Software Watermarking

Ginger Myles

mylesg@cs.arizona.edu

University of Arizona

Department of Computer Science

November 21, 2003

Introduction

- In this talk we will discuss...
  - What software watermarking is
  - Why we use software watermarking
  - Techniques used in software watermarking

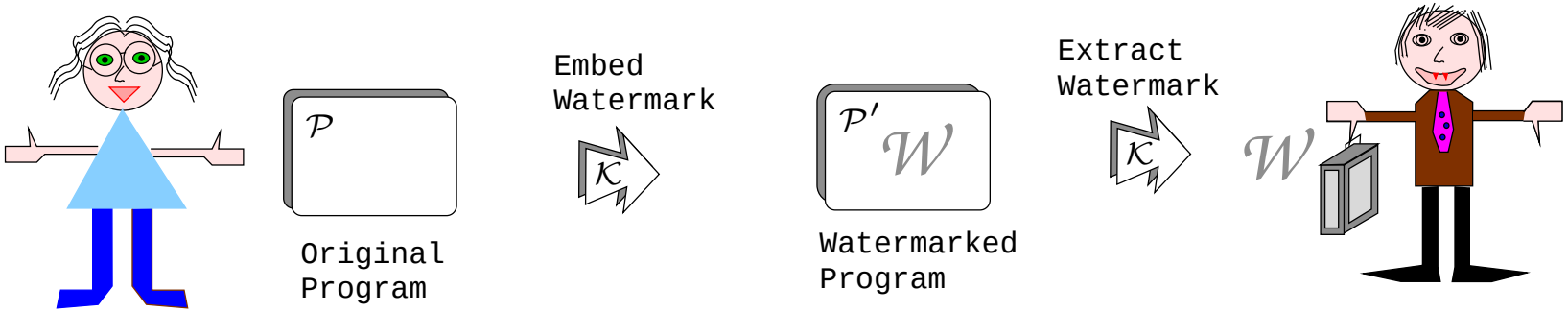
What is Software Watermarking?

- A technique used to aid in the prevention of software piracy.
- The idea is to embed a message w (the "watermark") into a program P, such that w uniquely identifies the owner of P (w is a copyright notice) or the purchaser of P (w is a fingerprint).

Software Watermarking

A watermarking system consists of two functions:

- `embed`$(P, w, key) \rightarrow P'$
- `recognize`$(P', key) \rightarrow w$



Why use software watermarking?

- Discourages illegal copying and redistribution.
- If we embed a copyright notice software watermarking can be used to provide proof of ownership.
- If we embed a fingerprint it can be used to trace the source of the illegal redistribution.
- Does not prevent illegal copying and redistribution.

Categories

- Static: the watermark is stored directly in the data or code sections of a native executable or class file.
- Dynamic: the watermark is stored in the run-time structures of the program.

Static Watermarking

- Make use of the features of an application that are available at compile-time.
  - Java application: constant pool table, method names, instruction sequence.

Static Watermarking

- Make use of the features of an application that are available at compile-time.
  - Java application: constant pool table, method names, instruction sequence.
- Advantages:
  - There are a variety of locations to embed a watermark.
  - Fairly simple to modify these features and still maintain the semantics of the application.

Static Watermarking

- Make use of the features of an application that are available at compile-time.
  - Java application: constant pool table, method names, instruction sequence.
- Advantages:
  - There are a variety of locations to embed a watermark.
  - Fairly simple to modify these features and still maintain the semantics of the application.
- Disadvantage:
  - Fairly simple to modify these features and still maintain the semantics of the application.

