

Data Embedding using Object-Code Instruction Reordering

Peter Meerwald

January 7, 2004

Abstract

Ways to embed information in object-code for the Intel x86 architecture.
Instruction reordering (IR) for data hiding, steganography and watermarking.

Data Embedding

Def.: Data Embedding: Adding a message to some host data (eg. image, audio, software, . . .) without changing its size.

perceptible / imperceptible?

distortion (perceptual quality; correctness, runtime)

Characteristics

- capacity (Data Hiding)
- detectability (Steganography)
- robustness (Watermarking)

data embedding is often just called 'watermarking'

Multi-media vs. Software Watermarking

multi-media watermarking (and its applications) well known: copyright protection, tamper detection, steganography

- have perceptual models (due to compression research)

software watermarking less known, but gaining interest due to on-line software distribution

- have correctness and runtime constraints (compiler research)

why?

control software distribution; eg. fingerprinting, owner identification, authenticity, . . .

but: alternatives available, eg. hashing, certificates, . . .

Software Watermarking Overview

Characteristics

- watermark embedding in source-, intermediate- or object-code
- dynamic (at runtime) or static detection of watermark [2]
- local (instruction rewriting) or global (control-flow) host modification [5]
- can use spread-spectrum or quantization embedding

related field: software obfuscation (software transformations to prevent reverse-engineering and tampering) [1]

Example: Instruction Rewriting

used by hydan¹ and Stern et al.[4]

substitute instruction $i \in S$ with equivalent instruction i_w from set S to encode information

eg. single bit, $|S| = 2$

to embed 0		to embed 1
add <i>reg</i> , CONST		sub <i>reg</i> , -CONST

eg. multi-bit, $|S| = 4$

to embed 00		to embed 01		to embed 10		to embed 11
xor r/m32, r32		xor r32, r/m32		sub r/m32, r32		sub r32, r/m32

hydan uses 12 different replacement patterns (redundancies in instruction set)

¹<http://www.crazyboy.com/hydan/>

Data Embedding with Instruction Reordering (IR)

local modification of object-code (rewriting of instruction groups), static detection
reorder instructions within an instruction group to embed information; can build replacement codebook for efficient embedding

instructions must be independent (detail on next slide)

instructions must be within basic block² of execution, need program analysis (later)

Example

push CONST		mov <i>reg</i> , CONST
mov <i>reg</i> , CONST		push CONST

²A basic block is a sequence of instructions with a single entry point (and a single exit point).

Instruction Dependencies (1)

looking at instruction tuples (pairs, triples, . . .); similar to peephole optimization

data dependency: instructions read (R) or write (W) the same resource (R-W, W-R, W-W conflicts)

flag dependency: instructions may set or test processor flags; not explicitly denoted by instruction's mnemonic

control dependency: can not reorder instructions `jmp`, `call`, `jcc`, `ret`, . . .

identical instructions: can not be reordered :-)

with larger context, certain dependencies can be ignored (next slide)

Instruction Dependencies (2)

data dependency conflict (W-R, W-W)

<code>mov reg, CONST</code>		W: reg
<code>add reg, reg</code>		R: reg, W: reg (write-read & write-write conflict)

flag dependency conflict (W-W)

<code>add reg, CONST</code>		W: flag
<code>cmp [MEM], CONST</code>		W: flag (write-write conflict)

instruction replacement ignoring data dependency

<code>mov reg1, reg2</code>		<code>xchg reg1, reg2</code>
<code>mov reg2, reg3</code>		<code>mov reg2, reg3</code>

instruction replacement ignoring flag dependency

<code>add reg1, CONST</code>		<code>add reg2, CONST</code>
<code>add reg2, CONST</code>		<code>add reg1, CONST</code>
<code>add reg3, CONST</code>		<code>add reg3, CONST ; W: flag</code>

Program Analysis (1)

best-possible disassembly: use hybrid recursive traversal / linear sweep algorithm [3]

recursive traversal yields CFG, recovery by linear sweep

1000	add eax, ebx	1000	add eax, ebx
1002	ret	1002	ret
1003	db 0 ; padding	1003	db 0 ; padding
1004	mov eax, 1	1004	mov eax, 1
1009	ret	1009	ret
100a	db 0 ; padding	100a	db 0 ; padding
100b	db 0 ; padding	100b	db 0 ; padding
100c	xor eax, eax ; entry point	100c	xor eax, eax ; entry point
100e	call 1000	100e	call 1000
1013	call [eax*4] ; ???	1013	call [eax*4] ; ???
101a	hlt	101a	hlt
101b	jmp [eax] ; ???	101b	jmp [eax] ; ???
101d	call [1022]	101d	call [1022]
1021	hlt	1021	hlt
1022	dd 101b ; data	1022	dd 101b ; data
1026	call 1004	1026	call 1004

