の集合
  - 条件1. プログラムpのみから得られる
  - 条件2. プログラムqがpのコピーならば，p, q のバースマークは同じ



バースマークが満たすべき性質

保存性 もしpを等価変換（難読化，最適化など）し，p'を得たのならば，pとp'のバースマークは同一である

弁別性 もしpとqが全く独立に実装されたなら，pとqのバースマークは異なる



単純なバースマーク

```
xor  eax, eax
push ebp
push $004466aa
mov  eax, [ebp - $04]
call InitComport
mov  mov eax, [ebp - $04]
mov  eax, [eax + $000002f4]
mov  dl, $01
call SetTimerEnabled
push $000001f4
call Sleep
```

バースマーク

- xorの個数
- pushの個数
- movの後に最もよく来る命令
- pushの後に最もよく来る命令

このような単純なバースマークは，攻撃によって消されやすい．
（例：等価な命令に置き換える）

プログラムの持つ特徴

- 特徴
  - 属性（フィールド）を持つ
  - 操作（メソッド）を持つ
  - 継承関係がある
  - 頻繁に標準ライブラリを使用する
  - 変数名，メソッド名など様々な名前を含む
  - 数値演算がある
- 玉田によるJavaクラスファイルを対象としたバースマーク
  - フィールド初期値バースマーク(CVFV)
  - メソッド呼び出し系列バースマーク (SMC)
  - 継承関係バースマーク (IS)
  - 使用クラスバースマーク (UC)

http://se.naist.jp/jbirth/

フィールド初期値バースマーク

- クラスのフィールドの型と値の集合をバースマークとして扱う
  - フィールドの初期値を変更するとプログラムの出力も変わってしまう可能性が高い
  - フィールド名は容易に変更可能なのでバースマークとして扱わない

```
public class SampleClass{
  public static int field1 = 10;
  private Object field2;
}
```

SampleClassのフィールド初期値バースマーク
(int, 10)
(Object, null)

メソッド呼び出し系列バースマーク

- メソッド呼び出しの系列をバースマークとして扱う
  - プログラム中には既知のクラスのメソッド呼び出しが多数含まれる
  - 既知のクラスのメソッドを置き換えることは一般に困難
  - メソッド呼び出し順序の変更は機械的にできない

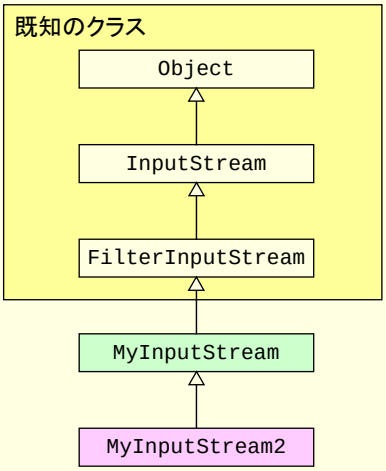
```
ArrayIteratorの
メソッド呼び出し系列バースマーク

Class#isArray
IllegalArgumentException#<init>
Array#get
Array#getLength
```

```
public class ArrayIterator extends Object
                implements Iterator{
  private Object array;
  private int index = 0;
  public ArrayIterator(Object array){
    if(!Class.isArray(array.getClass())){
      throw new IllegalArgumentException(
        "not array type");
    }
    this.array = array;
  }
  public Object next(){
    return Array.get(array, index++);
  }
  public boolean hasNext(){
    return index<Array.getLength(array);
  }
  public void remove(){ }
}
```

継承関係バースマーク

- 継承ツリーをルートまで辿り，既知のクラスの名前の系列をバースマークとして扱う
- 全てのクラスはクラス階層構造を持つ
  - クラス階層の系列は既知のクラスとユーザ定義のクラスの両方を含む
  - ユーザ定義のクラスの名前を変えることは容易である
    - ユーザ定義のクラスの名前をnullに置き換える



継承関係バースマーク

```
MyInputStream:  FilterIputStream, InputStream, Object
MyInputStream2: null, FilterInputStream, InputStream, Object
```

