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

POPL®'2004, Venice, Italy 14–16 Jan 2004

Software Watermarking: Principle & Motivation

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
- Authentification
- Version control (fingerprinting)
- Copyrights protection
- Intellectual property protection
- ...

$\Rightarrow$ extract the signature from the watermarked program, and

$\Rightarrow$ compare with original signature (kept secret by a trusted third party since watermark embedding time)

Requirements

Requirements on Software Watermarking

- The watermarked program source is clear and public (but *not* the subject program)
- The end-user semantics is preserved by watermarking
- Program executability is left unchanged
- Signature embedding and extraction is algorithmic
- Signature embedding and extraction algorithms are public

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)

Fingerprinting

- The subject program can be successively watermarked by several different signatures
- Each of these signatures can be individualy extracted (by their respective watermarkers)

Making the Software Watermarking Algorithms Public

- More confidence in public algorithms;

$\Rightarrow$ Make the embedding/extraction algorithms public;

$\Rightarrow$ By parameterizing with a secret:

Watermark embedding:

Program $\times$ Signature $\times$ Secret $\longrightarrow$ Watermarked program

Watermark extraction:

Watermarked program $\times$ Secret $\longrightarrow$ Signature

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

Stegomark Approach to Software Watermarking

1) Hide the signature in a program (so called stegomark)
2) Incrust the stegomark in the subject program:

Stegomark maker:

Signature $\times$ Secret $\longrightarrow$ Stegomark

Watermark embedding:

Subject program $\times$ Stegomark $\longrightarrow$ Watermarked program

Watermark extraction:

Watermarked program $\times$ Secret $\longrightarrow$ Signature

Existing Solutions

Static Software Watermarking

- Syntax-based approach
- The syntax of the stegomark contains the signature, such as:
  - in the data (e.g. watermarked image)
  - in the control (e.g. order of the branches of a switch case)

$\Rightarrow$ not pervasive

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])

Principle of Abstract Software Watermarking

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)

Abstract Software Watermarking Framework

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.b) Ingredients of a classical static analyzer

- Parameterized abstract domain: $\langle \mathcal{D}^\sharp[\text{Secret}], \sqsubseteq[\text{Secret}] \rangle$
- Parameterized abstraction: $\alpha[\text{Secret}] \in \mathcal{D} \longmapsto \mathcal{D}^\sharp[\text{Secret}]$
- Parameterized abstract semantics of programs:

$$S^\sharp[\text{Secret}] \in \text{Program} \longmapsto \mathcal{D}^\sharp[\text{Secret}]$$

such that:

$$S^\sharp[\text{Secret}][\![P]\!] \quad \sqsupseteq[\text{Secret}] \quad \alpha[\text{Secret}](S[\![P]\!])$$

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)

2) Embedding

Stegomark generator:

Signature × Secret ⟶ Stegomark

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

Watermark embedding:

Subject program × Stegomark ⟶ Watermarked program

$$I[\text{Secret}](P, Q)$$

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:

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

Formalization of Abstract Software Watermarking (Cont'd)

4) Requirements on the watermarking ingredients

- Embedding/extraction of signatures in abstract domains

$$E[\text{Secret}](A[\text{Secret}](\text{Signature})) = \text{Signature}$$

Formalization of Abstract Software Watermarking (Cont'd)

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

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

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

- Observable semantics is unchanged by stegomark inlaying:

$$P \equiv_{\mathcal{O}} I[\mathrm{Secret}](P, M[\mathrm{Secret}](D)))$$

Formalization of Abstract Software Watermarking (Cont'd)