Static Watermarking Example 1

```
class C{
    method m1(){...}
    method m2(){...}
}
```

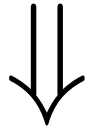
```
class C{
    method m1#hel(){...}
    method m2#lo(){...}
}
```

Static Watermarking Example 2

```
char V;
switch e{
    case 1 : V = 'G'
    case 2 : V = 'I'
    case 3 : V = 'N'
    case 4 : V = 'G'
    case 5 : V = 'E'
    case 6 : V = 'R'
}
```

Static Watermarking Example 3

```
class C{
    int a = 1;
    void m1(){...}
    void m2(){...}
}
```



```
class C{
    int a = 1;
    int wildcat# = 5;
    void m1(){...}
    void m2(){...}
}
```

Dynamic Watermarking

- Easter Egg Watermark
- Data Structure Watermark
- Execution Trace Watermark

Easter Egg Watermark

- A piece of code that gets activated for a highly unusual input to the application.
- The watermark is generally immediately perceptible by the user.
- Typically the watermark displays a copyright message or an unexpected image on the screen.

Easter Egg Watermark

- A piece of code that gets activated for a highly unusual input to the application.
- The watermark is generally immediately perceptible by the user.
- Typically the watermark displays a copyright message or an unexpected image on the screen.
- Disadvantages:
  - They are obvious.
  - They are easy to locate (using debugging techniques).
  - Once they have been located they are easy to remove.

Easter Egg Example

- Adobe Acrobat 4.0
- Select Help → About Plug-ins → Acrobat Forms and hold Ctrl+Alt+Shift while clicking on the credits button

Easter Egg Example

- Adobe Acrobat 4.0
- Select Help → About Plug-ins → Acrobat Forms and hold Ctrl+Alt+Shift while clicking on the credits button

Data Structure Watermark

- Embeds the watermark in the state of a program as the program is executed with a particular input sequence.
  - e.g. global, heap, and stack data
- Far more stealthy than easter egg watermark since no output is produced.

Execution Trace Watermark

- Embeds the watermark within the trace of the application as it is executed with a special input sequence.
- Differs from the data structure watermark in that the watermark is embedded in the application's instructions or address instead of the application's state.

Watermark Evaluation Properties

- Credibility: The watermark should be readily detectable for proof of authorship while minimizing the probability of coincidence.

Watermark Evaluation Properties

- Credibility: The watermark should be readily detectable for proof of authorship while minimizing the probability of coincidence.
- Data-rate: Maximize the length of message that can be embedded.

Watermark Evaluation Properties

- Credibility: The watermark should be readily detectable for proof of authorship while minimizing the probability of coincidence.
- Data-rate: Maximize the length of message that can be embedded.
- Perceptual Invisibility (Stealth): A watermark should exhibit the same properties as the code around it so as to make detection difficult.

Watermark Evaluation Properties

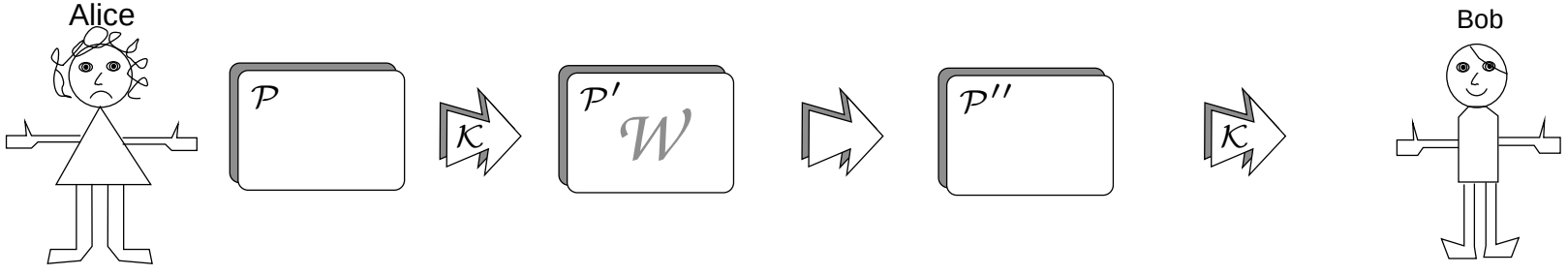
- Credibility: The watermark should be readily detectable for proof of authorship while minimizing the probability of coincidence.
- Data-rate: Maximize the length of message that can be embedded.
- Perceptual Invisibility (Stealth): A watermark should exhibit the same properties as the code around it so as to make detection difficult.
- Part Protection: A good watermark should be distributed throughout the software in order to protect all parts of it.

