Software Protection: How to Crack Programs, and Defend Against Cracking
Lecture 8: Watermarking
Minsk, Belarus, Spring 2014

Christian Collberg
University of Arizona

www.cs.arizona.edu/~collberg
© June 23, 2014 Christian Collberg
Today’s lecture

Watermarking
Embed a unique identifier into the executable of a program.
Embed a unique identifier into the executable of a program.

A watermark is much like a copyright notice.
Embed a unique identifier into the executable of a program.

A watermark is much like a copyright notice.

Won’t prevent an attacker from reverse engineering or pirating it the program.
Embed a unique identifier into the executable of a program.

A watermark is much like a *copyright notice*.

Won’t prevent an attacker from reverse engineering or pirating it the program.

Allows us to show that the program the attacker claims to be his, is actually ours.
Embed a unique identifier into the executable of a program.

A watermark is much like a *copyright notice*.

Won’t prevent an attacker from reverse engineering or pirating it the program.

Allows us to show that the program the attacker claims to be his, is actually ours.

*Software fingerprinting*: every copy you sell will have a different unique mark in it.
Embed a unique identifier into the executable of a program.

A watermark is much like a *copyright notice*.

Won’t prevent an attacker from reverse engineering or pirating it.

Allows us to show that the program the attacker claims to be his, is actually ours.

*Software fingerprinting*: every copy you sell will have a different unique mark in it.

Trace the copy back to the original owner, and take legal action.
History and Applications

p. 468
Customer #27182818

Fingerprint mark (invisible, robust)

Authorship mark ((in)visible, robust)

Meta-data mark (visible, fragile)

Licensing mark (invisible, robust)

Validation mark (visible, fragile)

<license object="kitten.jpg">
  <grant to="Alice">
    <right> copy-once </right>
  </grant>
</license>

© 2006 Collberg

PG-13

MD5(kitten.jpg)

0xc6ba8f25d2dfe4cf518d7f327c8e83f

"Cute kitten in window in Venice"

"Attack mice at dawn"

Customer #31415926

© 2006 Collberg
Audio marking: Echo hiding

Embed echoes that are short enough to be imperceptible to the human ear:
Audio: Least Significant Bit

- LSB of an audio sample is the one that contributes least to your perception,
Audio: Least Significant Bit

- LSB of an audio sample is the one that contributes least to your perception,
- Alter without adversely affecting quality!
Audio: Least Significant Bit

- LSB of an audio sample is the one that contributes least to your perception,
- Alter without adversely affecting quality!
- *Attack*: randomly replace the least significant bit of every sample!
Image: Patchwork

- Embed a single bit by manipulating the brightness of pixels.
Image: Patchwork

- Embed a single bit by manipulating the brightness of pixels.
- Use a pseudo-random number sequence to trace out pairs \((A, B)\) of pixels.
Image: Patchwork

- Embed a single bit by manipulating the brightness of pixels.
- Use a pseudo-random number sequence to trace out pairs \((A, B)\) of pixels.
- During embedding adjust the brightness of \(A\) up by a small amount, and \(B\) down by the same small amount:
Patchwork: Embedding And Recognition

**EMBED**(\(P, key\)):

1. \texttt{Init\_RND(key); } \(\delta \leftarrow 5\)
2. \(i \leftarrow \text{RND}(); j \leftarrow \text{RND}()\)
3. Adjust the brightness of pixels \(a_i\) and \(b_i\):
   \(a_i \leftarrow a_i + \delta; b_j \leftarrow b_j - \delta\)
4. repeat from 2 \(\approx 10000\) times

**RECOGNIZE**(\(P, key\)):

1. \texttt{Init\_RND(key); } \(S \leftarrow 0\)
2. \(i \leftarrow \text{RND}(); j \leftarrow \text{RND}()\)
3. \(S \leftarrow S + (a_i - b_j)\)
4. repeat from 2 \(\approx 10000\) times
5. if \(S \gg 0 \Rightarrow 0\) output "marked!"
Blind vs. Informed

- Watermarking recognizers are either *blind* or *informed*.
Blind vs. Informed

- Watermarking recognizers are either *blind* or *informed*.
- To extract a blind mark you need the marked object and the secret key.
Blind vs. Informed

Watermarking recognizers are either *blind* or *informed*.

To extract a blind mark you need the marked object and the secret key.

To extract an informed mark you need extra information, such as original, unwatermarked, object.
Watermarking text

- Cover object types:
  - the text itself with formatting (ASCII text); or
  - free-flowing text;
  - an *image* of the text (PostScript or PDF).
Watermarking Text: PDF

- Similar to marking images.

I saw the best minds ______
of my generation, ______
starving hysterical naked

\[
\begin{align*}
12pt \{ & I \text{ saw the best minds} \\
12pt \{ & \text{of my generation,} \\
14pt \{ & \text{starving hysterical naked}
\end{align*}
\]
Watermarking Text: PDF

- Similar to marking images.
- Example: encode 0-bit or a 1-bit by hanging word/line spacing.

