



# Breaking the x86 ISA


{ domas / @xoreaxeaxeax / DEF CON 2017



& Christopher Domas

✂ Cyber Security Researcher @  
Battelle Memorial Institute

./bio



& 8086: 1978

& A long, tortured history...

# The x86 ISA

A decorative background pattern of white lines and circles on a black background, resembling a circuit board or a network diagram. The lines are vertical and horizontal, with some diagonal connections. The circles are of varying sizes and are placed at various points along the lines.

& Modes:

- ⌘ Real (Unreal)

- ⌘ Protected mode (Virtual 8086, SMM)

- ⌘ Long mode (Compatibility, PAE)

# x86: evolution



& Instruction sets

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

x86: evolution

A decorative background pattern of a circuit board, consisting of thin white lines and small circles on a black background, primarily visible on the left side of the slide.

& We don't trust software.


⌘ We audit it

⌘ We reverse it

⌘ We break it

⌘ We sandbox it

Trust.

- 
- ⌘ But the processor itself?
  - ⌘ We blindly trust

Trust.



- & *Why?*
- & Hardware has  
all the same problems as software
- & Secret functionality?
  - ⌘ Appendix H.
- & Bugs?
  - ⌘ FOOF, 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

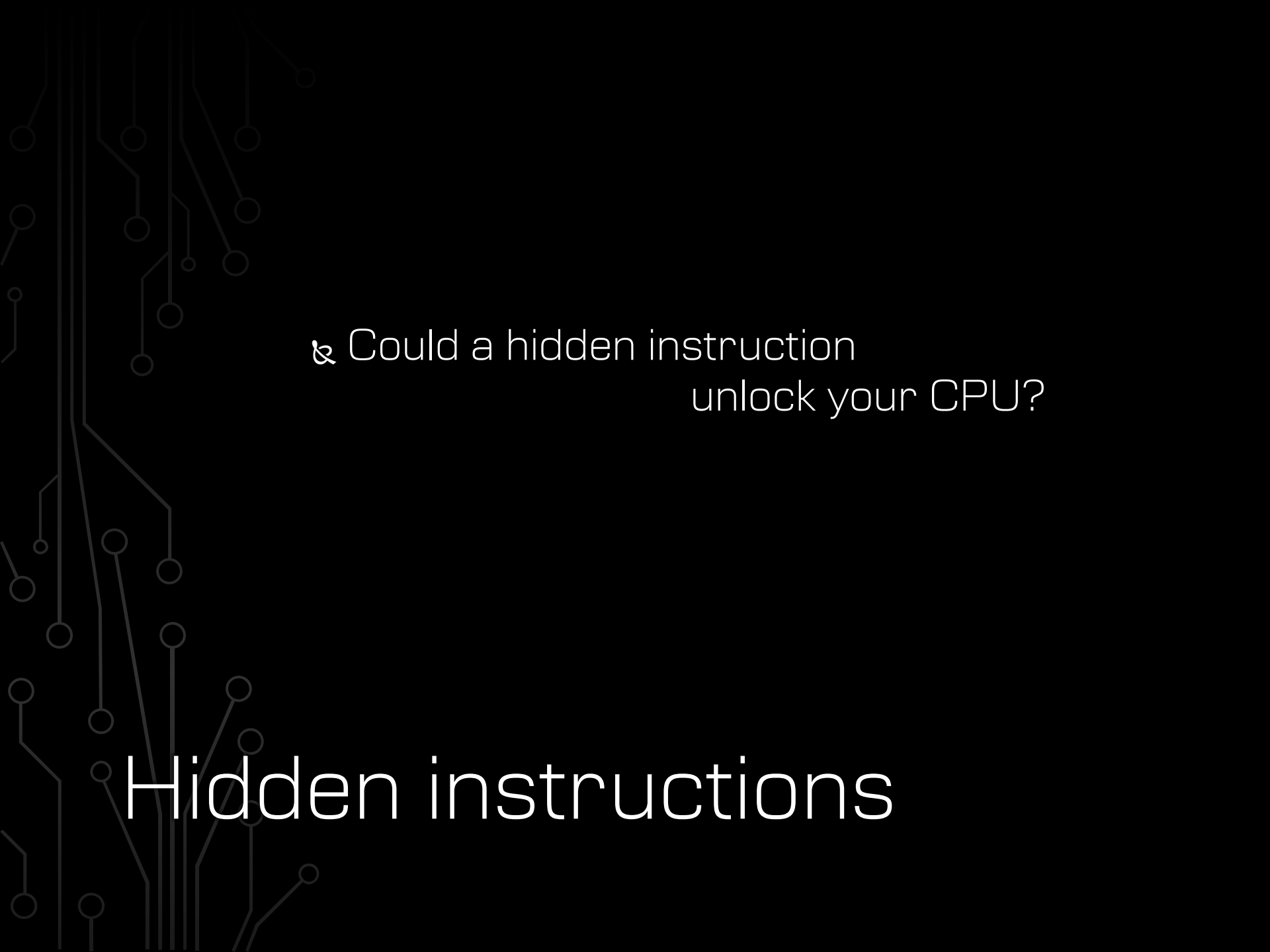
# Backdoors

A decorative background pattern of a circuit board with various lines and circular nodes, rendered in a light gray color against a black background.