Watermark Evaluation Properties

- Resilience: A watermark should withstand a variety of attacks

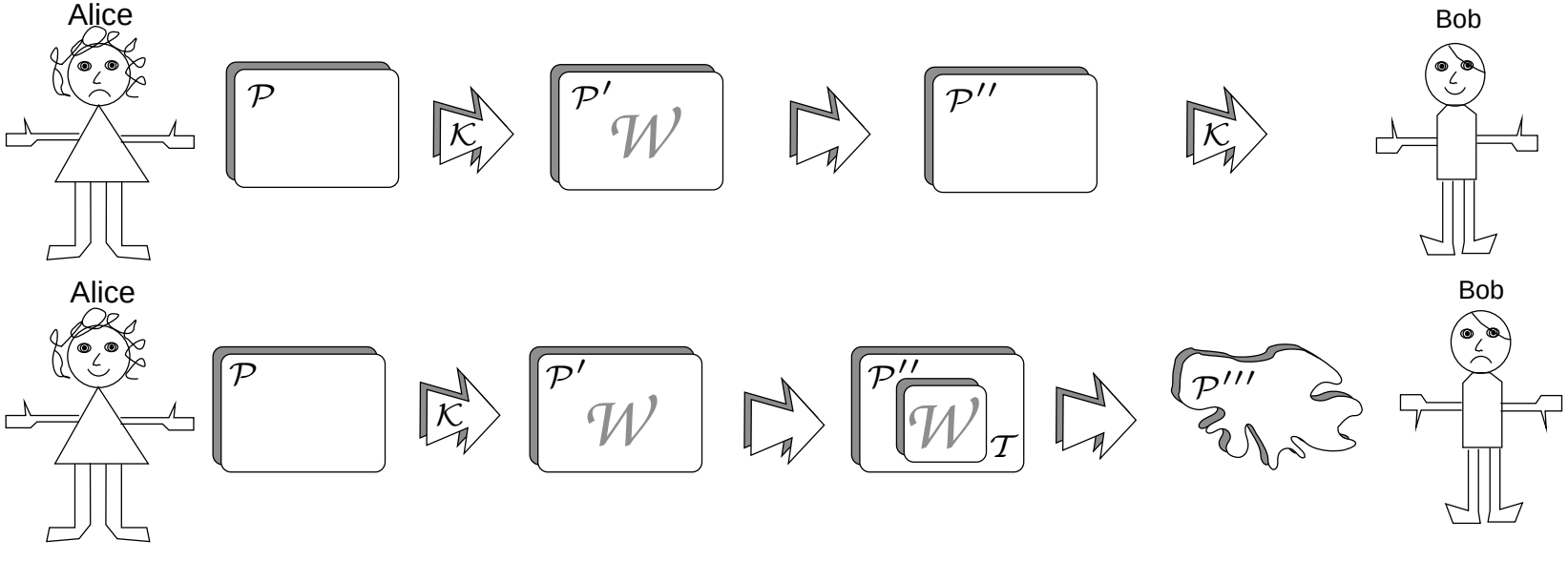
Watermark Evaluation Properties

- Resilience: A watermark should withstand a variety of attacks
  - Subtractive Attack: The adversary examines the (disassembled/de-compiled) program in an attempt to discover the watermark and to remove all or part of it from the code.



