DIFFERENTIATING CODE FROM DATA IN X86 BINARIES

Richard Wartell, Yan Zhou, Kevin W. Hamlen, Murat Kantarcioglu, and Bhavani Thuraisingham

{rhw072000, yan.zhou2, hamlen, muratk, bhavani.thuraisingham} @utdallas.edu

UT DALLAS

Work funded by AFOSR contracts FA9550-09-1-004, FA9550-10-1-0088, and FA9550-08-1-0265, NIH grant 1R01LM009989, and NSF grants Career-0845803, CNS-0984350, CNS-1016343

Why Disassemble

- Problems Requiring Accurate Disassemblies
  - Reverse Engineering
  - Malware Analysis
  - Reference Monitor In-lining
  - Software Fault Isolation
- Disassemblers
  - ObjDump [GNU]
  - OllyDbg [Yuschuk]
  - IDA Pro [Hex-Rays; '11]

Disassembling Java & RISC

- Java Byte Code and RISC Architectures
  - 1 byte opcodes
  - Code and data stored in separate sections

Opcode	Parameter	Address	Instruction
0x05		0x00	iconst 2
0x3C		0x01	istore 1
0x1B		0x02	iload 1
0x11	0x1000	0x03	sipush 1000
0xA2	0x0044	0x06	if icmpge 44
0x05		0x09	iconst 2
0x3D		0x0A	istore 2

Disassembling x86

- Variable length opcodes
- Code and Data interleaved in code sections

Raw Hex
E9 21 60 30 00 50 E8 0C 55 C7 45 FC 55 00 00 00 55 FF 35 08 C3

Correct Disassembly

E9 21 60 30 00	Jmp 306021h
50	push eax
E8 0C	jmp 0Ch
55 C7 45 FC 55 00 00 00 55 FF 35 08	*DB (12 bytes)*
C3	retn

Disassembling x86

- Variable length opcodes
- Code and Data interleaved in code sections

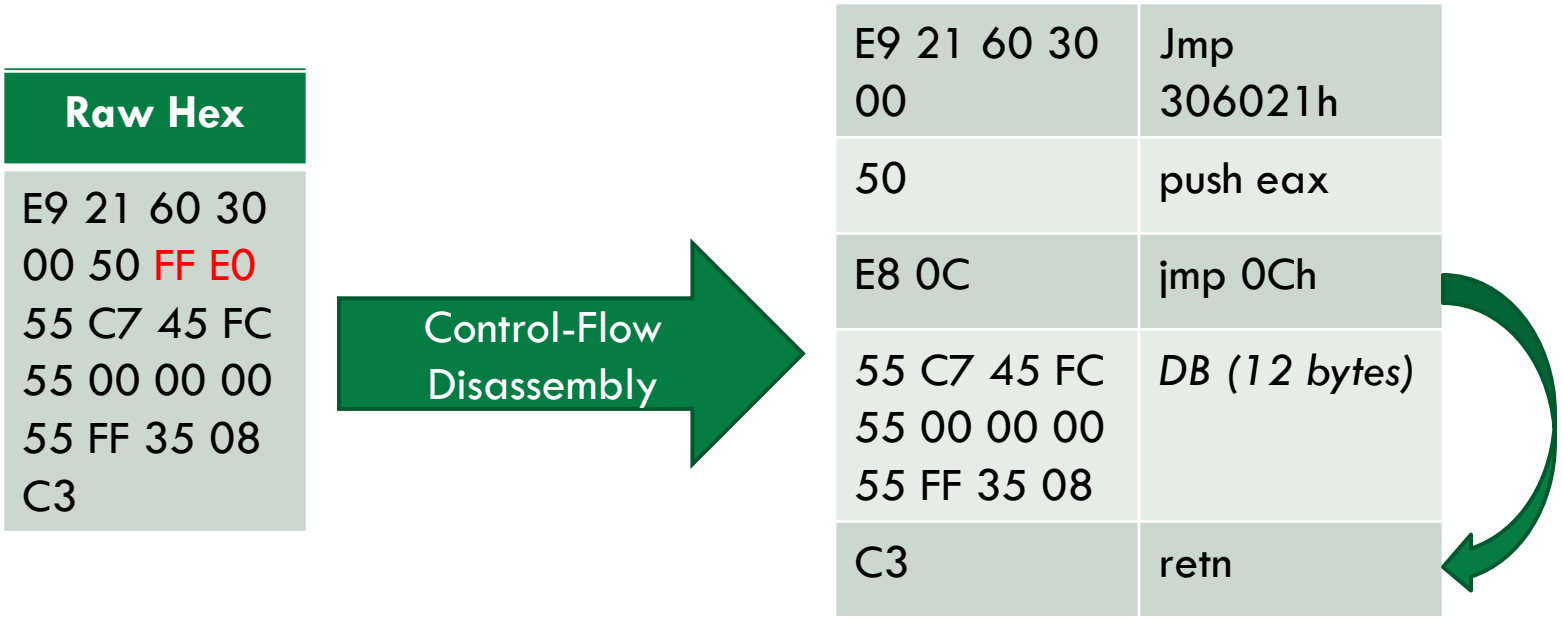
Raw Hex
E9 21 60 30 00 50 E8 0C 55 C7 45 FC 55 00 00 00 55 FF 35 08 C3