5) Resistance to attacks

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

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

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

An Instance of the Abstract Software Watermarking Framework

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

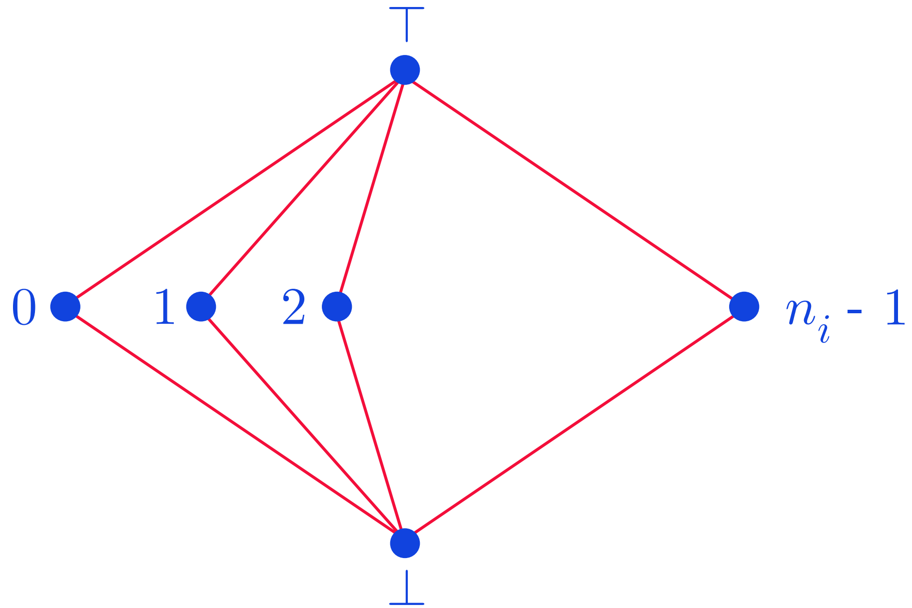
Signatures, Secret

- Signatures: anything encryped into an integer $\leq m$, large
- Secret: $\langle n_1, \ldots, n_\ell \rangle$, machine representable, relatively prime and $n_1 \times \ldots \times n_\ell > m$

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.

Embedding Signatures in the Abstract Domain

- Embedding/extraction of signatures in abstract domains (Chinese remainder theorem)

$$A[\langle n_1, ..., n_\ell \rangle](s) = \langle c_1, ..., c_\ell \rangle$$
$$E[\langle n_1, ..., n_\ell \rangle](\langle c_1, ..., c_\ell \rangle) = s$$

where $0 \leq c_i < n_i$, $i = 1, \ldots, \ell$

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)

False recognition

- A terminating static analysis is always approximate
- Nevertheless, the static analysis of the stegomark will always reveal c_i
- The analysis of the watermarked program might reveal a false positive, that is a c'_i different from the intented c_i;

  $\Rightarrow$ check at watermarking time

  $\Rightarrow$ in the (rare) case of false positive, just change n_i

  $\Rightarrow$ don't care, anyway you also get c_i

Obfuscating the stegomark for c_i

Obfuscation for 2nd 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 {
  public Fibonacci() {}
  public static void main(String[] args){
    int n=Integer.parseInt(args[0]);
    int a=0; int b=1;  int d=1; int e=35538;
    int f=1; int g=-111353;
    e=d*e; d=e+11445; g=f*g; f=g-47305;
    for (int i=1;i<n;i++)
    {int c=a+b; e=d*658; f=f*4; a=b; g=g+1566;
     e=e+971; g=g*f; e=e*d; b=c; d=e+4623; f=g+21494;}
    System.out.println("Fib("+n+") = "+b); }}
```

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 and Counter-Attacks

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

1) Tamper-proofing (fail when program altered)

2) Anchor the stegomark:

- make the subject program dependent upon the stegomark
- so that the watermarked program becomes unusable when erasing this stegomark

($\longrightarrow$ see the proceedings)

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

Obfuscate the program, by:

- code reordering,
- proceduralization,
- parallelization,
- globalization of variables,
- data heap reallocation,
- variable splitting and merging,
- etc

to puzzle the static analysis

Counter-attack on obfuscating the stegomark for c_i

1) obfuscate the watermarked program before distribution

2) refine the static analyzer

Conclusion

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!

One Suggestion for Future Work (Among Many)

Watermark embedding:

> Program $\times$ Private signature $\times$ Private Key $\longrightarrow$ Watermarked program

Watermark checking:

> Watermarked program $\times$ Public Key $\longrightarrow$ Public signature

Watermark extraction:

> Watermarked program $\times$ Private Key $\longrightarrow$ Private signature