使用クラスバースマーク

- クラスが使用する既知のクラスの集合をバースマークとして扱う
- クラスは機能を実現するため，既知のクラスを使用する
  - スーパークラス
  - メソッドの引数・返り値
  - メソッド呼び出し
  - フィールド参照・代入

```
import java.util.Iterator;
import java.lang.reflect.Array;

public class ArrayIterator extends Object
                    implements Iterator{
  private Object array;
  private int index = 0;
  public ArrayIterator(Object array){
    if(!Class.isArray(array.getClass())){
      throw new IllegalArgumentException(
        "not array type");
    }
    this.array = array;
  }
  public Object next(){
    return Array.get(array, index++);
  }
  public boolean hasNext(){
    return index < Array.getLength(array);
  }
  public void remove(){
  }
```

ArrayIteratorの使用クラスバースマーク

```
java.lang.reflect.Array
java.lang.Class
java.lang.IllegalArgumentException
java.lang.Object
java.lang.String
java.util.Iterator
```

バースマークの類似度

- バースマークが変更されても，オリジナルのものと非常によく似ている場合がある
- 同じかそうでないかでバースマークを比較すると，手法そのものが敏感になりすぎる

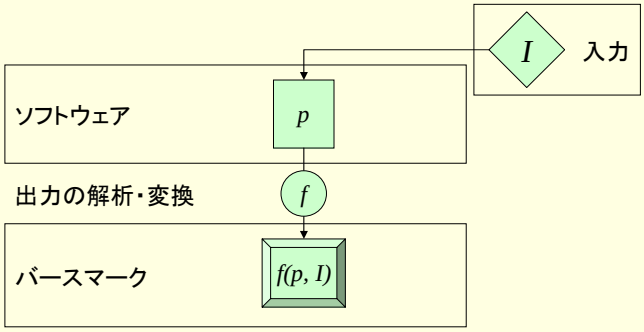
プログラムpとqの
バースマーク$f(p)$と$f(q)$の類似度

$$\frac{2k}{m+n}$$

m: バースマーク$f(p)$の要素数
n: バースマーク$f(q)$の要素数
k: 一致した要素の数
$(m \neq 0 \wedge n \neq 0)$

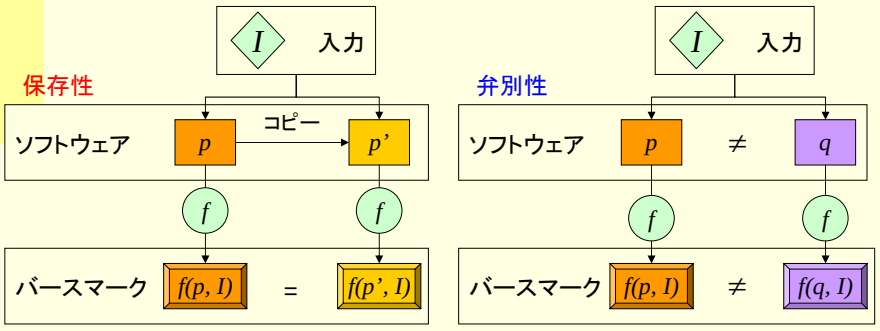
動的バースマークの定義

- 定義
  - 入力 I を与えてソフトウェア p を実行したときの出力から得られる $f(p, I)$
    - 出力は画面出力、計算結果とは限らない、ソフトウェアの動作の結果全て



動的バースマークが満たすべき性質

- p' が p のコピーであれば，p と p' のバースマークは同一 (保存性)
- p と q が独立であれば(コピーでなければ)，p と q のバースマークは異なる (弁別性)



動的バースマークの例

- 岡本らによって提案された動的バースマーク
  - ソフトウェアの実行時にAPI の呼び出しを観測，ソフトウェア固有の情報とするもの
- API: Application Program Interface
  - オペレーティングシステムに実装されたプログラムインターフェース
  - OS上で動作するアプリケーションは，APIによりOSの様々な機能を利用
    - ファイルの入出力，プロセスやスレッドの制御，セマフォやミューテクスなどの同期機能，GUI機能
  - 簡単に消去することはできない

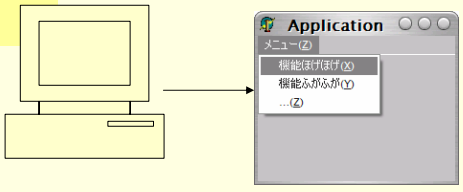
岡本, 玉田, 中村, 門田, 松本, "ソフトウェア実行時のAPI呼び出し履歴に基づく動的バースマークの実験的評価," プログラミングシンポジウム, Jan. 2005.

