Beyond MOV ADD XOR

the unusual and unexpected in x86

Mateusz "j00ru" Jurczyk, Gynvael Coldwind

CONFidence 2013, Kraków

Who

- Mateusz Jurczyk
  - Information Security Engineer @ Google
  - http://j00ru.vexillium.org/
  - @j00ru
- Gynvael Coldwind
  - Information Security Engineer @ Google
  - http://gynvael.coldwind.pl/
  - @gynvael

Agenda

- Getting you up to speed with new x86 research.
- Highlighting interesting facts and tricks.
- Both x86 and x86-64 discussed.

Security relevance

- Local vulnerabilities in CPU ↔ OS integration.
- Subtle CPU-specific information disclosure.
- Exploit mitigations on CPU level.
- Loosely related considerations and quirks.

x86 is everywhere.

x86 - introduction not required

- Intel first ships 8086 in 1978
  - 16-bit extension of the 8-bit 8085.
- Only 80386 and later are used today.
  - first shipped in 1985
  - fully 32-bit architecture
  - designed with security in mind
    - code and i/o privilege levels
    - memory protection
    - segmentation

x86 - produced by...

Intel, AMD, VIA - yeah, we all know these.

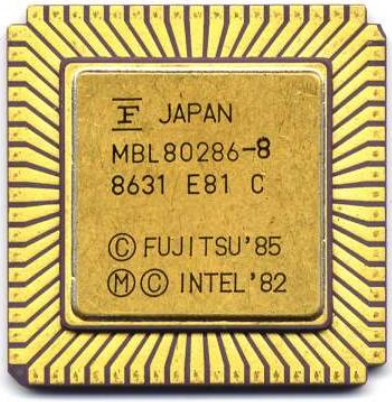
"Honorable" mentions (via Wikipedia)

- **Chips and Technologies** - *left market after failed 386 compatible chip failed to boot the Windows operating system.*
- **NEC** - sold early Intel architecture compatibles such as NEC V20 and NEC V30; product line transitioned to NEC internal architecture

http://www.cpu-collection.de/

x86 - other manufacturers

Eastern Bloc KM1810BM86 (USSR)



http://www.cpu-collection.de/

x86 - other manufacturers

Transmeta, Rise Technology, IDT, National Semiconductor, Cyrix, NexGen, Chips and Technologies, IBM, UMC, DM&P Electronics, ZF Micro, Zet IA-32, RDC Semiconductors, Nvidia, ALi, SiS, GlobalFoundries, TSMC, Fujitsu, SGS-Thomson, Texas Instruments, ...

(via Wikipedia)

At first, a *simple* architecture...

... but then:

32-bit instruction set, MMU with paging.
RISC-like pipelining, integrated x87 FPU (80-bit), on-chip cache.
In-order, integrated FPU, some models with on-chip L2 cache, MMX, SSE.
Superscalar, 64-bit databus, faster FPU, MMX (2× 32-bit).
μ-op translation.
μ-op translation, conditional move instructions, Out-of-order, register renaming, speculative execution, PAE (Pentium Pro), in-package L2 cache (Pentium Pro).
L3-cache support, 3DNow!, SSE (2× 64-bit).
optimized for low power.
Superscalar FPU, wide design (up to three x86 instr./clock).
deeply pipelined, high frequency, SSE2, hyper-threading.
VLIW design with x86 emulator, on-die memory controller.

At first, a *simple* architecture...

... but then:

Very deeply pipelined, very high frequency, SSE3, 64-bit capability (integer CPU) is available only in LGA 775 sockets.
64-bit (integer CPU), low power, multi-core, lower clock frequency, SSE4 (Penryn).
Out-of-order, superscalar, 64-bit (integer CPU), hardware-based encryption, very low power, adaptive power management.
x86-64 instruction set (CPU main integer core), on-die memory controller, hypertransport.
Monolithic quad-core, SSE4a, HyperTransport 3 or QuickPath, native memory controller, on-die L3 cache, modular.
In-order but highly pipelined, very-low-power, on some models: 64-bit (integer CPU), on-die GPU.
Out-of-order, 64-bit (integer CPU), on-die GPU, low power (Bobcat).
SSE5/AVX (4× 64-bit), highly modular design, integrated on-die GPU.

x86 bursted with new functions

Security relevant examples

- **N**o e**X**ecute bit (W^X, DEP)
  - completely redefined exploit development, together with ASLR
- **S**upervisor **M**ode **E**xecution **P**revention
- **RDRAND** instruction
  - cryptographically secure prng
- **Related**: TPM, VT-d, IOMMU

Overall...

- **Gigantic market share**
  - millions of x86 CPUs shipped every year.
- **Dramatic development**
  - most basic functionality already invented and implemented.
  - noticeable trend: more and more high-level, abstract features.
- **Vast complexity**
  - open the Intel manuals at a random page and you'll likely find something interesting or worth further investigation.

x86 is a great ~~attack~~ research target.

just look where nobody has looked yet.

surprisingly, there's many places like that.

CPU ↔ OS vulnerabilities

historical perspective

or how important it is to integrate correctly.

(for local system security)

Security model in modern x86 computing at the lowest level

- Architecture provides means to create a secure environment.
  - primarily by splitting execution between supervisor (kernel) and client (applications).
  - a set of rules and assumptions an OS can take for granted.
- **None** of CPU security features make the environment safe by themselves.
- The operating system must fully and correctly make use of them to accomplish security.

Essential requirements

1. CPU's must function exactly as advertised.
    a. similarly to whatever emulates them, e.g. VMM.
2. OS must be **fully** aware of **all** functionality provided by the CPU.
3. OS must correctly interpret information provided by the CPU.

Local system security is hard

- Userland applications can interact with the CPU in however way they choose.
  - within privilege-enforced boundaries.
- Assuming ring-3 code is hostile, the OS needs to predict **all** unusual, faulty or weird states a program can put the system in.

The problem

CPU vendors are not of much help.

- Very frequently manuals contain *vague* statements regarding abnormal conditions.
  - ... or not discuss them at all.
- Virtually **no** explicit warnings addressing special situations.
- This leaves low-level system developers on their own.
  - very smart guys.
  - writing code in the 90's...
  - but they are not security people.

We could expect many conditions to be overlooked before a proper review from some passionate researcher.

And there are quite a few examples from the past.

SYSRET vulnerability

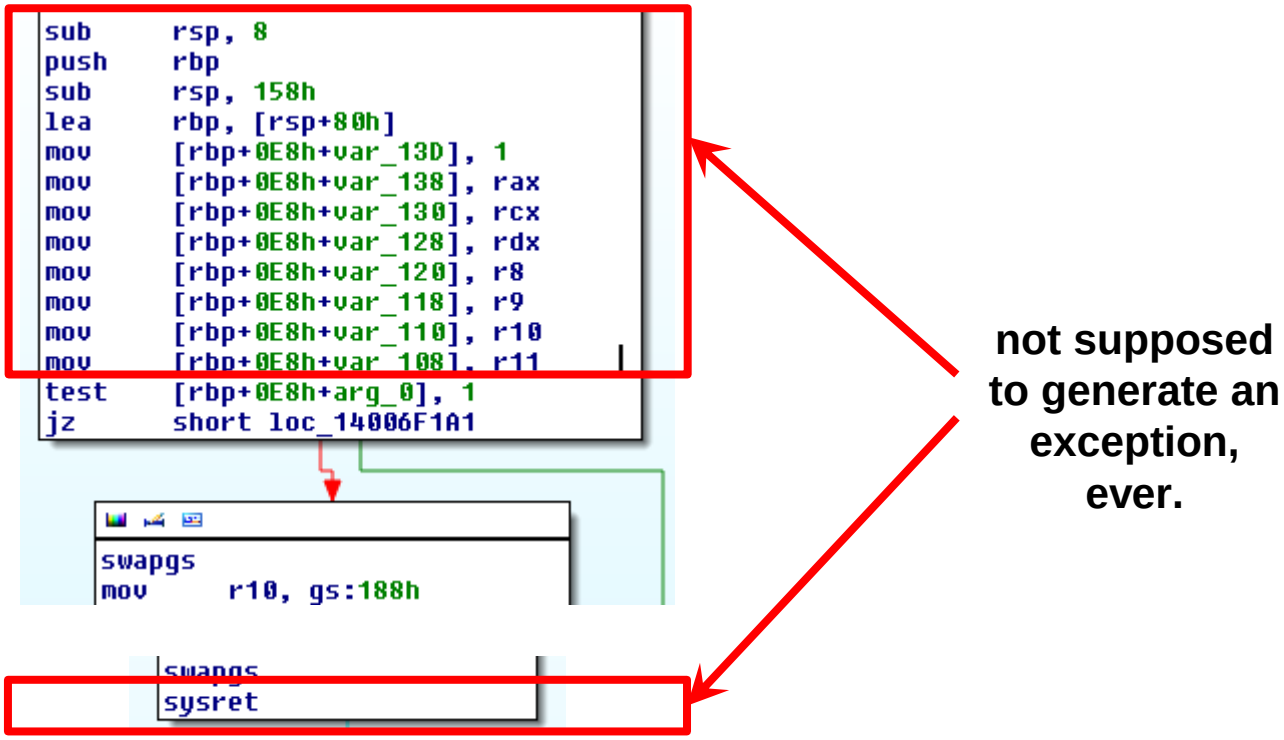
- (Re)discovered in April 2012 by **Rafał Wojtczuk.**
- Applicable only for Intel 64-bit platforms.
  - AMD not affected.
- Part of the SYSRET instruction functionality was unaccounted for in Windows, Linux and BSD:

64-Bit Mode Exceptions

#UD	If IA32_EFER.SCE bit = 0.
	If the LOCK prefix is used.
#GP(0)	If CPL ≠ 0.
	If ECX contains a non-canonical address.

SYSRET vulnerability

- ECX is user-controlled.
  - SYSRET is used for kernel → user transitions.



SYSRET vulnerability

- The `gs:` segment plays a special role in 64-bit kernel-mode.
  - used by kernels to address per-CPU structures.
- Switched by kernel upon entry to ring-0.
  - switched back when returning.
- Nested exceptions assume switched `gs:` if previous mode was kernel.
- If we trigger an exception in kernel before first or after second `SWAPGS` instruction, game over.

SYSRET vulnerability

References

http://blog.xen.org/index.php/2012/06/13/the-intel-sysret-privilege-escalation/

http://www.vupen.com/blog/20120806.Advanced_Exploitation_of_Windows_Kernel_x64_Sysret_EoP_MS12-042_CVE-2012-0217.php

http://media.blackhat.com/bh-us-12/Briefings/Wojtczuk/BH_US_12_Wojtczuk_A_Stitch_In_Time_WP.pdf

While we're at SWAPGS...

In 2008, Derek Soeder discovered that the code:

```
JMP non-canonical-address
```

executed in VMware generates #GP at

```
Rip = non-canonical-address
```

instead of

```
Rip = address-of-faulty-jmp
```

VMware SWAPGS - exploitation

- The #GP handler does not IRETQ to a non-canonical address.
  - passes the exception to dispatcher directly.

If you keep trying to jump at the address:

```
MOV     RAX, 0x8000000000000000
PUSH    RAX
JMP     QWORD PTR [RSP]
```

a hardware interrupt will eventually preempt the thread at Rip=8000... and later return back to it.

VMware SWAPGS vulnerability

Confused `gs:` value in nested #GP handler → elevation of privileges in Windows and FreeBSD.

References

http://lists.grok.org.uk/pipermail/full-disclosure/2008-October/064860.html

`nt!Kei386EoiHelper` vulnerability

- The function is a generic syscall / interrupt kernel → user exit routine.
  - same as `nt!KiExceptionExit`
- It used `KTRAP_FRAME.SegCs & 0xfff7 = 0` to indicate a special kernel trap frame condition.
- In all 32-bit Windows, `cs:=7` can point to a valid "code" LDT segment.

nt!Kei386EoiHelper vulnerability

- Result: use of uninitialized KTRAP_FRAME fields.
  - extremely tricky (but possible) to reliably exploit.

References

http://j00ru.vexillium.org/blog/20_05_12/cve_2011_2018.pdf

http://pwnies.com/winners/

LDT itself is troublesome

Prior research

- In 2003, Derek Soeder that the "Expand Down" flag was not sanitized.
  - *base* and *limit* were within boundaries.
  - but their semantics were reversed
- User-specified selectors are not trusted in kernel mode.
  - especially in Vista+
- But Derek found a place where they did.
  - write-what-where → local EoP

LDT Expand Down vulnerability

References

http://www.eeye.com/Resources/Security-Center/Research/Security-Advisories/AD20040413D

Be careful about virtual-8086, too

- The virtual-8086 compatibility mode allows ring-3 code to forge somewhat unusual conditions
  - `CS: & 3` can be 0
  - the semantics of segment registers in 16-bit environments are different.
- Quite a few vulnerabilities found around the area
  - VMM logical bugs
  - Miscellaneous issues in the Windows implementation of v8086: NTVDM.

virtual-8086 mode vulnerabilities

References

https://www.cr0.org/paper/to-jt-party-at-ring0.pdf

http://www.securityfocus.com/archive/1/522141

http://seclists.org/fulldisclosure/2004/Oct/404

http://seclists.org/fulldisclosure/2004/Apr/477

http://seclists.org/fulldisclosure/2007/Apr/357

Trap handlers

Trap handlers

Every operating system must provide handlers for all traps.