Straight – Line Disassembly

E9 21 60 30 00	Jmp 306021h
50	push eax
E8 0C	jmp 0Ch
55	push ebp
C7 45 FC 55 00 00 00	mov [ebp-4], 55h
55	push ebp
FF 35 08 C3 00 01	push 0100C308h

Control Flow Disassembly

- Follows execution path of binary to determine disassembly



Control Flow Disassembly

- Follows execution path of binary to determine disassembly

Raw Hex
E9 21 60 30 00 50 FF E0 55 C7 45 FC 55 00 00 00 55 FF 35 08 C3

Correct Disassembly

E9 21 60 30 00	Jmp 306021h
50	push eax
FF E0	jmp eax
55 C7 45 FC 55 00 00 00 55 FF 35 08	*DB (12 bytes)*
C3	retn

Control Flow Disassembly

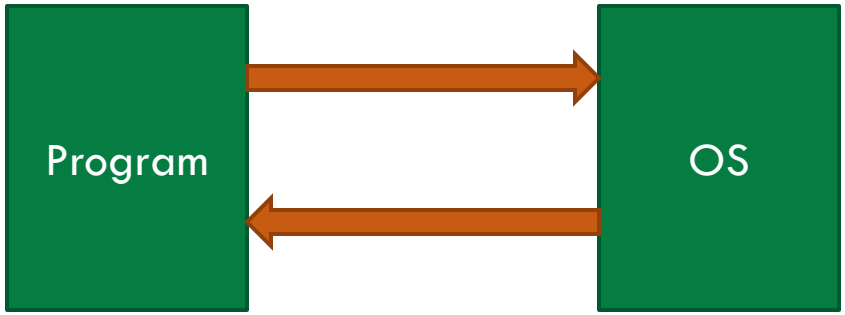
- Follows execution path of binary to determine disassembly

Raw Hex
E9 21 60 30 00 50 FF E0 55 C7 45 FC 55 00 00 00 55 FF 35 08 C3

E9 21 60 30 00	jmp 306021h
50	push eax
FF E0	jmp eax
55 C7 45 FC 55 00 00 00 55 FF 35 08 C3	*DB (13 bytes)*

Hidden Entrypoints

Binary	Address	Instruction
0x89 65 E8	.text:00401000	mov [ebp+ms_exc.old_esp], esp
0x8B F4	.text:00401003	mov esi, esp
0x89 3E	.text:00401005	mov [esi], edi
0x56	.text:00401007	push esi
0x8B 3D CC E2 71 00	.text:00401008	mov edi, ds:GetModuleHandleA
0xFF D7	.text:0040100E	call edi ; GetModuleHandleA



