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
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.
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

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