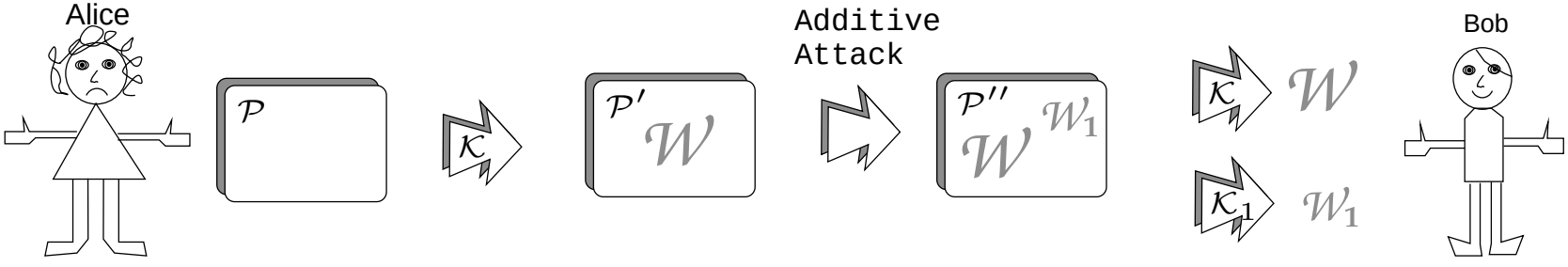
Watermark Evaluation Properties

- Resilience: A watermark should withstand a variety of attacks
  - Subtractive Attack: The adversary examines the (disassembled/de-compiled) program in an attempt to discover the watermark and to remove all or part of it from the code.



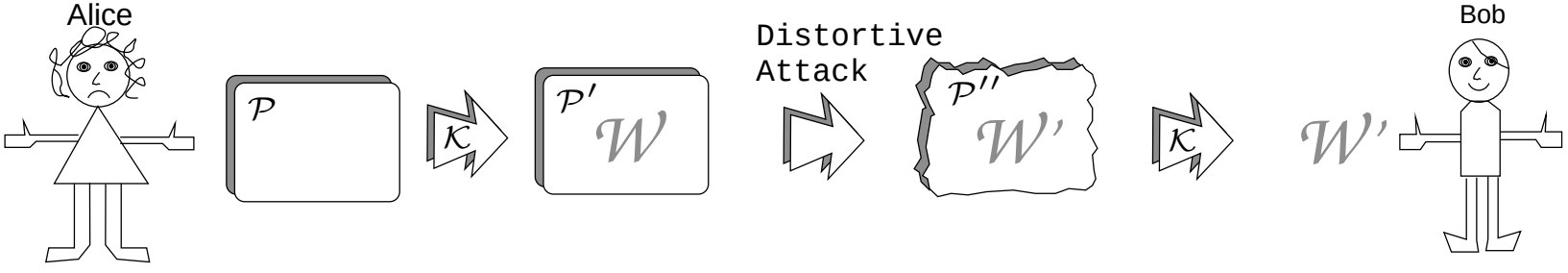
Watermark Evaluation Properties

- Resilience: A watermark should withstand a variety of attacks
  - Additive Attack: The adversary adds a new watermark in order to make it hard for the IP owner to prove that her watermark is actually the original.



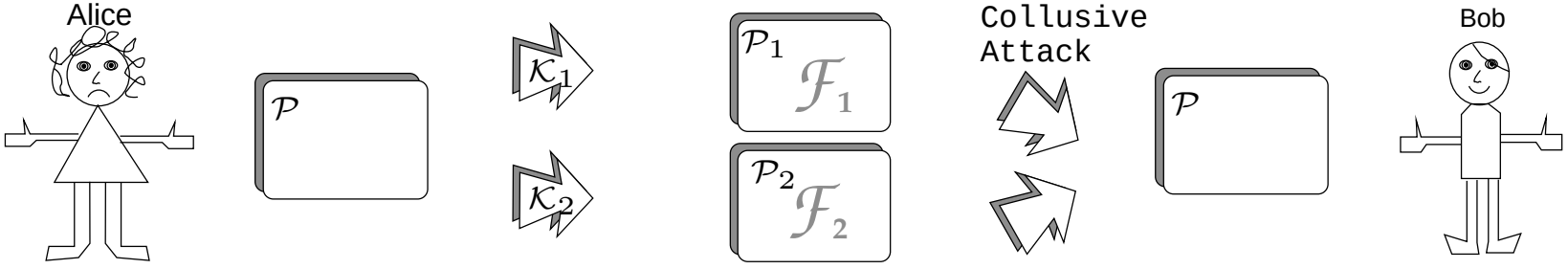
Watermark Evaluation Properties

- Resilience: A watermark should withstand a variety of attacks
  - Distortive Attack: A series of semantics-preserving transformations are applied to the software in an attempt to render the watermark useless.



