# A Goal-Oriented Approach to Software Obfuscation Techniques

**A Case Study to Hide Software Watermarking**

**Software Engineering Lab.**

D2 Hiroki Yamauchi

## Background

- Recent software products often contain "Intellectual Property" of a software development company.
  - In-house software component library
  - Algorithms
  - Customer Data
- Such intellectual properties often stolen when the company outsources a part of development.
- Intellectual properties should be protected by software protection techniques.

## Software Protection Techniques 1/2

- Obfuscation
  - Translates a program so that it is more difficult to understand, yet is functionally equivalent to the original.

```
int n = 52;
int i, k, p=1;

for(i=1;i<=31;i++)
{
   k = n – i + 1;
   p = p * k / i;
}
return p;
```

Program for $_{52}C_{31}$

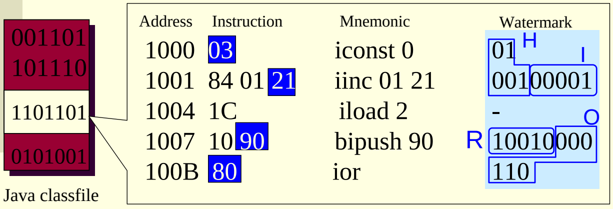
Translation →

```
int n=105,k,i=1,p=1;
L1: if(i <= 31){ for(;;){
k=n–2*i+2;p=(p*k-p)/2/i;
if(++i>31){k=n–2*i+2;
p=(p*k-p)/2/i++; }else
break;
p=p*(n–2*i+1)/2/i++;}
goto L1;}
return p;
```

Obfuscated program

## Software Protection Techniques 2/2

- Software Watermarking
  - A process of embedding a small amount of identifying information into a program.
- Example of static code watermark



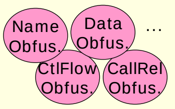
| Address | Instruction | Mnemonic | Watermark |
|---|---|---|---|
| 1000 | 03 | iconst 0 | 01 |
| 1001 | 84 01 21 | iinc 01 21 | 00100001 |
| 1004 | 1C | iload 2 | - |
| 1007 | 10 90 | bipush 90 | 10010000 |
| 100B | 80 | ior | 110 |

- When the program was stolen, watermark proves the fact of program theft.

## Problem

There is no systematic method on how to apply software protection techniques appropriately.

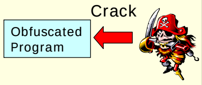
- Which obfuscation technique should be used?



- Which part of the program should be obfuscated?

```
int fact=1, upper=16
int  i;
for(i=1;i<=upper;i++){
  fact  *=  i;
}
printf("%d",fact);
```

- How much effects of obfuscation can be expected?



These problems are caused because the conventional techniques do not count the purpose and target of the cracker.

## Research Objective

Establish a goal-oriented analysis framework for proper use of the existing obfuscation techniques.

*Key idea*

- Assume an imaginary cracker with his purpose and target (i.e., goal).
- Break down the goal into pieces, each of which an appropriate obfuscation is applied to.

*Approach*

**Step1.** Determine a capability of an imaginary cracker.

**Step2.** Identify a cracker's goal.

**Step3.** Conduct a goal-oriented analysis.

**Step4.** For every terminal sub-goal, select an obfuscation.

**Step5.** Apply the selected obfuscations to the program.

## Case study

We have applied the proposed framework to hide a watermark embedded in a program.

Target program

- A Java program with static code watermark embedded by jmark [1].

Cracker's Capability Model

Knowledge: Know jmark algorithm.

Observation: Watch class file and input/output values.

Control: Use debuggers and disassemblers.

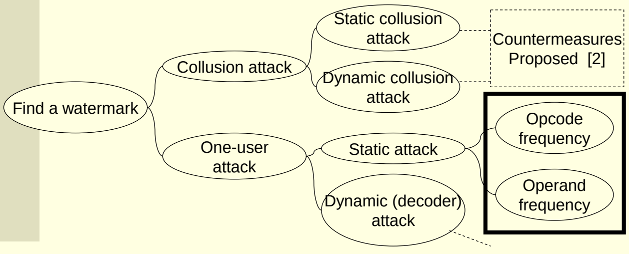
Goal

- Find a watermark

[1] jmark home page , http://se.naist.jp/jmark/

## Goal-Oriented Analysis

- A goal tree for finding a watermark



In this case study, We protect software from an attack based on opcode/operand frequency.

[2] K. Fukushima, T. Tabata, K. Sakurai, "A Software Fingerprinting Scheme for Java Using Class Structure Transformation", IPSJ-Journal, Vol.46 No.8, pp. 2042-2052, 2005.

## Opcode/operand frequency attack

- An ordinal Java class has a biased opcode/operand frequency, while watermarked method shows unique frequency.
- Preliminary analysis with rt.jar (a Java runtime library)

| Rank | rt.jar | |
|---|---|---|
| 1 | aload_0 | 10.01% |
| 2 | invokevirtual | 7.85% |
| 3 | getfield | 5.50% |
| 4 | dup | 4.49% |
| 5 | aload_1 | 3.57% |
| 6 | invokespecial | 3.31% |
| 7 | aload | 3.24% |
| 8 | ldc | 2.98% |
| 9 | iload | 2.76% |
| 10 | iconst_0 | 2.51% |
| rest | | 53.28% |

## Opcode/operand frequency of watermarked method

- Find unique instruction and its frequency, check out operands.

