Breaking the x86 ISA

{ domas / @xoreaxeaxeax / DEF CON 2017

./bio

- Christopher Domas
  - Cyber Security Researcher @ Battelle Memorial Institute

The x86 ISA

- 8086: 1978
- A long, tortured history…

- Modes:
  - Real (Unreal)
  - Protected mode (Virtual 8086, SMM)
  - Long mode (Compatibility, PAE)

x86: evolution

- Instruction sets

x86: evolution

x86: evolution

- Modern x86 chips are a complex labyrinth of new and ancient technologies.
  - Things get lost…
- 8086: 29,000 transistors
- Pentium: 3,000,000 transistors
- Braodwell: 3,200,000,000 transistors

Trust.

- We don’t trust software.
  - We audit it
  - We reverse it
  - We break it
  - We sandbox it

- But the processor itself?
  - We blindly trust

Trust.

- *Why?*
- Hardware has
  all the same problems as software
- Secret functionality?
  - Appendix H.
- Bugs?
  - F00F, TSX, Hyperthreading.
- Vulnerabilities?
  - SYSRET, cache poisoning, sinkhole

Trust.

We should stop blindly trusting our hardware.

Trust.

- What do we need to worry about?

- Well known from software
  - Examples

Backdoors

Backdoors

- Hardware
  - FPGAs
  - Hypervisors
  - Microcode
  - Supply chain

Could a hidden instruction unlock your CPU?

Hidden instructions

Hidden instructions

- Historical examples
  - ICEBP
  - apicall

Table A-2. One-byte Opcode Map: (00H — F7H) *

	0	1	2	3	4	5	6	7
0	ADD Eb, Gb	Ev, Gv	Gb, Eb	Gv, Ev	AL, Ib	rAX, Iz	PUSH ES[i64]	POP ES[i64]
1	ADC Eb, Gb	Ev, Gv	Gb, Eb	Gv, Ev	AL, Ib	rAX, Iz	PUSH SS[i64]	POP SS[i64]
2	AND Eb, Gb	Ev, Gv	Gb, Eb	Gv, Ev	AL, Ib	rAX, Iz	SEG=ES (Prefix)	DAA[i64]
3	XOR Eb, Gb	Ev, Gv	Gb, Eb	Gv, Ev	AL, Ib	rAX, Iz	SEG=SS (Prefix)	AAA[i64]
4	INC[i64] general register / REX[o64] Prefixes eAX REX	eCX REX.B	eDX REX.X	eBX REX.XB	eSP REX.R	eBP REX.RB	eSI REX.RX	eDI REX.RXB
5	PUSH[d64] general register rAX/r8	rCX/r9	rDX/r10	rBX/r11	rSP/r12	rBP/r13	rSI/r14	rDI/r15
6	PUSHA[i64]/ PUSHAD[i64]	POPA[i64]/ POPAD[i64]	BOUND[i64] Gv, Ma	ARPL[i64] Ew, Gw MOVSXD[o64] Gv, Ev	SEG=FS (Prefix)	SEG=GS (Prefix)	Operand Size (Prefix)	Address Size (Prefix)
7	Jcc[f64], Jb - Short-displacement jump on condition O	NO	B/NAE/C	NB/AE/NC	Z/E	NZ/NE	BE/NA	NBE/A
8	Immediate Grp 1[1A] Eb, Ib	Ev, Iz	Eb, Ib[i64]	Ev, Ib	TEST Eb, Gb	Ev, Gv	XCHG Eb, Gb	Ev, Gv
9	NOP PAUSE(F3) XCHG r8, rAX	XCHG word, double-word or quad-word register with rAX rCX/r9	rDX/r10	rBX/r11	rSP/r12	rBP/r13	rSI/r14	rDI/r15
A	MOV AL, Ob	rAX, Ov	Ob, AL	Ov, rAX	MOVS/B Yb, Xb	MOVS/W/D/Q Yv, Xv	CMPS/B Xb, Yb	CMPS/W/D Xv, Yv
B	MOV immediate byte into byte register AL/R8L, Ib	CL/R9L, Ib	DL/R10L, Ib	BL/R11L, Ib	AH/R12L, Ib	CH/R13L, Ib	DH/R14L, Ib	BH/R15L, Ib
C	Shift Grp 2[1A] Eb, Ib	Ev, Ib	near RET[f64] Iw	near RET[f64]	LES[i64] Gz, Mp VEX+2byte	LDS[i64] Gz, Mp VEX+1byte	Grp 11[1A] - MOV Eb, Ib	Ev, Iz
D	Shift Grp 2[1A] Eb, 1	Ev, 1	Eb, CL	Ev, CL	AAM[i64] Ib	AAD[i64] Ib		XLAT/ XLATB
E	LOOPNE[f64]/ LOOPNZ[f64] Jb	LOOPE[f64]/ LOOPZ[f64] Jb	LOOP[f64] Jb	JrCXZ[f64]/ Jb	IN AL, Ib	eAX, Ib	OUT Ib, AL	Ib, eAX
F	LOCK (Prefix)		REPNE XACQUIRE (Prefix)	REP/REPE XRELEASE (Prefix)	HLT	CMC	Unary Grp 3[1A] Eb	Ev

Hidden instructions

Hidden instructions

- Traditional approaches:
  - Leaked documentation
  - Reverse engineering software
  - NDA
- But what if it's something stealthy?

