Watermark & Fingerprinting

Watermark: a secret message embedded into a cover message.

- Visible or invisible marks.
- Watermarking discourages theft.
- Fingerprinting allows us to trace violators.

Copyright © 2002 C. Collberg
Software Watermarking

data rate: \( \leq 1000 \) bits?

cover program: source code/object code? typed/untyped?
architecture-neutral/native binary?

threat-model: semantics-preserving transformations
(translation, optimization, obfuscation)?

logistics: generation, distribution, bug-reports?

Watermarking Overview II

Attacks on Media Watermarks

- We trade-off between
  1. stealth (we want imperceptible marks),
  2. bit-rate (we want to embed much data),
  3. resilience (we want the mark to withstand attacks).
- Attacks: compression, scaling, cropping, blurring, rotation...
Static Code Watermarks

Reorder statements

\[ S_1 \Rightarrow S_2 \Rightarrow S_3 \Rightarrow S_4 \]

Reorder switches

\[ \text{case e of } \{ \begin{array}{c} \text{case 1 : } S_1 \\ \text{case 2 : } S_2 \\ \text{case 3 : } S_3 \\ \text{case 1 : } S_1 \end{array} \} \]

- \( m \)-cases \( \Rightarrow O(m \log m) \) watermarking bits.

- Kirowski et al.: Store watermarks in the register allocation.

Static Software Watermarks

1. **Static Data Watermarks** are embedded in the initialized data (string) section of the program:
   > strings ./bin/netscape | \ grep -i copyright
   Copyright (C) 1995, Thomas G. Lane

2. **Static Code Watermarks** are embedded in the text (code) section of the program.

Static Code Watermarks – Microsoft

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

Static Data Watermarks – DICE Method

```java
class Main {
    const Picture C = 
        [it looks like a picture like this]
        ... 
    Code R = Decode(C);
    Execute(R);
}
```

- A watermarked media object is embedded in the program’s static data segment.
- “Essential” parts of the program are steganographically encoded into the static data.
Dynamic Software Watermarks

The program is run with input $I = I_1 \ldots I_k$. The watermark is embedded in

1. **Easter Egg**
   - the unexpected behavior of the program.
2. **Data Structure**
   - a variable $V$.
3. **Execution Trace**
   - the program’s instruction or address trace.

Easter Egg Watermarks

- The watermark performs an action that is immediately perceptible.

  
  \[
  \downarrow
  \]

  Extraction is trivial.

- Effects must not be too subtle.

  - www.eeggs.com/1r.html

Attacks on Static Watermarks

- Effective threat models:
  1. locality-improving optimizations,
  2. code/data obfuscations,
  3. decompile + recompile,
  4. binary translation,
  5. ...

C–Dynamic

Security Through Obscurity

CSCG 620

C–Dynamic

Copyright © 2002 C. Colberg

University of Arizona
Dynamic Graph Watermarks

- We embed watermarks in the topology of a graph:
  1. Hard for Bob to analyze (aliasing),
  2. Easier for Alice to tamper-proof.
- A recognizer $\mathcal{R}$ extracts the watermark by examining heap-objects after input $I$.

Dynamic Data Structure Watermark

- The watermark is embedded within the state (globals, heap, stack) of the program.
- A recognizer $\mathcal{R}$ extracts the watermark by examining the state after input $I$.
- No “special” output is produced.
- $\mathcal{R}$ is not shipped.

Dynamic Execution Trace Watermark

- The watermark is embedded within the instruction or address trace.
- Watermark extraction:
  - the actual trace,
  - some statistical property of the trace.
- Threat model:
  - optimizations,
  - obfuscations,...
Graph Embedding: Enumeration

- $n$ is represented by the index of the graph $G$ in some enumeration.
- We must, efficiently, be able to
  1. given $n$, generate the $n$th graph,
  2. given $G$, find $G_n$ index $n$.
- Oriented parent-pointer trees
  1. 655 nodes $\Rightarrow 1024$-bit integer,
  2. bit-rate: 1.36 bits per word.

Graph Recognition

- For each input $I_i$, one graph component is built.
- After input $I_i$ the entire graph has been assembled.
- To find the root of $G$ we only consider nodes created during the processing of $I_i$. 
- $G$ is a circular linked list.
- An extra pointer field encodes a base-$k$ digit:
  - null-pointer $\Rightarrow 0$
  - self-pointer $\Rightarrow 1$
  - next node pointer $\Rightarrow 2, \ldots$
Tamperproofing Against Node-Splitting

1. Alice expands each node into a 4-cycle.
2. Bob splits two nodes.
3. Alice shrinks the underlying graph's biconnected components.

