

Hardware Bugs and Fuzzing

Mengjia Yan

Spring 2024

sysret slides credit: Will Liu (MIT)



What is Errata?



8th and 9th Generation Intel® Core™ Processor Family

Specification Update

Supporting 8th Generation Intel® Core™ Processor Families for S/H/U Platforms, formerly known as Coffee Lake

Supporting 9th Generation Intel® Core™ Processor Families Processors for S/H Platforms, formerly known as Coffee Lake Refresh

November 2019

Revision 002

It is a compilation of device and document **errata** and **specification clarifications and changes**, which is intended for hardware system manufacturers and for software developers of applications, operating system, and tools.

Errata are **design defects or errors**. Errata may cause the processor's behavior to **deviate from published specifications**. Hardware and software designed to be used with any given stepping must assume that all errata documented for that stepping are present on all devices.

Errata Table Example

3.2 Errata Summary Information

Table 4-3. Errata Summary Table

ID	Processor Line / Stepping			Title
	S	H	U	
001				Reported Memory Type May Not Be Used to Access the VMCS and Referenced Data Structures
001	Problem	Bits 53:50 of the IA32_VMX_BASIC MSR report the memory type that the processor uses to access the VMCS and data structures referenced by pointers in the VMCS. Due to this erratum, a VMX access to the VMCS or referenced data structures will instead use the memory type that the memory-type range registers (MTRRs) specify for the physical address of the access.		
001	Implication	Bits 53:50 of the IA32_VMX_BASIC MSR report that the write-back (WB) memory type will be used, but the processor may use a different memory type.		
001	Workaround	Software should ensure that the VMCS and referenced data structures are located at physical addresses that are mapped to WB memory type by the MTRRs.		
	Status	For the steppings affected, refer the Summary Table of Changes.		

More Errata

Occasionally, AMD identifies product errata that cause the processor to deviate from published specifications. Descriptions of identified product errata are designed to assist system and software designers in using the processors described in this revision guide. This revision guide may be updated periodically.



Revision Guide for AMD Family 10h Processors

Publication # 41322 Revision: 3.92
Issue Date: March 2012

Advanced Micro Devices

298 L2 Eviction May Occur During Processor Operation To Set Accessed or Dirty Bit

Description

The processor operation to change the accessed or dirty bits of a page translation table entry in the L2 from 0b to 1b may not be atomic. A small window of time exists where other cached operations may cause the stale page translation table entry to be installed in the L3 before the modified copy is returned to the L2.

In addition, if a probe for this cache line occurs during this window of time, the processor may not set the accessed or dirty bit and may corrupt data for an unrelated cached operation.

Potential Effect on System

One or more of the following events may occur:

- Machine check for an L3 protocol error. The MC4 status register (MSR0000_0410) is B2000000_000B0C0Fh or BA000000_000B0C0Fh. The MC4 address register (MSR0000_0412) is 26h.
- Loss of coherency on a cache line containing a page translation table entry.
- Data corruption.

Suggested Workaround

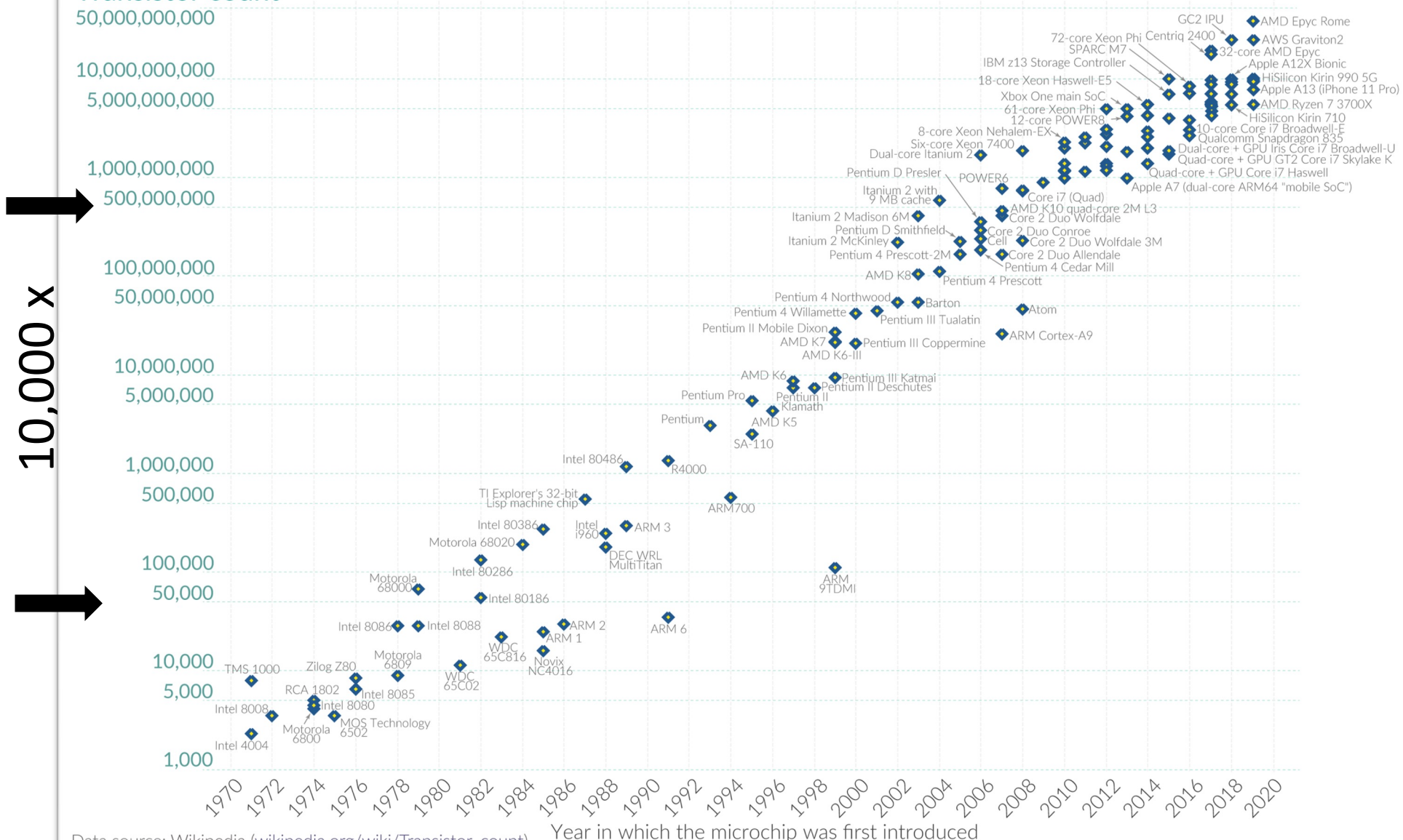
BIOS should set MSRC001_0015[3] (HWCR[TlbCacheDis]) to 1b and MSRC001_1023[1] to 1b. 4

In a multiprocessor platform, the workaround above should be applied to all processors regardless of

Moore's Law: The number of transistors on microchips doubles every two years

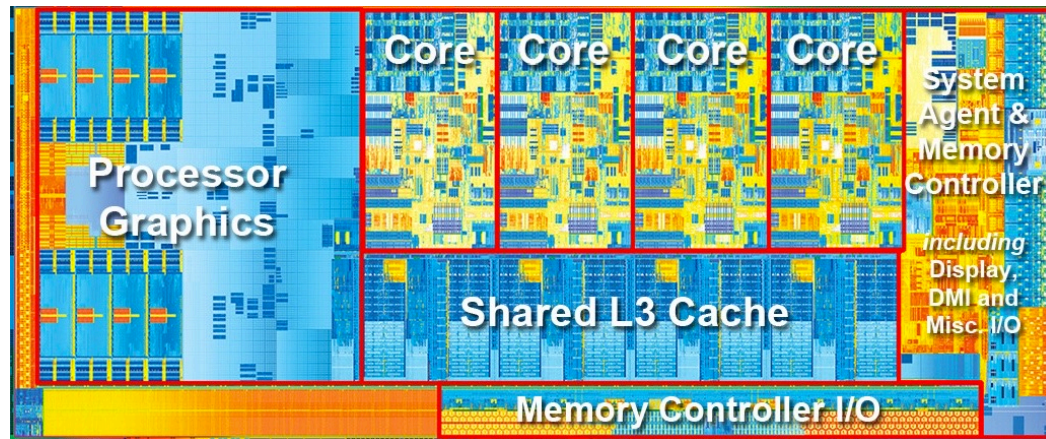
Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important for other aspects of technological progress in computing – such as processing speed or the price of computers.

