An Abstract Interpretation-Based Framework for Software Watermarking

Patrick Cousot
École normale supérieure
Patrick.Cousot@ens.fr
www.di.ens.fr/~cousot

Radhia Cousot
CNRS & École polytechnique
Radhia.Cousot@polytechnique.fr
www.stix.polytechnique.fr/~rcousot

Principle of Software Watermarking

Watermark embedding:

$$\text{Program} \times \text{Signature} \longrightarrow \text{Watermarked program}$$

Watermark extraction:

$$\text{Watermarked program} \longrightarrow \text{Signature}$$

The signature should be invisible in the watermarked program.

Motivating Applications of Software Watermarking

- Identification

Requirements

Requirements on Software Watermarking (Contn'd)

- Resistance to attacks:
  - Signature is secret (it cannot be extracted — but by the watermark extraction program)
  - Signature is persistent (semantics and executability preserving transformations cannot prevent signature extraction)

Making the Software Watermarking Algorithms Public

- More confidence in public algorithms;

⇒ Make the embedding/extraction algorithms public;

⇒ By parameterizing with a secret:

Watermark embedding:

$$\text{Program} \times \text{Signature} \times \text{Secret} \longrightarrow \text{Watermarked program}$$

Watermark extraction:

$$\text{Watermarked program} \times \text{Secret} \longrightarrow \text{Signature}$$

Both the signature and secret should be invisible in the watermarked program.

Stegomark Approach to Software Watermarking

Existing Solutions

Dynamic Software Watermarking

- Semantics-based approach
- Watermark embedding: the signature is hiden in the semantics of the stegomark
- Watermark extraction: execution of watermarked program with the secret input reveals the signature:

  Dynamic data structure watermarking: by building a data structure containing the signature

  Dynamic execution trace watermarking: by generating a succession of events (adresses/operations/...) encoding the signature

⇒ more robust (Collberg & Thomborson [POPL'97 & 98])

Abstract Software Watermarking

- Abstract interpretation-based approach
- Watermark embedding: the signature is hiden in the abstract semantics of the stegomark (hence that of the watermarked program)
- Watermark extraction: the extraction of the signature is by static analysis of the watermarked program (which always succeeds because of the inlayed stegomark)

Formalization of Abstract Software Watermarking
1.a) Ingredients of a concrete semantics

- Programs: $P \in \mathbf{Program}$
- Concrete semantic domain: $\mathcal{D}$
- Concrete semantics of programs: $S \in \mathbf{Program} \mapsto \mathcal{D}$
- Observability abstraction: $\alpha_{\mathcal{O}}$

such that:

$\forall P \in \mathbf{Program}$, only $\alpha_{\mathcal{O}}(S[\![P]\!])$ is of interest

- Observability equivalence: $\equiv_{\mathcal{O}}$

$$P \equiv_{\mathcal{O}} P' \Leftrightarrow \alpha_{\mathcal{O}}(S[\![P]\!]) = \alpha_{\mathcal{O}}(S[\![P']\!])$$

Formalization of Abstract Software Watermarking (Cont'd)

1.c) Watermarking ingredients

- Signature abstractor:

$$A[\text{Secret}] \in \text{Signature} \longmapsto \mathcal{D}^\sharp[\text{Secret}]$$

- Signature extractor:

$$E[\text{Secret}] \in \mathcal{D}^\sharp[\text{Secret}] \longmapsto \text{Signature}$$

- Stegomark generator:

$$M[\text{Secret}] \in \mathcal{D}^\sharp[\text{Secret}] \longmapsto \text{Stegomark}$$

- Stegoinlayer:

$$I[\text{Secret}] \in \begin{array}{c}\text{Subject}\\ \text{Program}\end{array} \times \text{Stegomark} \longmapsto \begin{array}{c}\text{Watermarked}\\ \text{Program}\end{array}$$

Formalization of Abstract Software Watermarking (Cont'd)

3) Extraction

Watermark extraction:

$$\text{Watermarked program} \times \text{Secret} \longrightarrow \text{Signature}$$

Signature extraction from P is by static analysis:

$$E[\text{Secret}](S^{\sharp}[\text{Secret}][\![P]\!])$$

4) Requirements on the watermarking ingredients (Cont'd)

- Abstract values hidden in the stegomark are extractable by static analysis

$$S^{\sharp}[\text{Secret}][\![M[\text{Secret}](D)]\!] = D$$

- Extraction of hidden abstract values is preserved by inlaying:

$$S^{\sharp}[\text{Secret}][\![I[\text{Secret}](P, M[\text{Secret}](D))]\!] = D$$

⇒ The hidden signatures are extractable from watermarked programs by static analysis:

if $Q = M[\text{Secret}](A[\text{Secret}](\text{Signature}))$ then

$= E[\text{Secret}](S^{\sharp}[\text{Secret}][\![I[\text{Secret}](P, Q)]\!]) = \text{Signature}$

Formalization of Abstract Software Watermarking (Cont'd)

5) Resistance to attacks

- Signature extraction without the secret is hard:
  - Computing $S^\sharp[?][\![I[?](P,Q)]\!]$ is hard
- Recovering the original program/stegomark elimination is hard:
  - Computing P from $I[\text{Secret}](P,Q)$ is hard (without Q)
- Ideally, stegomark obfuscation should be effectless:

$$\begin{array}{lrcl} \text{If} & Q & = & M[\text{Secret}](A[\text{Secret}](\text{Signature})) \\ \text{and} & P' & \equiv_{\mathcal{O}} & I[\text{Secret}](P,Q) \end{array}$$

$$\text{then } S^\sharp[\text{Secret}][\![P']\!] = S^\sharp[\text{Secret}][\![I[\text{Secret}](P,Q))]\!]$$

