Software Theft Detection Through Software Watermarking

Ginger Myles
mylesg@cs.arizona.edu

University of Arizona
Department of Computer Science
and
IBM Almaden Research Center
Introduction

- In this talk we will discuss...
  - Our motivation
  - Definition of software watermarking
  - Various software watermarking techniques
  - More in depth look at the Branch-Based algorithm.
Motivation

3 Major Threats

- **Software Tampering:** the illegal modification of a program to circumvent licence checks, to obtain access to digital media protected by the software, etc.

- **Malicious Reverse Engineering:** the extracting of a piece of a program in order to reuse it in one's own.

- **Software Piracy:** the illegal reselling of legally obtained copies of a program.
Motivation

- Piracy is widespread and decentralized
  - Rich distribution formats
  - Availability of high speed Internet and peer-to-peer systems
  - Illegal copies are difficult to detect
Motivation

Ramifications

- Obvious: Software companies
- Peripheral: Hardware companies
- Not-obvious: Everyday people
What is Software Watermarking?

- A technique used to aid in the prevention of software piracy.
- Embed a unique identifier in a program.

<table>
<thead>
<tr>
<th>Watermarking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Same identifier</td>
</tr>
<tr>
<td>Copyright notice</td>
</tr>
<tr>
<td>Discourages theft</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Fingerprinting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Different identifier</td>
</tr>
<tr>
<td>Customer identification</td>
</tr>
<tr>
<td>Trace illegal copies</td>
</tr>
</tbody>
</table>

- Discourages but does not prevent illegal copying and redistribution.
A watermarking system consists of two functions:

- \( \text{embed}(P, w, key) \rightarrow P' \)
- \( \text{recognize}(P', key) \rightarrow w \)
Software Watermarking

We want to develop an algorithm such that when we embed the watermark $W$ in the program $P$

- $W$ is resilient to various attacks.
- $W$ is stealthy.
- $W$ is large (high bit-rate).
- The overhead (space and time) is low.
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.
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.
**Attacks on Software Watermarks**

**Distortive Attack:** A series of semantics-preserving transformations are applied to the software in an attempt to render the watermark useless.
Collusive Attack: The adversary compares two copies of the software which contain different fingerprints in order to identify the location.
Naive Watermarking Techniques

Constant String

```java
String watermark = "Copyright 2006 ...";
String fingerprint = "CC Number 1234 ...";
```

Easy to attack, low stealth, high bit-rate, little overhead.
Switch Encoding

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

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

- Easy to attack, **high stealth**, low bit-rate, no overhead.
Watermarking Transformations

Naive approaches:

- Renaming

![Diagram](image)
Watermarking Transformations

Naive approaches:

- Renaming

- Reordering
Watermarking Transformations

Advanced approaches:

- Alter program statistics

![Diagram showing watermarking process]
Watermarking Transformations

Advanced approaches:

- Alter program statistics

- Extend program semantics
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.

- **Abstract**: the watermark is stored in the semantics of the program through the use of abstract interpretation.
Static Watermarking

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

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

  **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 sequences.

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.
Reordering transformation.

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

Easy to attack, low stealth, low bit-rate, little overhead.

Renaming transformation.

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

Easy to attack by random register re-numbering.

Easy to attack, high stealth, low bit-rate, no overhead.


Bowling Green State University
Statistics altering transformation.

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

Moderately easy to attack, fairly stealthy, high bit-rate, minimal impact on performance, moderate impact on size.

Semantics extending transformation.

Bogus branches tie the watermark CFG to the program.

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

Moderately difficult to attack, low stealth, high bit-rate, large overhead.

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
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.
Semantics extending transformation.

Embed the mark in the topology of a dynamic graph structure, built at runtime for a special input sequence (the secret key).

Why? Shape-analysis is hard.

Hard to attack, medium stealth, high bit-rate, medium overhead.

POPL 1999.

Bowling Green State University
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.
Dynamic WMs — Collberg et al.

Semantics extending transformation.

Watermark is embedded in the dynamic branching behavior of the application by modifying the sequence of branches taken and not taken on the secret input.

Hard to attack, low stealth, high-bit rate, medium overhead.

PLDI 2004
Abstract Watermark

- Watermark is embedded through the use of abstract interpretation.
- Watermark is extracted by analyzing the concrete semantics of the code.
- Is static in that recognition does not require execution of the program.
- Is dynamic in that the watermark is hidden in the semantics of the program.
Dynamic WMs — Cousot and Cousot

Semantic extending transformation.

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

Algorithm introduces many “weird” constants.

93% of all integers are less than 1000, 63% of all literal integers in real programs are $2^n$, $2^n + 1$, $2^n - 1$

Hard to attack, low stealth, high bit-rate, low overhead.

POPL 2004.

Bowling Green State University
Dynamic execution trace technique.
Semantics extending transformation.
Incorporates ideas from code obfuscation and tamper detection to embed an authorship mark and a fingerprint mark.
International Information Hiding Workshop, 2005.
Code is embedded which generates the fingerprint as the program executes on a secret input sequence $I_1, \ldots, I_n$.