User-mode can trigger most of them with controlled KTRAP_FRAME.

Trap handlers

Windows implements special handling of different corner-cases in some handlers.

in hacky ways.

nt!KiTrap0d vulnerability

`nt!KiTrap0d` vulnerability

- Found by Tavis Ormandy in 2010
- The default `#GP` handler was expecting:
  - previous `KTRAP_FRAME.Eip` to be `nt!Ki386BiosCallReturnAddress`
  - previous `KTRAP_FRAME.SegCs` to be `0xB` (`CPL=3`)
- Both conditions can be forged from ring-3.
- Allowed for a kernel stack switch → local **elevation of privileges**.

nt!KiTrap0d vulnerability

References

http://seclists.org/fulldisclosure/2010/Jan/341

http://pwnies.com/archive/2010/winners/

nt!KiTrap01, nt!KiTrap0e flaws

nt!KiTrap01, nt!KiTrap0e flaws

- The 32-bit #DB and #PF handlers deal with special cases at magic KTRAP_FRAME.Eip:
  - nt!KiFastCallEntry (#DB)
  - nt!KiSystemServiceCopyArguments (#PF)
  - nt!KiSystemServiceAccessTeb (#PF)
  - nt!ExpInterlockedPopEntrySListFault (#PF)
- They don't check previous CPL
  - kernel-mode condition: KTRAP_FRAME.SegCs=8
- Try to restart execution at a different Eip (but same previous privilege level)

`nt!KiTrap01`, `nt!KiTrap0e` flaws

By altering the exception context, the handlers disclose four static kernel-mode addresses within ntoskrnl.exe

nt!KiTrap0e vulnerability

That's not the end!

- The #PF handler also blindly trusts KTRAP_FRAME.Ebp
  - fully controlled through the Ebp register for a ring-3 origin.
  - can be used to crash system (bugcheck) or read the least significant bit of any kernel byte in two instructions.

```
xor ebp, ebp
jmp 0x8327d1b7
```

nt!KiSystemServiceAccessTeb

nt!KiTrap0e vulnerability

References

http://www.nosuchcon.com/talks/D1_01_j00ru_Abusing_the_Windows_Kernel.pdf

http://j00ru.vexillium.org/blog/21_05_13/kitrap0e.html

http://j00ru.vexillium.org/?p=1767

CPU-specific information leaks

where the kernel cannot defend.

(usually)

GDT, IDT

- Two essential, CPU-wide structures.
  - pointed to by dedicated (abstract) `GDTR`, `IDTR` registers.
- Their addresses can be disclosed using standard `SGDT` and `SIDT` instructions.
  - available at every privilege level.
  - access not controlled via a `CR4` bit
    - should be, similarly to `CR.TSD` enabling/disabling unprivileged `RDTSC`
- Information about kernel address space can be used in attacks against local vulnerabilities.
  - CPU structures are cross-platform, thus useful.

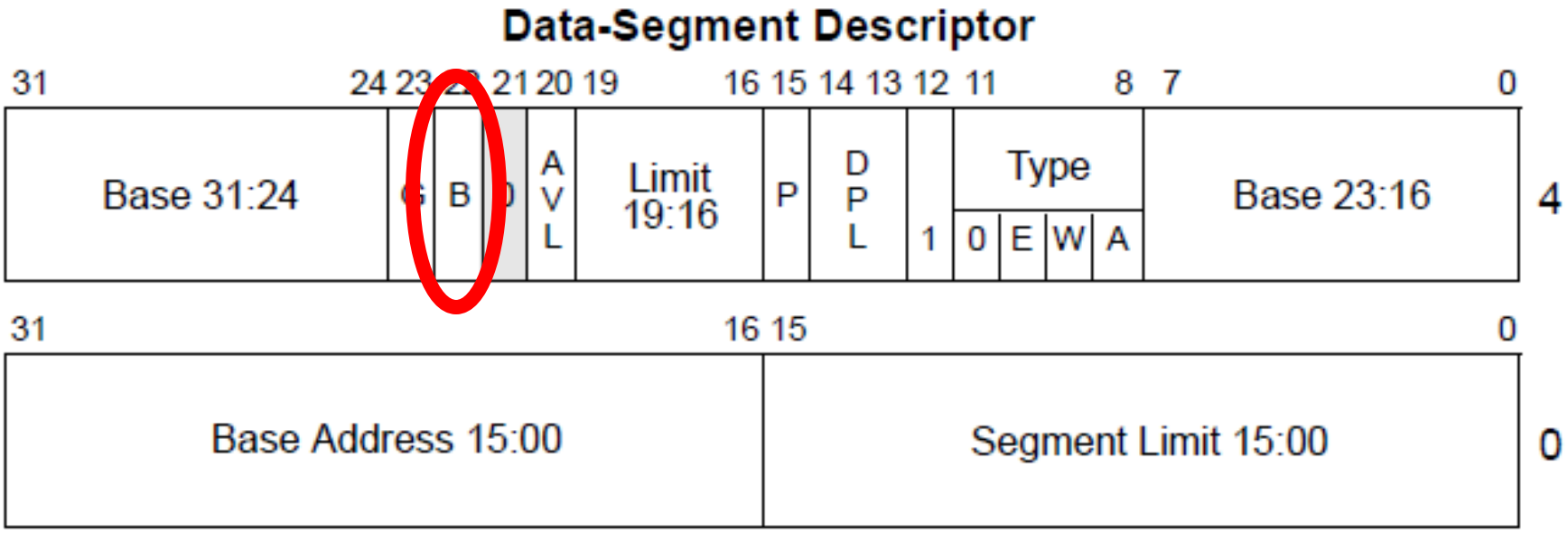
References

http://vexillium.org/dl.php?call_gate_exploitation.pdf

Disclosing kernel stack pointer

- Back to custom LDT entries ☺

The "Big" flag



Different functions

D/B (default operation size/default stack pointer size and/or upper bound) flag

Performs different functions depending on whether the segment descriptor is an executable code segment, an expand-down data segment, or a stack segment. (This flag should always be set to 1 for 32-bit code and data segments and to 0 for 16-bit code and data segments.)

Stack segment

- **Stack segment (data segment pointed to by the SS register).** The flag is called the B (big) flag and it specifies the size of the stack pointer used for implicit stack operations (such as pushes, pops, and calls). If the flag is set, a 32-bit stack pointer is used, which is stored in the 32-bit ESP register; if the flag is clear, a 16-bit stack pointer is used, which is stored in the 16-bit SP register. If the stack segment is set up to be an expand-down data segment (described in the next paragraph), the B flag also specifies the upper bound of the stack segment.

Kernel-to-user returns

- On each interrupt and system call return, system executes `IRETD`
  - pops and initializes `cs`, `ss`, `eip`, `esp`, `eflags`

Or that's what everyone thinks!

IRETD algorithm

```
IF stack segment is big (Big=1)
   THEN
         ESP ←tempESP
   ELSE
         SP ←tempSP
FI;
```

- Upper 16 bits of are not cleaned up.
  - Portion of kernel stack pointer is disclosed.
- Behavior not discussed in Intel / AMD manuals.

Address space leaks via cache examination

- Different types of shared cache are used to store information about user and kernel address space
  - L1, L2, L3 cache
  - Translation Lookaside Buffer
- Arbitrary native code running locally has means to partially examine cache contents.
  - reversing hash algorithm used to store entries in cache.
  - timing attacks.
  - some methods are specific to particular CPU vendors.

Not just addresses can be leaked (side channels)

- The Hyper-Threading technology enables tow logical CPUs within a single physical core.
- Side channels between them exist
  - a controlled, rogue thread can infer information about what a *secret* thread is currently doing.
  - e.g. what private key OpenSSH is currently processing.

Cache attacks

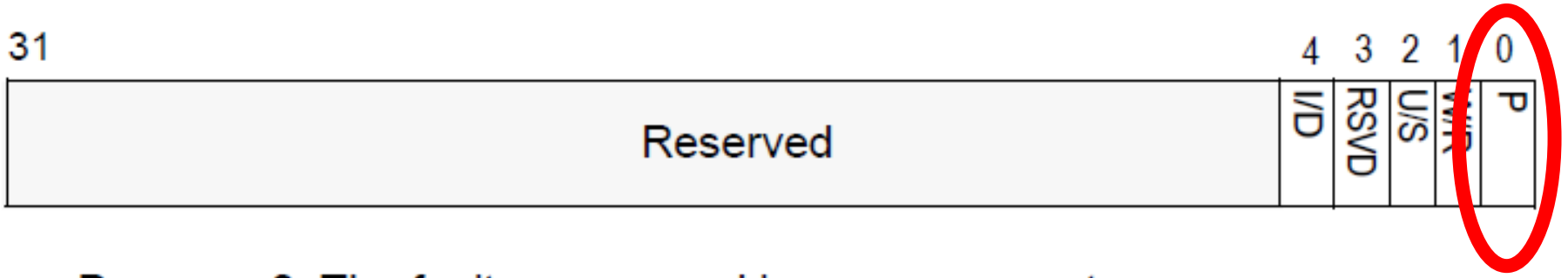
References

Hund, Willems, Holz: *“Practical Timing Side Channel Attacks Against Kernel Space ASLR”*

http://www.daemonology.net/papers/htt.pdf

http://www.daemonology.net/hyperthreading-considered-harmful/

Kernel memory layout through the “Present” #PF flag



P
0 The fault was caused by a non-present page.
1 The fault was caused by a page-level protection violation.

Kernel memory layout through the “Present” #PF flag

- The “P” flag in the error code of the Page-Fault Exception is accurate even for userland code accessing ring-0 memory areas.
  - even if the reason of the #PF was caused by insufficient privileges.
- In Linux, the error code is propagated down to syslogs.
  - readable from ring-3.

Kernel memory layout through the “Present” #PF flag

References

http://vulnfactory.org/blog/2013/02/06/a-linux-memory-trick/

Exploit mitigations

detection and prevention measures made possible by x86

Integer overflow detection

Suppose we wish to detect all integer overflows in compiled assembly at run-time.

(adding some checks at compile time)

Similarly to what the IOC LLVM patch does.

INTO to the rescue

The way -ftrapv (gcc) and similar are usually implemented

```
COMPILER_RT_ABI si_int
__addvsi3(si_int a, si_int b)
{
    si_int s = a + b;
    if (b >= 0)
    {
        if (s < a)
            compilerrt_abort();
    }
    else
    {
        if (s >= a)
            compilerrt_abort();
    }
    return s;
}
```

http://svnweb.freebsd.org/base/vendor/compiler-rt/dist/lib/addvsi3.c?view=co

INTO to the rescue

Simple solution to detect (signed) integer overflows.

```
[bits 32]
mov eax, 0x7fffffff
add eax, 5
into
```

Interrupt #OF if flag OF is set. Translates to:

- C0000095 (STATUS_INTEGER_OVERFLOW)
- Signal 11 (SIGSEGV)

One instruction. Doesn't work for unsigned types (CF vs OF).
Removed in AMD64. Stupid AMD :(

BOUND Instruction

```
BOUND r16, m16&16
BOUND r32, m32&32
```

- Dedicated instruction to check a complicated bounds checking condition:

```
IF (ArrayIndex < LowerBound OR ArrayIndex > UpperBound)
  THEN #BR; FI;
```

- Removed from x86-64 (together with `INTO`)

BOUND Instruction

- Otherwise implemented using at least four x86 instructions.
- A great optimization for potential run-time memory error detection.
  - e.g. AddressSanitizer (uses a different concept).
  - no known detectors are known to use the mechanism.

Interrupt 5—BOUND Range Exceeded Exception (#BR)

Exception Class Fault.

Description

Indicates that a BOUND-range-exceeded fault occurred when a BOUND instruction was executed. The BOUND instruction checks that a signed array index is within the upper and lower bounds of an array located in memory. If the array index is not within the bounds of the array, a BOUND-range-exceeded fault is generated.

Performance counters: taming ROP on Sandy Bridge

- Presented by Georg Wicherski at SyScan 2013

Concept

- Branch predictor holds 16 entries for recent returns
  - populated by calls.
- using PMC (0x8889), you can get the CPU to yield an interrupt upon too many prediction misses.
- Implement a custom interrupt handler
  - check for `CALL` instructions directly prior to return addresses.
  - not found? it (most likely) is a ROP chain!