Transistor count



Data source: Wikipedia (wikipedia.org/wiki/Transistor_count)

Errata Statistics



The Core-i7 processor with integrated graphics card
in 2012 with 1,400M Transistors

4 years; 136 errata; 3 bugs/month

*SPECS: A Lightweight Runtime Mechanism for Protecting Software
from Security-Critical Processor Bugs; Hicks et al; ASPLOS'15
<https://github.com/impedimentToProgress/specs>*

A **4-fold increase** in bugs in Intel processor designs
per generation. Approximately 8000 bugs
designed into the Pentium 4 ('Willamette')

from <https://www.cl.cam.ac.uk/~jrh13/slides/nijmegen-21jun02/slides.pdf>

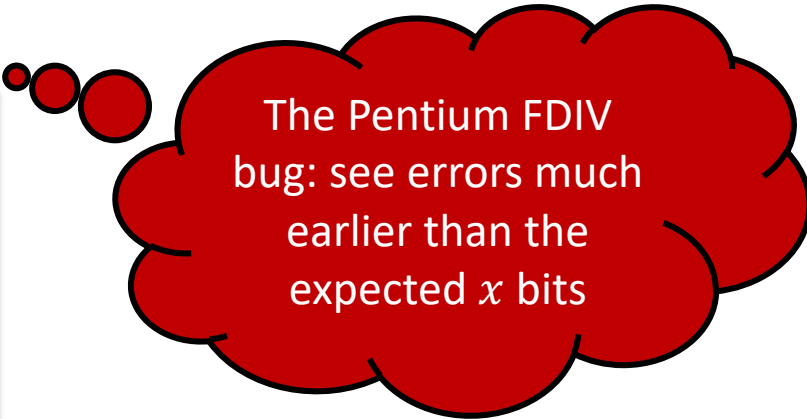
Outline

- Hardware Bug Examples
 - How do they look like? The discovery process? Impact?
 - #1: The famous Pentium FDIV bug
 - #2: SYSRET 64-bit OS privilege escalation vulnerability on Intel CPU
 - #3: Branch history injection attack
- How to discover hardware bugs?
 - Manual efforts, testing
 - Fuzzing
 - Formal verification (next lecture)

Bug #1: Pentium FDIV Bug

- What is the specification for floating-point computation?
 - Floating is encoded as $(1 + f) \times 2^e, 0 \leq f < 1, e \in \mathbb{Z}$
 - Example: $1/10 = 1.9999 \dots 9a \times 2^{-4}$ (in hexadecimal)
 - We always have errors when doing floating-point computation, because we have limited number of bits for each floating number
- The specification allows error to occur after bit x

	Single precision	Double precision	Extended precision
Word size in bits	32	64	80
Bits for f	23	52	63
Bits for e	8	11	15
Relative accuracy	$2^{-23} \approx 1.2 \cdot 10^{-7}$	$2^{-52} \approx 2.2 \cdot 10^{-16}$	$2^{-63} \approx 1.1 \cdot 10^{-19}$
Approximate range	$2^{\pm 127} \approx 10^{\pm 38}$	$2^{\pm 1023} \approx 10^{\pm 308}$	$2^{\pm 16383} \approx 10^{\pm 4964}$



The Pentium FDIV bug: see errors much earlier than the expected x bits

The Discovery Process #1: Nicely's Prime

- Thomas Nicely, a mathematics professor, tried to compute reciprocal of prime numbers: $p = 824,633,702,441$

- The correct result:

$$1/p = 1.212659629408667 \times 10^{-12}$$

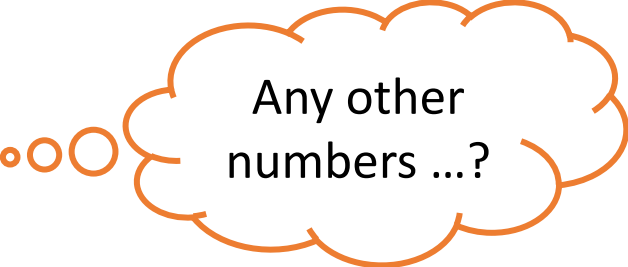

- But the new Pentium processor gives:

$$1/p = 1.21265962\mathbf{4891158} \times 10^{-12}$$

- Took him four months to confirm the problem was NOT in his program -> math libraries -> compilers -> operating system, but in the hardware



Differ after the 9th digit



Any other numbers ...?

The Discovery Process #2: Kaiser's List

- Andreas Kaiser, a computer consultant
 - Generate *25 billion* random integers and checked the accuracy of the computed reciprocals. *23* are incorrect.

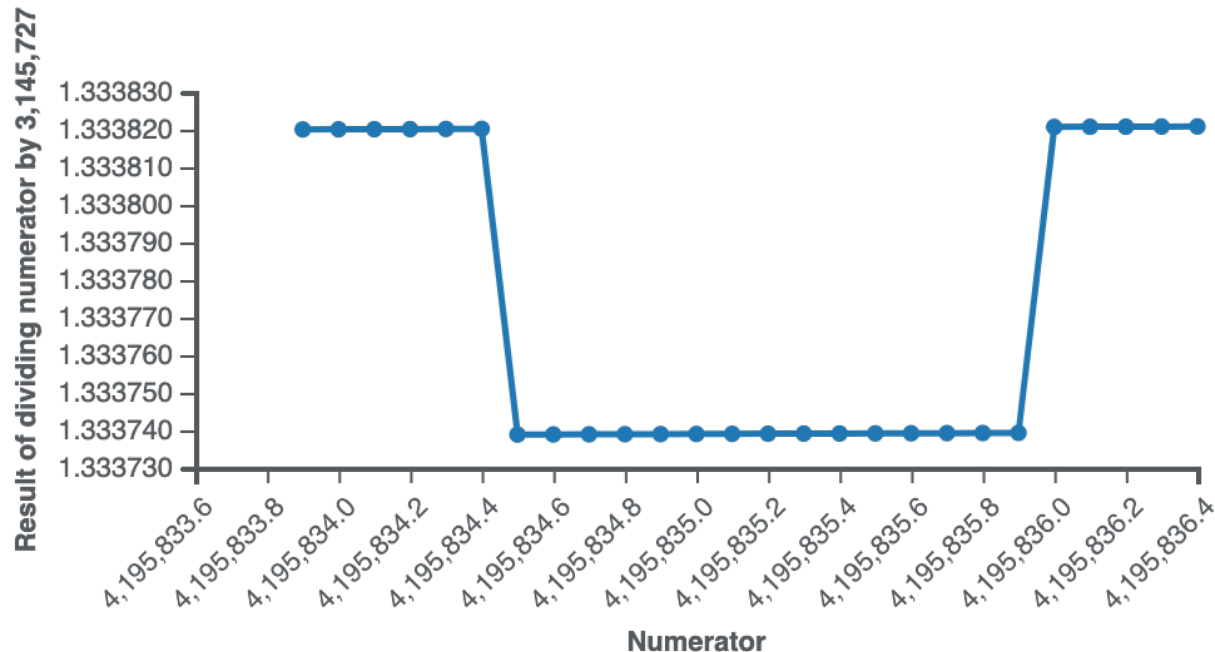
```
3221224323 = 1.7ffff70600000 . 231
12884897291 = 1.7ffff70580000 . 233
206158356633 = 1.7ffff704c8000 . 237
824633702441 = 1.7ffff7052000 . 239
1443107810341 = 1.4fffedac25000 . 240
6597069619549 = 1.7ffff7057400 . 242
9895574626641 = 1.1fffc6bc2a200 . 243
13194134824767 = 1.7ffff704e7e00 . 243
26388269649885 = 1.7ffff704fdd00 . 244
52776539295213 = 1.7ffff7046f680 . 245
```

Patterns?

- Many are started with *1.7ffff*
- In another word, the first 20 bits after the leading bit have to be a single zero, followed by at least 19 ones

The Discovery Process #3: Coe's Ratio

- Tim Coe, electrical engineer, has designed floating-point chips
 - $\frac{4,195,835}{3,145,727} = 1.33382044 \dots$ (correct) $1.33373906 \dots$ (Pentium)



The errors involve y/x where x and y 's **bit patterns conspire** to excite the bug at an early stage in the division.