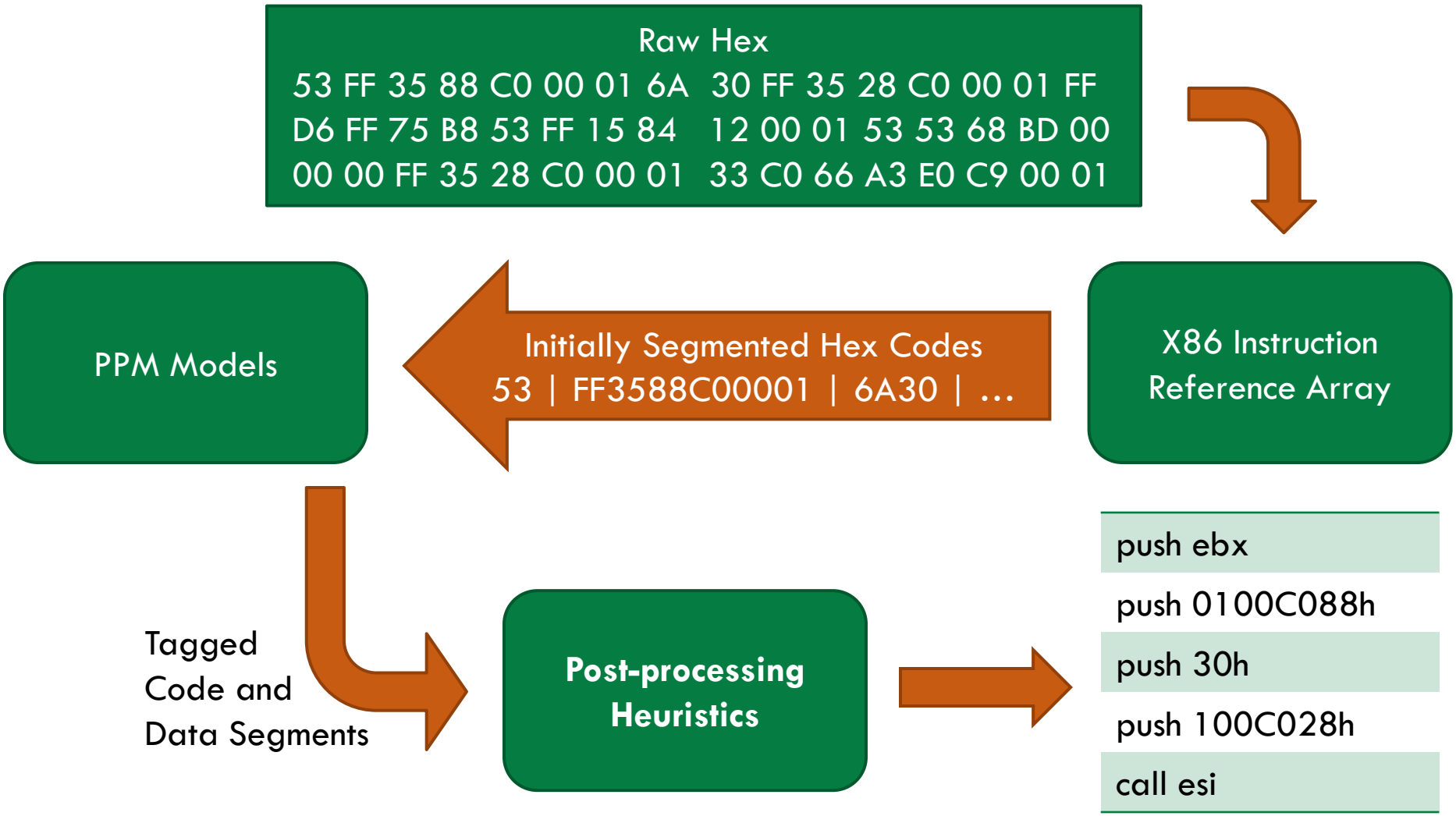
Disassembling x86 Successfully

Binary	Address	Instruction
0x51	.text:00524CB6	push ecx
0xFF 15 B4 FA 7D 00	.text:00524CB7	call off_7DFAB4
0x8B E8	.text:00524CBD	mov ebp, eax
0x8B 83 DC 00 00 00	.text:00524CBF	mov eax, [ebx+0DCh]
0x8B 48 04	.text:00524CC5	mov ecx, [eax+4]
0x3B CE	.text:00524CC8	cmp ecx, esi
0x0F 8E 91 00 00 00	.text:00524CCA	jle loc_524D61
0xC3	.text:00524CD0	retn
0x6A 00 E8 0F 7D FC FF 0F 85 F2 00 0F 00	.text:00524CD1	*Data Bytes (13 bytes)* (used as an indirect jump table)

Disassembling x86 Gone Wrong

Binary	Address	Instruction
0x51	.text:00524CB6	push ecx
0xFF	.text:00524CB7	*Data Byte FFh*
0x15 B4 FA 7D 00	.text:00524CB8	adc eax, 7DFAB4h
0x8B E8	.text:00524CBD	mov ebp, eax
0x8B	.text:00524CBF	*Data Byte 8Bh*
0x83 DC 00	.text:00524CC0	sbb esp, 0
0x00 00 8B 48 04 3B CE 0F 8E 91 00 00 00	.text:00524CC3	*Data Bytes (13 bytes)*
0xC3	.text:00524CD0	retn
0x6A 00	.text:00524CD1	push 0
0xE8 0F 7D FC FF	.text:00524CD3	call FFFC7D0Fh
0x0F 85 F2 00 0F 00	.text:00524CD1	jnz 0F00F2h

The Solution



Instruction Reference Array

- Segment executable byte sequence
- Given an opcode, how long is the instruction?

Algorithm 1: Tagging

Input: $x_0 \dots x_i \dots x_{n-1}$ // input string of bytes
M_c // language model
Output: $x_0 \dots x_i | x_{i+1} \dots x_j | \cdots | x_k \dots x_{n-1}$ // segmented string