I saw the best minds ________
12pt { of my generation, ________
12pt { starving_hysterical_naked

I saw the best minds ________
12pt { of my generation, ________
14pt { starving_hysterical_naked
Watermarking Text: formatted ASCII

- Encode the mark in white-space: 1 space = 0-bit, 2 spaces = 1-bit:

```
I saw the best minds of my generation, starving hysterical naked
```

```
I saw the best minds of my generation, starving hysterical naked
```
Watermarking Text: Synonym replacement

- Replace words with synonyms.
- Insert spelling or punctuation errors.

I saw the best minds of my generation, starving hysterical naked

\[
\downarrow
\]

I observed the choice intellects of my generation, famished hysterical nude
Watermarking Text: Syntax

Encode a mark in the syntactic structure of an English text:

1. Devise an extract function which computes a bit from a sentence,
2. Modify the sentence until it embeds the right bit.

I saw the best minds of my generation, starving hysterical naked

It was the best minds of my generation that I saw, starving hysterical naked
Watermarking Text: Atallah et al.

1. Chunk up the watermark, embed one piece per sentence.
2. A function computes one bit per syntax tree node.
3. Modify sentence until these bits embed a watermark chunk.

I saw the best minds of my generation, starving hysterical naked
I saw the best minds of my generation, starving hysterical naked.

I saw the best minds of my generation. They were starving hysterical naked. None, baby, none were smarter than them. Nor more lacking in supply of essential nutrients or in more need of adequate clothing. Baby.
Watermarking Software

p. 478
Static watermarks

You care about
- Encoding bitrate
- Stealth
- Resilience to attack
Ideas for Software Watermark Algorithms

Encode the watermark
  in a permutation of a language structure
Ideas for Software Watermark Algorithms

Encode the watermark
- in a permutation of a language structure
- in an embedded media object
Ideas for Software Watermark Algorithms

Encode the watermark
- in a permutation of a language structure
- in an embedded media object
- in a statistical property of the program
Ideas for Software Watermark Algorithms

Encode the watermark

- in a permutation of a language structure
- in an embedded media object
- in a statistical property of the program
- as a solution to a static analysis problem
Ideas for Software Watermark Algorithms

Encode the watermark
- in a permutation of a language structure
- in an embedded media object
- in a statistical property of the program
- as a solution to a static analysis problem
- in the topology of a CFG
Dynamic watermarks

Encode the watermark in the runtime state of the program
Dynamic watermarks

- Encode the watermark in the runtime state of the program
- Dynamic marks appear more robust, but are more cumbersome to use
Attacks against software watermarks

The adversary knows the algorithm
Attacks against software watermarks

- The adversary knows the algorithm
- The adversary has complete access to the program
Attacks against software watermarks

- The adversary knows the algorithm
- The adversary has complete access to the program
- The adversary doesn’t know the key
Attacks against software watermarks

- The adversary knows the algorithm
- The adversary has complete access to the program
- The adversary doesn’t know the key
- The adversary doesn’t know the embedding location (it’s key dependent)
Attacks — Rewrite attack

- Alice has to assume that Bob will try to destroy her marks before trying to resell the program!
- One attack will always succeed...
Attacks — Rewrite attack

Alice has to assume that Bob will try to destroy her marks before trying to resell the program!

One attack will **always** succeed...

Ideally, this is the *only* effective attack.
Attacks — Additive attack

- Bob can also add his own watermarks to the program:

- An additive attack can help Bob to cast doubt in court as to whose watermark is the original one.
Attacks — Distortive attack

A **distortive attack** applies semantics-preserving transformations to try to disturb Alice’s recognizer:
Attacks — Distortive attack

A **distortive attack** applies semantics-preserving transformations to try to disturb Alice’s recognizer:

Transformations: code optimizations, obfuscations,…
Attacks — Collusive attack

Bob buys two differently marked copies and comparing them to discover the location of the fingerprint:

P1

Collusive Attack

Bob

42

P2

17

P''

Extract

?
Attacks — Collusive attack

- Bob buys two differently marked copies and comparing them to discover the location of the fingerprint:

- Alice should apply a different set of obfuscations to each distributed copy, so that comparing two copies of the same program will yield little information.
Algorithm wMDM: Reordering Basic Blocks
Algorithm $\text{WMDM}$: Reordering Basic Blocks

- Performance overhead of 0-11% for three standard high-performance computing benchmarks.
Algorithm **WMDM**: Reordering Basic Blocks

- Performance overhead of 0-11% for three standard high-performance computing benchmarks.
- Negligible slowdown for a set of Java benchmarks.
Algorithm $\text{WMDM}$: Reordering Basic Blocks

- Performance overhead of 0-11% for three standard high-performance computing benchmarks.
- Negligible slowdown for a set of Java benchmarks.
- If you have $m$ items to reorder you can encode

$$\log_2(m!) \approx \log_2(\sqrt{2\pi m}(m/e)^m) = \mathcal{O}(m \log m)$$

watermarking bits.
Algorithm \texttt{WM\textsubscript{DM}}: Reordering Basic Blocks

