Experience with Software Watermarking

Jens Palsberg, Sowmya Krishnaswamy, Minseok Kwon, Di Ma, Qiuyun Shao, Yi Zhang

Properties of Watermarks

- Easy to create
- Easy to verify
- Difficult to remove
- Difficult to alter

Static Software Watermarks

- Static data watermarks are easy to alter and remove
  - Can be attacked by static code analyzers
  - Many semantics-preserving modifications will automatically remove them.

Dynamic Software Watermarks

- Much more difficult to attack
  - Nearly impossible to statically analyze
  - Altering final runtime structure by changing the program is very difficult
- Examples
  - “Easter Egg” watermarks
  - Watermarks which depend on the object graph

Graph based watermarking

- Inserting the watermark
  - Create a watermark graph
  - Insert it into the program's object graph
- Recovering the watermark
  - Create a copy of the runtime object graph
  - Find a subgraph isomorphic to the watermark graph
  - Without prior knowledge, this is an NP Complete Problem

What are PPCTs?

- Stands for "Planted Plane Cubic Tree
- A binary tree structure, with an extra "Origin" node
- Origin node and leaf nodes form a circularly linked list

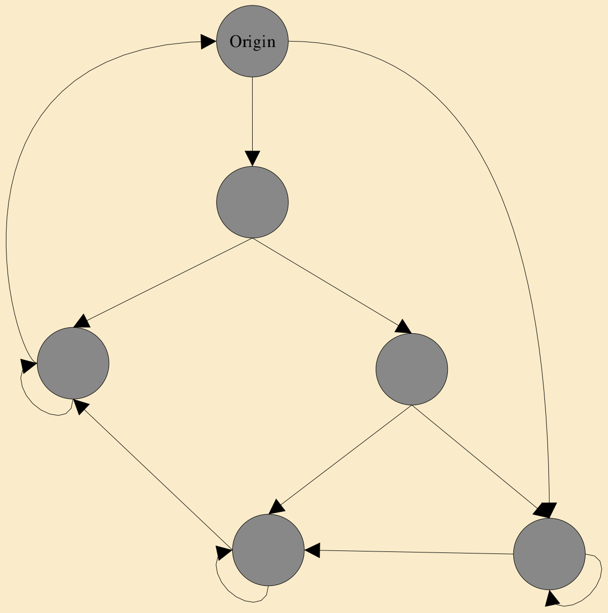


Figure 1. PPCT

What are PPCTs?

- Each leaf node points to itself
- Each node has two pointers in it
- Note that from any node, you can reach the origin node.

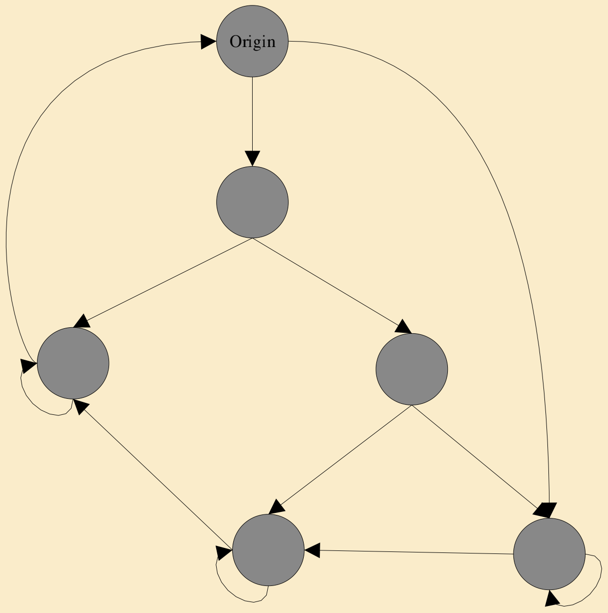
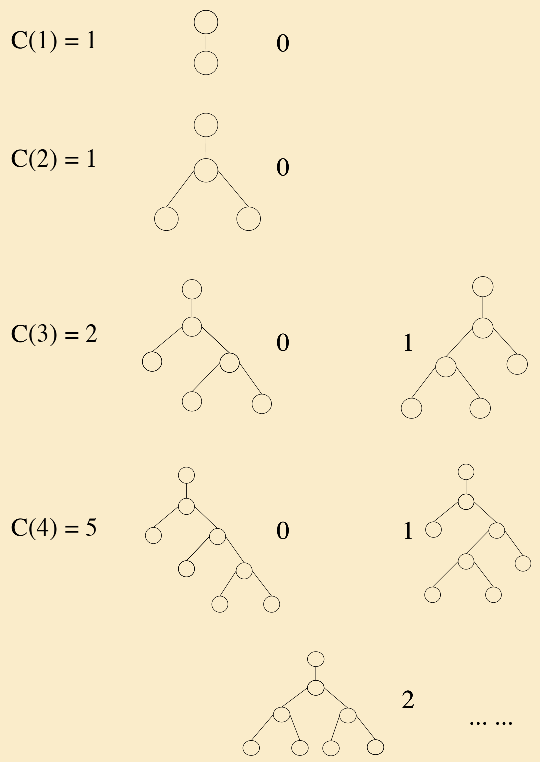


Figure 1. PPCT

How to represent a watermark with a PPCT

- Each PPCT with a certain number of nodes has an enumerable set of trees
- Make a tree large enough to represent your number



How do we create the object graph?