$t \leftarrow 0$
while $t < n$ **do**
  $\ell \leftarrow 0$
  if $x_t \in M_c$ **then**
    $\ell \leftarrow codeLength(x_t \dots x_{\min\{t+4,n-1\}})$ // lookup instruction length
  if $(\ell = 0) \vee (t + \ell > n)$ **then** $\ell \leftarrow 1$ // tag as possible data
  print $x_t \dots x_{t+\ell-1}$ // output the segment
  $t \leftarrow t + \ell$

Machine Learning + x86?

- Treat byte sequence as input
- Train on corpus of correctly tagged byte sequences

Tag	Hex	Instruction (or Data)
Code	0x3B CE	cmp ecx, esi
Code	0x0F 8E 91 00 00 00	jle loc_524D61
Code	0xC3	retn
Data	0x6A 00 E8 0F 7D FC FF 0F 85 F2 00 0F	*Data Bytes (12 bytes)*

- Break bytes into code and data
- Test model against executable byte sequence

Prediction by Partial Matching (PPM)

[Cleary, Teahan; TCJ '97] [Cleary, Witten; IEEE ToC '84] [Cormack, Horspool; TCJ '08]

- *k*th order Markov model
  - *k*th order context c_i^k includes only the k prior symbols
- To predict the probability of seeing x_i
  - Match c_i^k in the context tree
    - On a match: return $p(x_i \mid c_i^k)$
    - Otherwise record an escape event and step backwards:
      - Attempt to match c_i^{k-1}
      - On a match: return $p(\text{Esc} \mid c_i^k) \bullet p(x_i \mid c_i^{k-1})$
      - Otherwise, record another escape event and repeat