- Performance overhead of 0-11% for three standard high-performance computing benchmarks.
- Negligible slowdown for a set of Java benchmarks.
- If you have $m$ items to reorder you can encode

$$\log_2(m!) \approx \log_2(\sqrt{2\pi m}(m/e)^m) = \Theta(m \log m)$$

watermarking bits.
- What about stealth?
Code decode (Image m) {
  ...
}

Image tudou =

  int foo(int x){
    ...
  }

Copyright 2007, Collberg

void main() {
  Code c = decode(tudou);
  int x = c.execute(42)
}
Algorithm $\text{wmMC: Media watermark}$

```c
void main() {
    int x = foo(42);
}
```

```c
int foo (int x) {
    ...
}
```
Algorithm \texttt{WMMC}: Media watermark

```c
int foo (int x) {
    ...
}

void main() {
    int x = foo(42);
}
```

Bob uses \textbf{Stirmark} to destroy marks in embedded images!
Algorithm \textnormal{WMMC}: Media watermark

Bob uses \textnormal{Stirmark} to destroy marks in embedded images!
Algorithm WMVVS

Watermarks in CFGs
Algorithm \textsc{wMVVS}: Watermarks in CFGs

Basic idea:

1. Embed the watermark in the CFG of a function.
Algorithm WmVVS: Watermarks in CFGs

Basic idea:

1. Embed the watermark in the CFG of a function.
2. Tie the CFG tightly to the rest of the program.
Algorithm wMVVS: Watermarks in CFGs

Basic idea:
1. Embed the watermark in the CFG of a function.
2. Tie the CFG tightly to the rest of the program.

Issues:
1. How do you encode a number in a CFG?
2. How do you find the watermark CFG?
3. How do you attach the watermark CFG to the rest of the program?
Algorithm \textit{wmVVS:} Embedding

- Generate a stealthy watermark CFG:
  - basic blocks have out-degree of one or two
Algorithm $\text{wmVVS: Embedding}$

- Generate a stealthy watermark CFG:
  1. basic blocks have out-degree of one or two
  2. it is reducible
Algorithm $\text{wmVVS}$: Embedding

- Generate a stealthy watermark CFG:
  1. basic blocks have out-degree of one or two
  2. it is reducible
  3. it is shallow (real code isn’t deeply nested)
Algorithm $wMVVS$: Embedding

- Generate a stealthy watermark CFG:
  1. basic blocks have out-degree of one or two
  2. it is reducible
  3. it is shallow (real code isn’t deeply nested)
  4. it is small (real functions aren’t big)
Algorithm $\mathrm{wMVVS}$: Embedding

- Generate a stealthy watermark CFG:
  1. basic blocks have out-degree of one or two
  2. it is reducible
  3. it is shallow (real code isn’t deeply nested)
  4. it is small (real functions aren’t big)
  5. it is resilient to edge-flips:

\[
\begin{align*}
\text{if } a < b & \text{ goto } B_k \\
\text{if } a \geq b & \text{ goto } B_j
\end{align*}
\]
Algorithm \textbf{WMVVS}: Embedding

- Generate a stealthy watermark CFG:
  1. basic blocks have out-degree of one or two
  2. it is reducible
  3. it is shallow (real code isn’t deeply nested)
  4. it is small (real functions aren’t big)
  5. it is resilient to edge-flips:

- Reducible Permutation Graphs (RPGs)
public static int bogus;
public static int m4(int i) {
    i = i & 0x7BFF;
bogus+=2; i-=i>>2;
    do {
        i = i >> 3;
        label:
        {
            if (++bogus <= 0) {
                i = i | 0x1000;
                if ((bogus += 6) == 0)
                    break label;
            }
            ++bogus;
            i = i * 88 >>> 1;
        }
    } while((bogus += 6)<0);
    i = i | 0x4;
} while((bogus += 6)<0);
bogus+=2; return i;
public void P(boolean S) {
    if (S)
        System.out.println("YES");
    else
        System.out.println("NO");
}

public void main (String args[]) {
    for (int i=1; i<args.length; i++) {
        if (args[0].equals(args[i])) {
            if (m4(3)<0)
                P(false);
            return;
        }
    }
    m3(-1);
    P(false);
}

public int m3(int i) {
    i = i ^ i >> 0x1F;
    i = i / 4 * 3;
    do {
        i -= i >> 3;
        if ((bogus += 11) <= 0)
            break;
        i = i / 5;
    } while (((bogus += 6)<0) && (m3(9)>=0));
    return i;
}

public int m4(int i) {
    i = i & 0x7BFF;
    bogus += 2;
    i -= i >> 2;
    do {
        if (i<6)
            P(bogus<i);
        i = i >> 3;
        label: {
            if (++bogus <= 0) {
                i = i | 0x1000;
                m3(0);
                if (((bogus+=6)==0)
                    break label;
            }
            ++bogus;
            i = i * 88 >>> 1;
        }
    } while (((bogus += 6)<0) && (m3(9)>=0));
    i = i | 0x4;
    m3(0);
Algorithm $\text{wMVVS: Recognition}$

- So, how do you find the watermark CFG among all the “real” CFGs?
Algorithm \texttt{wMVVS}: Recognition

- So, how do you find the watermark CFG among all the “real” CFGs?
- Idea:
  - Mark the basic blocks,
  - A 0 for every cover program block, a 1 for every watermark block.
Algorithm wMWVS: Recognition

So, how do you find the watermark CFG among all the “real” CFGs?