Bug Explanation: FDIV

- Shift-and-subtract

```
          1.333?  
3145727 √ 4195835  
          3145727  
          -----  
          10501080  
           9437181  
           -----  
           10638990  
            9437181  
            -----  
            12018090  
             9437181  
             -----  
             25809090  
              ????????
```

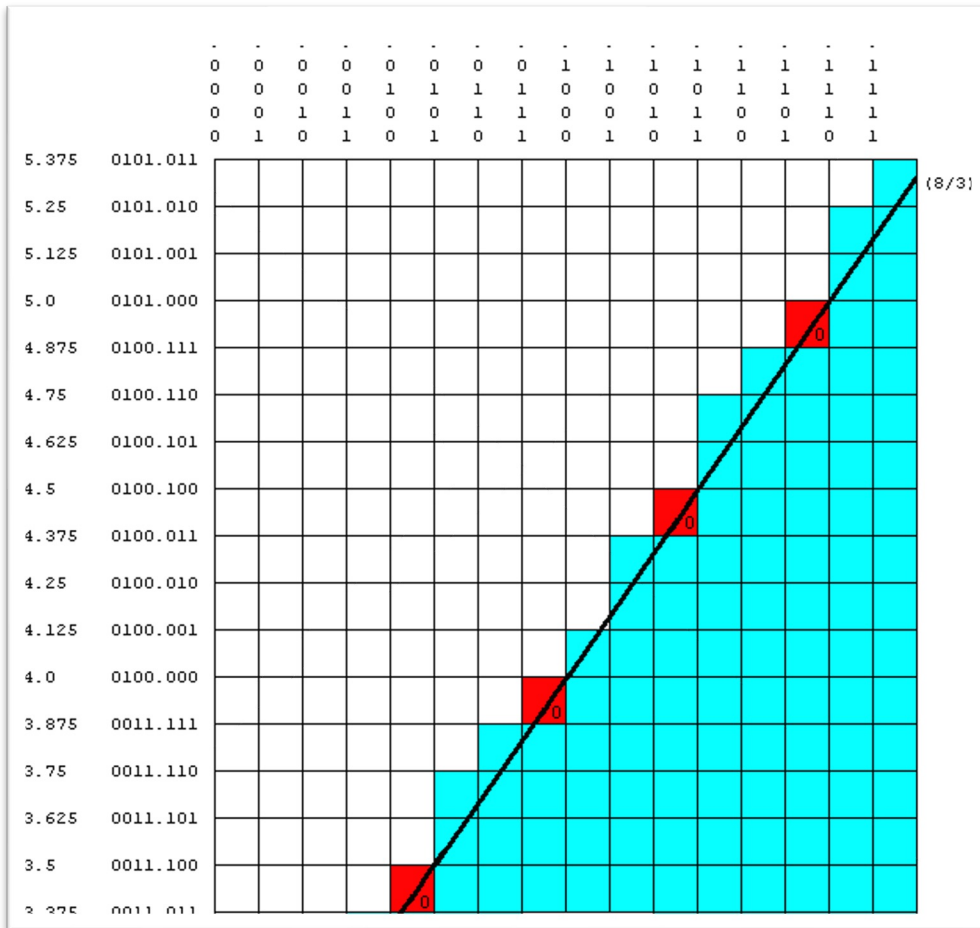
How to choose
quotient as a
human being?

- Old processors: choose quotient from 0, 1
- Faster [Sweeney, Robertson, and Tocher](#) (SRT) algorithm Radix-4:
 - Choose quotient from 0, +1, -1, +2, -2;
 - If the current quotient is incorrectly chosen, we can recover it from the next iteration
 - Guess the quotient based on the first few digits => use a 2D table to lookup

A combination of trial and error, experience, pattern matching and luck.

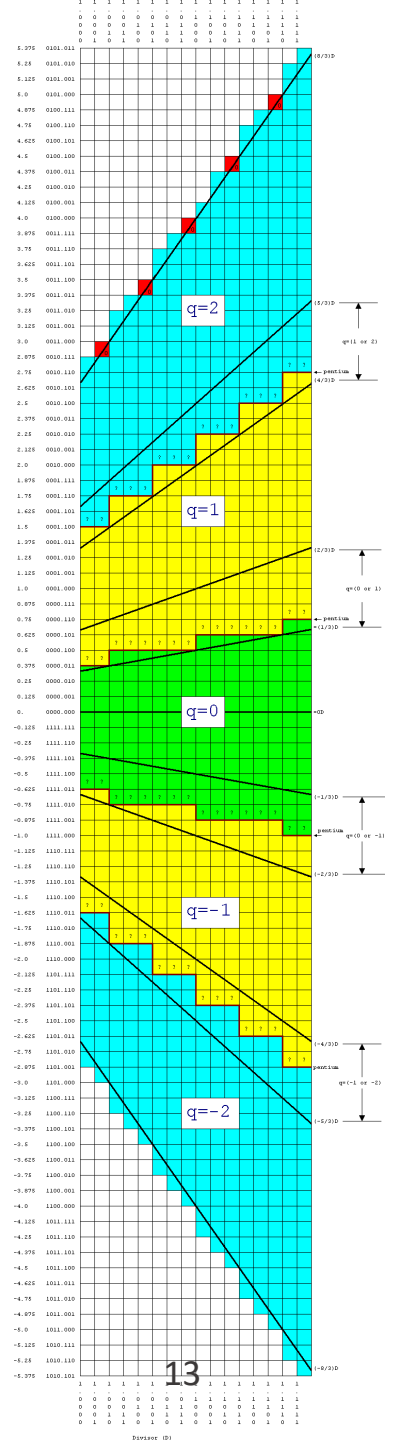
Bug Explanation: SRT Table

first 5 bits of the divisor



first 7 bits of the remainder

- 2048 cells in total
- 1066 cells in use
- **5 cells are not initialized**
- When the bug will be triggered?



How Frequently the bug can be triggered?

- Intel: an average spreadsheet user could encounter this flaw **once in every 27,000 years**, assuming 1,000 divisions per day.
- IBM: suspended sales of Pentium-based models and said it is as many as **20 mistakes per day**.
- Who actually got affected?
 - Normal users?
 - Wall street? Financial pre-diction programs? Did the Pentium bug flip a trading decision from buy to hold to sell?
 - Difficult to calibrate

Consequences/Impacts

- Intel's bad responses
 - Conditional replacement (customers need to claim they do get influenced by the bug) → disastrous press
 - No-questions-asked replacement → \$475M cost in 1994, 10% replacements
- Potential long-term impact:
 - Random test is not be a good idea. Exhaustive test has scalability problem.
 - A marked increase in the use of formal verification and number theory in hardware design

Some humor for you:

Q: How many Pentium designers does it take to screw in a light bulb?

A: 1.99904274017, but that's close enough for non-technical people.

Q: What do you get when you cross a Pentium PC with a research grant?

A: A mad scientist.

Do you think it bothers x86 users that the 486 is a functional upgrade to the Pentium?

In response to the Pentium bug, PowerMac officials have announced that they will be adding the control panel "Pentium Switcher" that allows users to decide whether the PowerMac should emulate pre-Pentium or post-Pentium FDIV behaviour.

TOP TEN NEW INTEL SLOGANS FOR THE PENTIUM

9.9999973251 **It's a FLAW, Dammit, not a Bug**

8.9999163362 **It's Close Enough, We Say So**

7.9999414610 **Nearly 300 Correct Opcodes**

6.9999831538 **You Don't Need to Know What's Inside**

5.9999835137 **Redefining the PC--and Mathematics As Well**

4.9999999021 **We Fixed It, Really**

3.9998245917 **Division Considered Harmful**

2.9991523619 **Why Do You Think They Call It *Floating* Point?**

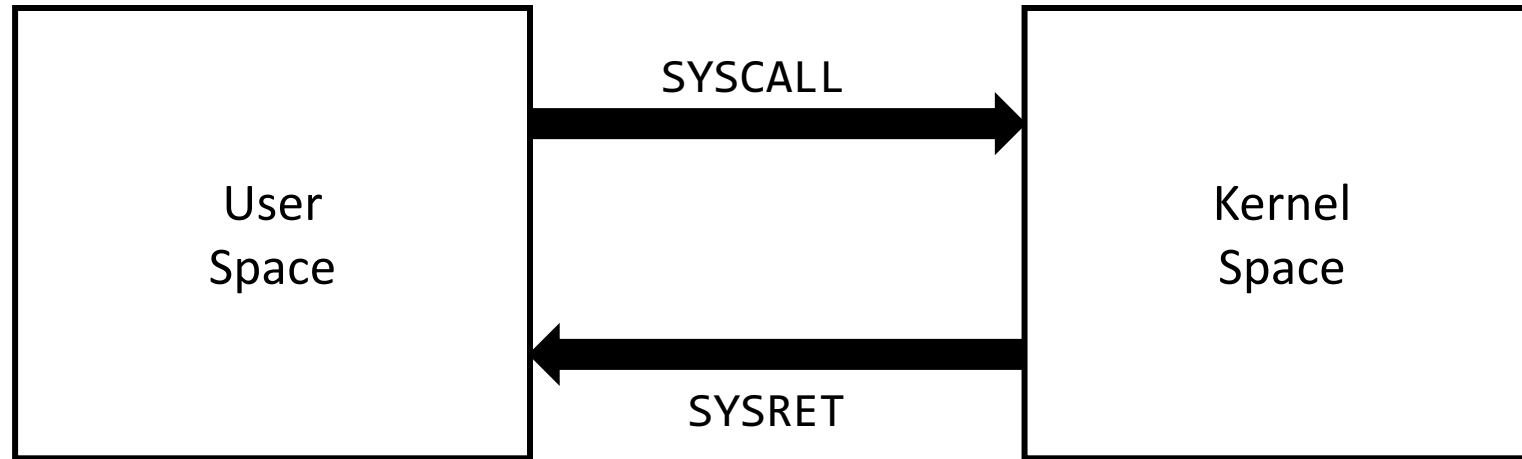
1.9999103517 **We're Looking for a Few Good Flaws**