Watermark Evaluation Properties

- Resilience: A watermark should withstand a variety of attacks
  - Collusive Attack: The adversary compares two copies of the software which contain different fingerprints in order to identify the location.



Watermark Evaluation Properties

- Credibility
- Data-rate
- Perceptual Invisibility (Stealth)
- Part Protection
- Resilience
  - Subtractive Attack
  - Additive Attack
  - Distortive Attack
  - Collusive Attack

Published Algorithms

- Monden, et. al.
- Davidson and Myhrvold
- Qu and Potkonjak
- Arboit

SandMark

- A research tool for studying software protection techniques for Java bytecode.
  - software watermarking, code obfuscation, and tamper-proofing
- Includes a variety of tools to study the strength of a watermarking algorithm.

Java Bytecode

- Java classes are compiled to class-files which contain the bytecodes of each method and a symbol table.

```
void whileI( ){
   int i = 0;
   while( i < 100 ){
      i++;
   }
}
```

```
0 iconst_0       // push int constant 0
1 istore_1       // store into local variable 1
2 goto 8         // first time no increment
5 iinc 1 1       // add 1 to local variable 1
8 iload_1        // load from local variable 1
9 bipush 100     // push a small int (100)
11 if_icmplt 5   // compare, if true goto 5
14 return        // return void when done
```

Opaque Predicates

- Opaque Predicate: A predicate P is opaque at a program point p, if at point p the outcome of P is known at embedding time. If P always evaluates to `True` we write P_p^T, for `False` we write P_p^F, and if P sometimes evaluates to `True` and sometimes to `False` we write $P_p^?$.

Opaque Predicates

- **Opaque Predicate:** A predicate P is opaque at a program point p, if at point p the outcome of P is known at embedding time. If P always evaluates to `True` we write P_p^T, for `False` we write P_p^F, and if P sometimes evaluates to `True` and sometimes to `False` we write $P_p^?$.
- **Opaque Method:** A boolean method M is opaque at an invocation point p, if at point p the return value of M is known at embedding time. If M always returns the value of `True` we write M_p^T, for `False` we write M_p^F, and if M sometimes returns `True` and sometimes `False` we write $M_p^?$.

Opaque Predicates

- Opaque Predicate: A predicate P is opaque at a program point p, if at point p the outcome of P is known at embedding time. If P always evaluates to `True` we write P_p^T, for `False` we write P_p^F, and if P sometimes evaluates to `True` and sometimes to `False` we write $P_p^?$.
- Opaque Method: A boolean method M is opaque at an invocation point p, if at point p the return value of M is known at embedding time. If M always returns the value of `True` we write M_p^T, for `False` we write M_p^F, and if M sometimes returns `True` and sometimes `False` we write $M_p^?$.
- Inserted to make it difficult for an adversary to analyze the control-flow of the application.

Monden Algorithm

- Embeds the watermark in a dummy method that is added to the application.
- The embedding is accomplished through a specially constructed sequence of instructions.
- Since the inserted method is never executed there is flexibility in how the instructions are constructed.
- Can disguise the method by adding a call to the method which is regulated by an opaque predicate.

Monden Algorithm

- SandMark implementation
  - Encode 8 bits of the watermark by replacing the operand of every BIPUSH instruction.
  - Encode 3 bits of the watermark by replacing each arithmetic instruction.

Monden Algorithm

- SandMark implementation
  - Encode 8 bits of the watermark by replacing the operand of every BIPUSH instruction.
  - Encode 3 bits of the watermark by replacing each arithmetic instruction.