Idea:

- **Mark** the basic blocks,
  - A 0 for every cover program block, a 1 for every watermark block.

Recognition procedure:
- compute the mark value for each basic block in the program
Algorithm \texttt{VVMVSS: Recognition}

So, how do you find the watermark CFG among all the “real” CFGs?

\textbf{Idea:}
- Mark the basic blocks,
  - A 0 for every cover program block, a 1 for every watermark block.

\textbf{Recognition procedure:}
1. compute the mark value for each basic block in the program
2. assume that any function with more than $t\%$ blocks marked is a watermark function
Algorithm wM VVS: Recognition

So, how do you find the watermark CFG among all the “real” CFGs?

Idea:

- **Mark** the basic blocks,
- A 0 for every cover program block, a 1 for every watermark block.

Recognition procedure:

1. compute the mark value for each basic block in the program
2. assume that any function with more than $t\%$ blocks marked is a watermark function
3. construct CFGs for the watermark functions
Algorithm \textsc{wMVVS}: Recognition

So, how do you find the watermark CFG among all the “real” CFGs?

Idea:

- **Mark** the basic blocks,
- A 0 for every cover program block, a 1 for every watermark block.

Recognition procedure:

1. compute the mark value for each basic block in the program
2. assume that any function with more than \( t\% \) blocks marked is a watermark function
3. construct CFGs for the watermark functions
4. decode each one into an integer watermark
Algorithm wM VVS: Recognition

So, how do you find the watermark CFG among all the “real” CFGs?

Idea:

- **Mark** the basic blocks,
- A 0 for every cover program block, a 1 for every watermark block.

Recognition procedure:

1. compute the mark value for each basic block in the program
2. assume that any function with more than $t\%$ blocks marked is a watermark function
3. construct CFGs for the watermark functions
4. decode each one into an integer watermark

The embedder can split the watermarking into pieces, for higher bitrate.
Steganographic Embeddings

Authorship mark
((in)visible,robust)

Fingerprint mark
(invisible,robust)

Licensing mark
(invisible,robust)

Validation mark
(visible,fragile)

Meta-data mark
(visible,fragile)

Filtering mark
(visible,robust)

Assignment mark
(invisible,robust)

Metadata mark
(visible,fragile)

MD5(kitten.jpg)

PG−13

<license object="kitten.jpg">
  <grant to="Alice">
    <right>copy−once</right>
  </grant>
</license>

© 2006 Collberg

Customer #27182818

Customer #31415926

"Cute kitten in window in Venice"

"Attack mice at dawn"

0xc6ba8f25d2dfe518d7f327c8e83f
Watermark Embeddings

- Watermarks are
  - short identifiers
  - difficult to locate
  - hard to destroy
Watermark Embeddings

- Watermarks are
  - short identifiers
  - difficult to locate
  - hard to destroy
- The adversary
  - knows that the object is marked
  - knows the algorithm used
  - doesn’t know the key
  - is active
Watermark Embeddings

- Watermarks are
  - short identifiers
  - difficult to locate
  - hard to destroy

- The adversary
  - knows that the object is marked
  - knows the algorithm used
  - doesn’t know the key
  - is active

- You care about
  - data-rate
  - stealth
  - resilience
Steganographic Embeddings

- Stegomarks are
  - long identifiers
  - difficult to locate
Steganographic Embeddings

Stegomarks are
- long identifiers
- difficult to locate

The adversary
- wants to know if the object is marked
- knows the algorithm used
- doesn’t know the key
- is passive
Steganographic Embeddings

- Stegomarks are
  - long identifiers
  - difficult to locate

- The adversary
  - wants to know if the object is marked
  - knows the algorithm used
  - doesn’t know the key
  - is passive

- You care about
  - data-rate
  - stealth
Steganography — Prisoners’ Problem

Alice

Bob
Steganography — Prisoners’ Problem

Alice

Bob
Steganography — Prisoners’ Problem
Steganography — Prisoners’ Problem
Steganography — Prisoners’ Problem

Alice

Wendy

Bob
Steganography — Prisoners’ Problem

ESCAPE AT DAWN!

Alice

Wendy

Bob
Steganography — Prisoners’ Problem

Wendy

ESCAPE AT DAWN!

Alice

Bob
Steganography — Null cipher

Easter is soon, dear! So many flowers! Can you smell them? Are you cold at night? Prison food stinks! Eat well, still! Are you lonely? The prison cat is cute! Don’t worry! All is well! Wendy is nice! Need you! :}
WMASB: Hidden Messages in x86 Binaries

Basic idea: **Play compiler!**

*whenever the compiler has a choice in which code to generate, or the order in which to generate it, pick the choice that embeds the next bits from the message \( W \).*
Basic idea: Play compiler!

whenever the compiler has a choice in which code to generate, or the order in which to generate it, pick the choice that embeds the next bits from the message $W$.