0.9999999998 **The Errata Inside**

<http://davefaq.com/Opinions/Stupid/Pentium.html#glitch>

Bug #2: A SYSRET Bug

64-bit x86 instruction set: AMD64, Intel 64



SYSCALL

- HW transits from user mode to kernel mode
- Save the userspace next-PC to the RCX register
- Jump to a kernel syscall entry point

SYSRET

- HW transits from kernel mode to user mode
- Restore the userspace next-PC from the RCX register

Two Different Specifications for SYSRET

AMD
SYSRET

HW transits from kernel mode
to user mode

Restore the userspace next-PC
from the RCX register

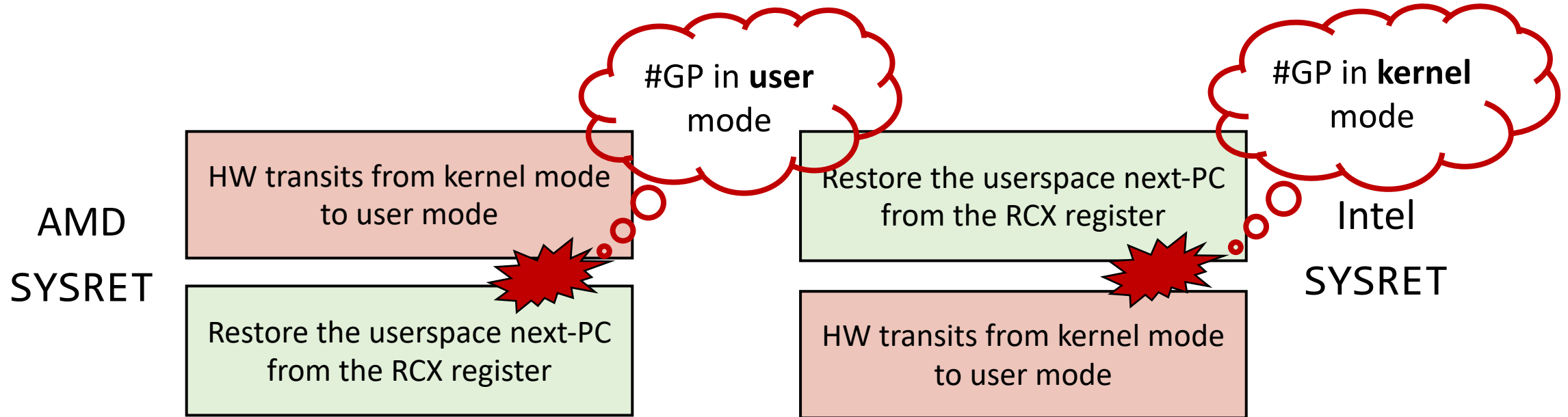
Restore the userspace next-PC
from the RCX register

HW transits from kernel mode
to user mode

Order is
flipped

Intel
SYSRET

SYSRET Vulnerability



If RCX holds a non-canonical address, the SYSRET will generate a #GP (general protection fault). Canonical means that given 48-bit virtual address space, the high 16 bits (bits 63-48) of a virtual address have the same value as bit 47.

How SYSRET is used in kernel code?

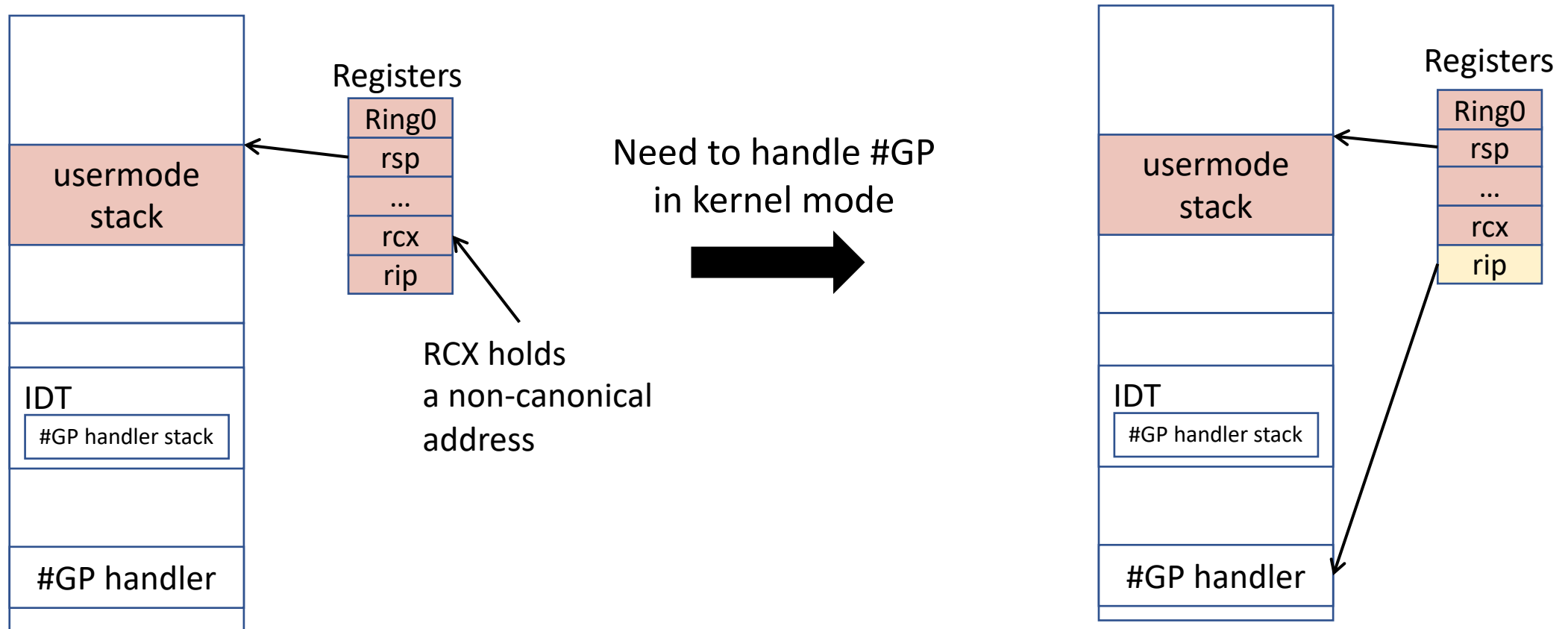
- What do we do before we transition from kernelspace to userspace?

```
movq RCX(%rsp), %rcx  
movq RIP(%rsp), %r11  
cmpq %rcx, %r11 /* SYSRET requires RCX == RIP */  
jne swaps_restore_regs_and_return_to_usermode  
  
...  
/* populate all the registers using data from userstack */  
sysret
```

At this point,
all the registers are
user-controlled
(attacker-controlled)



SYSRET Attack on Intel Processors



Before executing SYSRET, all registers have been restored using usermode context

Assume rip points to kernel stack and start using it --> can **overwrite kernel data**

Longtime Intel x86 OS Bug

2012-07-05

CVE-2012-0217: Intel's sysret Kernel Privilege Escalation (on FreeBSD)

By iZsh

Filed under [vulnerability](#) [exploit](#) [FreeBSD](#)

[PATCH] x86_64: Wh
Intel EM64T CPUs han
from AMD CPUs.

The exception is rep
This leads to the ke
with the wrong GS be
on this instruction.

This version of the p
version fixed.

This is [CVE-2006-0744](#)

Thanks to Ernie Petrides and Asit B. Mallick for analysis and initial patches.

CVE-2012-0217 Detail

MODIFIED

This vulnerability has been modified since it was last analyzed by the NVD. It is awaiting real changes to the information provided.