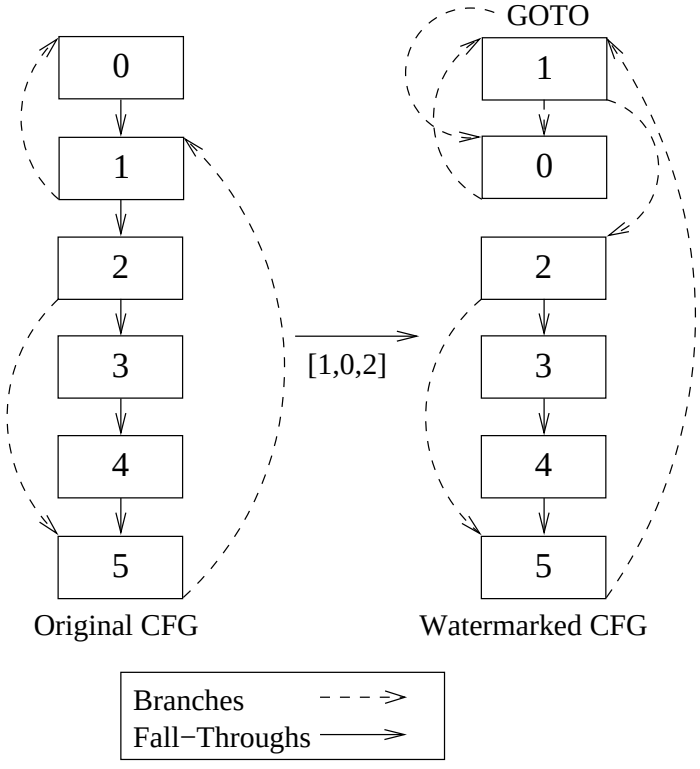
iadd	000
iand	001
ior	010
ixor	011
irem	100
idiv	101
imul	110
isub	111

DM Algorithm

- Embeds the watermark by reordering the basic blocks of the control flow graph.
- Since the functionality of the program must be maintained the blocks are relinked.

DM Algorithm

- Embeds the watermark by reordering the basic blocks of the control flow graph.
- Since the functionality of the program must be maintained the blocks are relinked.



QPS Algorithm

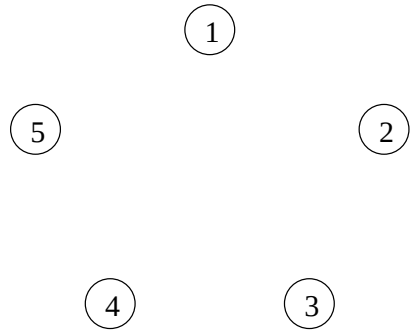
- Use the interference graph and the graph coloring problem to embed a watermark in the register allocation of an application.

Interference Graph

- Models the relationship between the variables in the procedure.
- Each variable in the procedure is represented by a vertex.
- If two variables have overlapping live ranges then the vertices are joined by an edge.
- The graph is colored so that we can assign the variables to registers so that we minimize the number of registers required and variables that are live at the same time do not share a register.

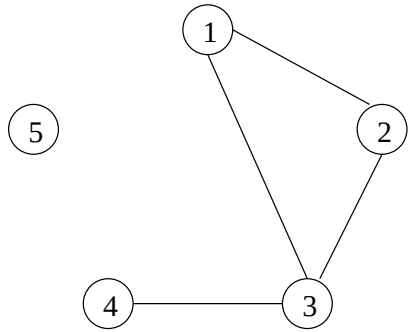
Interference Graph Example

```
v1 := 2 * 2
v2 := 2 * 3
v3 := 2 * v2
v4 := v1 + v2
v5 := 3 * v3
```



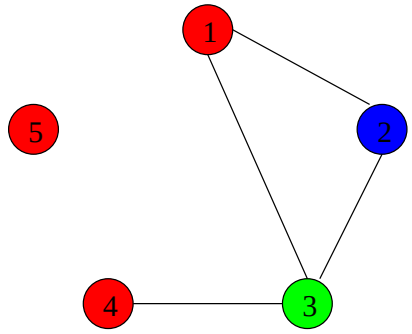
Interference Graph Example

```
v1 := 2 * 2
v2 := 2 * 3
v3 := 2 * v2
v4 := v1 + v2
v5 := 3 * v3
```



Interference Graph Example

```
v1 := 2 * 2
v2 := 2 * 3
v3 := 2 * v2
v4 := v1 + v2
v5 := 3 * v3
```