Taming ROP on Sandy Bridge

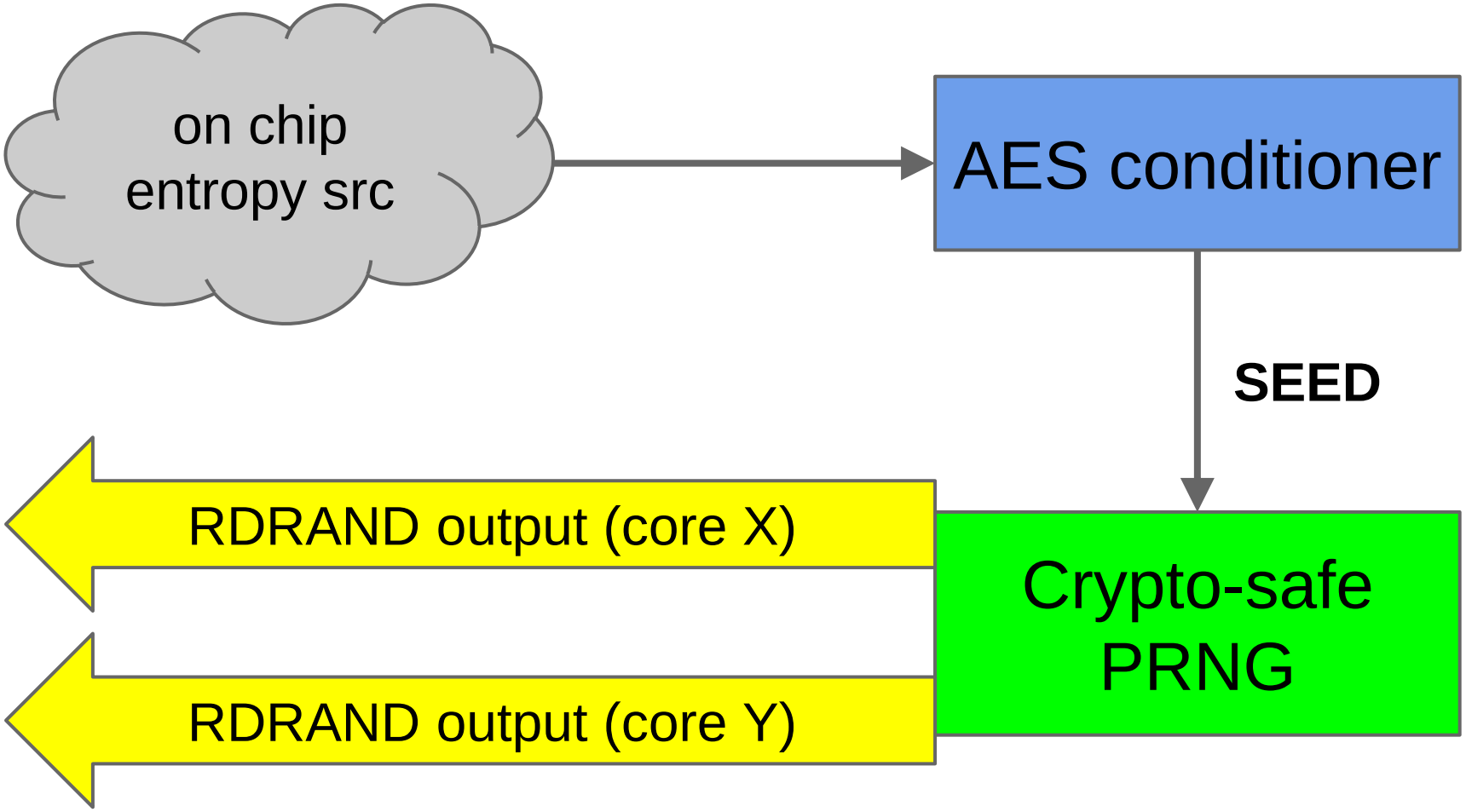
- Related work: **Mitigating ROP via Last Branch Recording**
  - BlueHat 1st prize

References

http://syscan.org/index.php/download/get/3c6891f2e90e661ea23224cd8f419262/SyScan2013_DAY1_SPEAKER05_Georg_WIcherski_Taming_ROP_ON_SANDY_BRIDGE_syscan.zip

http://blogs.technet.com/b/srd/archive/2012/07/23/technical-analysis-of-the-top-bluehat-prize-submissions.aspx

RDRAND on Ivy Bridge

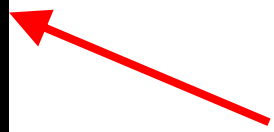


http://software.intel.com/sites/default/files/m/d/4/1/d/8/441_Intel_R__DRNG_Software_Implementation_Guide_final_Aug7.pdf

RDRAND on Ivy Bridge

- Sets CF if a random number was ready.
  (CF not set -> output is 0)
- Frequent reseeds (upper limit: 511 * 128-bit reads). You can even force a reseed:
  - call `RDRAND` over 511 times
  - call `RDRAND` over 32 times with 10 us delay inbetween

```
gen_rand:
 rdrand eax
 jnc gen_rand
```



don't forget to check CF!

RDRAND on Ivy Bridge

- **Windows 8**
  - `nt!ExGenRandom` (exported `nt!RtlRandomEx`)
  - used for generation of secret values
    - stack cookies for the nt image
    - kernel module image base relocations
    - replaced the old `RDTSC` entropy source
- **Linux:** not actually used anywhere?

http://lxr.free-electrons.com/source/arch/x86/kernel/cpu/rdrand.c

RDRAND on Ivy Bridge

References

http://smackerelofopinion.blogspot.co.uk/2012/10/intel-rdrand-instruction-revisited.html

http://software.intel.com/sites/default/files/m/d/4/1/d/8/441_Intel_R__DRNG_Software_Implementation_Guide_final_Aug7.pdf

Miscellaneous

amusing, interesting and potentially scary facts about x86

Microsoft VirtualPC 2004 detection

- A number of techniques for detection of VM environment
  - differences in functioning of the CPU are some of them.

Gynvael's technique from 2006:

```
rep rep rep rep rep rep rep rep rep rep rep rep
rep rep rep rep movsb
```

- Generates an #UD on host machines
- Generated no exception within a VirtualPC 2004 guest.
  - likely due to x86 translator inconsistency.

Generic VM detection

j00ru's technique from 2008:

```
rep rep rep rep rep rep rep rep rep rep rep
rep rep rep rep rep rep rep rep rep movsb
```

- Generates an `#GP` on host machines
- Generated an `#UD` on a majority of VM back then.
  - discrepancy can be used to distinguish between host and guest

Generic VM detection

References

http://www.openrce.org/forums/posts/247

http://www.woodmann.com/forum/archive/index.php/t-11245.html

http://www.openrce.org/blog/view/1029

http://www.symantec.com/avcenter/reference/Virtual_Machine_Threats.pdf

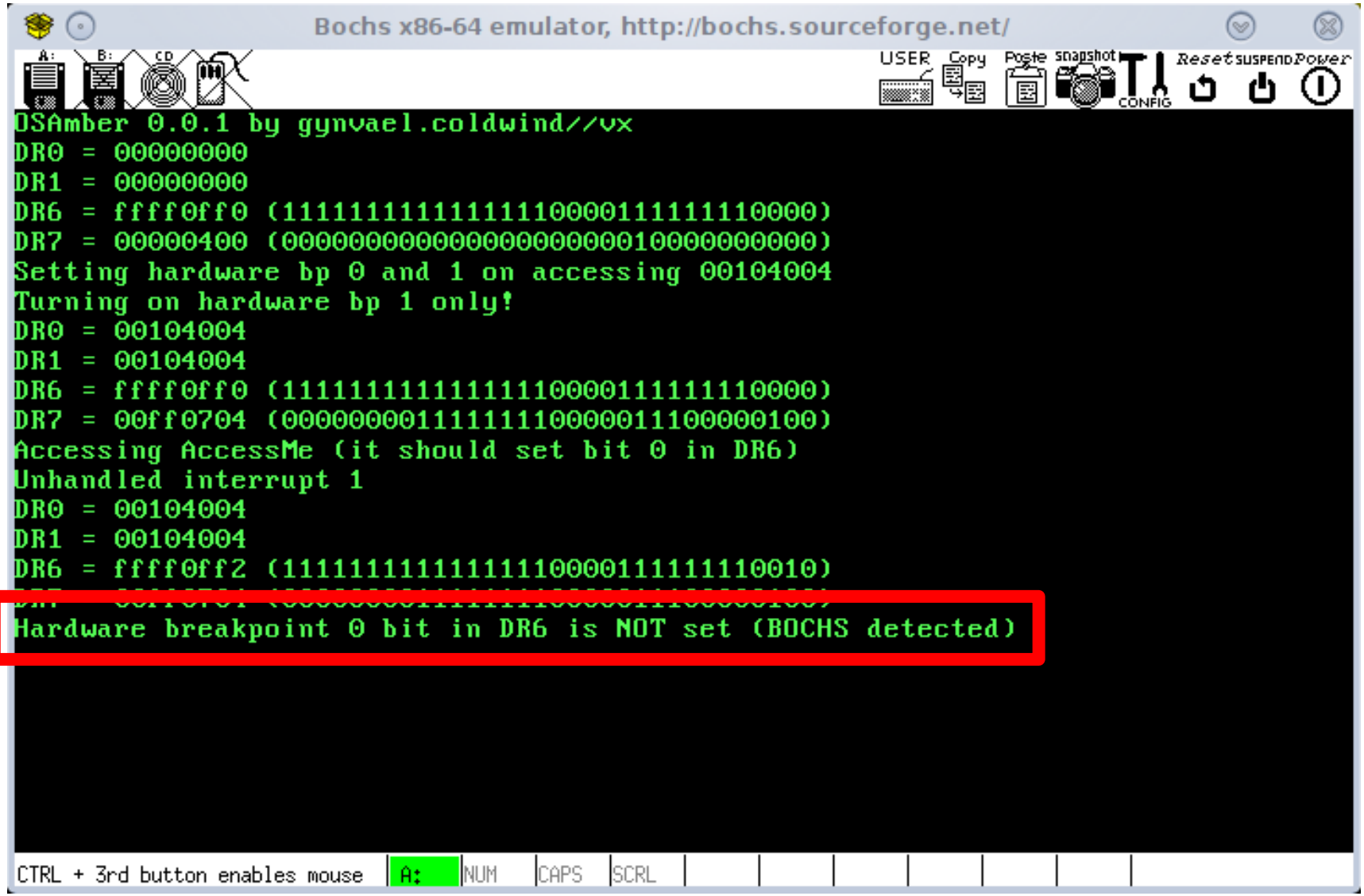
A historical note on DR6

- According to Intel Manuals from 2006:

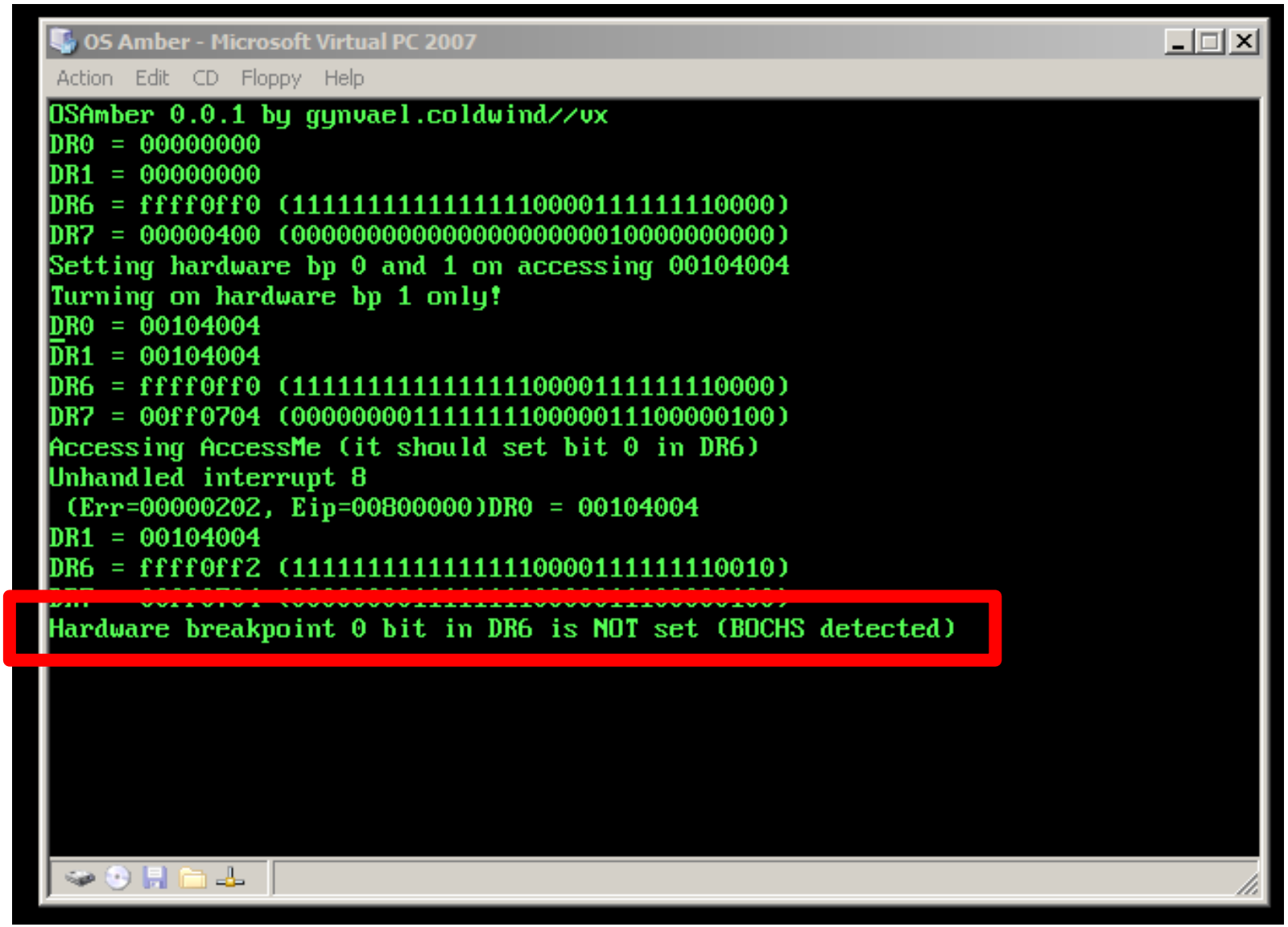
*"**B0 through B3** (breakpoint condition detected) flags (bits 0 through 3) — **Indicates (when set) that its associated breakpoint condition was met** when a debug exception was generated. [...]. **They are set even if the breakpoint is not enabled** by the Ln and Gn flags in register DR7.“*

- **Question**: Do VMs actually set these bits?