Description

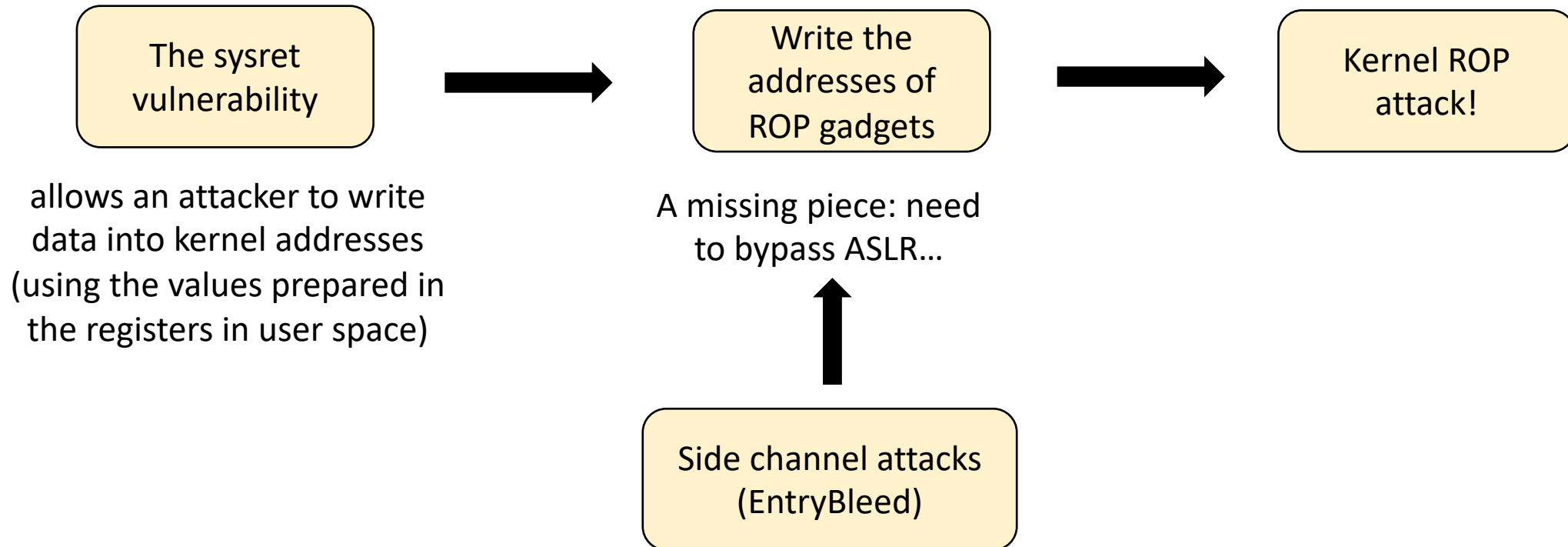
The x86-64 kernel system-call functionality in [Xen 4.1.2 and earlier](#), as used in Citrix XenServe, [Oracle Solaris 11 and earlier](#); [illumos before r13724](#); [Joyent SmartOS before 20120614T184600Z](#); [FreeBSD before 9.0-RELEASE-p3](#); [NetBSD 6.0 Beta and earlier](#); [Microsoft Windows Server 2008 R2 and R2 SP1 and Windows 7 Gold and SP1](#); and possibly other operating systems, when running on an Intel processor, incorrectly uses the sysret path in cases where a certain address is not a

CVE-2014-4699: Linux Kernel ptrace/sysret vulnerability analysis

by Vitaly Nikolenko

🕒 Posted on July 21, 2014 at 6:52PM

Exploiting Sysret on Linux in 2023



It's not a bug, it's a feature

Description

SYSRET is a companion instruction to SYSCALL. In 64-bit mode, SYSRET remains at privilege level 0, and the registers are loaded.

Operation

```
IF (CS.L ≠ 1) or (IA32_EFER.LMA ≠ 1) or (IA32_EFER.SCE ≠ 1)
(* Not in 64-Bit Mode or SYSCALL/SYSRET not enabled in IA32_EFER *)
  THEN #UD; FI;
IF (CPL ≠ 0) THEN #GP(0); FI;

IF (operand size is 64-bit)
  THEN (* Return to 64-Bit Mode *)
    IF (RCX is not canonical) THEN #GP(0);
    RIP := RCX;
  ELSE (* Return to Compatibility Mode *)
    RIP := ECX;

FI;
RFLAGS := (R11 & 3C7FD7H) | 2;          (* Clear RF, VM, reserved bits; set bit 1 *)

IF (operand size is 64-bit)
  THEN CS.Selector := IA32_STAR[63:48]+16;
  ELSE CS.Selector := IA32_STAR[63:48];
FI;
CS.Selector := CS.Selector OR 3;        (* RPL forced to 3 *)
(* Set rest of CS to a fixed value *)
CS.Base := 0;                          (* Flat segment *)
CS.Limit := FFFFFFFH;                   (* With 4-KByte granularity, implies a 4-GByte limit *)
CS.Type := 11;                          (* Execute/read code, accessed *)
CS.S := 1;
CS.DPL := 3;
CS.P := 1;
IF (operand size is 64-bit)
  THEN (* Return to 64-Bit Mode *)
```

OS system-call handler to user mode from R11.¹ With a 64-bit operand only the low 32 bits of the register are used.

It's not a bug, it's a feature

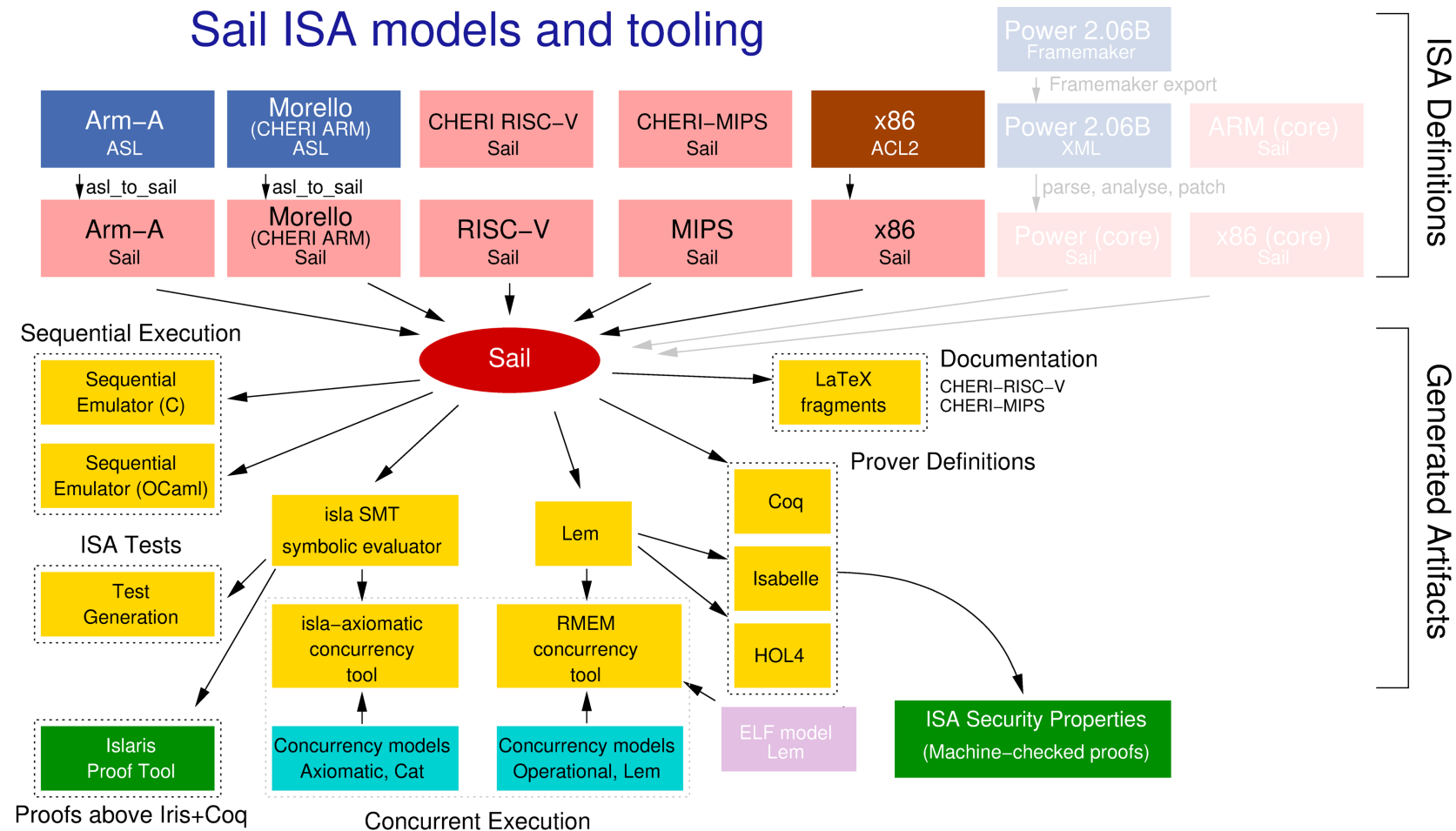
Intel claims that this vulnerability is a software implementation issue, as their processors are functioning as per their documented specifications. However, software that fails to take the Intel-specific SYSRET behavior into account may be vulnerable.