Goal: Audit the Processor

- Find out what's really there

Approach

- How to find hidden instructions?

- Instructions can be one byte …
  - inc eax
  - 40
- … or 15 bytes …
  - lock add qword cs:[eax + 4 * eax + 07e06df23h], 0efcdab89h
  - 2e 67 f0 48 818480 23df067e 89abcdef
- Somewhere on the order of 1,329,227,995,784,915,872,903,807,060,280,344,576 possible instructions

Approach

https://code.google.com/archive/p/corkami/wikis/x86oddities.wiki

Approach

- The obvious approaches don't work:
  - Try them all?
    - Only works for RISC
  - Try random instructions?
    - Exceptionally poor coverage
  - Guided based on documentation?
    - Documentation can't be trusted (that's the point)
    - Poor coverage of gaps in the search space

Tunneling

- A depth-first-search algorithm
- (Overview)

Instruction lengths

- Catch: requires knowing the instruction length
- Simple approach: trap flag
  - Fails to resolve the length of faulting instructions
  - Necessary to search privileged instructions:
    - ring 0 only: mov cr0, eax
    - ring -1 only: vmenter
    - ring -2 only: rsm
  - It's hard to even auto-generate a successfully executing ring 3 instruction:
    - mov eax, [random_number]
- Solution: page fault analysis

Page Fault Analysis

- (Overview)

Cleanup

- Trap flag
  - Catch branching instructions
  - Differentiate between fault types

Tunneling

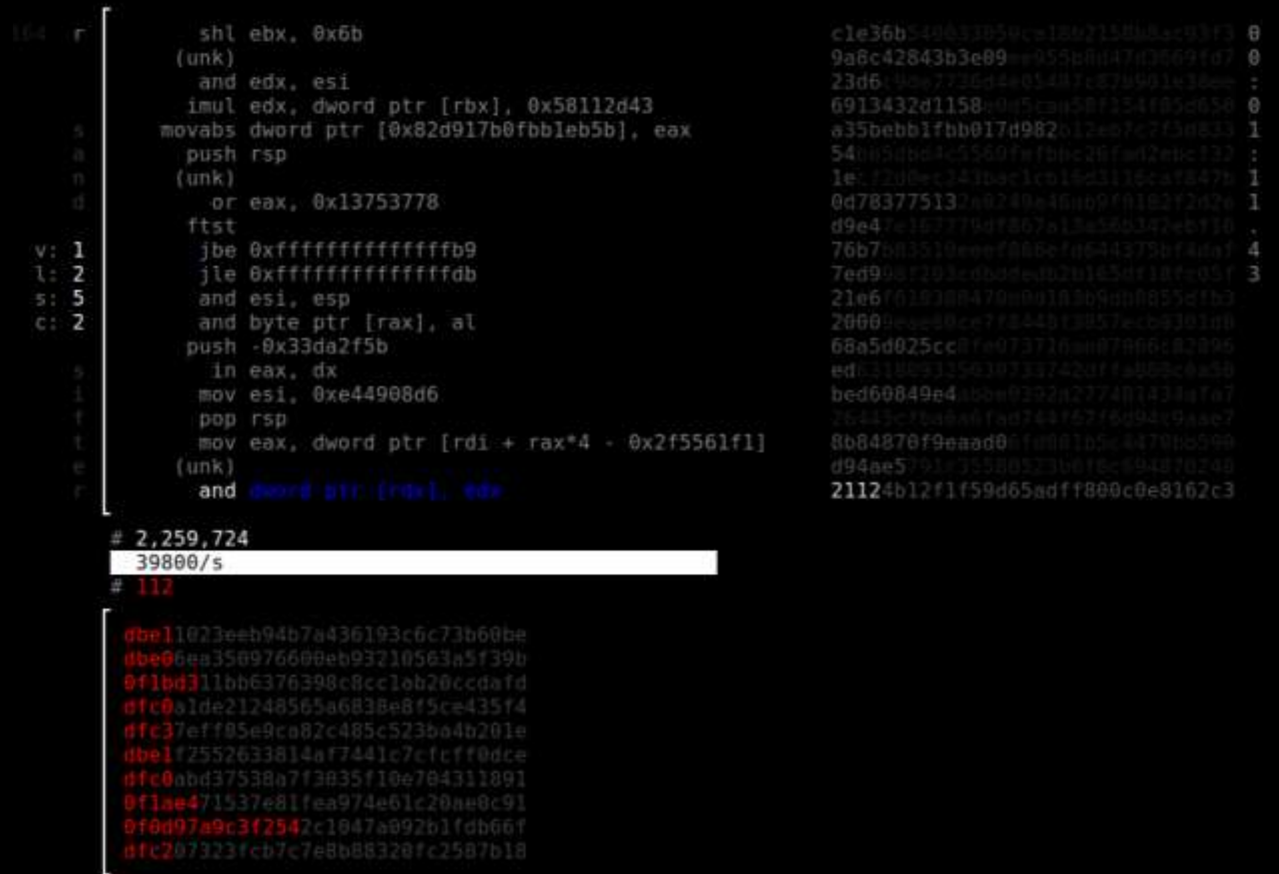
- Reduces search space from 1.3×10^{36} instructions to ~100,000,000 (one day of scanning)
- This gives us a way to *search* the instructions space.
  - How do we make sense of the instructions we execute?