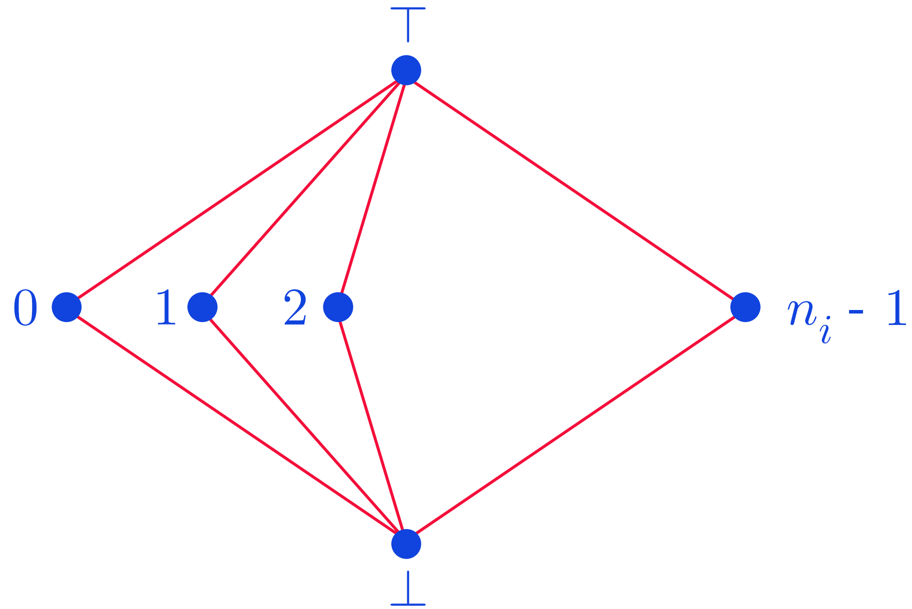
Programs, Semantics, Observability

- Programs: Java methods (classes, programs)
- Concrete semantics: reachable states
- Observability:
  - end-user visible effects of method invocation
  - but not the internal computations
  - same complexity

Static analysis

- ℓ times a variant of Kildall's constant propagation modulo the secret n_i:

$$\mathcal{D}^\sharp = \prod_{i=1}^{\ell} \mathcal{D}_i^\sharp \quad \text{where} \quad \mathcal{D}_i^\sharp =$$



- Extend pointwise/componentwise to environments, program points, etc.

Stegomark for c_i

ℓ stegomarks, each hiding c_i, $i = 1, \ldots, \ell$:

- Declaration:

  ```
  int W;
  ```

- Initialization part:

  $\mathtt{W} = P(1)$ (in $\mathbb{Z}$, such that $P(1) = c_i \; in \; \mathbb{Z}/_{n_i}\mathbb{Z}$)

- Iteration part:

  $\mathtt{W} = Q(\mathtt{W})$ (in $\mathbb{Z}$, such that $c_i = Q(c_i)$ in $\mathbb{Z}/_{n_i}\mathbb{Z}$)

- `W` is constant in $\mathbb{Z}/_{n_i}\mathbb{Z}$ whence c_i is extractable by constant propagation in $\mathcal{D}_i$;
- `W` is not constant in $\mathbb{Z}$ (looks stochastic at execution)

Obfuscating the stegomark for c_i

Obfuscation for 2[nd] degree polynomials (computed by Horner method):

- $P(x) = (x - k_1)x + k_2$
  where $k_1 = (1 + c_i) + r_1.n_i)$
  $k_2 = (c_i + r_2.n_i)$
  r_1 and r_2 are random numbers;
- idem for $Q(x)$.

Example of Watermarked Program

```
public class Fibonacci {
```

Confidentiality

- Assume the stegomark was extracted from the program
- Can the signature be extracted from the stegomark?

  Find c_i, $i = 1, \ldots, \ell$ from $M[?](A[?](?))$:

  – extract the polynomials P and Q for c_i, then

  – amounts to the factoring problem

  – hard for large factors
- Indeed useless anyway since the signature contains encrypted information only
- So, the only interesting attacks are those erasing or obfuscating the stegomark

Attacks on erasing the stegomark for c_i

The stegomark contains:

- unusual large integer constants
- auxiliary variables with almost stochastic integer values in $\mathbb{Z}$

that might be recognized by monitoring the watermarked program execution to reveal the stegomark components for some c_i where $i \in [1, \ell]$

Counter-attack on erasing the stegomark for c_i (Cont'd)

3) Hide operations on large integers as non-standard semantics of operations on other types:

- floating point operations
- list, tree operations
- etc

interpreting these operations:

- on the original data types in the concrete semantics
- on large integers during the extracting static analysis

Secret = $\langle n_1, \ldots, n_\ell \rangle$ + Non-standard concrete semantics

Attacks on obfuscating the stegomark for c_i

Counter-attack on obfuscating the stegomark for c_i

1) obfuscate the watermarked program before distribution

2) refine the static analyzer

Pronostics on Attacks

When knowing:

- The embedder and/or extractor: attacks are easy
- The embedder and/or extractor algorithm principle but not the underlying non-standard semantics: attacks are harder, may be feasible (?)
- Nothing but that abstract watermarking might have been used: good luck!