Program Analysis (2)

need control-flow graph (CFG) to determine basic blocks

can associate basic blocks to source-level functions; locally complete CFG sufficient

data-flow analysis difficult but necessary for correct / complete CFG

```
void f(void) {  
    switch (iChoice) {  
        case 0: ...  
        case 1: ...  
    }  
    ...  
}
```

```
_f proc near  
    mov eax, iChoice  
    jmp [eax * 4 + offset  
        switch_jump_table]  
switch_case_0: ...  
switch_case_1: ...  
    ...  
    ret ; end of f()  
switch_jump_table:  
    dd offset switch_case_0  
    dd offset switch_case_1
```

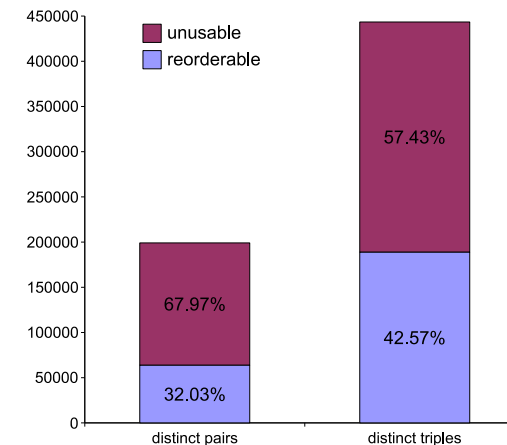
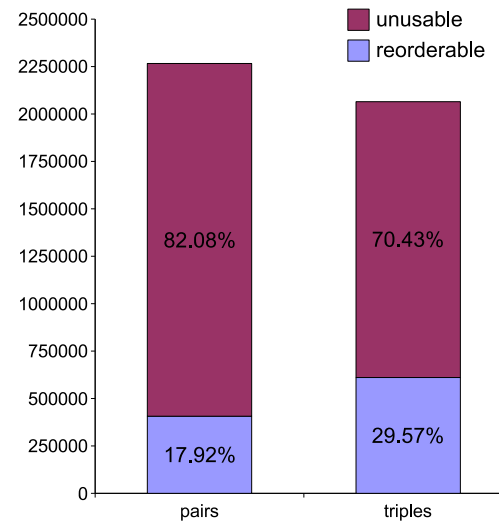
can reduce code-book size by normalizing instructions (set immediate values, offsets, ... to zero); store only normalized instructions

Building an IR Codebook

analyzing 50 programs from Debian /usr/bin

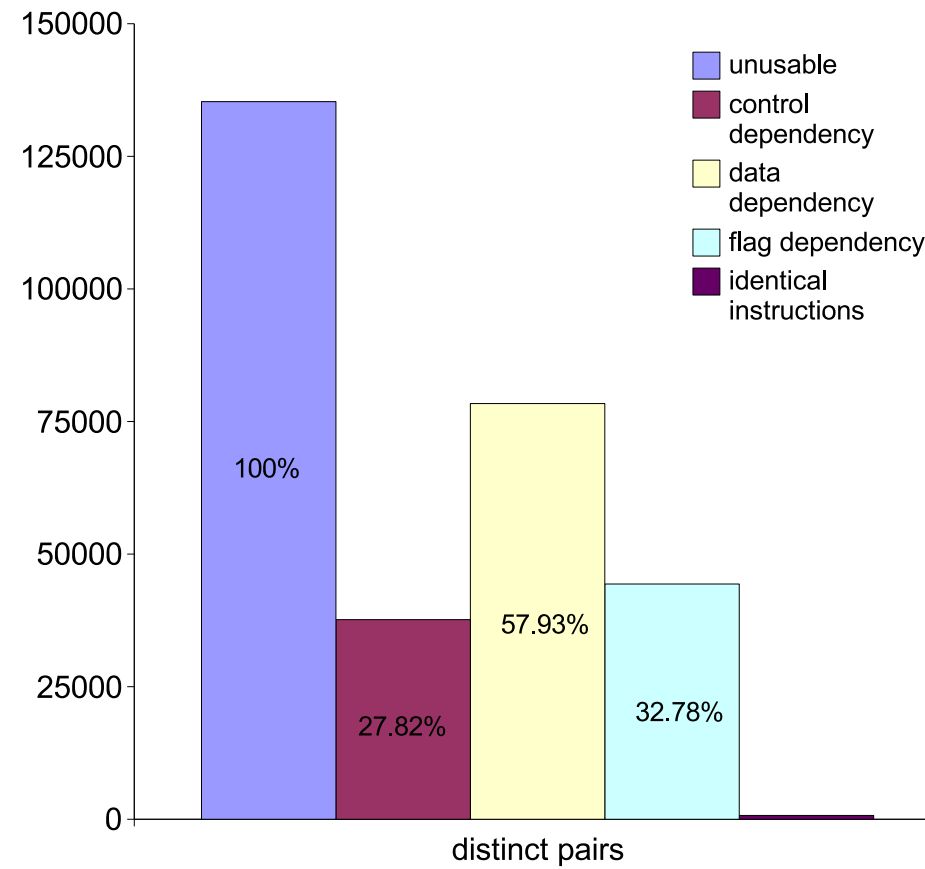
using custom disassembler written in Python building on libdisasm³