Four sources of ambiguity:

1. code layout (ordering of chains of basic blocks)
Basic idea: Play compiler!

*whenever the compiler has a choice in which code to generate, or the order in which to generate it, pick the choice that embeds the next bits from the message $W$."

Four sources of ambiguity:
1. code layout (ordering of chains of basic blocks)
2. instruction scheduling (instruction order within basic blocks)
WMASB: Hidden Messages in x86 Binaries

Basic idea: **Play compiler!**

*whenever the compiler has a choice in which code to generate, or the order in which to generate it, pick the choice that embeds the next bits from the message W.*

Four sources of ambiguity:

1. code layout (ordering of chains of basic blocks)
2. instruction scheduling (instruction order within basic blocks)
3. register allocation
WMASB: Hidden Messages in x86 Binaries

Basic idea: **Play compiler!**

*whenever the compiler has a choice in which code to generate, or the order in which to generate it, pick the choice that embeds the next bits from the message W.*

Four sources of ambiguity:
1. code layout (ordering of chains of basic blocks)
2. instruction scheduling (instruction order within basic blocks)
3. register allocation
4. instruction selection
WMASB: Embedding

1 Construct:
   1 codebook $\mathcal{B}$ of equivalent instruction sequences

\[
\begin{align*}
&\text{mul } r_i, x, 5 \\
&\text{shl } r_i, x, 2 \\
&\text{add } r_i, r_i, x \\
&\text{add } r_i, x, x \\
&\text{add } r_i, r_i, r_i \\
&\text{add } r_i, r_i, x
\end{align*}
\]

2 statistical model $\mathcal{M}$ of real code
WMASB: Embedding

1. Construct:
   1. codebook $B$ of equivalent instruction sequences

   $\begin{array}{c}
   \text{mul} \quad r_i, x, 5 \\
   \text{shl} \quad r_i, x, 2 \\
   \text{add} \quad r_i, r_i, x \\
   \text{add} \quad r_i, x, x \\
   \text{add} \quad r_i, r_i, r_i \\
   \text{add} \quad r_i, r_i, x \\
   \end{array}$

2. statistical model $M$ of real code

2. Encrypt $W$ with key.
WMASB: Embedding

1 Construct:
   1 codebook $B$ of equivalent instruction sequences

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>mul</td>
<td>$r_i, x, 5$</td>
</tr>
<tr>
<td>shl</td>
<td>$r_i, x, 2$</td>
</tr>
<tr>
<td>add</td>
<td>$r_i, r_i, x$</td>
</tr>
<tr>
<td>add</td>
<td>$r_i, x, x$</td>
</tr>
<tr>
<td>add</td>
<td>$r_i, r_i, r_i$</td>
</tr>
<tr>
<td>add</td>
<td>$r_i, r_i, x$</td>
</tr>
</tbody>
</table>

2 statistical model $M$ of real code

2 Encrypt $W$ with key.

3 Canonicalize $P$:
   1 Sort block chains, procedures, modules
   2 Order instructions in each block in standard order
WMASB: Embedding

**Code layout**: Embed bits from $W$ by reordering code segments within the executable.
WMASB: Embedding

4. **Code layout**: Embed bits from $W$ by reordering code segments within the executable.

5. **Instruction scheduling**:
   1. Build dependency graph
   2. Generate all valid instruction schedules
   3. Embed bits from $W$ by picking a schedule

Use $\mathcal{M}$ to avoid picking unusual schedules.
WMASB: Embedding

**Code layout:** Embed bits from $W$ by reordering code segments within the executable.

**Instruction scheduling:**
1. Build dependency graph
2. Generate all valid instruction schedules
3. Embed bits from $W$ by picking a schedule

Use $M$ to avoid picking unusual schedules.

**Instruction selection:** Use $B$ to embed bits from $W$ by replacing instructions. Use $M$ to avoid unusual instruction sequences.
WMASB: Stealth

Instruction selection:
- There are 3078 different encodings of three instructions for $EAX = (EAX/2)$!
- Most don’t occur in real code...
WMASB: Stealth

- **Instruction selection:**
  - There are 3078 different encodings of three instructions for $\text{EAX} = (\text{EAX}/2)$!
  - Most don’t occur in real code...

- **Instruction scheduling:**
  - Avoid bad schedules: no compiler would generate it!
  - Avoid generating different schedules for two blocks with the same dependency graph!
wMASB: Stealth

Instruction selection:
- There are 3078 different encodings of three instructions for \( EAX = (EAX/2) \).
- Most don’t occur in real code.

Instruction scheduling:
- Avoid bad schedules: no compiler would generate it!
- Avoid generating different schedules for two blocks with the same dependency graph!

