Software Watermarking

Christian Collberg
collberg@cs.arizona.edu
Department of Computer Science
University of Arizona

Software Watermarks & Fingerprints

- Embed a unique identifier in a program to trace software pirates.
- Watermarking
  1. discourages theft,
  2. allows us to prove theft.
- Fingerprinting
  3. allows us to trace violators.
- We want to develop algorithms that produce marks that are hard to destroy, are stealthy, have a high bit-rate, and have little performance overhead.

Malicious Reverse Engineering

- Alice and Bob are competing software developers.
- Bob reverse engineers Alice’s program and includes parts of it in his own code.
- Easier with Java bytecode, .NET, ANDF…
- ⇒ Alice obfuscates her code.
Alice is a media publisher. She packages her media into a **cryptolope**.

Bob tampers with the software player to extract the decrypted media.

InterTrust, Intel, IBM, Xerox, Microsoft, . . .

⇒ Alice **obfuscates**, **watermarks**, and **tamper-proofs** the player.

Alice is a software developer.

Bob buys one copy of Alice's application and sells copies to third parties.

⇒ Alice **watermarks/fingerprints** her program.

**Software Watermarking**

- **Software Protection Overview**
- **Software Watermarking Overview**
- **Static Software Watermarking Algorithms**
- **Attacks on Software Watermarks**
- **Dynamic Software Watermarking**
- **The SANDMARK tool**
- **Conclusion**

Embed an integer $W$ in program $P$ such that:
- $W$ is resilient against automated attacks
- $W$ is stealthy
- $W$ is large (high bitrate)
- the overhead (space and time) is low
Naive Approaches

```java
String watermark = "Copyright 2004...";
```

- High bitrate, little overhead, unstealthy.

```
switch (E) {
    case 1: {...}
    case 5: {...}
    case 9: {...}
}
```

- Low bitrate, no overhead, stealthy, easy to destroy.

Watermarking Transformations

- Naive approaches typically use **reordering** (of statements, basic blocks, ...) or **renaming** (of registers, methods, ...):

```java
switch (E) {
    case 1: {...}
    case 5: {...}
    case 9: {...}
}
```

- More powerful approaches extend program semantics or alter program statistics:

Attacks on Software Watermarks

- **Additive** attack
- **Distortive** attack
- **Semantics-preserving transformations**
- **Collusive** attack

Software Protection Overview

Software Watermarking Overview

Static Software Watermarking Algorithms

Attacks on Software Watermarks

Dynamic Software Watermarking

The **S**AND**M**ARK tool

Conclusion
A watermarked media object is embedded in the program’s static data segment.

“Essential” parts of the program are steganographically encoded into the media.

If the watermarked image is attacked, the embedded code will crash.

—— Moskowitz & Cooperman


**Stern et al.**

Embed mark by adjusting frequency of instruction patterns:
1. Replace instruction groups by semantic equivalents.
2. Insert redundant instruction groups.


—— Qu-Potkonjak

Embed the mark by adding constraints (extra edges) to the register interference graph.

Easy to attack by random register re-numbering.


**REORDER**

— Davidson & Myhrvold

The watermark is encoded in the basic block sequence \(\langle B_5, B_2, B_1, B_6, B_3, B_4 \rangle\).

US Patent 5,559,884, 1996, Microsoft

**EXTEND SEMANTICS**

— Moskowitz & Cooperman

```cpp
class Main {
    const Picture C =
        Code R = Decode(C);
        Execute(R);
    }
```


**ALTER STATS**

— Stern et al.

```
C = A + 1

X = A + B

C = A + 1
```


**RENAME**

— Qu-Potkonjak

Original

```
5
  2
  |
  1
```

Marked

```
5
  2
  |
  1
```

New coloring

```
5
  2
  |
  1
```

Bogus branches tie the watermark CFG to the program.

Basic blocks are marked so the watermark graph can be found.

Embed the mark in the result of a static analysis problem.

Algorithm introduces many “weird” constants. Unstealthy, since 92% of all literal integers are $2^n$, $2^n + 1$, $2^n - 1$.
Obfuscating Transformations

- SPLIT
- Increase Dimensions
- Increase Nesting
- REF
- DEREF
- RENAME
- REORDER
- MERGE

Boolean Splitting Obfuscation

```java
public class C {
    static int gcd(int i, int j) {
        int t9, t8, q7, q6, q4, q3;
        q7=9;
        for (; ; ) {
            if ( i%j ==0) { t9 =1; t8 = 0 ; }
            else { t9 =0; t8 =0; }
            q4=t8 ; q6=t9 ;
            if ( ((q4*q6)!=0) )
                return j ;
            else {
                k=i%; i=j; j=k;
            }
        }
        public static void main(String[] Z1) {
            System.out.print("Answer:");
            System.out.println(gcd(100, 10));
        }
    }
}
```