<https://www.kb.cert.org/vuls/id/649219>

Who to blame?

- Intel claims it is not an errata
 - *Errata are design defects or errors that may cause ... behavior to deviate from published specifications.*
 - This behavior is consistent with Intel's specification
 - **So the problem is the specification is incorrect**
- Intel SDM (software development manual) 3400 pages. We cannot assume the specification is always correct.
- Research question: how can we know the ISA specification is correct?
 - Some research efforts to verify ISA specification

The Sail ISA specification language



Bug #3: eIBRS Vulnerability

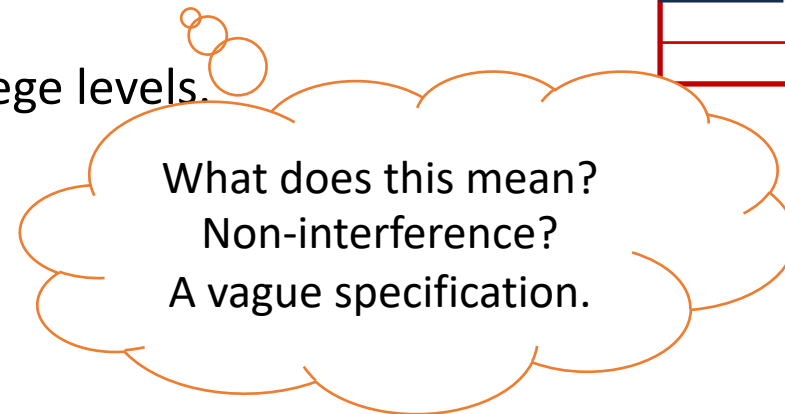
- Recap Spectre v2
- eIBRS: Enhanced Indirect Branch Restricted Speculation. Advertised as a mitigation against Spectre v2.

Specification:

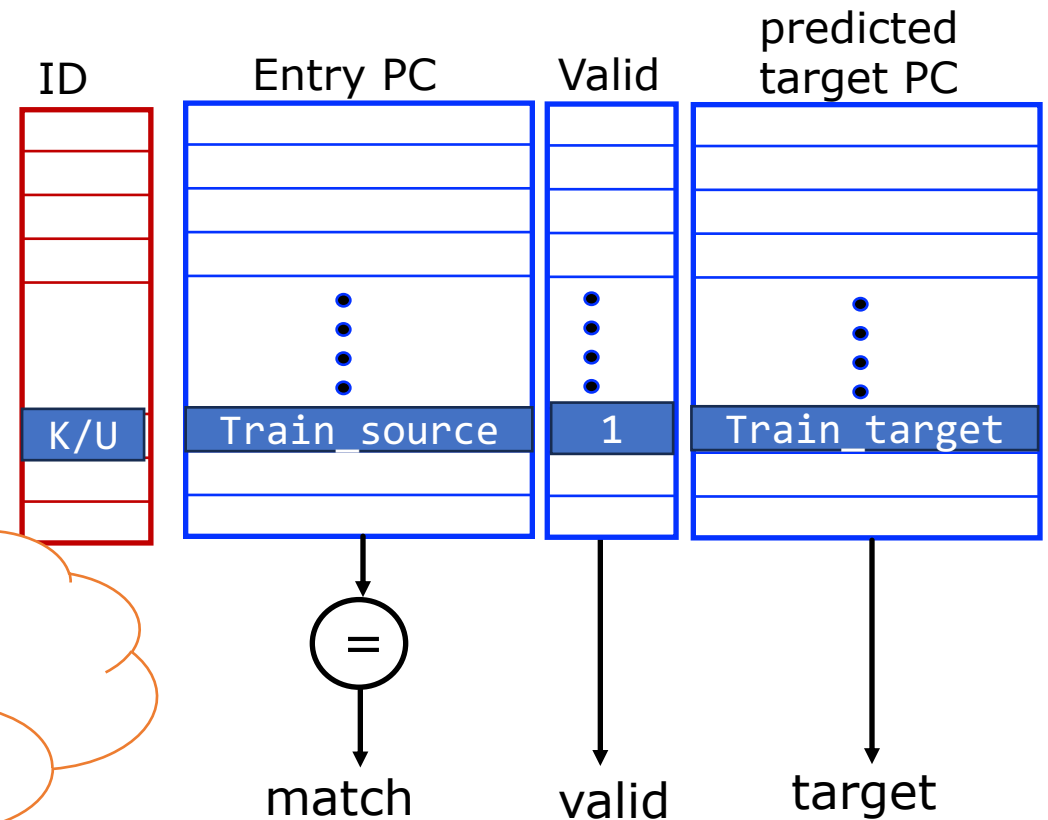
Do not let lower-privileged code to interfere the branch prediction target of the high-privilege code.

OR

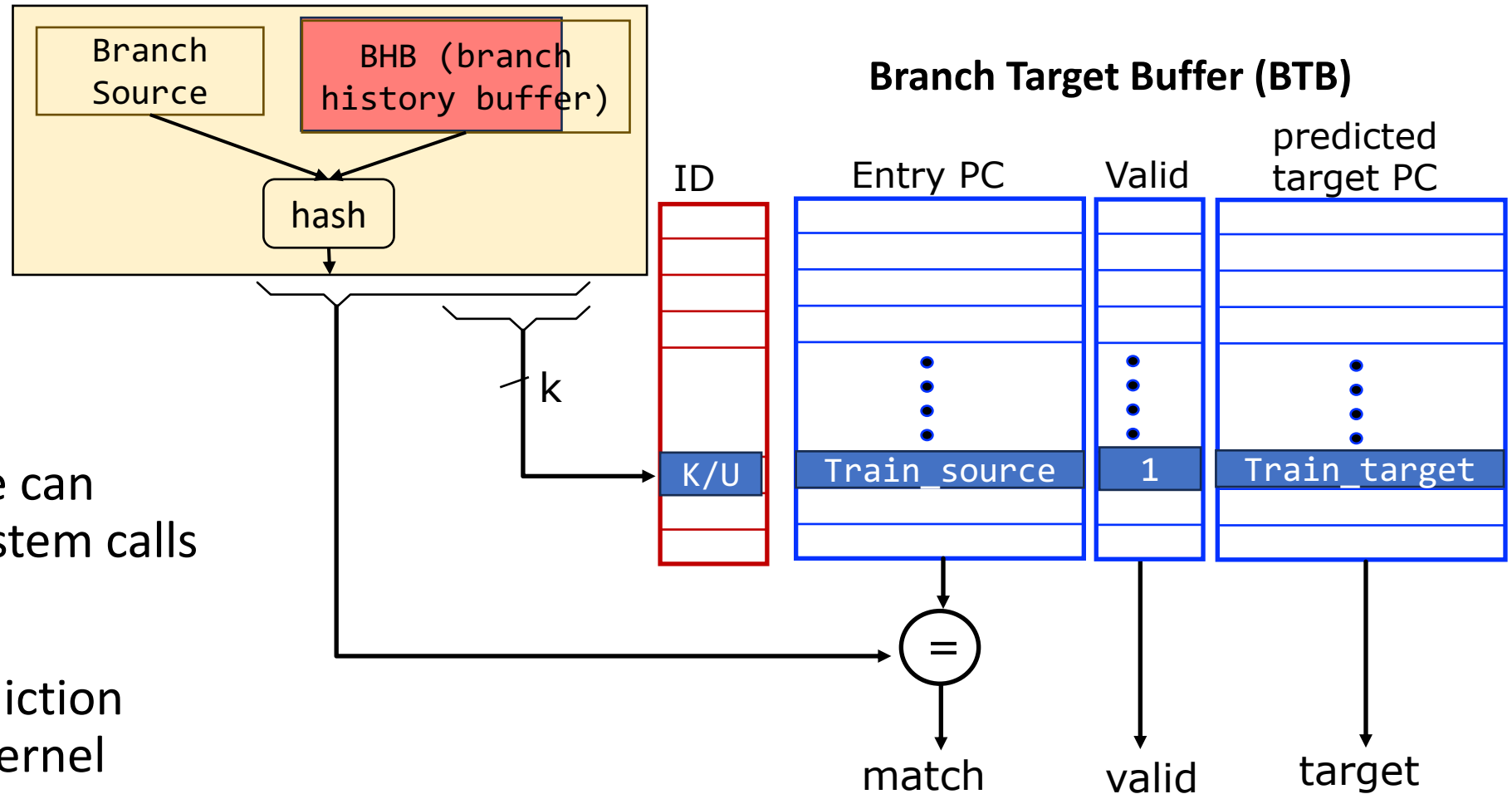
Isolate BTB entries across privilege levels.