Code layout:
- Compilers lay out code for locality: don’t deviate too much from that!
wmASB: Stealth

Encoding rate

- Unstealthy code: $\frac{1}{27}$
- Stealthy: $\frac{1}{89}$. 
\textbf{WMASB: Stealth}

- **Encoding rate**
  - Unstealthy code: \( \frac{1}{27} \)
  - Stealthy: \( \frac{1}{89} \).

- **Encoding space:**
  - 58\% from code layout
  - 25\% from instruction scheduling
  - 17\% from instruction selection
wmASB: Stealth

- Encoding rate
  - Unstealthy code: $\frac{1}{27}$
  - Stealthy: $\frac{1}{89}$

- Encoding space:
  - 58% from code layout
  - 25% from instruction scheduling
  - 17% from instruction selection

- Real code doesn’t use unusual instruction sequences.

- Real code contains many schedules for the same dependency graph
Wanna design a watermarking algorithm?

- Find a **language structure** into which to encode the mark (CFGs, threads, dynamic control flow...)

⟨language structure, encoder/decoder, tracer/locator, embedder/extractor, attacker/protector⟩
Wanna design a watermarking algorithm?

- Find a **language structure** into which to encode the mark (CFGs, threads, dynamic control flow...)
- Construct an **encoder/decoder** (number $\leftrightarrow$ CFG,...)

$\langle$ **language structure**, **encoder/decoder**, **tracer/locator**, **embedder/extractor**, **attacker/protector$\rangle$
Wanna design a watermarking algorithm?

- Find a **language structure** into which to encode the mark (CFGs, threads, dynamic control flow . . .)
- Construct an **encoder/decoder** (number ↔ CFG, . . .)
- Construct a **tracer/locater** to find locations for the mark (using key, every function, . . .)

⟨**language structure**, **encoder/decoder**, **tracer/locater**, **embedder/extractor**, **attacker/protector**⟩
Wanna design a watermarking algorithm?

- Find a **language structure** into which to encode the mark (CFGs, threads, dynamic control flow...)
- Construct an **encoder/decoder** (number↔CFG,...)
- Construct a **tracer/locator** to find locations for the mark (using key, every function,...)
- Construct a **embedder/extractor** to tie the mark to surrounding code

\[
\langle \text{language structure, encoder/decoder, tracer/locator, embedder/extractor, attacker/protector} \rangle
\]
Wanna design a watermarking algorithm?

- Find a **language structure** into which to encode the mark (CFGs, threads, dynamic control flow . . .)
- Construct an **encoder/decoder** (number ↔ CFG, . . .)
- Construct a **tracer/locater** to find locations for the mark (using key, every function, . . .)
- Construct a **embedder/extractor** to tie the mark to surrounding code
- Decide on an **attack model**.

\[ \langle \text{language structure, encoder/decoder, tracer/locater, embedder/extractor, attacker/protector} \rangle \]
Dynamic Watermarking
Static watermarking?

Embedding ideas: Encode the watermark

1. as a permutation of the original code, or
2. in new but non-functional code
Static watermarking?

- **Embedding ideas**: Encode the watermark
  1. as a permutation of the original code, or
  2. in new but non-functional code

- **Recognition**: Extract the mark by analyzing the code itself.
Static watermarking?

- **Embedding ideas**: Encode the watermark
  1. as a permutation of the original code, or
  2. in new but non-functional code

- **Recognition**: Extract the mark by analyzing the code itself.

- **Attack ideas**: Disrupt the recognizer by
  1. permuting the original code, or
  2. embedding your own watermark, or …
Dynamic watermarking!

**Embedding idea**: Embed the mark by adding code that changes the program’s behavior.
Dynamic watermarking!

- **Embedding idea**: Embed the mark by adding code that changes the program’s behavior.
- **Recognition idea**: Extract the mark by running the program and analyzing its behavior.
Dynamic watermarking!

- **Embedding idea**: Embed the mark by adding code that changes the program’s behavior.
- **Recognition idea**: Extract the mark by running the program and analyzing its behavior.
- ⇒ The program produces the watermark into its *state*. 
Dynamic watermarking API

- **Secret key**: a special input sequence 
  \[ I_1, \ldots, I_k \]
Dynamic watermarking API — Example

- Secret input sequence to word processor application:

  \langle \text{spell check, enter text, change font, ...} \rangle.
Dynamic watermarking API —
Example

- **Secret input sequence** to word processor application:

  \langle \text{spell check, enter text, change font, ...} \rangle.

- **Recognition procedure**: 
  1. Run the watermark program with the secret input,
Dynamic watermarking API — Example

- **Secret input sequence** to word processor application:

\(\langle\text{spell check, enter text, change font, \ldots}\rangle\).

- **Recognition procedure**:
  1. Run the watermark program with the secret input,
  2. Examine the state for the watermark.
Example

Simple semantics-preserving transformations are less likely to affect a dynamic watermark.

Secret watermarking input sequence: \langle "hello", "world" \rangle.