PPM Classifier

- Code classifier, M_c
  - Trained from pre-tagged code corpus
- Data classifier, M_d
  - Trained from pre-tagged data corpus
- Estimate the probability of the sequence B:

$$p(B|M_\alpha) = -\log \prod_{i=1}^{|B|} p(b_i|b_{i-k}^{i-1}, M_\alpha)$$

- Class membership α of B is predicted by minimizing the cross entropy:

$$\alpha = \arg\min_{\alpha \in \{c,d\}} -\frac{1}{|B|} p(B|M_\alpha)$$

Classification Heuristics

- Word Data Tables

0x	20 A9 40 12 80 48 4E 12	30 CC 56 12 A0 F4 52 12
0x	A0 4B 47 12 A0 C0 44 12	F0 0F 44 12 F0 66 4B 12

- 16-bit addressing modes rare

```
push ax
mov [sp], 644Fh
and di, si
```

- Expect data after unconditional branches

```
DB (N bytes)
pop ecx
DB (N bytes)
```

```
pop eax
pop ecx
DB (N bytes)
```

Results

	IDA Pro 5.5			Ours		
File Name	FP	FN	Accuracy	FP	FN	Accuracy
7zFM.exe	0	1	99.999%	0	0	100%
notepad.exe	4	0	99.943%	0	0	100%
DosBox.exe	0	26	99.997%	0	0	100%
WinRAR.exe	0	23	99.989%	0	39	99.982%
Mulberry.exe	0	202	99.986%	0	0	100%
scummvm.exe	0	65	99.998%	0	0	100%
emule.exe	0	681	99.931%	0	117	99.988%
Mfc42.dll	0	1216	99.658%	0	47	99.987%
Mplayerc.exe	0	2065	99.751%	0	307	99.963%
RevelationClient.exe	0	1781	97.320%	0	71	99.893%
Vmware.exe	0	183	99.950%	0	45	99.988%

FP – False Positive (Data misclassified as code)

FN – False Negative (Code misclassified as data)

Results (eMule)

1	missed opcode prefixes	`push offset 701268h`
		db 64h
		`mov eax, large ds:0`
			`push offset 701268h`
			`mov eax, large fs:0`

2	code following unconditional branches	`jmp 526396h`
		db 8Bh
		`or eax, 9CAF08h`
			`jmp 526396h`
			`mov ecx, 9CAF08h`

3	code following returns	`retn`
		db C4h 83h
		`sub al, CDh`
		`push es`
			`retn`
			`add esp, 2Ch`
			`int 6`

4	code following conditional branches	`jz 52518Fh`
		db 8Bh
		`or eax, 9CAF04h`
			`jz 52518F`
			`mov ecx, 9CAF04h`

Conclusion and Future Work

- Improved accuracy over control-flow based approaches to disassembly
- Improvements still to be made
  - More sophisticated heuristics
  - Block entropy analysis
  - Improved corpus from perfectly labeled binaries
  - Applications in security
    - Source-free x86 SFI [Hamlen, Mohan, Wartell; UTDCS-10]

References

- Hamlen, K.W., Mohan, V., Wartell, R.: Reining in Windows API abuses with in-lined reference monitors. Tech. Rep. UTDCS-18-10, The University of Texas at Dallas, Richardson, Texas (June 2010)
- Cleary, J.G., Teahan, W.J.: Unbounded length contexts for PPM. The Computer Journal 40(2/3), 67-75 (1997)
- Cleary, J.G., Witten, I.H.: Data compression using adaptive coding and partial string matching. IEEE Transactions on Communications 32(4), 396-402 (1984)
- Cormack, G.V., Horspool, R.N.: Data compression using dynamic Markov modeling. The Computer Journal 30(6), 541-550 (1987)
- Hex-Rays: The IDA Pro disassembler and debugger. www.hex-rays.com/idapro (2011)

Instruction Aliasing