バースマークの記録方法

- 対象アプリケーションの動作中の API呼び出しを記録

2種類のバースマーク

- 実行系列バースマーク
  - APIの呼び出し順序に基づく
- 実行頻度バースマーク
  - APIそれぞれの呼び出し回数に基づく



API呼び出しシーケンス

OpenFile
ReadFile
CloseFile
CreateWindow
ShowWindow
FileOpen
ReadFile
CloseFile
SendMessage
…

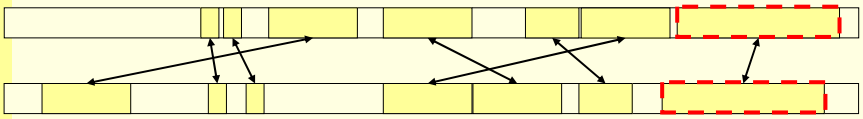
実行系列バースマーク

APIの種類	呼出回数
FileOpen	696
FileClose	696
CreateWindow	43
ShowWindow	66
SendMessage	45903
ReadFile	4849

実行頻度バースマーク

バースマークの比較方法

- 実行系列バースマーク
  - 比較する実行系列バースマークに共通に存在する部分列を発見する
  - 最大の部分列の占める割合を類似度として採用



- 実行頻度バースマーク
  - APIごとの呼び出し回数をベクトルの要素とし，ベクトルのなす角のコサインを類似度とする（単純類似度）

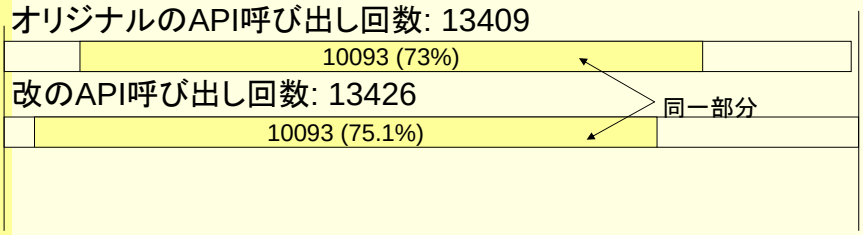
実験：使用したソフトウェア

- 全てMP3オーディオのメタ情報タグの編集ソフトウェア
- Super Tag Editor 2.00b7
  - MERCURY, 2001/04/30
- Super Tag Editor改
  - haseta, 2004/06/25
  - STE 2.00b7をベースに改造を施したもの，37回の更新

ソフトウェア名	作者
Super Tag Editor 2.00b7	MERCURY
Super Tag Editor改(Rev.32)	haseta
つゆたぐ 2.02	P's Soft
Mp3tag 2.2.6.0	Florian Heidenreich
Teatime 2.525	黒田 徹

Super Tag Editor: http://www5.wisnet.ne.jp/~mercury/supertag/index.html
Super Tag Editor改：http://hp.vector.co.jp/authors/VA012911/mp3DB/ste.html

実験結果 実行系列バースマーク



ソフトウェア名	呼び出し回数	最長一致	占める割合
STE2.00b7	13286	ー	ー
STE改	13447	10093	75.1%
Mp3tag	10834	31	0.3%
TeaTime	1647	11	0.7%
Tuyutag	191783	10	0.0%

単位：API呼び出し回数（回）

まとめ

- ソフトウェア盗用の抑止技術
  - 電子透かし
  - バースマーク

本日の演習課題

- 電子透かしの新たな方法を考案し，1つのプログラムに対する適用事例を説明せよ.
  - 透かし埋め込み対象プログラムの説明
    - ソースコード or 実行ファイル
  - 透かし埋め込み方法の説明
  - 透かし取り出し方法の説明
    - 手動，もしくは，自動
  - どういう場合に役立つか
    - 想定する攻撃者
  - 電子透かしに対する攻撃手段とその対策
  - 正当な透かしであることを裁判所で立証できるか？

本日の演習課題

- 課題
  - 適当なプログラムを用意し，電子透かしを挿入せよ.
- レポート提出期限
  - 2006年2月2日（金）2限
  - 希望者は課題発表を行うこと
    - レポートの内容について説明する.
    - レポート用紙をスクリーンに映す，もしくは，パワーポイント等のプレゼン資料を用いる.
- 連絡先
  - 門田暁人 akito-m@is.naist.jp
  - B303室，内線5311