Bogus Branch Obfuscation

```java
public class C {
    static int gcd(int i, int j) {
        int t9, t8, q7, q6, q4, q3;
        q7=9;
        for (; ; ) {
            if ( i%j ==0) { t9 =1; t8 = 0 ; }
            else { t9 =0; t8 =0; }
            q4=t8 ; q6=t9 ;
            if ( ((q4*q6)!=0) )
                return j ;
            else {
                if ( (((q7+q7*q7)%2!=0)?0:1)!=1 ) return 0;
                q3=i%; i=j; j=q3;
            }
        }
        public static void main(String[] Z1) {
            System.out.print("Answer:");
            System.out.println(gcd(100, 10));
        }
    }
}
```
String Encoding Obfuscation

```java
public class C {
    static int gcd(int i, int j) {
        // As before
    }

    public static void main(String[] a)
    {
        System.out.print(Obfuscator.DecodeString("u00AB\u00CD\u00AB\u00CD" +
                     "\uFF84\u2A16\u5D68\u2AA0" +
                     "\u388E\u91CF\u5326\u5604");
        System.out.println(gcd(100, 10));
    }
}
```

Primitive Promotion Obfuscation

```java
public class C {
    static Integer get0(Integer i, Integer j) {
        Integer K, L, M, N;
        int t9, t8; K = new Integer(9);
        for (; ; ) {
            if ((i.intValue() % j.intValue()) == 0) { t9 = 1; t8 = 0; }
            else { t9 = 0; t8 = 0; }
            M = new Integer(t8);
            L = new Integer(t9);
            if (((M.intValue()) ^ (L.intValue())) != 0)
                return new Integer(j.intValue());
            else {
                if (((K.intValue()) + K.intValue()) % 2 != 0)
                    return new Integer(0);
                N = new Integer(i.intValue());
                i = new Integer(j.intValue());
                j = new Integer(N.intValue());
            }
        }
    }

    public static void main(String[] Z1)
    {
        System.out.print(Obfuscator.get0((String)new Object[] { "String as before" }[0]));
        System.out.println((Integer) get0(new Object[] {
            new Integer(100), new Integer(10)}[0], new Integer(10))[1].intValue());
    }
}
```

Method Signature Obfuscation

```java
public class C {
    static Object get0(Object[] I) {
        Integer K, L, M, N;
        int t9, t8; K = new Integer(9);
        for (; ; ) {
            if (((Integer) I[0]).intValue() % ((Integer) I[1]).intValue() == 0)
                return y;
            t = x % y; x = y; y = t;
        }
    }

    public static void main(String[] Z1)
    {
        System.out.print((Object[]) get0(new Object[] {
            new Integer(100), new Integer(10)}[0], new Integer(10))[1].intValue());
    }
}
```

This is what we started out with...

```java
public class C {
    static int gcd(int x, int y) {
        int t;
        while (true) {
            boolean b = x % y == 0;
            if (b) return y;
            t = x % y; x = y; y = t;
        }
    }

    public static void main(String[] a)
    {
        System.out.print("Answer:");
        System.out.println(gcd(100, 10));
    }
}
```
Collusion Protection by Obfuscation

Obfuscation can also be used to protect against collusive attacks.

```
public class C {
    static Object get0(Object[] I) {
        int r, q, j; K = new Integer(9);
        j = 2; j = 60 - (j + 1); ++j; j = 60 - j;
        for (; ; ) {
            if (((Integer) I[0]).intValue() % ((Integer) I[1]).intValue()) == 0)
                r = 1; q = 0;
            else
                r = 0; q = 0;
            M = new Integer(r);
            J = new Integer(q);
            if (((Integer) I[0]).intValue() == 0)
                return new Integer(((Integer) I[1]).intValue());
            else {
                if (((K.intValue() + K.intValue()) % 2) != 0)
                    return new Integer(0);
                N = new Integer(((Integer) I[0]).intValue() % ((Integer) I[1]).intValue());
                I[0] = new Integer(N.intValue());
                I[1] = new Integer(N.intValue());
                } }
    public static void main(String[] Z1) {
        int j = 2; int i = 2; i = 80 - (i + 1); ++i; i = 80 - i;
        System.out.println(((Integer) get0(new Object[] {
            (String) new Object[] { "String as before" }[0],
            (Integer) new Object[] {
                new Integer(90), new Integer(90) }[0],
                (Integer) new Object[] {
                    new Integer(100), new Integer(100) }[1],
                new Integer(100), new Integer(100) }[1]),
            intValue()));
    }
}
```

Static vs. Dynamic Watermarking

- Static algorithms are vulnerable to semantics-preserving code transformations.
- Dynamic algorithms extract the mark from the state of the program when run on a secret key input sequence.