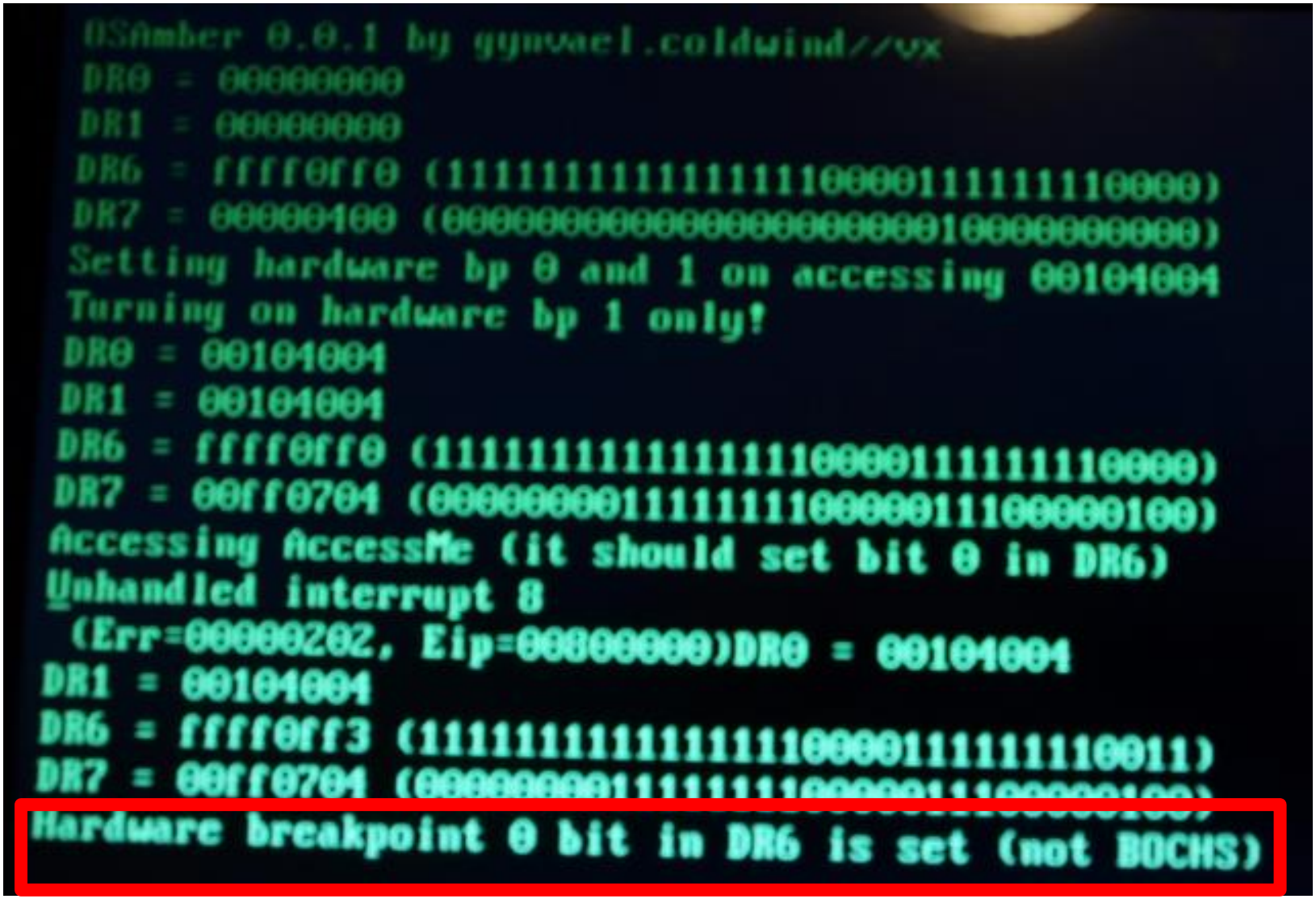
A historical note on DR6



A historical note on DR6



A historical note on DR6



A historical note on DR6

- Changed in Intel manuals since 2009:

  "They may or may not be set if the breakpoint is not enabled by the Ln or the Gn flags in register DR7"

- AMD does not mention it at all (or we're not aware)
- This technique may or may not be useful.

```
2010-01-06 08:02:00 = Schrödinger's Cat
{
   Thanks for the post. Lol.
}
```

TF flag modified behavior

- Normally TF flag is used for single-instruction step.
- **`MSR_DEBUGCTLA`** can change this behavior:
  - BTF (single-step on branches) flag (bit 1)
  - On Windows you can use `NtSystemDebugControl` for setup.

References

- Intel Manuals 3a / 3b
- Pedram's post: http://www.openrce.org/blog/view/535/Branch_Tracing_with_Intel_MSR_Registers

TF flag modified behavior

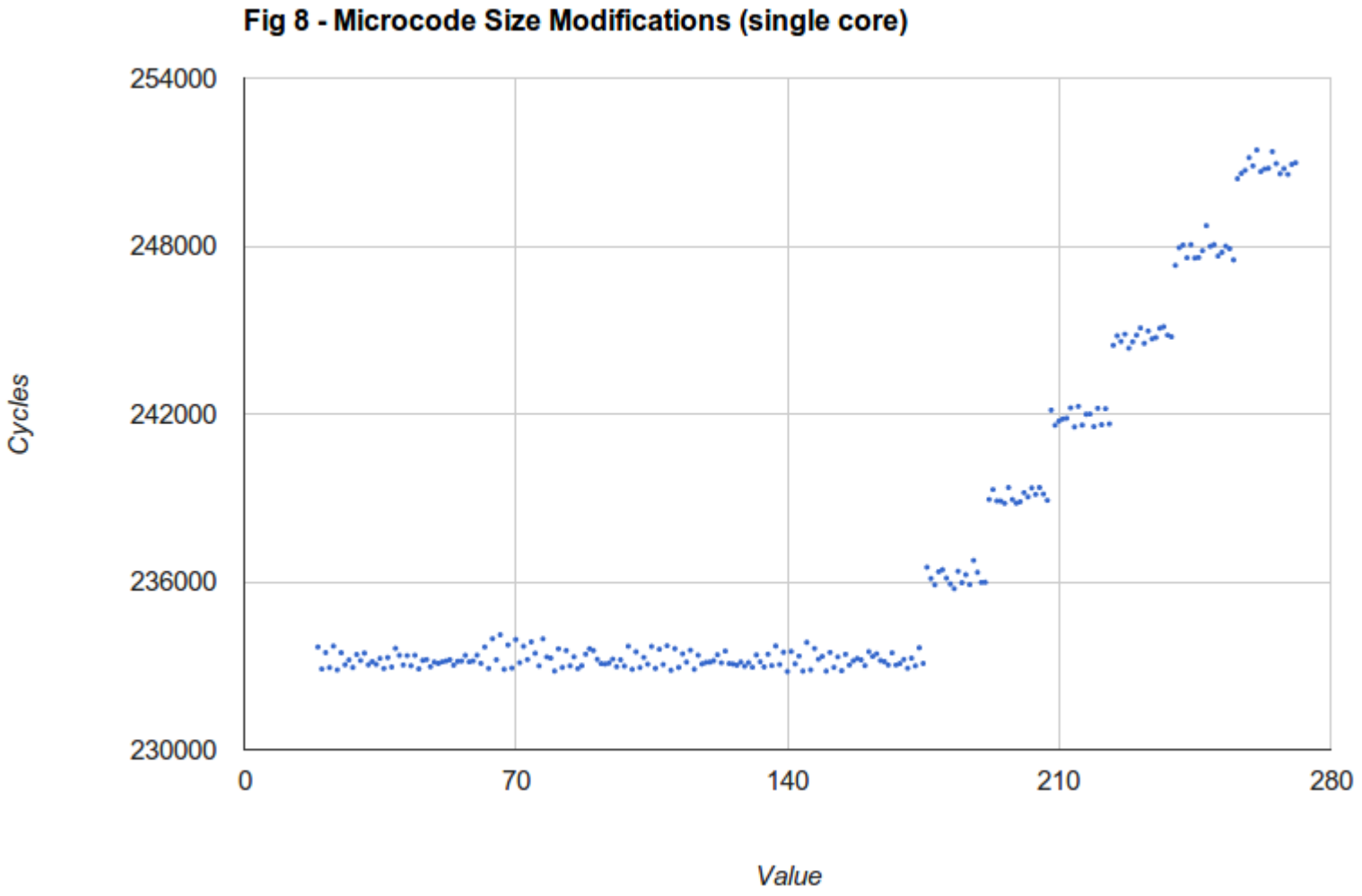
Random note

Since this is still slow for tracing, some debuggers implement simple instruction emulation for tracing.

"*Internal emulation of simple commands (Options|Run trace|Allow fast command emulation) has made run and hit trace 15 (fifteen!) times faster*"

http://www.ollydbg.de/version2.html

Notes on Intel Microcode Updates



http://inertiawar.com/microcode/ (Ben Hawkes)

Notes on Intel Microcode Updates

- File format and data structures further described.
- Results suggest that update is authenticated using 2048 RSA signature.

Notes on Intel Microcode Updates

- *Timing analysis reveals 512-bit steps correlating to supplied microcode length. This is a common message block size for cryptographic hash functions such as SHA1 and SHA2.*

- *The RSA signature was located, and the signed data is a PKCS#1 1.5 encoded hash value. Older processor models use a 160-bit digest (SHA1), and newer process models use a 256-bit digest (SHA2).*

Historical note: LOADALL

- 286: `0F 05` - read data from 0x800 to `MSW, TR, IP, LDTR`, segment regs (including hidden part), general, `GDT, LDT, IDT, TSS`
- 386: `0F 07` – a 32-bit aware version of the above.
- Later: invalid opcode. (`#UD`)

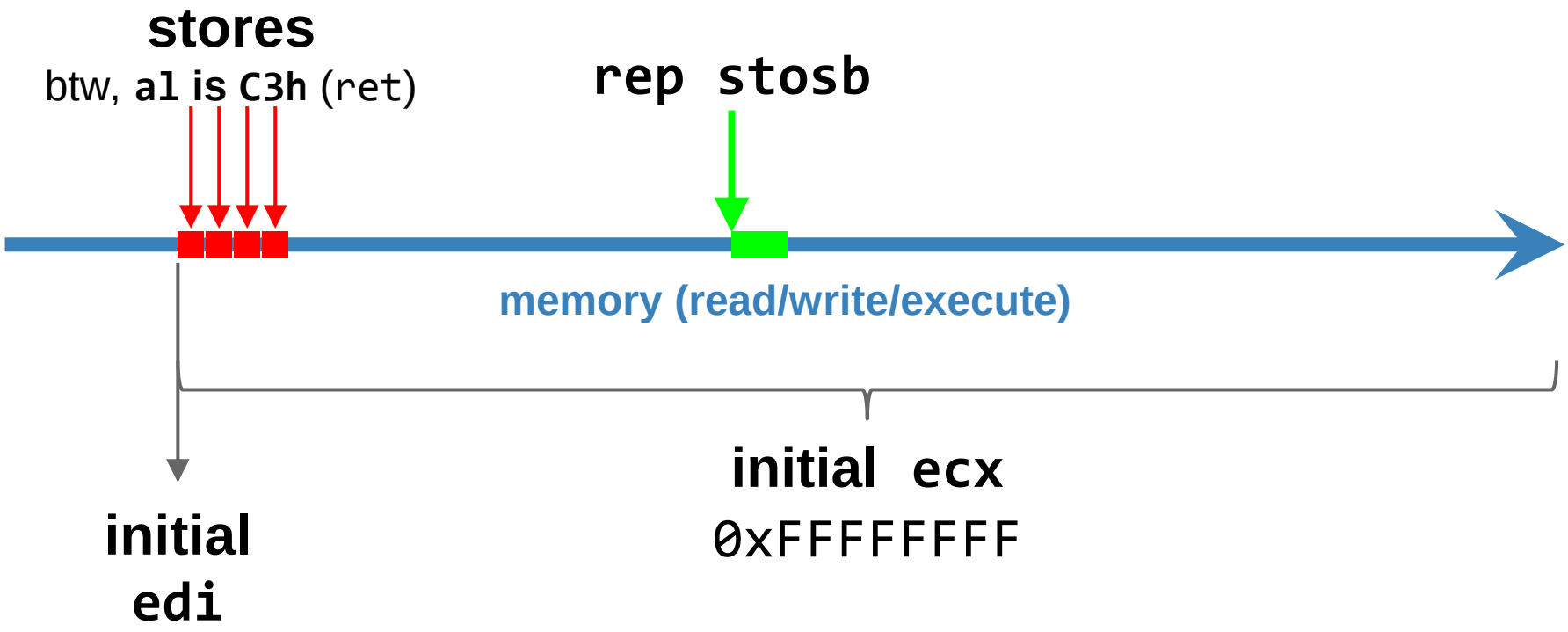
Used to gain access above 1MB of memory.
(`himem.sys`, `emm386.exe`, Windows 2.1, etc)

Currently these opcodes are occupied by `SYSCALL`, `SYSRET`.