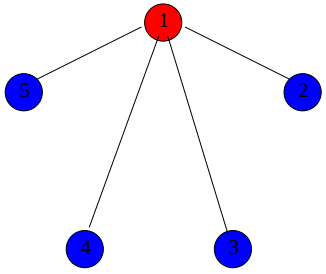


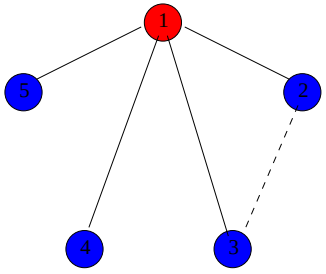
QPS Algorithm

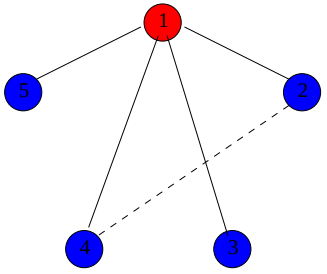
- Edges are added between chosen vertices in the graph based on the value of the message.
- Since the vertices are now connected, they cannot be assigned to the same register.

QPS Embedding Algorithm

for each vertex $v_i \in V$ which is not already in a triple
    if possible find the nearest two vertices v_{i_1} and v_{i_2}
    such that
        v_{i_1} and v_{i_2} are the same color as v_i,
        and v_{i_1} and v_{i_2} are not already in triple.
    if $m_i = 0$
        add edge (v_i, v_{i_1})
    else
        add edge (v_i, v_{i_2})
end for





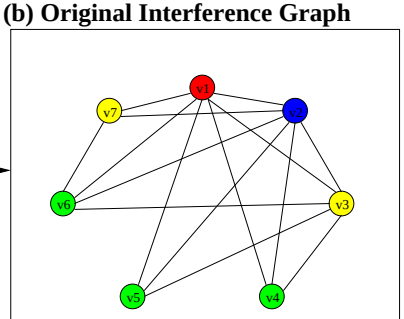


QPS Example

(a) Original Bytecode

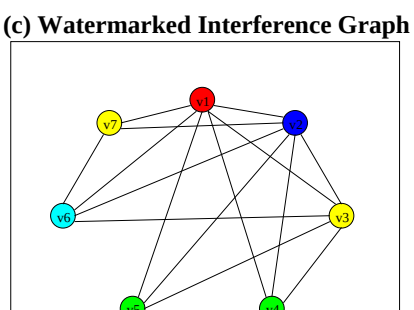
```
METHOD: fast_memcmp:([B[BI)Z
 0 : iconst_0
 1 : istore 3
 3 : iconst_0
 4 : istore_3
 5 : iconst_1
 6 : istore 3
 8 : iload_2
 9 : iconst_1
10 : isub
11 : istore_2
12 : iload_2
13 : iconst_0
14 : if_icmplt -> 21
17 : iconst_0
18 : goto -> 22
21 : iconst_1
22 : ifne -> 33
25 : iload 3
27 : invokestatic
30 : goto -> 34
33 : iconst_1
34 : ifne -> 60
37 : aload_0
38 : iload_2
39 : baload
40 : aload_1
41 : iload_2
42 : baload
43 : if_icmpeq -> 50
46 : iconst_0
47 : goto -> 51
50 : iconst_1
51 : istore 3
53 : iload_2
54 : iconst_1
55 : isub
56 : istore_2
57 : goto -> 12
60 : iload 3
62 : ireturn
```