The mark is stored in a global variable watermark:

```c
int watermark=0;
...
if (read() == "hello")
    if (read() == "world")
        watermark=42;
...
```
Example

An attacker might apply various static code transformations to destroy the mark:

```c
int watermark=0;
int x = 3;
...
x = x*2;
if (read() == "hello")
    if (read() == "world")
        watermark = x*6+6;
...
```

- Static recognizers — get confused!
- Dynamic recognizers — run the code, look at the value of `watermark`, read out the mark.
Attack against dynamic state

- The attacker could split the watermark variable!
- Careful in choosing what state in which you store the mark!
  1. It should be difficult for the adversary to modify the state.
  2. Modifying the state would make the program too slow or too large to be useful.
  3. It should be easy to tamperproof the state.
Disadvantages of dynamic watermarks

1. Dynamic algorithms can’t protect parts of programs: to recognize the mark the program needs to be executed!
2. Dynamic watermarks have to be executed to be recognized.
3. Non-determinism in the program can affect the recognizer.
Algorithm $\text{WMCT}$: Exploiting aliasing

- Basic insight: Pointer analysis is hard!
Algorithm \texttt{WMCT}: Exploiting aliasing

- Basic insight: Pointer analysis is hard!
- Basic embedding idea:
Algorithm \texttt{wMCT}: Exploiting aliasing

- Basic insight: Pointer analysis is hard!
- Basic embedding idea:
  1. Let $n$ be the watermark number
Algorithm \texttt{wmCT}: Exploiting aliasing

- Basic insight: Pointer analysis is hard!
- Basic embedding idea:
  1. Let \( n \) be the watermark number
  2. Convert \( n \) to a graph \( G \) that encodes the number
Algorithm $\textit{WMCT}$: Exploiting aliasing

- Basic insight: Pointer analysis is hard!
- Basic embedding idea:
  1. Let $n$ be the watermark number
  2. Convert $n$ to a graph $G$ that encodes the number
  3. Convert $G$ to code $C$ that builds the graph
Algorithm \textsc{wmCT}: Exploiting aliasing

- Basic insight: Pointer analysis is hard!
- Basic embedding idea:
  1. Let $n$ be the watermark number
  2. Convert $n$ to a graph $G$ that encodes the number
  3. Convert $G$ to code $C$ that builds the graph
  4. Add $C$ to your program so that it’s executed for the special input
Algorithm \textit{wmCT}: Exploiting aliasing

- **Basic insight:** Pointer analysis is hard!
- **Basic embedding idea:**
  1. Let $n$ be the watermark number
  2. Convert $n$ to a graph $G$ that encodes the number
  3. Convert $G$ to code $C$ that builds the graph
  4. Add $C$ to your program so that it’s executed for the special input

- **Basic recognition idea:**
  1. Run the program for the special input
Algorithm \textsf{wmCT}: Exploiting aliasing

- Basic insight: Pointer analysis is hard!
- Basic embedding idea:
  1. Let $n$ be the watermark number
  2. Convert $n$ to a graph $G$ that encodes the number
  3. Convert $G$ to code $C$ that builds the graph
  4. Add $C$ to your program so that it’s executed for the special input
- Basic recognition idea:
  1. Run the program for the special input
  2. Dump all objects on the heap
Algorithm \textit{wmCT}: Exploiting aliasing

- Basic insight: Pointer analysis is hard!
- Basic embedding idea:
  1. Let $n$ be the watermark number
  2. Convert $n$ to a graph $G$ that encodes the number
  3. Convert $G$ to code $C$ that builds the graph
  4. Add $C$ to your program so that it’s executed for the special input
- Basic recognition idea:
  1. Run the program for the special input
  2. Dump all objects on the heap
  3. Reconstruct the graph $G$
Algorithm wMC\textsc{T}: Exploiting aliasing

- Basic insight: Pointer analysis is hard!
- Basic embedding idea:
  1. Let $n$ be the watermark number
  2. Convert $n$ to a graph $G$ that encodes the number
  3. Convert $G$ to code $C$ that builds the graph
  4. Add $C$ to your program so that it’s executed for the special input
- Basic recognition idea:
  1. Run the program for the special input
  2. Dump all objects on the heap
  3. Reconstruct the graph $G$
  4. Convert $G$ to the watermark number $n$
Algorithm \textsc{wmCT}: Example

```java
public class M {
    public void main (String args[]) {
        for (int i=1; i<args.length; i++) {
            if (args[0].equals(args[i])) {
                System.out.println("YES");
                return;
            }
        }
        System.out.println("NO");
    }
}
```

> java M 2 3 4 5 2
YES
> java M 2 3 4 5 3
NO
Algorithm WMCT: Example

- Here we’re using a radix encoding.
- Let $n = 2$, the base-2 expansion is $1 \cdot 2^1 + 0 \cdot 2^0$: we get the graph

```
WMNode n2 = new WMNode();
n2.digit = n2;
Node n1 = new WMNode();
n2.spine = n1;
n1.spine = n2;
n1.digit = n2;
```

