Watermarking, Tamper-Proofing and Obfuscation – Tools for Software Protection

Christian Collberg & Clark Thomborson
Computer Science Tech Report 170
University of Auckland

4 February 2000

Watermarking and Fingerprinting

Watermark: a secret message embedded into a cover message.

- Image, audio, video, text…
- Visible or invisible marks
- Fragile or robust
- Watermarking
  1. Discourages theft
  2. Allows us to prove theft
- Fingerprinting
  3. Allows us to trace violators

Watermarking Variants

- The watermark may be visible and robust (difficult to remove), providing a proof of ownership.
- The watermark may be fragile (obliterated by any modification), proving authenticity.
- ☞ The watermark may be invisible and robust, providing proof of ownership and security from theft.
- Fingerprinting is a variant of watermarking in which we put a unique customer-ID in each object we distribute. Piracy can be detected if we discover duplicate fingerprints, and these fingerprints identify the (witting or unwitting) source of the distribution.

Our Desiderata for WMs

- Watermarks should be stealthy -- difficult for an adversary to locate.
- Watermarks should be resilient to attack -- resisting attempts at removal even if they are located.
- Watermarks should have a high data-rate -- so that we can store a meaningful message without significantly increasing the size of the object.

Attacks on Watermarks

- **Subtractive** attacks: remove the WM without damaging the cover.
- **Additive** attacks: add a new WM without revealing "which WM was added first".
- **Distortive** attacks: modify the WM without damaging the cover.
- **Collusive** attacks: examine two fingerprinted objects, or a watermarked object and its unwatermarked cover; find the differences; construct a new object without a recognisable mark.

Defenses for Software Watermarks

- **Obfuscation**: we can modify the software so that a reverse engineer will have great difficulty figuring out how to reproduce the cover without also reproducing the WM.
- **Tamperproofing**: we can add integrity-checking code that (almost always) renders it unusable if the object is modified.

Classification of SW Watermarks

- Static code watermarks are stored in the section of the executable that contains instructions.
- Static data watermarks are stored in other sections of the executable.

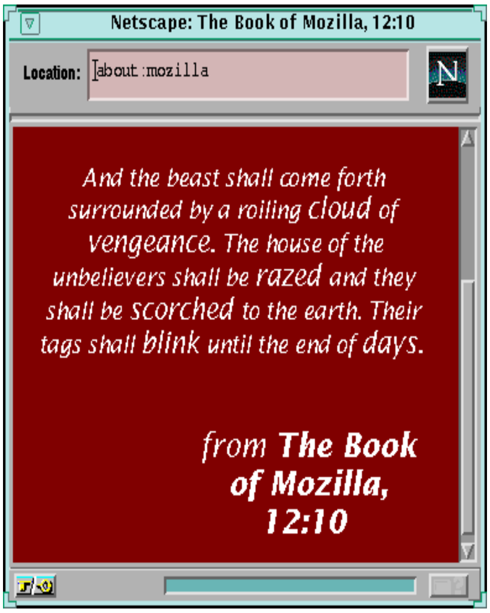
☞ Dynamic data watermarks are stored in a program's execution state. Such watermarks are resilient to distortive (obfuscation) attacks.

Dynamic Watermarks

- **Easter Eggs** are revealed to any end-user who types a special input sequence.
- **Execution Trace Watermarks** are carried (steganographically) in the instruction execution sequence of a program, when it is given a special input.

☞ **Data Structure Watermarks** are built (steganographically) by a program, when it is given a special input sequence (possibly null).

Easter Eggs



- The watermark is visible -- if you know where to look!
- Not resilient, once the secret is out.
- See www.eeggs.com

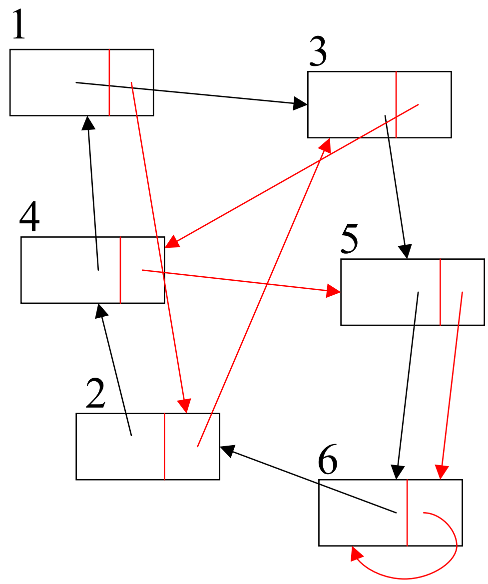
Our Goals for Dynamic DS WMs

- **Stealth**. Our WM should "look like" other structures created by the cover (search trees, hash tables, etc.)
- **Resiliency**. Our WM should have some properties that can be checked, stealthily and quickly at runtime, by tamperproofing code (triangulated graphs, biconnectivity, …)
- **Data Rate**. We would like to encode 100-bit WMs, or 1000-bit fingerprints, in a few KB of data structure. Our fingerprints may be 1000-bit integers that are products of two primes.