## & Hardware

- ⌘ FPGAs
- ⌘ Hypervisors
- ⌘ Microcode
- ⌘ Supply chain

# Backdoors

A decorative background pattern of a circuit board, consisting of thin white lines and small circles, resembling a PCB layout, set against a black background.

& Could a hidden instruction  
unlock your CPU?

Hidden instructions



& Historical examples

⌘ ICEBP

⌘ apicall


# Hidden instructions

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

	0	1	2	3	4	5	6	7
0	Es, Gb	Ev, Gv	Gb, Eb	ADD Gx, Ev	AL, Ib	rAX, Iz	PUSH Eb <sup>34</sup>	POP EIP <sup>34</sup>
1	Es, Gb	Ev, Gv	Gb, Eb	ADC Gx, Ev	AL, Ib	rAX, Iz	PUSH SI <sup>34</sup>	POP SI <sup>34</sup>
2	Es, Gb	Ev, Gv	Gb, Eb	AND Gx, Ev	AL, Ib	rAX, Iz	SEG=ES (Prefix)	DAA <sup>34</sup>
3	Es, Gb	Ev, Gv	Gb, Eb	XOR Gx, Ev	AL, Ib	rAX, Iz	SEG=SS (Prefix)	AAA <sup>34</sup>
4	eAX REX	eCX REX.B	eDX REX.X	eBX REX.XB	eSP REX.R	eBP REX.RB	eSI REX.RX	eDI REX.RXB
5	RQZ <sup>34</sup> general register / REX <sup>34</sup> Prefixes							
6	rAX:r8	rCX:r8	rDX:r10	rBX:r11	rSP:r12	rBP:r13	rSI:r14	rDI:r15
7	PUSHA <sup>34</sup> PUSHAD <sup>34</sup>	POPAA <sup>34</sup> POPAD <sup>34</sup>	BOUND <sup>34</sup> Gv, Ma	ARPL <sup>34</sup> Ew, Gw MOVX <sup>34</sup> Gx, Ev	SEG=FS (Prefix)	SEG=GS (Prefix)	Operand Size (Prefix)	Address Size (Prefix)
8	Jcc <sup>34</sup> , Jb - Short-displacement jump on condition							
	O	NO	BN/AEC	NB/AENC	ZE	NZNE	BE/NA	NBE/A
9	Eb, Ib	Immediate Grp 1 <sup>34</sup> Ev, Iz		Eb, Eb <sup>34</sup>	Ev, Ib	TEST Eb, Gb	XCHG Eb, Gb	Ev, Gv
A	NDP PAUSE(F3) XCHG r8, rAX	rCX:r8	rDX:r10	rBX:r11	rSP:r12	rBP:r13	rSI:r14	rDI:r15
B	AL, Gb	rAX, Gv	Gb, AL	Gv, rAX	MOVSB Yb, Xb	MOVSW/DQ Yx, Xx	CMPSB Xb, Yb	CMPSW/DQ Xx, Yx
C	AL, RB, Ib	CL, RB, Ib	DL, R10L, Ib	BL, R11L, Ib	AH, R12L, Ib	CH, R13L, Ib	DH, R14L, Ib	BH, R15L, Ib
D	MOV immediate byte into byte register							
E	Shift Grp 2 <sup>34</sup> Eb, Ib		Ev, Ib	near RET <sup>34</sup> Iw	near RET <sup>34</sup> Iw	LES <sup>34</sup> Gz, Mz VEX+2byte	LDS <sup>34</sup> Gz, Mz VEX+1byte	Op 1 <sup>34</sup> - MOV Eb, Ib
F	Shift Grp 2 <sup>34</sup> Eb, 1		Ev, 1	Op, CL	Ev, CL	AAM <sup>34</sup> Ib	AAD <sup>34</sup> Ib	XLAT/ XLATB
	LOOPNE <sup>34</sup> / LOOPNC <sup>34</sup>	LOOPE <sup>34</sup> / LOOPZ <sup>34</sup>	LOOP <sup>34</sup> Jb	JRCXZ <sup>34</sup> / Jb	AL, Ib	IN rAX, Ib	OUT Ib, AL	Ib, eAX
	LOCK Prefix		REPNE XACQUIRE Prefix	REPE/REP XRELEASE Prefix	HLT	CMC	Unary Grp 3 <sup>34</sup> Eb, Ev	

# Hidden instructions



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

# Hidden instructions

A decorative background pattern of white lines and circles on a black background, resembling a circuit board or a network diagram. The pattern is most dense on the left side and fades towards the right.