Tamperproofing by Reflection

class C { public int a; public C car, cdr; }

Field F = C.class.getFields();
if (F.length != 3) die();
if (F[i].getType() == C.class) die();

Field F = C.class.getFields();
for(int i=0; i<F.length; i++)
if (!F[i].isAssignableFrom(C.class))
  F[i].set(0, 0);
break;

{ 0.car = 0;  
  etc; 
}
**A Model of Software Watermarking**

**Definition 1 (Software Watermark)** Let $\mathcal{W}$ be a set of mathematical structures, and $p$ a predicate such that $\forall w \in \mathcal{W} : p(w)$. We choose $p$ and $\mathcal{W}$ such that the probability of $p(x)$ for a random $x \not\in \mathcal{W}$ is small.

---

**Software Watermarking: Programs**

**Definition 2 (Programs)** Let $\mathcal{P}$ be the set of programs. $P_w$ is an embedding of a watermark $w \in \mathcal{W}$ into $P \in \mathcal{P}$.

Let $\text{dom}(P)$ be the set of input sequences accepted by $P$. Let $\text{out}(P, I)$ be the output of $P$ on input $I$.

Let $S(P, I)$ be the internal state of program $P$ after having processed input $I$. Let $|S(P, I)|$ be the size of this state, in accessible words.

---

**Non-Semantics-Preserving Attacks**

- A planted plane cubic tree on 8 nodes.
- Bit-rate is 0.5 bits per word.
- Planarity check:
  - For each internal node $x$, the left-most child of $x$’s right subtree is $L$-linked to the right-most child of $x$’s left subtree.
**Software Watermarking: Coding Efficiency**

**Definition 5 (Watermark Coding Efficiency)**

\[ H(w) = \log_2 |\mathbb{W}| \] is the entropy of \( w \), in bits, when \( w \) is drawn with uniform probability from \( \mathbb{W} \).

Let \( |P|, P \in \mathbb{P} \) be the size (in words) of \( P \) as expressed in some encoding.

Let \( |S(P)| = \max_{i \in \text{dom}(P)} |S(P, I)| \) be the least upper bound on the size of \( P \).

An embedding of \( P_w \) of \( w \) in \( P \) has a high static data rate if \( \frac{H(w)}{|P_w| - |P|} \geq 1 \). An embedding \( P_w \) of \( w \) in \( P \) has a high dynamic data rate if \( \frac{H(w)}{|S(P_w)| - |S(P)|} \geq 1 \). □

---

**A Model of Watermarking: Transformations**

**Definition 3 (Program Transformations)** Let \( T \) be the set of transformations from programs to programs.

\( T_{\text{sem}} \subseteq T \) is the set of semantics preserving transformations:

\[
T_{\text{sem}} = \{ t : T \mid P \in \mathbb{P}, I \in \text{dom}(P), \text{dom}(P) = \text{dom}(t(P)), \text{out}(P, I) = \text{out}(t(P), I) \}.
\]

Similarly, \( T_{\text{stat}} \subseteq T \) is the set of state preserving transformations:

\[
T_{\text{stat}} = \{ t : T \mid P \in \mathbb{P}, I \in \text{dom}(P), \text{S}(P, I) = \text{S}(t(P), I) \}.
\]

---

**Software Watermarking: The Recognizer**

**Definition 4 (Watermark Recognizer)** \( R_T(P_w, S(P_w, I)) \) is a recognizer of \( w \in \mathbb{W} \) in \( P_w \in \mathbb{P} \) with input \( I \) wrt a set of transformations \( T \subseteq T \), if,

\[
\forall t \in T : p(R(t(P_w), S(P_w, I))) = p(w)
\]

- \( R_{\emptyset}(P_w, S(P_w, I)) \) is the trivial recognizer.
- \( R_T(P_w, \emptyset) \) is a static recognizer.
- \( R_T(\emptyset, S(P_w, I)) \) is a pure dynamic recognizer.
- \( R_{T_{\text{sem}}}(P_w, S(P_w, I)) \) is a strong recognizer.
### Dynamic Graph Watermarking – Summary

<table>
<thead>
<tr>
<th>Embed $n$ in $P$ such that</th>
<th>Threat Resistance:</th>
</tr>
</thead>
<tbody>
<tr>
<td>a) the embedding is imperceptible,</td>
<td>a) Distortive attacks</td>
</tr>
<tr>
<td>b) any successful attack incurs a large performance penalty,</td>
<td>• translation,</td>
</tr>
<tr>
<td>c) $n$ is large,</td>
<td>• optimization,</td>
</tr>
<tr>
<td>d) there is no performance penalty,</td>
<td>• obfuscation.</td>
</tr>
<tr>
<td>e) $n$ can be retrieved after any attack on $P$.</td>
<td>b) Collusive attacks.</td>
</tr>
</tbody>
</table>

data2proc

**Slide 12-P-Summary-1**

**Slide 12-P-Summary-2**