Software Protection Overview

- Software Watermarking Overview
- Static Software Watermarking Algorithms
- Attacks on Software Watermarks
- Dynamic Software Watermarking
- The SandMark tool

Conclusion
The watermark is embedded in the topology of a dynamic graph structure, built at runtime but only for the special input sequence $I_1, \ldots, I_k$.

Why? Shape-analysis is hard.

ACM Principles of Programming Languages, POPL'99

---

**CT — Example**

```java
public class Simple {
    static void P(String i) {
        System.out.println("Hello "+ i);
    }
    public static void main(String args[]) {
        P(args[0]);
    }
}

class Watermark extends java.lang.Object {
    public Watermark edge1, edge2;
}
```

---

**CT — Example...**

```java
public class Simple_W {
    static void P(String i, Watermark n2) {
        if (i.equals("World")) {
            Watermark n1 = new Watermark();
            Watermark n4 = new Watermark();
            n4.edge1 = n1; n1.edge1 = n2;
            Watermark n3 = (n2 != null)?n2.edge1:new Watermark();
            n3.edge1 = n1;
        }
        System.out.println("Hello "+ i);
    }
    public static void main(String args[]) {
        Watermark n3 = new Watermark();
        Watermark n2 = new Watermark();
        n2.edge1 = n3; n2.edge2 = n3;
        P(args[0], n2);
    }
}
```

---

**CT — Semantics-Preserving Attacks**
CT — Error-Correcting Graphs

Current work: Define classes of graphs with efficient embedding and error-correcting properties.

CT — Error-Correcting Graphs

PBW — Embedding

The watermark is split into a large number of redundant pieces using an error correcting code.

PBW — Code Generation

Several different types of code can be generated to increase stealth.

Current work: Define classes of graphs with efficient embedding and error-correcting properties.

PBW — Embedding

The watermark is split into a large number of redundant pieces using an error correcting code.

PBW — Code Generation

Several different types of code can be generated to increase stealth.

Ensure to protect against simple branch-flips!
The program is run with the secret input.
Branches are monitored and a bitstream extracted.
Using the error correcting code, the watermark pieces are extracted from the bitstream and recombined into the watermark.

Choose \( p_1, \ldots, p_k \) pairwise relatively prime, split watermark into \( \frac{k(k-1)}{2} \) pieces of the form \( W \equiv r \mod p_i p_k \).

Use an enumeration scheme to turn these into integers, run through a block-cipher, embed into program.

The Generalized Chinese Remainder Theorem allows \( W \) to be reconstructed from \( \lceil \frac{k}{2} \rceil \) pieces.

Slide a 64-bit window across the bitstream. Throw out those that don’t meet randomness criteria.

Reconstruct \( W \equiv r \mod p_i p_k \) by inverting enumeration scheme.

Build a graph of statements inconsistent wrt to GCRT. Compute “most consistent” subgraph.

With a 256-bit mark and 100 pieces, the attacker needs to double the number of branch instructions in the program in order to destroy the mark.
**PBW — Adding Branches Attack**

- How much does CaffeineMark slow down versus wrt the number of branches the attacker added?

  By doubling the number of branches, the attacker slows down the program by about 40%.

**PBW — Time Overhead**

- How does the program slow down as the number of watermark pieces is increased?
- The more pieces we insert, the more pieces the attacker needs to destroy.

  For Jess we avoid the hotspots, so slowdown is negligible.
  For CaffeineMark we can’t avoid the hotspots, so slowdown is > 50%.

**Nagra-Thomborson**

- Embed mark in which threads execute which basic blocks.
- Can have huge performance degradation.
- Why? Parallelism-analysis is hard.

Software Protection Overview
Software Watermarking Overview
Static Software Watermarking Algorithms
Attacks on Software Watermarks
Dynamic Software Watermarking
The SandMark tool
Conclusion
**SANDMARK — A Software Protection Tool**

- 33 obfuscation algorithms
- 16 watermarking algorithms
- 6 birthmarking algorithms
- 6 bytecode diff algorithms
- bytecode visualization tools
- 6 software complexity metrics
- large toolbox (static analysis, graphs, ...)

**A Session with SANDMARK**

We obfuscate to protect against reverse engineering and collusive de-watermarking attacks.

**A Session with SANDMARK**

We extract the watermark to prove ownership.

**A Session with SANDMARK**

To simulate a manual attack we examine the obfuscated/watermarked program using various static analysis tools.
To simulate an automatic attack we use SandMark’s obfuscators (“SoftStir”) to attack the watermark.

Many interesting problems left to work on!
- Formal models of attack and stealth.
- Combining error correction and tamper-proofing.
- Watermarking other languages.

Download from sandmark.cs.arizona.edu.