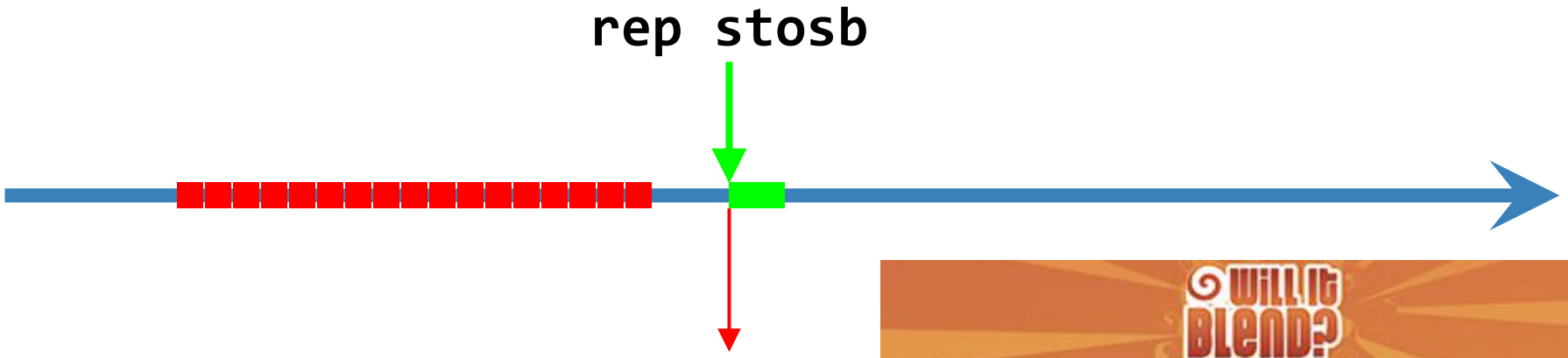
鼠

Kris Kaspersky's REP STOS PRNG

(Gynvael's version; Kris originally used `df=1` and `al=90`)

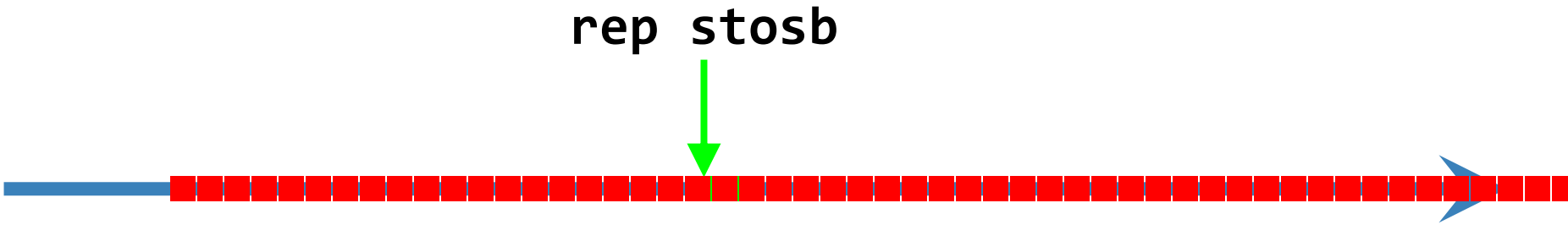


Kris Kaspersky's REP STOS PRNG



So... What happens when the store reaches this point?

Kris Kaspersky's REP STOS PRNG



It will just keep going and stop at the next interrupt*.
So, the ECX value after this is pseudo-random.

Let's see some generated values!

* Depends on CPU, new Intel Core i3/i5/i7 CPUs will actually stop after overwriting rep; the prefetch input queue bug seems to be fixed there.

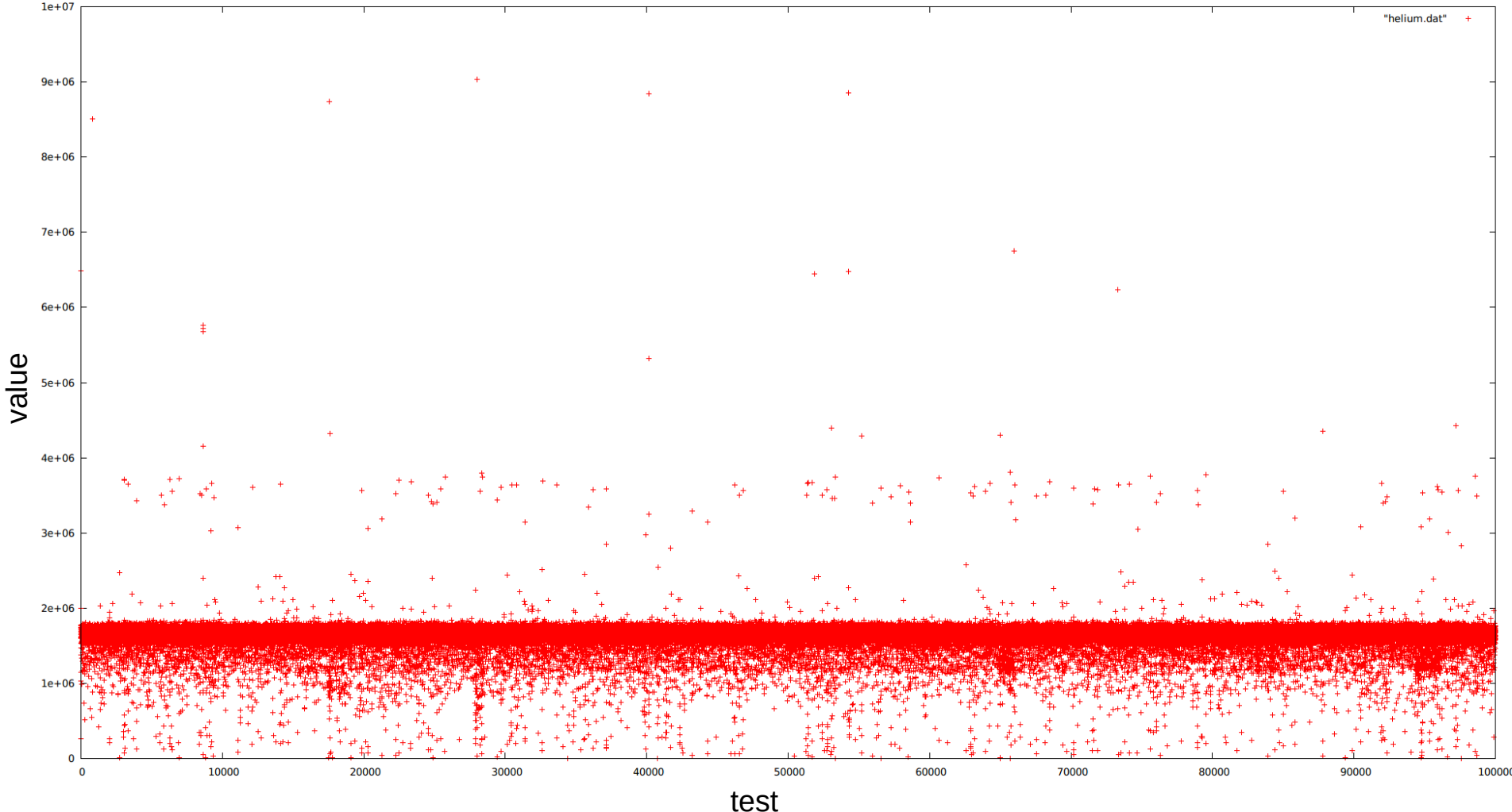
Kris Kaspersky's REP STOS PRNG

Intel(R) Core(TM)2 Duo CPU T5670

offset	min	avg	max
F00h	115CD0h	179624h	2866D0h
F01h	71DE1h	870FAh	91EC7h
F02h	56DF2h	83B2Eh	9216Fh
F03h	6EDAh	8028Ah	D3BFFh
F04h	68ECBh	83431h	918A1h
F05h	3DD17h	815D9h	900C3h
...			
F08h	10F5D0h	175D04h	18BE90h
...			
F10h	123E10h	1734BEh	19B110h