Fingerprint Branch Function (FBF)

**Key:** We link proper program execution and fingerprint generation.
Select a sequence of branch instructions.

Those branch instructions are replaced by calls to our fingerprint branch function.

FBF will return execution control to the target of the original branch.
Branch-Based WM Example

```c
void main(int argc, char *argv[]){
    int x = atoi(argv[1]);
    printf("%d! = %d\n", x, factorial(x));
}

int factorial(int x){
    if(x == 1)
        return x;
    return (x * factorial(x-1));
}

long key = seed;

void main(int argc, char *argv[]){
    int x = branchFunction(argv[1]);
    branchFunction("%d! = %d\n", x, branchFunction(x));
}

int factorial(int x){
    if(x == 1)
        return x;
    return (x * factorial(x-1));
}

void branchFunction(void *x){
    key = evolveKey(key);
    return;
}
```
Fingerprint Branch Function basis:

- Based on a code obfuscation technique designed to disrupt static disassembly by exploiting the assumption that a function call returns to the instruction immediately following the call instruction (Linn and Debray).

- Execution is rerouted through the branch function by converting `jmp` and `call` instructions to a call to a single branch function.

- The correct target is identified based on the call location.
  
  \[ T[h(j_i)] = t_i - j_i \]

- Once the target address is computed the return address on the stack is overwritten.
Fingerprint Branch Function

Performs three important tasks:

1. Performs an integrity check of the program producing the value $v_i$.

2. Generate the next key using a secure one-way function, the previous key, and the integrity check value.
   \[ k_{i+1} = SHA1(v_i, k_i) \]

3.Uses $k_{i+1}$ to identify the instruction where execution will resume.

Location for authorship mark embedding.

\[ k_{i+1} = SHA1[(k_i \oplus AM)||v_i)] \]
Branch-Based WM — Implementation Overview

Original Application

- $f_1$
- $f_2$
- $f_3$

$\Rightarrow$ annotate $\Rightarrow$ Annotated Application

- $f_1$ mark()
- $f_2$ mark()
- $f_3$ mark()

$\Rightarrow$ $I_0, I_1, \ldots$ trace $\Rightarrow$ execution trace

- $f_2$
- $f_3$

Annotated Application

- $f_1$ mark()
- $f_2$ mark()
- $f_3$ mark()

$\downarrow$ embed $\uparrow$

- $AM, seed$

Watermarked Application

- $f_1$
- $f_2$
- $f_3$

+ $FM$
Branch-Based WM — Embedding

- Native x86 executables and Java applications.
- Embedding process involves five steps:
  1. FBF construction
  2. Branch instruction selection
  3. Branch instruction replacement
  4. Table construction
  5. FM calculation
FBF Construction:

1. A simple checksum integrity check producing the value $v_i$.

2. Generation of the next key $k_{i+1}$ using the secure hash function SHA-1, $k_i$, $v_i$, and AM.

3. Identification of the displacement to the target instruction via $d_{i+1} = T[h(k_{i+1})]$.

4. Computation of the return location by adding the displacement $d_{i+1}$ to the return address.
Branch-Based WM — Embedding

Branch instruction selection

The execution trace identifies a set of function $F$ through which execution passes.

For each $f \in F$ we select $B_f = b_1, \ldots, b_k$.

Each $b_i$ is selected such that it resides on a deterministic path through the function.

Prevents irregular key evolution and improper program behavior.
Branch instruction replacement

- Each $b_i \in B_f$ is replaced with a call to the FBF.
- Maintain a mapping between the branch-target displacement and the current key $(k_i \rightarrow d_i)$.

$$\theta_f = k^f_1 \rightarrow d^f_1, \ldots, k^f_n \rightarrow d^f_n$$

- Because of this branch instructions are replaced in execution order.
Branch-Based WM — Embedding

Table construction

- Fingerprint generation and proper program execution are linked by using the current key to identify the branch-target displacement.
- Each branch-target displacement is stored in a table inserted in the data section of the binary.
- We use each key-displacement mapping $\theta_f$ to construct a table $T$.
- A perfect hash function is used to map each key to a unique slot in the table.
  \[ h = k_1, k_2, \ldots, k_m \rightarrow 1, 2, \ldots, n, m \leq n \]
- The displacements are stored in the table such that $T[h(k_i)] = d_i$. 

Bowling Green State University

February 13, 2006 – p. 4
Fingerprint mark calculation

Each \( f \in F \) will produce a final function key \( k_f \).

Each of these keys is combined in a commutative way to produce the fingerprint mark for the program.

Using the execution trace we know what order the functions will be executed and how many times. From this we combine the function keys to produce the fingerprint mark.
The only static variation in differently fingerprint instances of a program is in the displacement table.

By distributing the software without the table, software companies can link a specific copy of the software to the purchaser.
Evaluation

- High resistance to transformation
- Low stealth
- High bit-rate
- Minimal overhead
Summary

- Software piracy is widespread, decentralized problem.
- Software watermarking is one technique we can use to address the issue.
- Software watermarking can be used to provide proof of authorship or permit the tracing of illegal copying.
- A variety of algorithms have been proposed, but no perfect algorithm has yet to be developed.
Questions?