Permutation Graphs (Harary)

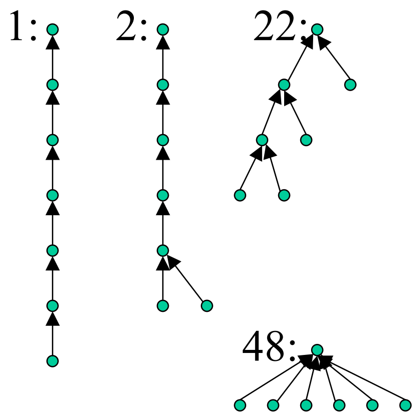
- The WM is 1-3-5-6-2-4.
- High data rate: lg(n!) ≈ lg(n/e) bits per node.
- High stealth, low resiliency (?)
- Tamperproofing may involve storing the same permutation in another data structure.
- What if an adversary changes the node labels?

☞ Node labels may be obtained from node positions on another list.



Oriented Trees

- Represent as "parent-pointer trees"
- There are
  $$c\alpha^{n-1}/n^{3/2} + O(\alpha^{n}/n^{5/2})$$
  oriented trees on n nodes, with
  c = 0.44 and α = 2.956,
  so the data rate is
  lg(α)/2 ≈ 0.8 bits/node.

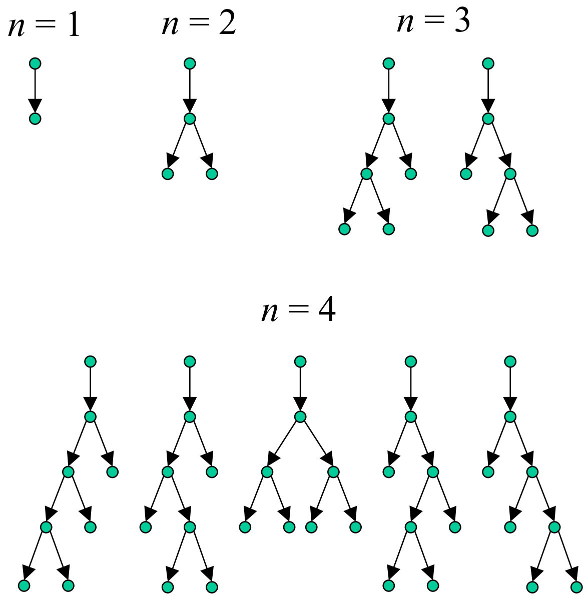


A few of the 48 trees for $n = 7$

Could you "hide" this data structure in the code for a compiler? For a word processor?

Planted Plane Cubic Trees

- One root node (in-degree 1).
- Trivalent internal nodes, with rotation on edges.
- We add edges to make all nodes trivalent, preserving planarity and distinguishing the root.
- Simple enumeration (Catalan numbers).
- Data rate is ~2 bits per leaf node.
- Excellent tamperproofing.



Open Problems in Watermarking

- We can easily build a "recogniser" program to find the WM and therefore demonstrate ownership… but can we release this recogniser to the public without compromising our watermarks?
- Can we design a "partial recogniser" that preserves resiliency, even though it reveals the location of some part of our WM?

State of the Art in SW Watermarking

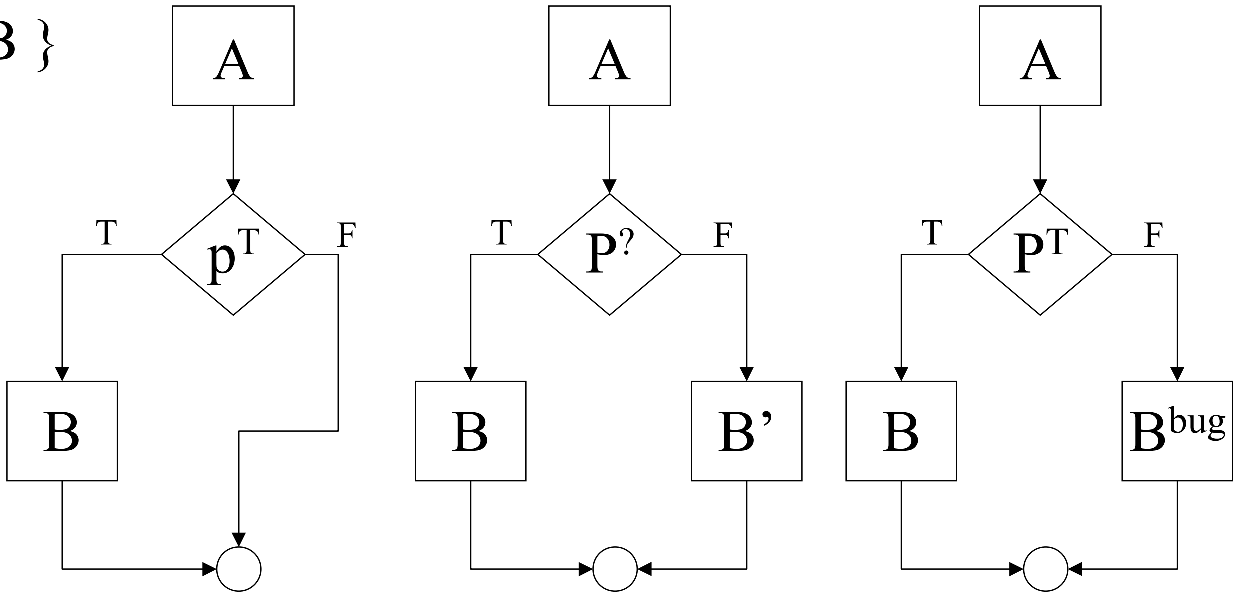
- First dynamic DS watermarks installed in 1999.
- Recognition SW being developed.
- Ongoing search for graph structures that are suitable for carrying fingerprints. Requirements:
  – easily enumerable
  – low outdegree (but high data rate)
  – quickly-checked properties (for tamperproofing)