- Here’s the corresponding graph-building code:
Algorithm \textsc{wmCT}: Example

```java
public class M {
    class WMNode {
        public Node spine, digit;
    }
    public static WMNode root;
    public void main (String args[]) {
        if (args[0].equals("2")) {
            Build the graph here!
        }
        for (int i=1; i<args.length; i++) {
            if (args[0].equals(args[i])) {
                System.out.println("YES");
                return;
            }
        }
        System.out.println("NO");
    }
}
```
Increasing bitrate

How do we increase the bitrate?
Increasing bitrate

- How do we increase the bitrate?
- Easy — just build a bigger graph!
Increasing bitrate

- How do we increase the bitrate?
  - Easy — just build a bigger graph!
  - But — bigger graph, less stealth!
Increasing bitrate

How do we increase the bitrate?
Easy — just build a bigger graph!
But — bigger graph, less stealth!
Idea:
1. split the graph into smaller pieces!
2. choose an efficient graph encoding!
Increasing bitrate — Choosing efficient graph encoding

![Graph showing size vs watermark for different graph bytecode sizes](image)
Increasing bitrate — Splitting graphs
public static Node root;
public static Node n2;
public void main (String args[]) {
    if (args[0].equals("2")) {
        n2 = new Node();
        Node n3 = new Node();
        n2.digit = n3;
        n2.spine = n3;
    }
    for (int i=1;i<args.length;i++) {
        if (found(args[0], args, i)) {
            System.out.println("YES");
            return;
        }
    }
    System.out.println("NO");
}
public boolean found (String value, String args[], int i) {
    if (value.equals(args[i])) {
        if ((i == 4) && (n2 != null)) {
            Node n1 = new Node();
            n1.digit = n1;
            n1.spine = n2;
            Node n3 = n2.spine;
            n3.spine = n1;
            n3.digit = n1;
            root = new Node();
            root.spine = n1;
        }
        return true;
    }
    return false;
}
Size of Java bytecode for building a split Radix Graph

Why does the total size grow with the number of components?
Increasing resilience — thwarting pointer analysis

Node n3 = new Node();
Node n2 = new Node();
n3.spine = n2; n2.digit = n3;
Node n1 = new Node();
n1.digit = n1; n1.spine = n3;
n3.digit = n1; n2.spine = n1;
root = new Node();
root.spine = n1;
...
Node tmp = root.spine.spine;
root.spine.spine = tmp.spine;
tmp.spine.spine = tmp;
tmp.spine = tmp.digit;
Increasing stealth — avoiding unstealthy classes

```java
class Person {
    public Object spouse;
}
class Node extends Person {
    public Object digit;
    ...
}

Node n1 = new Node();
n1.digit = n1;
java.awt.Event n3 =
    new java.awt.Event(n1, 0, n1);
Object n2[] = \{n3, n3\};
n1.spouse = n2;
root = new Person();
root.spouse = n1;
```
Avoiding weak cuts

```java
public static void main (String args[]) {
    int n = args.length;
    if (args[0].equals("2")) {
        Node root = new Node(); root.spine = n1;
        Node n2 = new Node(); n1.spine = n2;
        n1.digit = n2; n2.spine = n1; n2.digit = n2;
        ...
        Node p = root.spine; int k = 0;
        while (p.spine != root.spine) {p=p.spine; k++;}
        n *= k;
    }
    for (int i=1; i<n; i++) {
        if (args[0].equals(args[i]) &&
            ((root==null) || (root.spine!=root.spine.spine.spine)))
            System.out.println("YES"); return;
    }
}
```
Algorithm \text{WMCT}: Summary

- **Size overhead** — embed a 20-bit watermark only requires 141 bytes of Java bytecode. (Tamperproofing code adds more).
Algorithm wMCT: Summary

- **Size overhead** — embed a 20-bit watermark only requires 141 bytes of Java bytecode. (Tamperproofing code adds more).

- **Performance overhead** — the graph-building code is only executed for the special input.
Algorithm \textit{WMCT}: Summary

- **Size overhead** — embed a 20-bit watermark only requires 141 bytes of Java bytecode. (Tamperproofing code adds more).

- **Performance overhead** — the graph-building code is only executed for the special input.

- **Chief advantage** — tamperproofing is easy.
Algorithm wmCT: Summary

- **Size overhead** — embed a 20-bit watermark only requires 141 bytes of Java bytecode. (Tamperproofing code adds more).
- **Performance overhead** — the graph-building code is only executed for the special input.
- **Chief advantage** — tamperproofing is easy.
- **Attacks** — Scan the code for allocations and pointer manipulations.
Algorithm WMCT: Summary

- **Size overhead** — embed a 20-bit watermark only requires 141 bytes of Java bytecode. (Tamperproofing code adds more).
- **Performance overhead** — the graph-building code is only executed for the special input.
- **Chief advantage** — tamperproofing is easy.
- **Attacks** — Scan the code for allocations and pointer manipulations.
- **Stealth** — good in OO programs.
WMNT — Thread-Based Watermark

Embed mark in which threads execute which basic blocks.
Can have huge performance degradation.
Why? **Parallelism-analysis** is hard.