Dissassemble code

```
84 03 89  | iinc 03h 89h
84 02 5E  | iinc 02h 5Eh
84 03 78  | iinc 03h 78h
84 02 45  | iinc 02h 45h
84 03 78  | iinc 03h 78h
84 02 45  | iinc 02h 45h
```

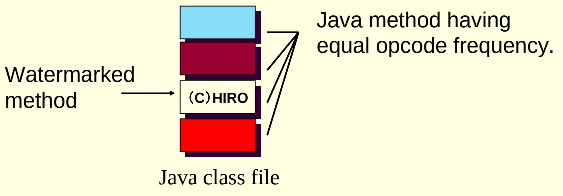
* iinc: increment instruction

- And then, search around this code, watermark (candidate) values can be found.

| Rank | watermaked method | |
|---|---|---|
| 1 | invokevirtual | 12.24% |
| 2 | bipush | 7.14% |
| 3 | iload_1 | 6.12% |
| 4 | iload_2 | 6.12% |
| 5 | iload_3 | 6.12% |
| 6 | iinc | 6.12% |
| 7 | goto | 6.12% |
| 8 | iconst_0 | 3.06% |
| 9 | iconst_3 | 3.06% |
| 10 | ldc | 3.06% |
| rest | | 40.82% |

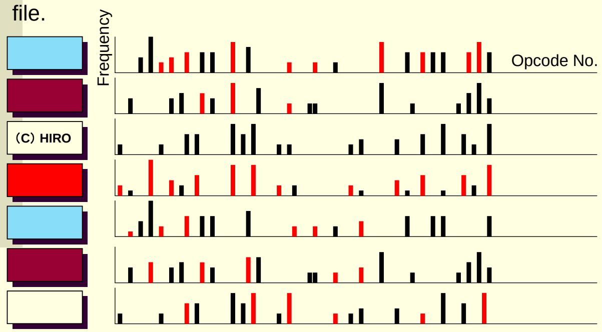
## A technique to hide a watermark

- Add dummy opcodes to all the methods so that opcode frequency of all methods become similar each other.

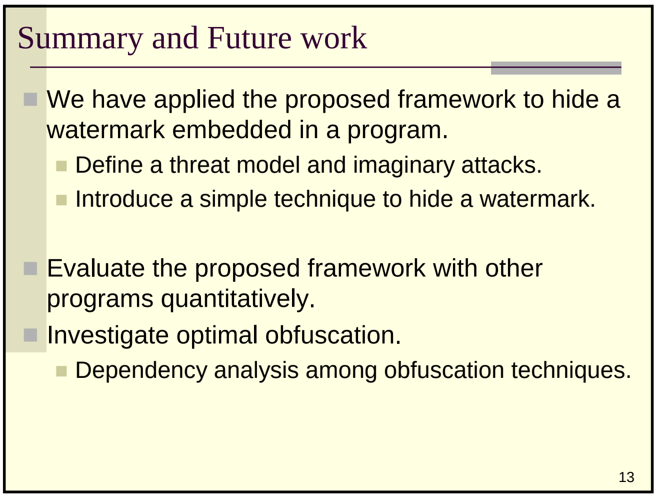


## Result of hiding a watermark

- Dummy opcodes were added to 10 methods of a Java class file.

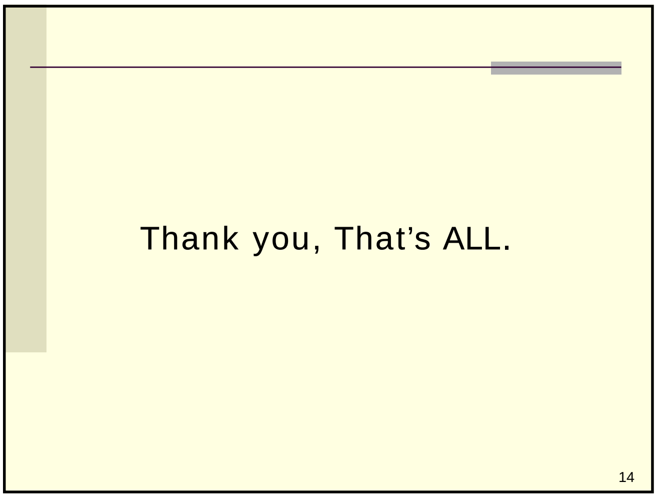


- It became quite difficult to find a watermarked method by inspecting opcode frequency.



## Summary and Future work

- We have applied the proposed framework to hide a watermark embedded in a program.
  - Define a threat model and imaginary attacks.
  - Introduce a simple technique to hide a watermark.
- Evaluate the proposed framework with other programs quantitatively.
- Investigate optimal obfuscation.
  - Dependency analysis among obfuscation techniques.



Thank you, That’s ALL.