Software Obfuscation

- Many authors, websites and even a few commercial products offer "automatic obfuscation" as a defense against reverse engineering.
- Existing products generally operate at the lexical level of software, for example by removing or scrambling the names of identifiers.
- We seem to have been the first (in 1997) to use "opaque predicates" to obfuscate the control structure of software.

Opaque Predicates

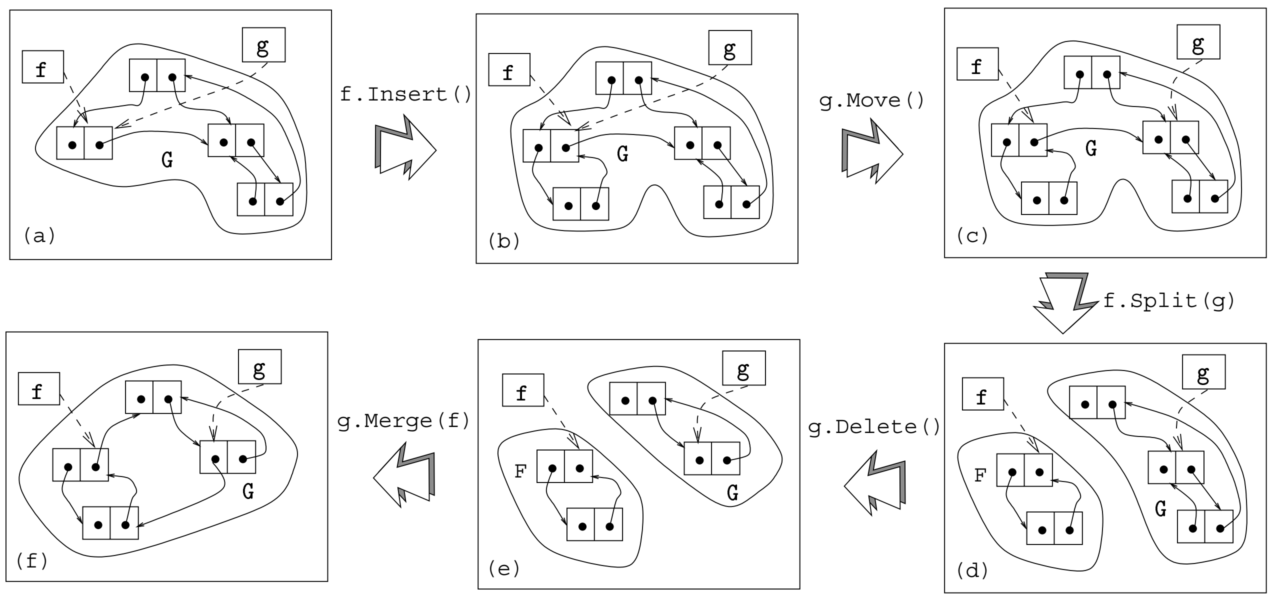
{A; B }



"always true" "indeterminate" "tamperproof"

("always false" is not shown)

if (f == g) then ?



Static alias analysis is intractable, so a de-obfuscator must use dynamic analysis to remove our opaque predicates.

Conclusion

- New art in software obfuscation can make it more difficult for pirates to defeat standard tamperproofing mechanisms, or to engage in other forms of reverse engineering.
- New art in software watermarking can embed "ownership marks" in software, that will be very difficult for anyone to remove.
- Much more R&D is required before robust obfuscating and watermarking tools are easy to use and readily available to software developers.