& Find out what's really there

Goal: Audit the Processor

A decorative background pattern of a circuit board, consisting of thin white lines and small circles, resembling a PCB layout, set against a dark grey background.

& How to find hidden instructions?

Approach

↳ 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

- 
- ⌘ 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

# Approach

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

# Tunneling

- ⌘ 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

# Instruction lengths



& (Overview)

# Page Fault Analysis






## & Trap flag

- ⌘ Catch branching instructions
- ⌘ Differentiate between fault types

# Cleanup

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

# Tunneling

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

# Sifting

A decorative background pattern of a circuit board, consisting of thin white lines and small circles on a black background, primarily located on the left side of the slide.

& Compare:

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

# Sifting

```

104 r      shl ebx, 0x6b                c1e36b548633859ca1862158b8ac93f3 0
      (unk)                    9a6c42843b3e09ee955b8d47d3669fd7 0
      and edx, esi              23d649ae7736d4e05487c879901e38ee :
      imul edx, dword ptr [rbx], 0x58112d43 6913432d1158e0d5ca58f134f05d8658 0
s      movabs dword ptr [0x82d917b0fbb1eb5b], eax a35ebbb1fbb017d982a12ea7c7f3d833 1
a      push rsp                  54a958804c5560f7fb0c26fad2ebc132 :
n      (unk)                    1e1f288ec143bae1c016d31136ca7847b 1
d      or eax, 0x13753778       0d783775132a0249e46ab9f8182f2d2e 1
      ftst                      d9e47e167779df867a13a56b342eb130 .
v: 1      jbe 0xffffffffffffb9     76b7b83518eeef880ef8644375bf4daf 4
l: 2      jle 0xffffffffffffdb     7ed998f293c8b8de802b165df18fe05f 3
s: 5      and esi, esp              21e6f619388470a9a183b9ab885d1b3
c: 2      and byte ptr [rax], al    26009ee880ce7f844b13857ecb930100
      push -0x33da2f5b         68a5d025cc8fe973718aa07886c82896
s      in eax, dx                edc3188e335638733742df1a868c8a58
i      mov esi, 0xe44908d6     bed60849e4abb09392a277401434a7a7
t      pop rsp                  76443c78a8a6fad744f67f6d94c9aae7
t      mov eax, dword ptr [rdi + rax*4 - 0x2f5561f1] 8b84870f9eaad06fd001b5e4470bc599
e      (unk)                    d94ae5791e76588523b0f6c694870248
r      and dword ptr [rdi], eax  21124b12f1f59d65adff800c0e8162c3

```

```

# 2,259,724
39800/s
# 112

```

```

dbe11023eeb94b7a436193c6c73b60be
dbe06ea350976600eb93210563a5f39b
0f1bd311bb6376390c8cc1ab20ccdafd
dfc0a1de21248565a6838e8f5ce435f4
dfc37eff05e9ca82c485c523ba4b201e
dbe1f2552633814af7441c7cfcff0dce
#fc0abd37538a7f3835f10e704311891
0f1ae471537e81fea974e61c20ae0c91
0f0d97a9c3f2542c1047a092b1fdb66f
dfc207323fcb7c7e8b88320fc2507b18


```

# sandsifter



& (Demo)

sandsifter

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

# Results

- ⌘ Of0dxx
  - ⌘ Undocumented for non-/1 reg fields
- ⌘ Of18xx, Of{1a-1f}xx
  - ⌘ Undocumented until December 2016
- ⌘ Ofae{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



A decorative background pattern of a circuit board with various lines and nodes, rendered in a light gray color against a black background.

& Catch:

⌘ Undocumented instructions recognized by the disassembler are not found

# Hidden instructions



& 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.

# Software 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

A decorative background on the left side of the slide, consisting of a vertical column of thin, light gray lines that branch out horizontally and vertically, ending in small circles, resembling a circuit board or a tree structure.

& 66e9xxxxxxxx (jmp)  
& 66e8xxxxxxxx (call)

# Software bugs

A decorative background on the left side of the slide, consisting of a vertical column of lines and circles that branch out horizontally, resembling a circuit board or a tree structure.