2492602	instr.
2266381	instr. pairs
2064811	instr. triples
20845	norm. instr.
199079	norm. pairs
443467	norm. triples



³<http://bastard.sourceforge.net/libdisasm.html>

Instruction Pair Dependencies



Hydan vs. IR Embedding Capacity

program	code (Bytes)	hydan (Bits)	hydan (Rate)	IR (Bits)	IR (Rate)
bison	134096	7136	1 / 150	4471	1 / 239
gcc	55474	2304	1 / 192	2424	1 / 183
cdrdao	281916	17264	1 / 130	8196	1 / 275
gnuplot	478912	23104	1 / 165	17977	1 / 213
unzip	66425	3352	1 / 158	2471	1 / 215
perl	921202	33200	1 / 221	24000	1 / 307
awk	160768	9720	1 / 132	4547	1 / 283
tr	16178	600	1 / 215	604	1 / 214
diff	38337	1832	1 / 167	1521	1 / 201
lame	267336	6928	1 / 308	13040	1 / 164
total / avg.	2420644	105440	1 / 184	79251	1 / 244

Hydan: Steganographic Analysis

replacement pattern	original frequency	frequency in marked code
add r/m32, imm8	16368	10702
sub r/m32, -imm8	6757	12423
xor r/m32, r32	18566	4960
xor r32, r/m32	0	4548
sub r/m32, r32	0	4614
sub r32, r/m32	0	4444
test r32, r32	40274	10330
or r32, r32	0	9860
or* r32, r32	0	10056
and r32, r32	0	10028

only looking at global frequency; top-3 hydan replacements (78 % of capacity)

IR Steganographic Capacity

program	IR (Bits)	IR (Rate)	Stego-IR (Bits)	Stego-IR (Rate)
bison	4471	1 / 239	1281	1 / 837
gcc	2424	1 / 183	514	1 / 863
cdrdao	8196	1 / 275	1865	1 / 1209
gnuplot	17977	1 / 213	2961	1 / 1294
unzip	2471	1 / 215	370	1 / 1436
perl	24000	1 / 307	5539	1 / 1300
awk	4547	1 / 283	678	1 / 1896
tr	604	1 / 214	96	1 / 1348
diff	1521	1 / 201	291	1 / 1054
lame	13040	1 / 164	2467	1 / 867
total / avg.	79251	1 / 244	16062	1 / 1205

Steganographic capacity is 20% of embedding capacity using IR

Watermarking (1)

following Stern's [4] vector extraction paradigm – separate domain-specific vector extraction from watermark embedding:

- use frequency of most prominent replacable instruction tuples to build vector v (vector extraction step)
- chose w , suitable Gaussian random sequence
- embed watermark w in extracted vector v : $v' = v + w$ (vectors of length n)
- $w' = v'' - v$ and use standard detection (correlation), $c = \frac{w \cdot w'}{\sqrt{w \cdot w} \sqrt{w' \cdot w'}}$, compare with threshold δ

this is a spread-spectrum scheme; could also quantization method

Watermarking (2)

Robustness Assessment

- code reordering (compute CFG and shuffle basic blocks)
- code addition (need to determine amount of code, separability?)
- code decompiling (practical?)
- code compression (can only detect with decompressed code)
- code obfuscation (?)

but: there is no distortion constraint on attacker (maybe code runtime?), attacker can set all possible IRs to zero

solution may be to make program analysis hard: combine watermarking and obfuscation

Conclusion & Further Work

- ✓ IR is a mechanism for data embedding in object code
- ✓ have embedding rate comparable to hydan but probably multi-platform
- ✓ can achieve steganographic capacity – local analysis?
- ✗ need to build larger codebook, look at other architectures
- ✗ can improve disassembler and instruction dependency model
- ✗ watermarking? in combination with obfuscation?

References

- [1] C. Collberg and C. Thomborson. Watermarking, tamper-proofing, and obfuscation - tools for software protection. *IEEE Transactions on Software Engineering*, 28(6), June 2002.
- [2] Christian Collberg and Clark Thomborson. Software watermarking: Models and dynamic embedding. In *ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '99*, pages 311–324, San Antonio, TX, USA, January 1999.
- [3] Cullen Linn and Saumya Debray. Obfuscation of executable code to improve resistance to static disassembly. In *Proceedings of the 10th. ACM Conference on Computer and Communications Security, CCS '03*, Washington, DC, USA, October 2003.
- [4] Julien P. Stern, Gaël Hachez, François Koeune, and Jean-Jacques Quisquater. Robust object watermarking: Application to code. In Andreas Pfitzmann, editor, *Proceedings of the 3rd Information Hiding Workshop '99*, volume 1768 of *Lecture Notes in Computer Science*, pages 368–378, Dresden, Germany, September 1999. Springer.
- [5] Ramarathnam Venkatesan, Vijay Vazirani, and Saurabh Sinha. A graph theoretic approach for software watermarking. In *Proceedings of the 4th Information Hiding Workshop '01*, Portland, OR, USA, April 2001.