(b) Original Interference Graph



Embed Watermark

(c) Watermarked Interference Graph



(d) Register Assignment Table

variable	register number
v1	0
v2	1
v3	2
v4	3
v5	3
v6	4
v7	2

(e) Watermarked Bytecode

```
METHOD: fast_memcmp:([B[BI)Z
 0 : iconst_0
 1 : istore 3
 3 : iconst_0
 4 : istore_3
 5 : iconst_1
 6 : istore 4
 8 : iload_2
 9 : iconst_1
10 : isub
11 : istore_2
12 : iload_2
13 : iconst_0
14 : if_icmplt -> 21
17 : iconst_0
18 : goto -> 22
21 : iconst_1
22 : ifne -> 33
25 : iload 4
27 : invokestatic
30 : goto -> 34
33 : iconst_1
34 : ifne -> 60
37 : aload_0
38 : iload_2
39 : baload
40 : aload_1
41 : iload_2
42 : baload
43 : if_icmpeq -> 50
46 : iconst_0
47 : goto -> 51
50 : iconst_1
51 : istore 4
53 : iload_2
54 : iconst_1
55 : isub
56 : istore_2
57 : goto -> 12
60 : iload 4
62 : ireturn
```

Arboit Algorithm 1

- k branching points throughout the application are randomly selected.
- At each branching point either $\wedge P^T_{b_i}$, $\vee \neg P^T_{b_i}$, or $\vee P^F_{b_i}$ is appended to the predicate at that location.
- The bits of the watermark are embedded through the opaque predicate that has been chosen.
  - Within the opaque predicate the bits can be encoded either as constants or by assigning a rank to each of the opaque predicates.

Sample Opaque Predicates

$\forall x, y \in$.	$7y^2 - 1 \neq x^2$
$\forall x \in$.	$2 \mid \lfloor \frac{x^2}{2} \rfloor$
$\forall x \in$.	$2 \mid x(x+1)$
$\forall x \in$.	$x^2 \geq 0$
$\forall x \in$.	$3 \mid x(x+1)(x+2)$
$\forall x \in$.	$7 \not\mid x^2 + 1$
$\forall x \in$.	$81 \not\mid x^2 + x + 7$
$\forall x \in$.	$19 \not\mid 4x^2 + 4$
$\forall x \in$.	$4 \mid x^2(x+1)(x+1)$

Arboit Algorithm 1 Example

```
class C{
  void m1(int a, int b){
    ...
    if(a <= b){...}
    else{...}
    ...
  }
}
```

```
class C{
  void m1(int a, int b){
    ...
    int c=1;
    if((a <= b) && (c*c >= 0)){...}
    else{...}
    ...
  }
}
```

Arboit Algorithm 2

- k branching points throughout the application are randomly selected.
- At each branching point, $M_{b_i}^T$ or $M_{b_i}^F$ is created and a method call is appended.
- The bits of the watermark are encoded in the opaque method through the opaque predicate that it evaluates.

Arboit Algorithm 2 Example

```
class C{
    void m1(int a, int b){
        ...
        if(a <= b){...}
        else{...}
        ...
    }
}
```

$\overset{\mathcal{W}}{\Rightarrow}$

```
class C{
    boolean m2(){
        int c = 1;
        return (c*c >= 0);
    }
    void m1(int a, int b){
        ...
        if((a <= b) && m2()){...}
        else{...}
        ...
    }
}
```

Summary

- Software watermarking is a technique that can be used to provide proof of authorship or permit the tracing of illegal copying.
- There are two general categories: static and dynamic.
- There are 5 watermark evaluation properties we use to determine the overall strength of an algorithm.

Readings and References

- Software Watermarking: Models and Dynamic Embeddings, Collberg, Thomborson, Principles of Programming Languages 1999, POPL'99, `citeseer.nj.nec.com/collberg99software.html`.
- A Practical Method for Watermarking Java Programs, Monden, Iida, Matsumoto, Inoue, Torii, Compsac 2000.
- Method and System for Generating and Auditing a Signature for a Computer Program, Davidson, Myhrvold, US Patent 5,559,884.
- Software Watermarking Through Register Allocation: Implementation, Analysis and Attacks, Myles, Collberg, ICISC 2003.

Readings and References

- A Method for Watermarking Java Programs via Opaque Predicates, Arboit, ICECR-5.
- A Technique for Discouraging Piracy of Java Applications, Myles, Collberg, submitted to Financial Cryptography 2004.
- Sandmark–A Tool for Software Protection Research Collberg, Myles, Huntwork, IEEE Security and Privacy,July-August 2003 (Vol. 1, No. 4).
- The Easter Egg Archive, `www.eeggs.com`