Branch Target Buffer (BTB)

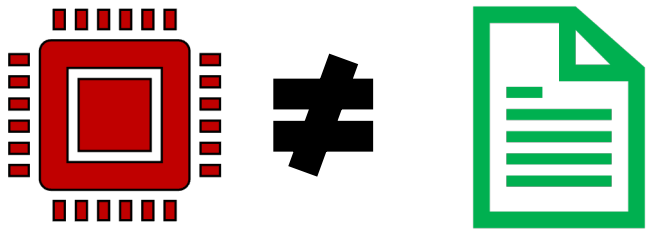


Recap the Problem: Branch History Injection



- #1: Userspace code can trigger different system calls
- #2: Userspace prediction history can affect kernel space BTB prediction

Summary



Implementation does not
match specification
(Errata)



Bugs in the specification



Vague specification

- Next: How to find hardware bugs?
 - Get ideas from the software

Software Bugs Hunting/Fixing

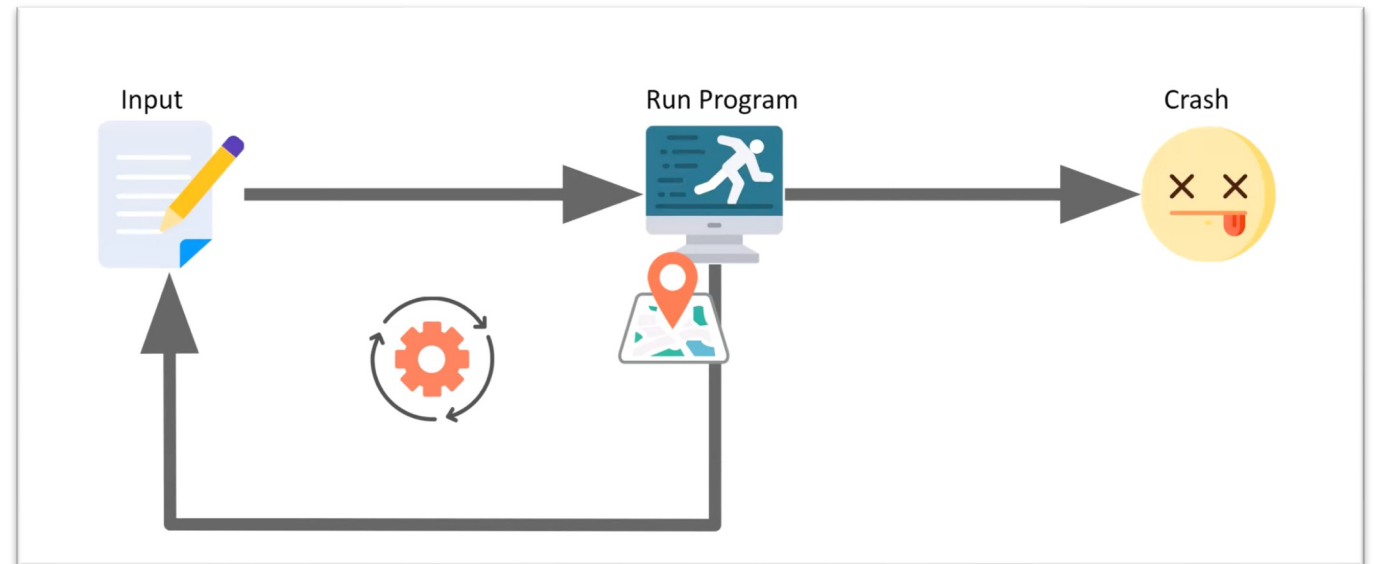
- Approach 1: Manual effort
 - Hire a lot of experts and stare at the code
 - Regression test → but need to be updated
- Approach 2: How about randomly generating test cases?
 - Fuzzing
- Approach 3: Formal verification



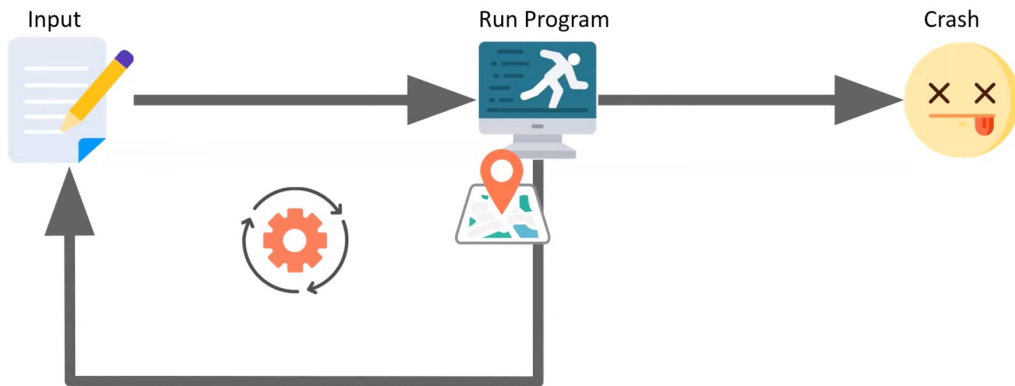
Fuzzing

Fuzzing In A Nutshell

- Automatic generate test examples
- 1999, Alan Cox at University of Wales discovered a vulnerability in Linux kernel by simply running a program generating random input and feed into the kernel
- Crash is generated by assertions/specifications
- Simple yet effective
- Industry standard



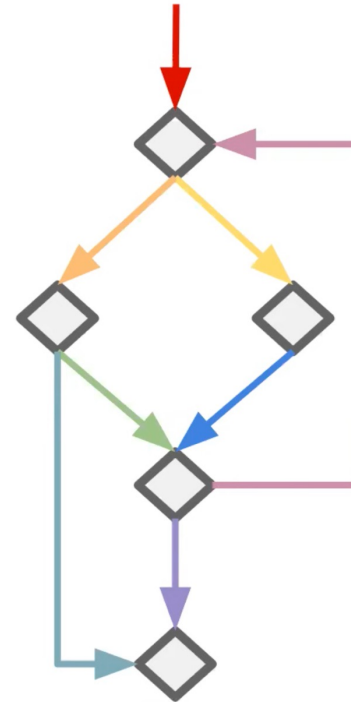
Fuzzing Components



- Random seeds
 - Sometimes need formatted inputs, e.g., PDF reader
- A criteria to check whether the outcome is as expected or not.
 - Specification
 - Security invariant (paper discussion SPECS)
 - Assertions (address sanitizer)
- Heuristics for generating new tests => feedback loop for better efficiency

Types of Fuzzing

- Blackbox
- Greybox
- Whitebox



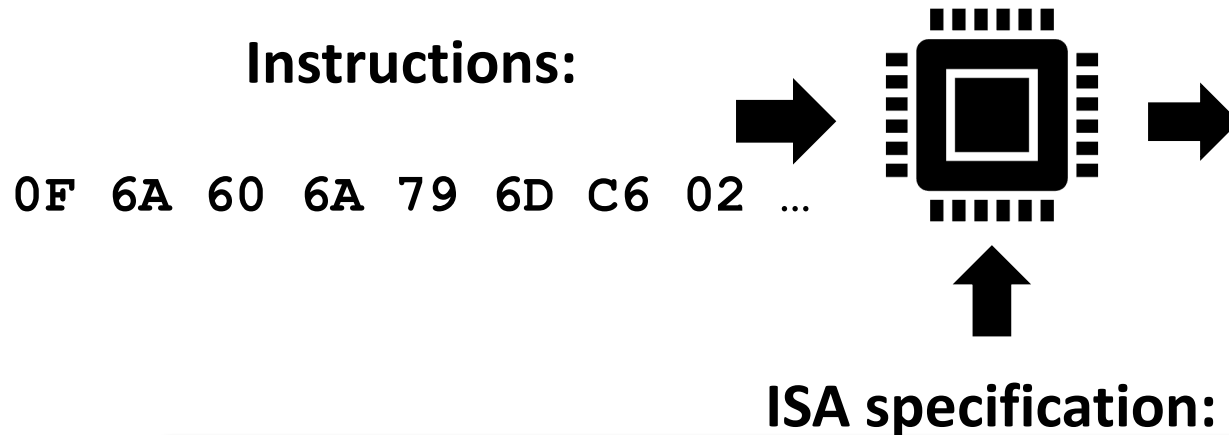
Collected coverage:



Example: Hidden Instructions

- Hidden instructions: secret instructions that give backdoor or powerful access to processor internals
- Secret processor functionality: Appendix H
- An example:
 - Pentium F00F bug, an invalid instruction freezes the cpu, discovered in 1997
 - A Ring 3 process can DOS (denial of service) a process
 - The invalid instruction encoding is: **F0 0F C7 [C8-CF]**

Search for Hidden Instructions



Valid instructions (in spec)

Invalid instructions
(#UD exception, invalid opcode)

Hidden instructions (not in spec,
but can execute, no #UD exception)

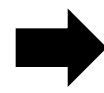
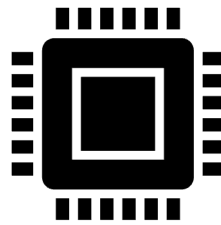
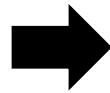
Table A-2. One-byte Opcode Map: (00H – F7H) *

	0	1	2	3	4	5	6	7
0	Eb, Gb	Ev, Gv	Gb, Eb	Gv, Ev	AL, Ib	rAX, Iz	PUSH ES ⁱ⁶⁴	POP ES ⁱ⁶⁴
1	Eb, Gb	Ev, Gv	Gb, Eb	Gv, Ev	AL, Ib	rAX, Iz	PUSH SS ⁱ⁶⁴	POP SS ⁱ⁶⁴
2	Eb, Gb	Ev, Gv	Gb, Eb	Gv, Ev	AL, Ib	rAX, Iz	SEG=ES (Prefix)	DAA ⁱ⁶⁴
3	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

Challenges #1: Detect Hidden Instruction

Instructions:

0F 6A 60 6A 79 6D C6 02 ...