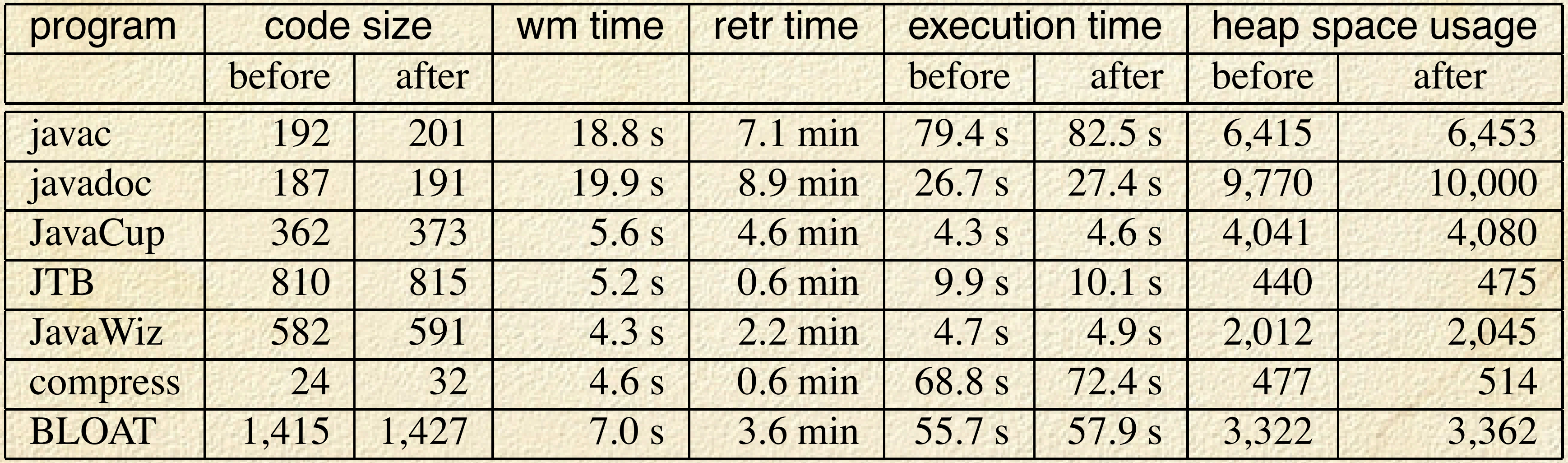
- Find all the non-library classes
  - Can’t rely on names, because they may have been obfuscated
- Find all objects in memory of those classes (nodes)
- Find pointers/references between these objects (edges)

How do we find the PPCT?

- In the object graph, find potential leaf nodes (nodes which have edges to themselves)
- Try to trace these nodes to find an origin node
- From the origin, see if you can find the watermark graph
    - You know the number of nodes in the subgraph, so search is bounded

Results

- Using a dual processor UltraSparc 200MHz



program	code size		wm time	retr time	execution time		heap space usage	
	before	after			before	after	before	after
javac	192	201	18.8 s	7.1 min	79.4 s	82.5 s	6,415	6,453
javadoc	187	191	19.9 s	8.9 min	26.7 s	27.4 s	9,770	10,000
JavaCup	362	373	5.6 s	4.6 min	4.3 s	4.6 s	4,041	4,080
JTB	810	815	5.2 s	0.6 min	9.9 s	10.1 s	440	475
JavaWiz	582	591	4.3 s	2.2 min	4.7 s	4.9 s	2,012	2,045
compress	24	32	4.6 s	0.6 min	68.8 s	72.4 s	477	514
BLOAT	1,415	1,427	7.0 s	3.6 min	55.7 s	57.9 s	3,322	3,362

How do we insert the watermark?

- We could just put the watermark generation code at the beginning of the program
  - Easy to find and remove
- Insert watermark creating in “Easter Egg”?
  - “Easter Egg” code may be discovered
- Randomly insert watermark code?
  - Can help avoid collusion attacks

Code Obfuscation

- Many different ways to do it
  - Padding
  - Opaque predicates
  - renaming
  - Method inlining/outlining
- We will look at the first two

Code Obfuscation

- Padding
  - Make a larger graph than necessary
  - Makes finding a graph much more difficult
  - Relatively inexpensive runtime and memory cost

Code Obfuscation

- Opaque Predicates
  - Predicates which regularly evaluate to either true or false
  - Come in Static and Dynamic flavors
  - Greatly hinders static code analysis
  - Can add significant runtime costs

Code Obfuscation

- Dynamic opaque predicates
  - Most effective for preventing static analysis
  - Can use the PPCT itself to create one
  - This causes problems.
    - Leaves parts of programs unobfuscated
    - Randomly generated PPCT may be attacked

Tamperproofing

- What if someone is able to change the watermark structure randomly?
  - Make the program behavior depend on watermark structure
  - Can be done with dynamic opaque predicates
  - Solves some of the problems with dynamic opaque predicates

Benefits of PPCT

- PPCTs have some properties which help many of these approaches:
  - Stealthy heap structure
  - Easy to enumerate
  - Source of dynamic opaque predicates
  - Have easy to check properties that don’t stand out
- Any other watermark graph representations should have these properties

Conclusion

- Dynamic software watermarks based on the object graph can be very effective
- Must be combined with other obfuscation and protection techniques to be secure
- Using the techniques in concert give the best results