Sifting

- We need a “ground truth”
- Use a disassembler
  - It was written based on the documentation
  - Capstone

Sifting

- Compare:
  - Observed length of instruction vs.
    disassembled length of instruction
  - Signal generated by instruction vs.
    expected signal


shl ebx, 0x6b
(unk)
and edx, esi
imul edx, dword ptr [rbx], 0x58112d43
movabs dword ptr [0x82d917b0fbb1eb5b], eax
push rsp
(unk)
or eax, 0x13753778
ftst
jbe 0xffffffffffffffb9
jle 0xffffffffffffffdb
and esi, esp
and byte ptr [rax], al
push -0x33da2f5b
in eax, dx
mov esi, 0xe44908d6
pop rsp
mov eax, dword ptr [rdi + rax*4 - 0x2f5561f1]
(unk)
and
v: 1
l: 2
s: 5
c: 2
2,259,724
39800/s
112
2112 4b12f1f59d65adff800c0e8162c3

sandsifter

(Demo)

sandsifter

- Hidden instructions
- Ubiquitous software bugs
- Hypervisor flaws
- Hardware bugs

Results

Hidden instructions

- 0f0dxx
  - Undocumented for non-/1 reg fields
- 0f18xx, 0f{1a-1f}xx
  - Undocumented until December 2016
- 0fae{e9-ef, f1-f7, f9-ff}
  - Undocumented for non-0 r/m fields until June 2014
- dbe0, dbe1
- df{c0-c7}
- f1
- {c0-c1}{30-37, 70-77, b0-b7, f0-f7}
- {d0-d1}{30-37, 70-77, b0-b7, f0-f7}
- {d2-d3}{30-37, 70-77, b0-b7, f0-f7}
- f6 /1, f7 /1

Hidden instructions

- Catch:
  - Undocumented instructions recognized by the disassembler are not found

Software bugs

- Issue:
  - Our “ground truth” (the disassembler) is also prone to errors

Software bugs

- Every disassembler we tried as the “ground truth” was littered with bugs.

- Most bugs only appear in a few tools, and are not especially interesting
- Some bugs appeared in *all* tools
  - These can be used to an attacker's advantage.

Software bugs

Software bugs

- 66e9xxxxxxxx (jmp)
- 66e8xxxxxxxx (call)

Software bugs

- 66 jmp
- Demo:
  - IDA
  - Visual Studio
  - objdump
  - QEMU

- 66 jmp
- Why does everyone get this wrong?
  - AMD designed the 64 bit architecture
  - Intel adopted… most of it.

Software bugs

- Issues when we can't agree on a standard
  - sysret bugs
- Either Intel or AMD is going to be vulnerable when there is a difference
- Complex architecture
  - Tools cannot parse a jump instruction

Software bugs

Hypervisor bugs

- Azure
  - CPUID / Trap flag bug

Hardware bugs

- Intel:
  - f00f bug on Pentium
- AMD:
  - Incorrect signals during decode
- Transmeta:
  - 0f{71,72,73}xxxx
  - Premature #GP0 signal during decode

Conclusions

- Our processors are not doing what we think they are
  - We need formal specifications
  - We need auditing tools
  - This is a start.

Conclusions

- Sandsifter lets us introspect what is otherwise a black box

Conclusions

- Open sourced:
  - The sandsifter scanning tool
  - github.com/xoreaxeaxeax/sandsifter

Conclusions

- Use sandsifter to audit your processor
- Reveal the instructions it really supports
- Search for hardware errata
- Break disassemblers, emulators, and hypervisors
- Send us the results

- github.com/xoreaxeaxeax
  - sandsifter
  - M/o/Vfuscator
  - REpsych
  - x86 0-day PoC
  - Etc.

- Feedback? Ideas?

- domas
  - @xoreaxeaxeax
  - xoreaxeaxeax@gmail.com