Valid instructions (in spec)

Invalid instructions
(#UD exception, invalid opcode)

Hidden instructions (not in spec,
but can execute, no #UD exception)

How to capture
this case?

Engineer the trap/exception handler
(Lab 6.A)

Challenges #2: Large Space

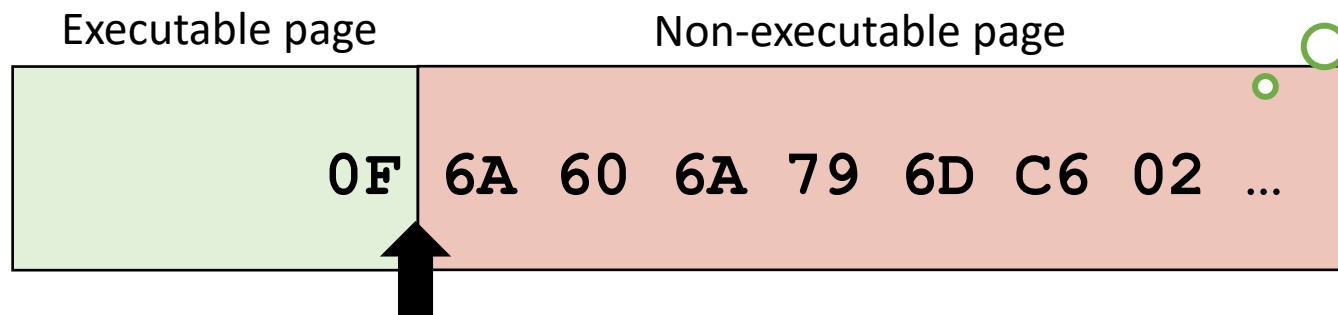
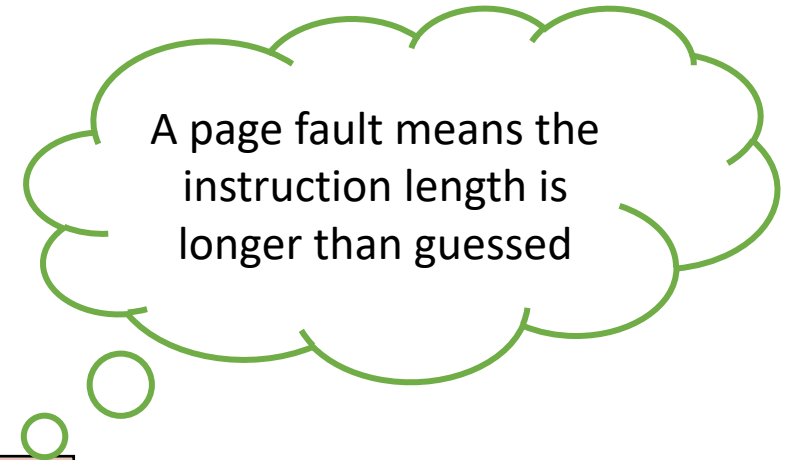
- CISC: Variable length instructions
 - One-byte instruction: `0x40 -> inc eax`
 - 15-byte instruction: `2e67f048 818480 23df067e 89abcdef -> lock add qword cs:[eax + 4 * eax + 07e06df23h], 0efcdab89h`
 - Worst-case exhaustive search: 256^{15}
- Observation: the **meaningful** bytes of an x86 instruction impact either its length or its exception behavior
- A potential solution: depth-first search

0F 6A 60 6A 79 6D C6 02 ...



Challenges #3: Measure Instruction Length

- Trap flag
 - Execute an instruction, set PC to the next instruction, and go to trap handler
 - Inside the trap handler, observe instruction length
- How to deal with privilege instructions?
 - Trap in user space. Will not advance the PC
- A potential solution: page fault analysis



Engineering Efforts to Survive

- Hack the kernel to hook page fault handler to catch the instruction
- Hack various fault handler inside the kernel in case the the hidden instruction traps
- A lot more...
 - watch the talk, learn in the system programming recitation and the fuzzing lab

<https://www.youtube.com/watch?v=KrksBdWcZgQ>

SandSifter and Findings

```
r      retf          cb0c31130032c2c039e4f4bc23c0a03 0
      sbb byte ptr [edi], ah      1827843c0e4a7b311aced8e01732d8e0 0
      in al, dx                    ec3e003478c7cc0c0c07e1e00243e423 :
      les ecx, ptr [ecx]          c4099a1ba7bd34e1b7aa7215c310889e 0
s      mov dword ptr [0x141a1726], eax a326171a14b83a4044384e1b6e9ae100 0
a      shr byte ptr [esi - 0x4fc2db0c], 0xfa c0ae94243db0fa8118e033273403e0 :
a      push esi                    50990c3e088713a2e24334e32a3370 0
d      xor eax, 0xd1aa9221         352192aad18d0ec03d4f9104f09c71 4
      inc ecx                      416e00000000000000000000000000 .
v: 1   adc dword ptr [esi + 0x46], 0x2084b8d1 815646d1b8842032e310932e037e70 2
l: 1   jmp 0x15                       eb1303000fa247ec0302547c0b1132e 9
s: 5   jmp 0x15                       eb1303000fa247ec0302547c0b1132e 9
c: 2   push ebp                       551e01000000000000000000000000 0
      stosb byte ptr es:[edi], al   aa3003010003000000000000000000 0
      scasd eax, dword ptr es:[edi] af0e78000000000000000000000000 0
      stosd dword ptr es:[edi], eax 0a40000101120d0077c00000000000 0
      imul esp, dword ptr [edx + ecx*4 - 0x17], -0x75 0e648ae98b03100000000000000000 0
      insd dword ptr es:[edi], dx   6d30c0001003300040710000000000 0
e      [unk]                       ff7840031470c00000000000000000 0
r      xlatb                       3ed729c7ade1730add527d4c85b0f9c6 0

# 318,405
71025/s
# 129

40c585b5400300059ccc036d7d1e07010
47c45b499f0ce20b0c13bd2em2307280
43c460307000a532309dad9c8c401d80
43c44d001f1f8a3e00b57e0509ad12ae
0fc08a11f59bb24d99573537aa8f529a
42c411e0880f9bc55b44170cad7dc187
0f17fcbb82ef09eee7d6d8db5d790713
0fc014e1a80d8d40e10091192f5513e0
0fc72c07a7cb902e2f3e98742e28d3e3
45c5175868804404fd341073dfe134de
```

- Hidden instructions across Intel and AMD processors
- Software bugs in disassemblers, such as IDA, objdump, VS, etc.
- Hardware errata, something like FOOF

More Hardware Fuzzing Examples

- Zenbleed: found a CPU bug via post-silicon fuzzing

```
movnti [rbp+0x0],ebx
```

```
rcr dh,1
```

```
sub r10, rax
```

```
rol rbx, cl
```

```
xor edi,[rbp-0x57]
```

```
movnti [rbp+0x0],ebx
```

```
sfence
```

```
rcr dh,1
```

```
lfence
```

```
sub r10, rax
```

```
mfence
```

```
rol rbx, cl
```

```
nop
```

```
xor edi,[rbp-0x57]
```

A randomly generated sequence of instructions, and the same sequence but with randomized alignment, serialization and speculation fences added.

- White-box fuzzing of hardware -> more in paper discussions

Summary



- Hardware bugs
 - Deviate from specification (errata)
 - Incorrect and vague specification
- Potential approaches to find hardware bugs
 - Manual analysis, testing
 - Fuzzing
 - Formal Verification (next lecture)

Program testing can be quite effective for showing the presence of bugs, but is hopelessly inadequate for showing their absence.
- Edsger Dijkstra