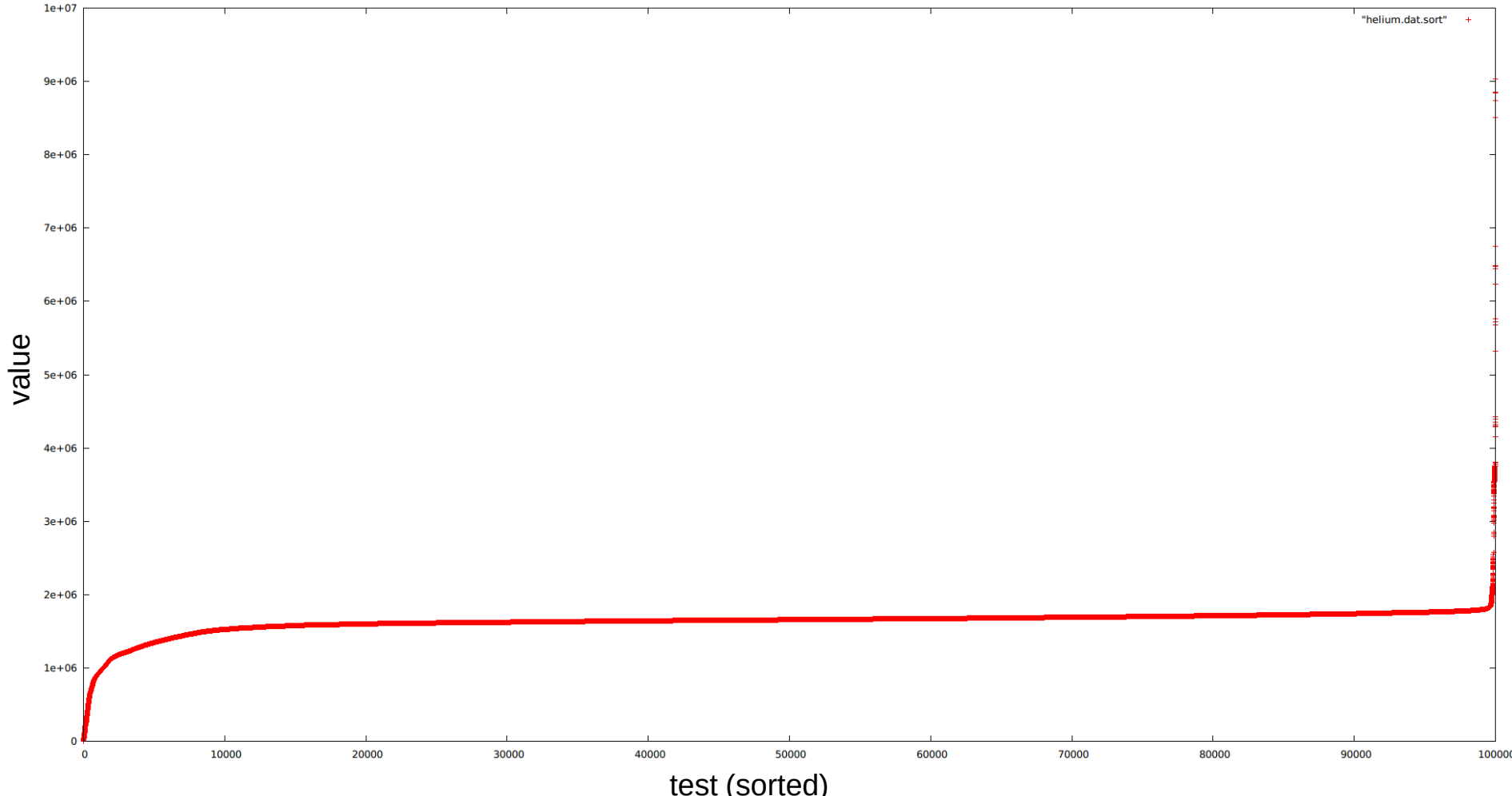
Kris Kaspersky's REP STOS PRNG

Intel(R) Core(TM)2 Duo CPU T5670



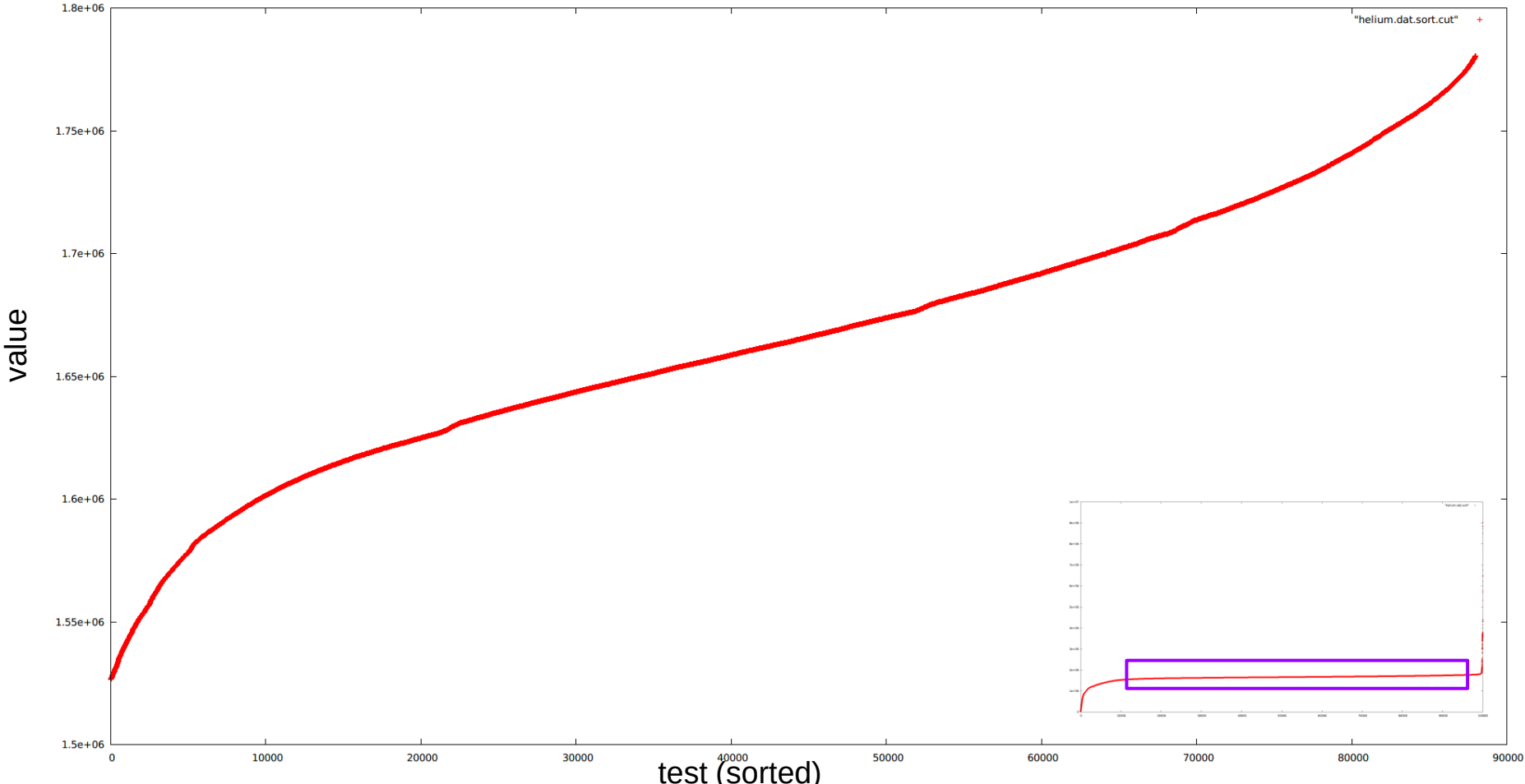
Kris Kaspersky's REP STOS PRNG

Intel(R) Core(TM)2 Duo CPU T5670



Kris Kaspersky's REP STOS PRNG

Intel(R) Core(TM)2 Duo CPU T5670



Kris Kaspersky's REP STOS PRNG

VIA Nano X2 U4025

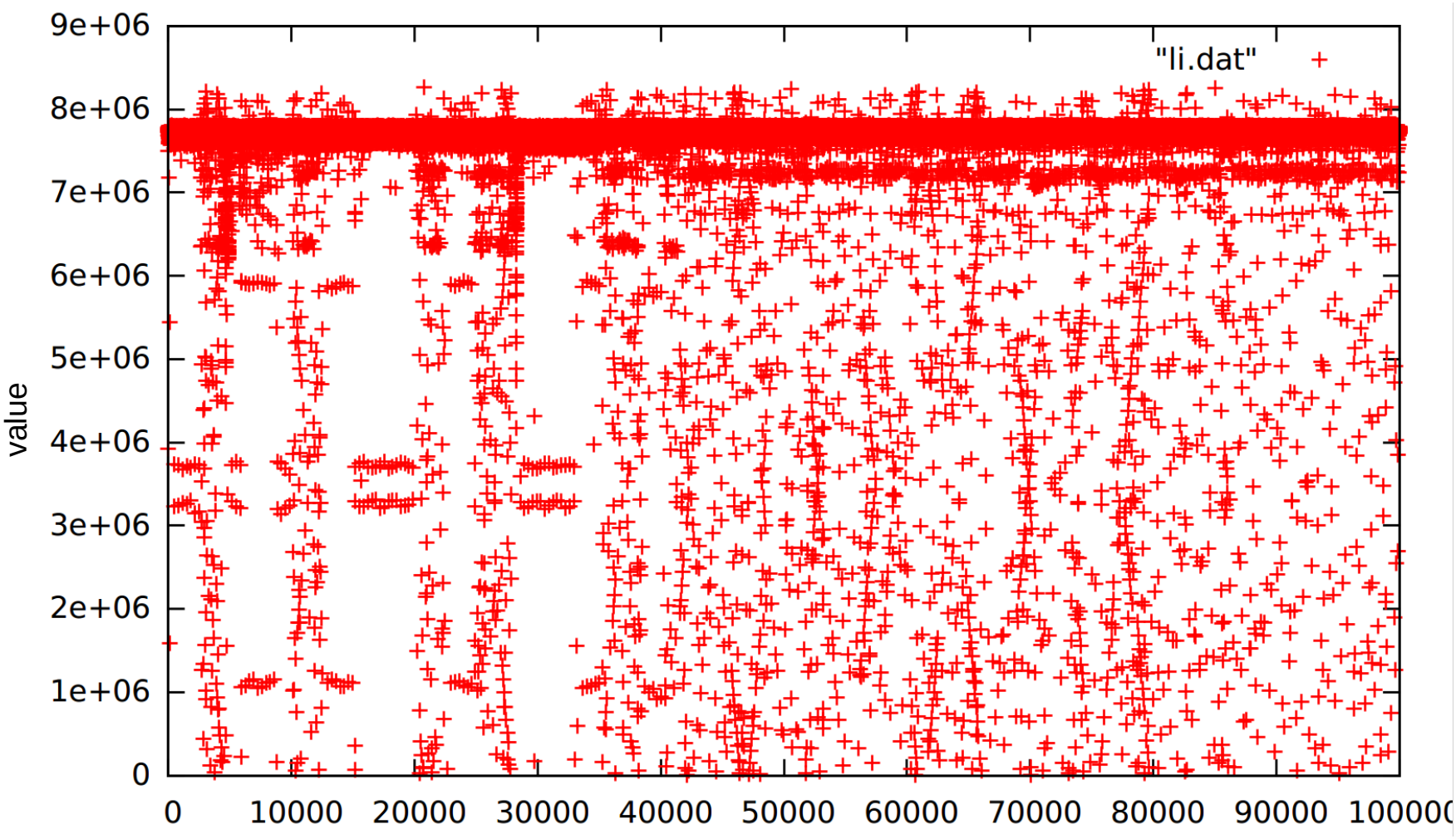
`offset = F00h`

individual test results at that offset (with a 80h "run way"):

89180h	79E180h	748180h	74C180h	74D180h
751180h	756180h	74C180h	730180h	BF180h
4A9180h	74B180h	74E180h	72B180h	74B180h
756180h	74E180h	749180h	755180h	74C180h
750180h	74D180h	749180h	759180h	741180h
739180h	74E180h	748180h	754180h	74C180h
755180h	74C180h			

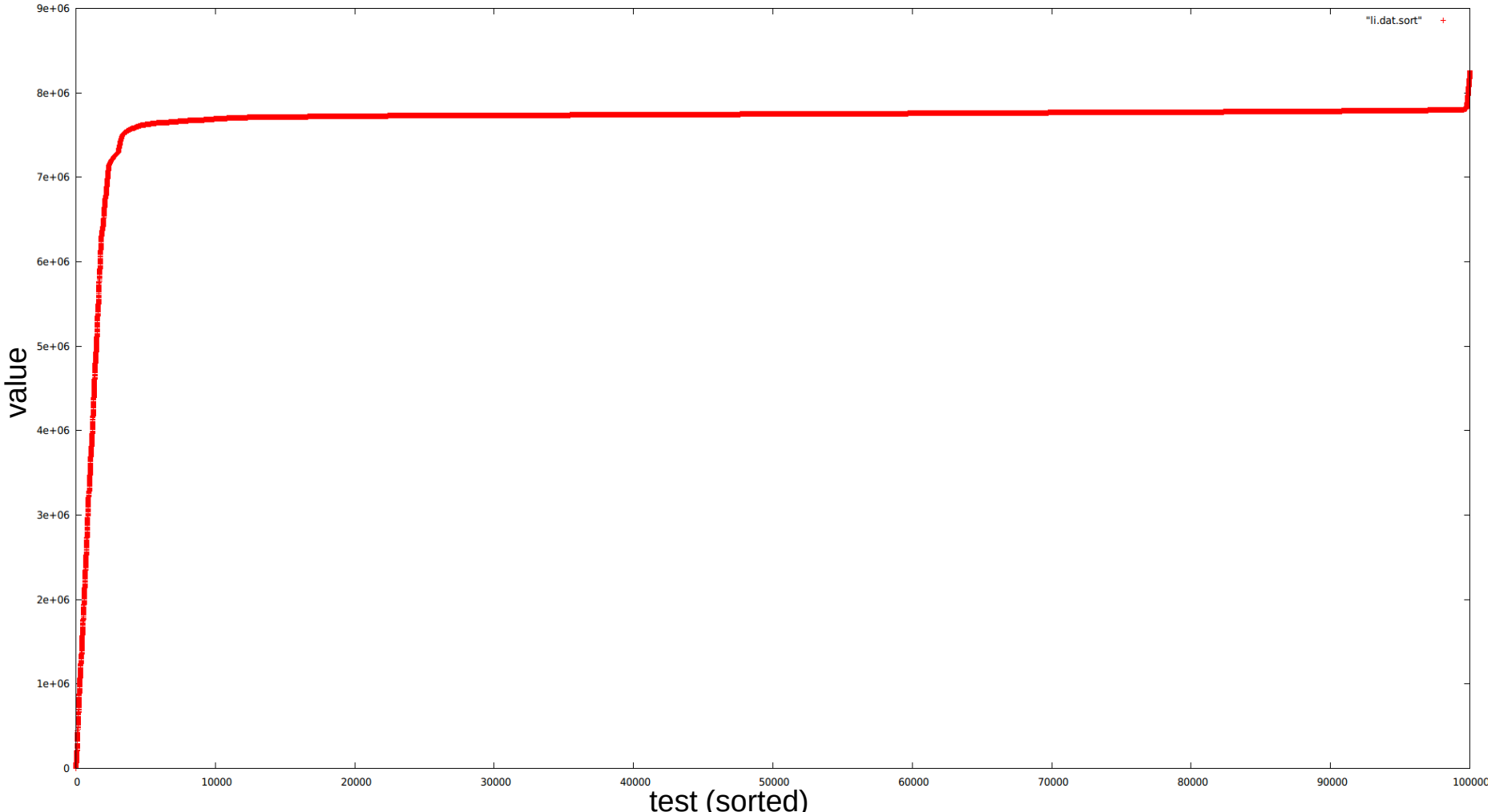
Kris Kaspersky's REP STOS PRNG

VIA Nano X2 U4025



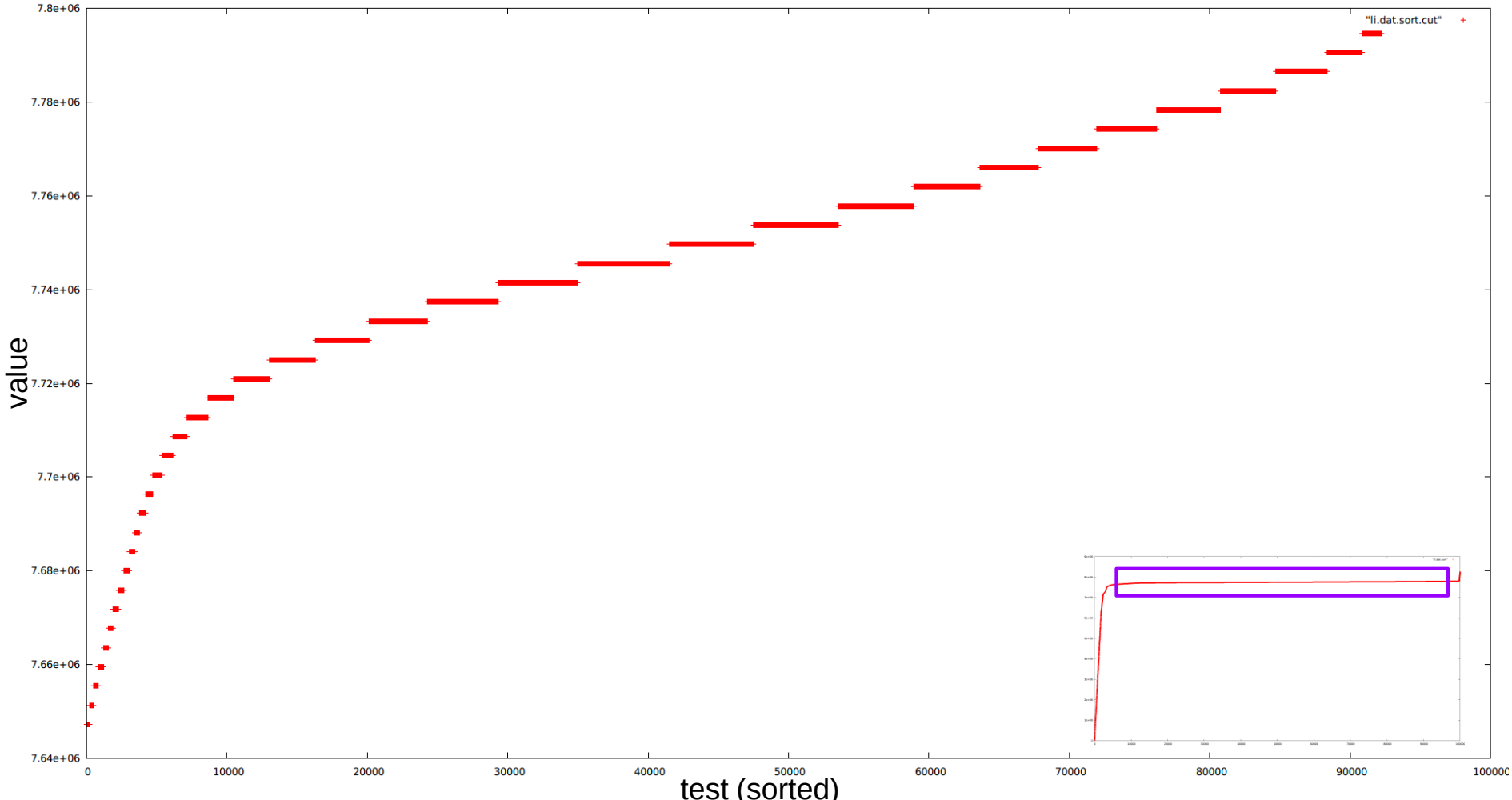
Kris Kaspersky's REP STOS PRNG

VIA Nano X2 U4025



Kris Kaspersky's REP STOS PRNG

VIA Nano X2 U4025



Kris Kaspersky's REP STOS PRNG

Read more on Kris' blog:

- http://nezumi-lab.org/blog/?p=136
- http://nezumi-lab.org/blog/?p=120

This trick no longer works on Intel Core i3/i5/i7
(aka the prefetch input queue bug seems to be fixed)

Machines in the machine

- **Return Oriented Programming** – everyone know it at this point – `ESP` / `RSP` becomes your `EIP` / `RIP`, and you re-use code that's already in memory.
  - initially by Solar Designer (1997)
    http://seclists.org/bugtraq/1997/Aug/63
  - more good stuff published later:
    http://cseweb.ucsd.edu/~hovav/papers/s07.html
    http://cseweb.ucsd.edu/~hovav/talks/blackhat08.html

Machines in the machine

- **Page Fault Liberation Army** - a trap-based 1-instruction VM
  - by Sergey Bratus and Julian Bangert
  - uses `#PF` / `#DF`, `TSS` mapped over `GDT`, `TSS` over page boundaries, etc; so crazy it's awesome ☺

http://conference.hitb.org/hitbseccconf2013ams/materials/D1T1%20-%20Sergey%20Bratus%20and%20Julian%20Bangert%20-%20Better%20Security%20Through%20Creative%20x86%20Trapping.pdf

Extending time windows for local kernel race condition exploitation

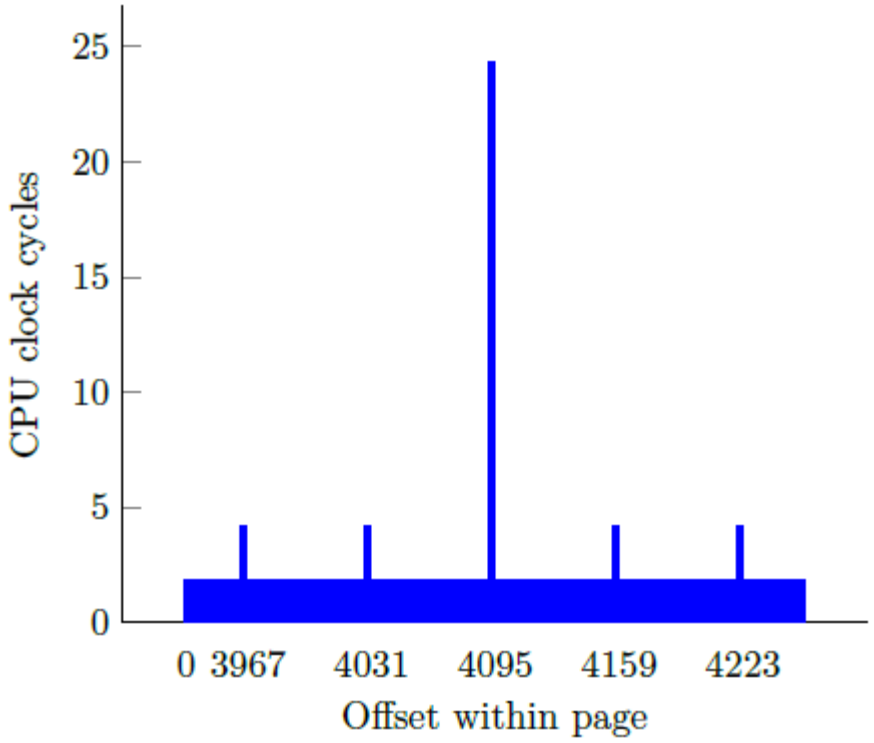
```
mov eax, [ecx]
```

- ECX is a controlled user-mode pointer.
  - points to cached memory, for simplicity.
- How to slow this down?
  - on Windows, but applicable anywhere.

Page boundaries

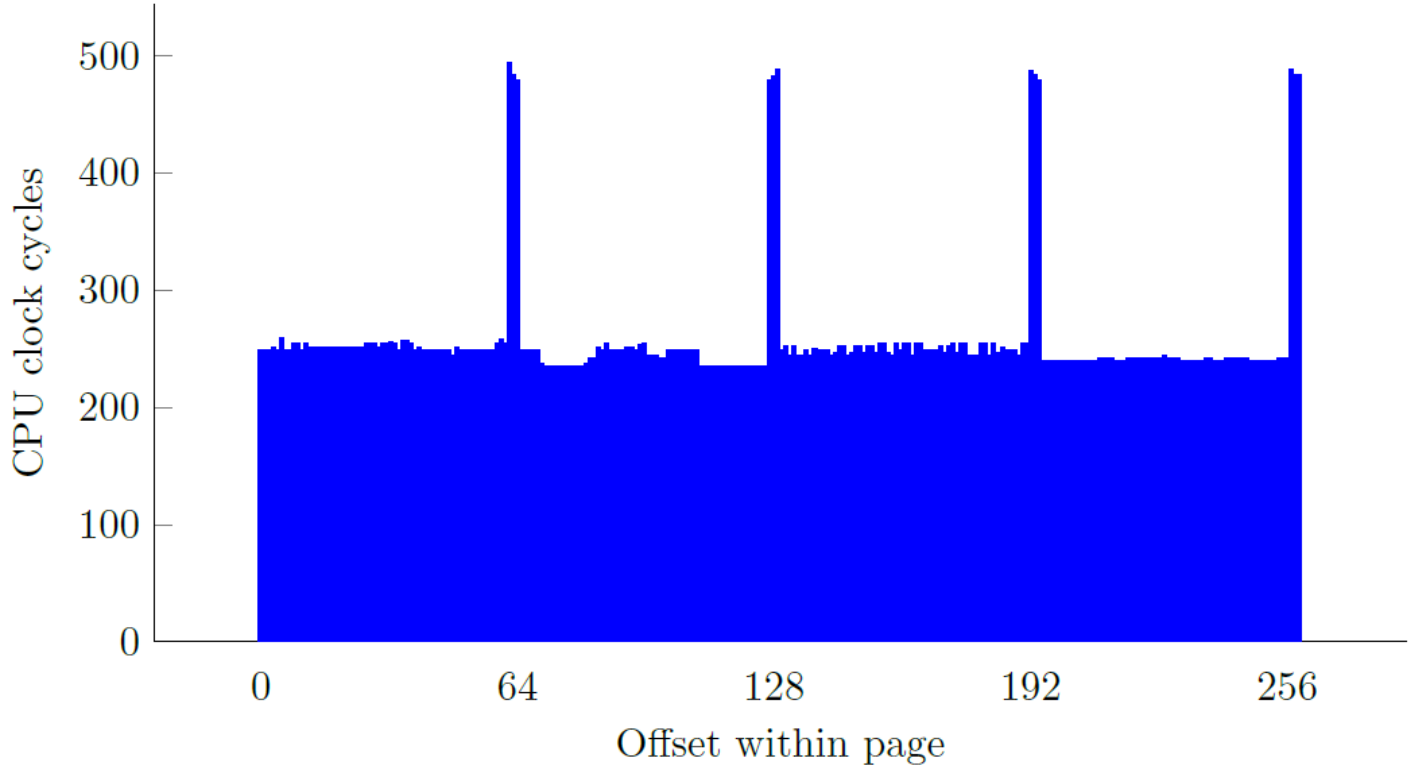
We've used this configuration for benchmarks everywhere (unless specified otherwise).

Test configuration: Intel i7-3930K @ 3.20GHz, DDR3 RAM CL9 @ 1333 MHz

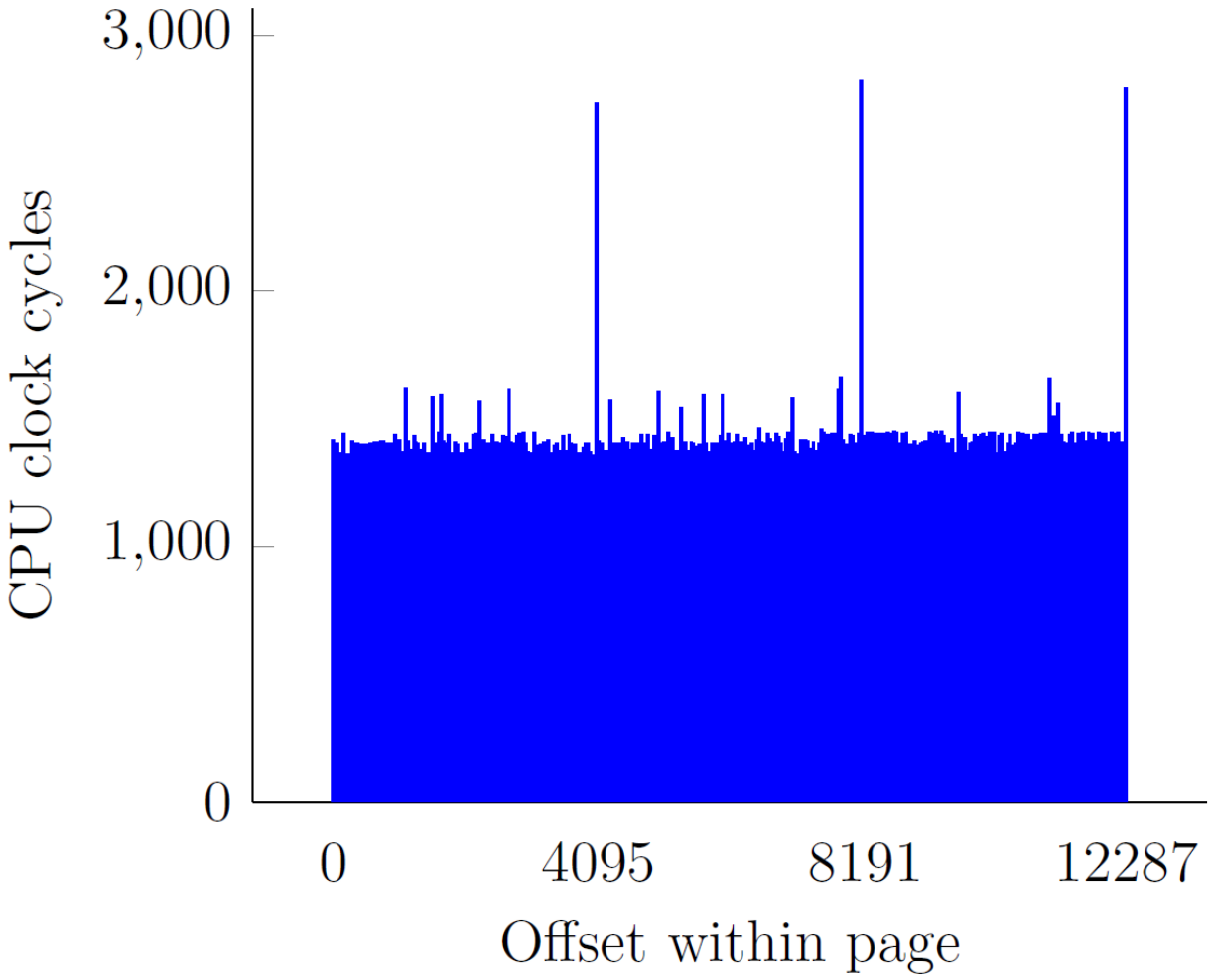


Disabling page cacheability

Visible RAM boundaries



TLB Flushing



More on advanced Windows kernel race condition exploitation

SySCan 2013 slides

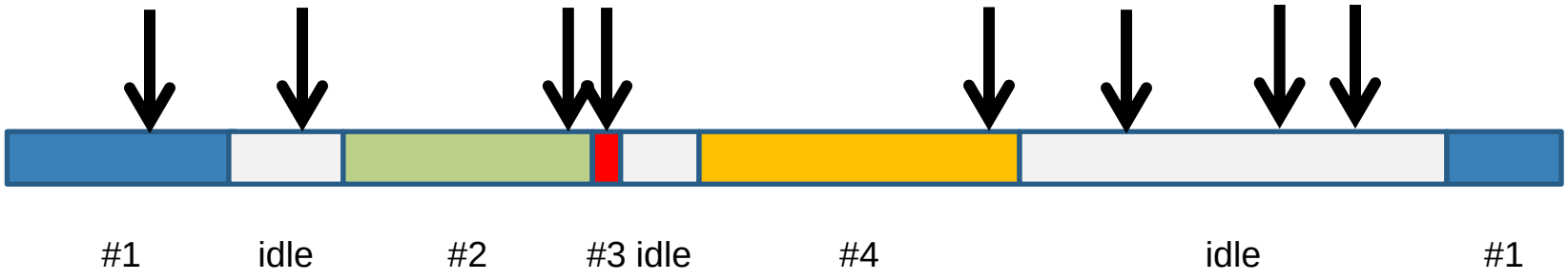
Bochspwn: Exploiting Kernel Race Conditions Found via Memory Access Patterns