& 66 jmp

& Demo:


⌘ IDA

⌘ Visual Studio

⌘ objdump

⌘ QEMU

# Software bugs

- 
- ⌘ 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





& Azure

⌘ CPUID / Trap flag bug

# Hypervisor bugs



& Intel:

⌘ f00f bug on Pentium

& AMD:

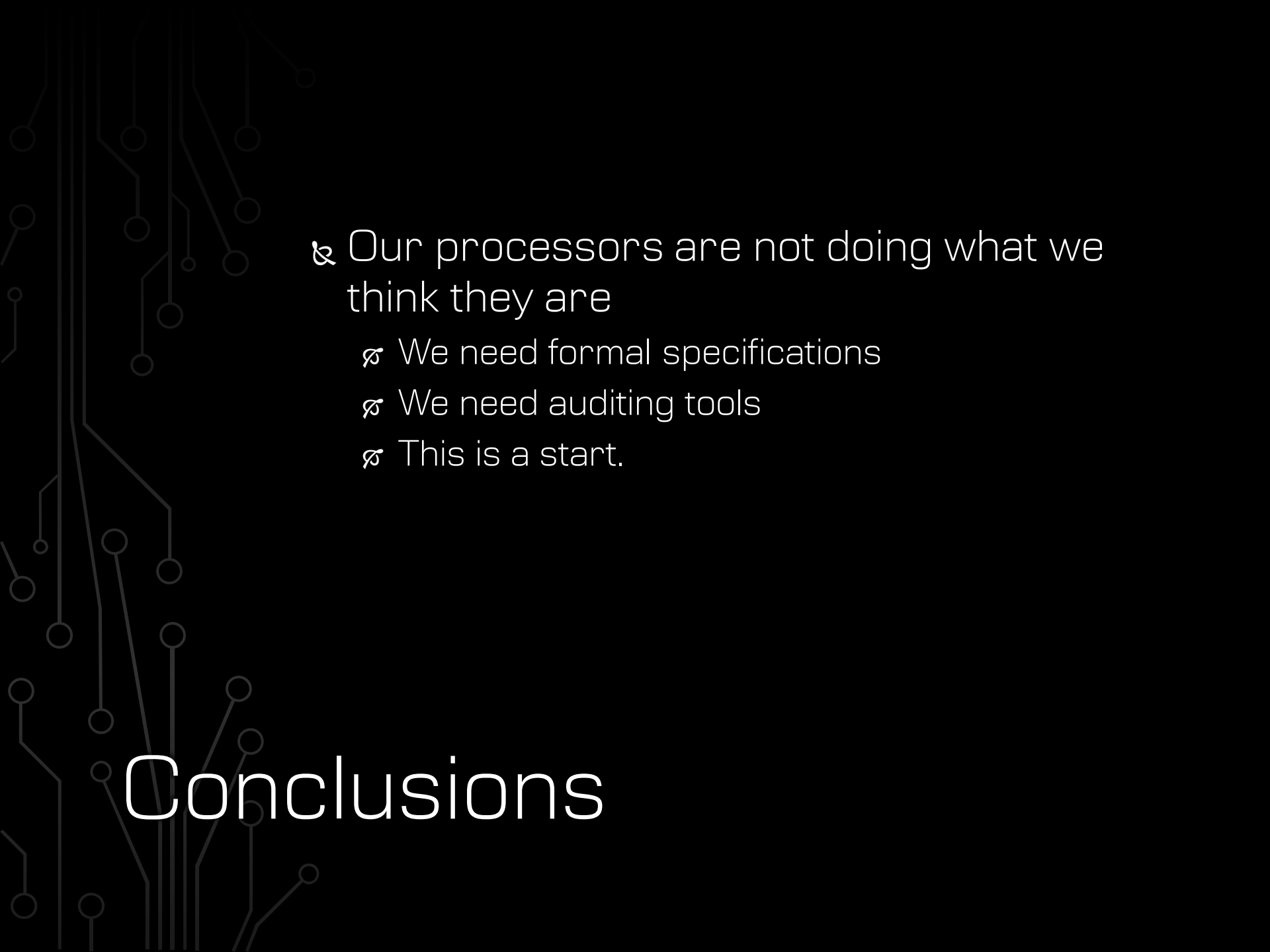
⌘ Incorrect signals during decode

& Transmeta:


⌘ Of{71,72,73}xxxx

⌘ Premature #GPO signal during decode

# Hardware bugs

- 
- A decorative background pattern of a circuit board, consisting of thin white lines and small circles on a black background, resembling a printed circuit board (PCB) layout.
- ⌘ 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](https://github.com/xoreaxeaxeax/sandsifter)

# Conclusions

- 
- A decorative background on the left side of the slide, consisting of a vertical column of thin, light-gray lines that branch out horizontally and vertically, ending in small circles, resembling a stylized circuit board or a tree structure.
- & 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

# Conclusions



& [github.com/xoreaxeaxeax](https://github.com/xoreaxeaxeax)

✂ sandsifter

✂ M/o/Vfuscator

✂ REpsych

✂ x86 0-day PoC

✂ Etc.

& Feedback? Ideas?

& domas

✂ @xoreaxeaxeax

✂ xoreaxeaxeax@gmail.com