SySCan 2013 whitepaper

Identifying and Exploiting Windows Kernel Race Conditions via Memory Access Patterns

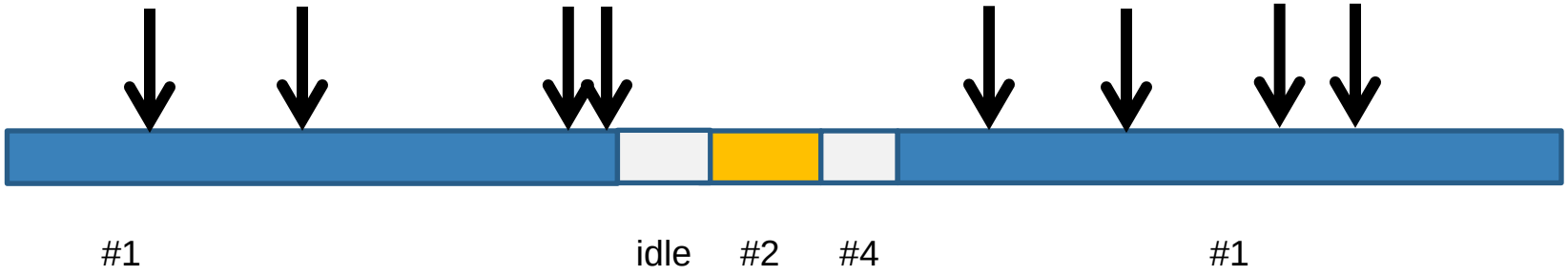
Spraying CPU time

- Hardware interrupts occur randomly during operating system execution.
  - at random times = inside of random thread contexts
  - they use kernel stack on top of the active thread's one.



Spraying CPU time

- Provided a stack-based memory disclosure primitive (e.g. buggy driver), we can read the interrupt's private data.
  - by taking 99% of available CPU time, most interrupts end up preempting our thread.
  - makes it possible e.g. to sniff PS/2 keyboard presses with high granularity.



AMD undocumented MSR

- In October 2010, a *Czernobyl* guy posted information about four undocumented, password protected AMD MSR registers.
  - `C0011024` (Control), `C0011025` (DataMatch), `C0011026` (DataMask), `C0011027` (AddressMask)
- Enabled extended debugging functionality.
  - Changes the semantics of part of the DR0 register.
  - Allows for more general types of hardware breakpoints.
    - i.e. matching bitmasks, not specific addresses

AMD undocumented MSR

- No local security impact, just an extension of existing functionality.
  - according to AMD
- Despite initial fuss, no significant progress made in further investigation.

References

http://www.woodmann.com/forum/showthread.php?13891-AMD-processors-quot-undocumented-quot-debugging-features-and-MSRs-%28DbgCtlMSR2-amp-al.%29

http://www.woodmann.com/collaborative/tools/index.php/AMD_dbg

Undocumented opcode: 0xF1

D	Shift Grp 2[1A]		
	Eb, 1	Ev, 1	Eb, C
E	LOOPNE[f64]/ LOOPNZ[f64] Jb	LOOPE[f64]/ LOOPZ[f64] Jb	LOOP[f] Jb
F	LOCK (Prefix)		REPN XACQU (Prefi

Intel Manual vol 2

0F1h... not defined?

Undocumented opcode: 0xF1 - INT1

	Eb, 1	Ev, 1	Eb, CL	
E	LOOPNE/NZ Jb	LOOPE/Z Jb	LOOP Jb	
F	LOCK Prefix	INT1	REPNE Prefix	F

AMD Manual vol. 3

defined in AMD Manuals

```
Program received signal SIGTRAP,
0x00030101 in ?? ()
(gdb) x/1i $eip-1
   0x30100:     icebp
(gdb)
```

RDTSC vs scheduling

- RDTSC can be used to detect beginning of a new time slice.
  - potentially used in very peculiar types of race conditions.

Algorithm

```
while 1:
    call RDTSC twice
    calculate delta
    if delta > E
        break
    store new value as old value
    continue
```

16-bit BSWAP, (un)documented

BSWAP

- 32 / 64-bit instruction introduced in 486, that does a LE↔BE swap. For example:

```
mov eax, 0x01020304
bswap eax
; eax = 0x04030201
```

- BSWAP overriden to operate on a 16-bit argument (prefix 66H) is undefined according to Intel.

16-bit BSWAP, (un)documented

So... let's do a 16-bit BSWAP!

```
mov eax, 01020304h
db 66h ; operand-size override
bswap eax ; ends up being bswap ax
```

ax = 0x0000

WTF?

16-bit `BSWAP`, (un)documented

This is explained as:

- AX being zero-extended to 32-bit
- then a normal 32-bit BSWAP happens
  (so the zero-extent ends up in lower 16-bit)
- the result is truncated to 16-bit
- and saved in AX

It's a commonly known behavior (even though "undefined").

Use **`xchg al, ah`** instead.

16-bit `BSWAP`, (un)documented

So... of course, everyone knows that right?

State for 2009:

- DOSBox did a normal `BSWAP EAX` (found by Peter Ferrie)
- So did Bochs and QEMU (found by Gynvael)

Conclusions, takeaway

Final word: x86 seems to be a fountain of interesting research areas.

Keywords: exploitation, exploit mitigation, vm detection, poorly documented functionality, undefined behavior, areas difficult to get right by OS developers.

Go and play with it.

Questions?

@j00ru

http://j00ru.vexillium.org/

j00ru.vx@gmail.com

@gynvael

http://gynvael.coldwind.pl/

gynvael@coldwind.pl

Further reading: HLE & RTM

- ## Transactional Synchronization Extensions

  "The hardware monitors multiple threads for conflicting memory accesses and aborts and rolls back transactions that cannot be successfully completed. Mechanisms are provided for software to detect and handle failed transactions" (via Wikipedia)

  - **Hardware Lock Elision** (HLE)

    `XACQUIRE`, `XRELEASE`

  - **Restricted Transactional Memory** (RTM)

    `XBEGIN`, `XEND`, `XABORT`, `XTEST`

Further reading: HLE & RTM

References

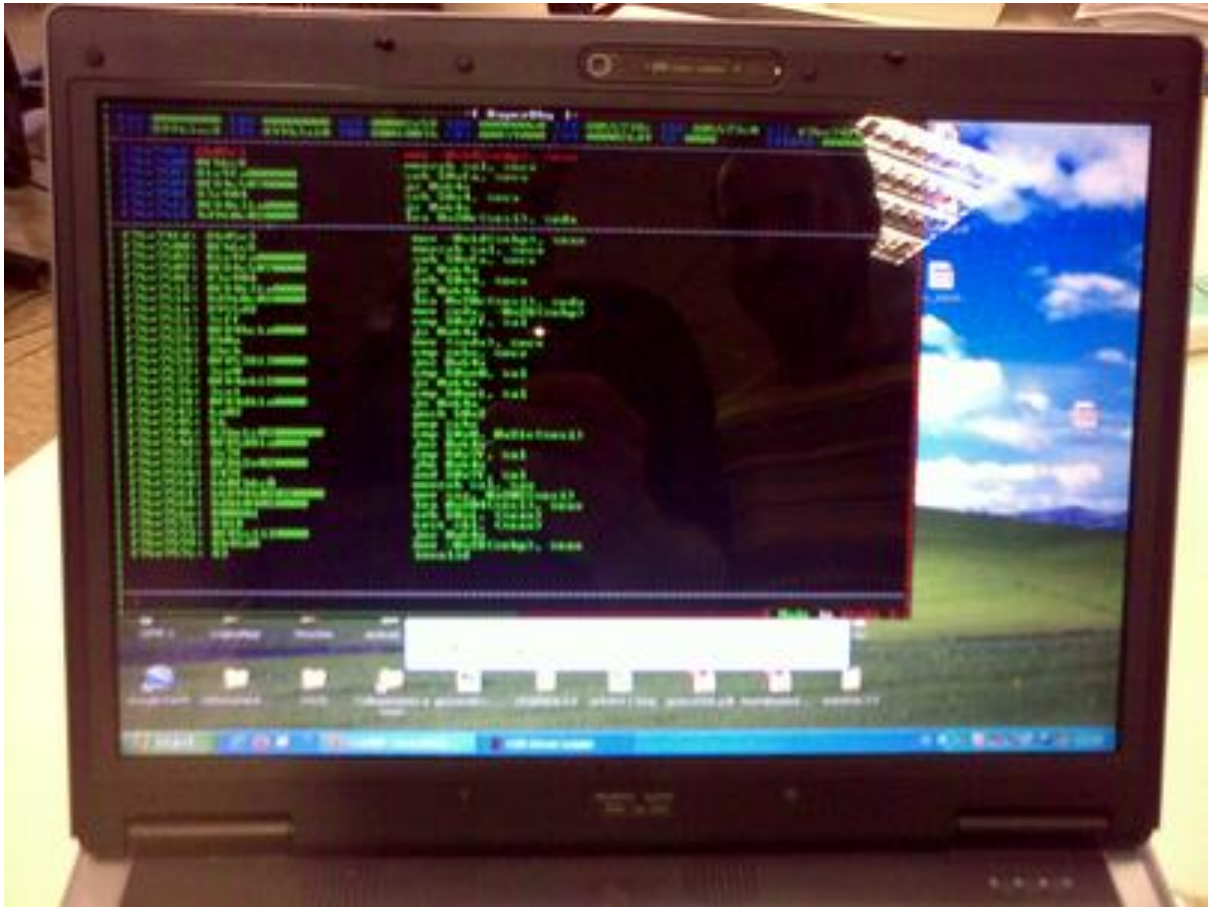
http://software.intel.com/en-us/blogs/2012/02/07/transactional-synchronization-in-haswell

http://halobates.de/adding-lock-elision-to-linux.pdf

http://en.wikipedia.org/wiki/Transactional_Synchronization_Extensions

Further reading: VM-based debugger

https://code.google.com/p/hyperdbg/



Further reading: Performance Counters

- Gaining a foothold in security research.
- Read more:
  - http://epic.hpi.uni-potsdam.de/pub/Home/TuKLecture2010/Dementiev_Processor__Performance__Counter_Monitoring_by_Roman_Dementiev_14-07-2010.pdf
  - http://developer.amd.com/wordpress/media/2012/10/Basic_Performance_Measurements.pdf

Table B-2. Invalid Instructions in 64-Bit Mode

Mnemonic	Opcode (hex)	Description
AAA	37	ASCII Adjust After Addition
AAD	D5	ASCII Adjust Before Division
AAM	D4	ASCII Adjust After Multiply
AAS	3F	ASCII Adjust After Subtraction
BOUND	62	Check Array Bounds
CALL (far)	9A	Procedure Call Far (far absolute)
DAA	27	Decimal Adjust after Addition
DAS	2F	Decimal Adjust after Subtraction
INTO	CE	Interrupt to Overflow Vector
JMP (far)	EA	Jump Far (absolute)
LDS	C5	Load DS Far Pointer
LES	C4	Load ES Far Pointer
POP DS	1F	Pop Stack into DS Segment
POP ES	07	Pop Stack into ES Segment
POP SS	17	Pop Stack into SS Segment
POPA, POPAD	61	Pop All to GPR Words or Doublewords
PUSH CS	0E	Push CS Segment Selector onto Stack
PUSH DS	1E	Push DS Segment Selector onto Stack
PUSH ES	06	Push ES Segment Selector onto Stack
PUSH SS	16	Push SS Segment Selector onto Stack
PUSHA, PUSHAD	60	Push All to GPR Words or Doublewords
Redundant Grp1	82 /2	Redundant encoding of group1 Eb,Ib opcodes
SALC	D6	Set AL According to CF

AMD Manuals, vol 3

Further reading: CPU bugs

- Real well known CPU bugs:
  - AMD bug found by Matthew Dillon
    http://permalink.gmane.org/gmane.os.dragonfly-bsd.kernel/14471
  - Pentium F00F bug
    http://en.wikipedia.org/wiki/Pentium_F00F_bug
  - Cyrix Coma bug
    http://en.wikipedia.org/wiki/Cyrix_coma_bug

Further reading: CPU bugs

well known CPU bugs

"*Do you know why Intel called Pentium "Pentium" and not 586?*

Because when they executed 486+100 on it they got

585.999999999999."

(Actually it was `FDIV`, but still funny :)

http://en.wikipedia.org/wiki/Pentium_FDIV_bug

Further reading: CPU bugs

Intel & AMD erratas

http://www.intel.com/content/www/us/en/search.html?keyword=specification+update

http://www.intel.com/content/www/us/en/search.html?keyword=errata

http://developer.amd.com/resources/documentation-articles/developer-guides-manuals/

Further reading: CPU bugs

Kris Kaspersky's slides on this topic

http://www.cs.dartmouth.edu/~sergey/cs108/2010/D2T1%20-%20Kris%20Kaspersky%20-%20Remote%20Code%20Execution%20Through%20Intel%20CPU%20Bugs.pdf