These are Not Your Grand Daddy's

CPU Performance Counters

black hat® USA 2015

CPU Hardware Performance Counters for Security

UBM Tech

Your Presenters Today

Nishad Herath – nherath@qualys.com
&
Anders Fogh – anders@protect-software.com

Performance Counters for Security

Performance Counters for Security

- For the x86/x64 architecture
  - Other architectures also likely to varying degrees
- Extremely flexible - thank you Intel!
- Hundreds of different low level events that are almost free
- Many great low level indicators of higher level security relevant "weirdness"
  - For example, use of ROP/JOP/COP/COOP etc.
- Need ring 0 or use (restricted) OS interfaces
- Think: an incredibly flexible low level instrumentation

Our Lineup…

- Introducing PMCs
  - CPU hardware Performance (Monitoring) Counters
- PMCs vs ROP (Return Oriented Programming)
- PMCs vs Rowhammer
- PMCs vs Rootkits
- PMCs vs Cache side channel attacks

Your Presenters: Nishad Herath

- Principal malware technologist at Qualys Inc.
- Architecture, research, technical innovation
- Information systems security space for more than 20 years
- Strategic technical partnerships
- Machine learning and complex adaptive systems
- Reverse engineering traditional martial arts, meditation etc.

Your Presenters: Anders Fogh

- Vice president of engineering and co-founder of Protect Software GmbH
- Innovation, research and leading development teams
- 15 years of professional experience with low level software
- Masters degree in economics
- Malware hobbyist and Intel architecture assembly language developer since '92
- Machine learning, malware and mountaineering

So How Well Do You Perform?

Performance Monitoring Counters

- Intel® 64 and IA-32 Architectures Software Developer's Manual
  - Volume 3 (3A, 3B & 3C): System Programming Guide
    - Chapter 17 – Debug, Branch Profile, TSC and Resource Monitoring Features
    - Chapter 18 – Performance Monitoring
    - Chapter 19 – Performance-monitoring Events
  - Volume 2 (2A, 2B & 2C): Instruction Set Reference, A-Z

    RDPMC, RDTSC, RDTSCP, RDMSR, WRMSR
  - Only 350+ pages ;-)

Performance Monitoring Counters

- Since mid '90s
- Early Intel Pentium processors
  - 2 x PMCs, as MSRs readable with RDMSR in ring 0
- Terje Mathisen reverse engineered EMON
  - *"Pentium Secrets: Undocumented features of the Intel Pentium can give you all the information you need to optimize Pentium code"*

    Byte Magazine, July 1994, Page 191
- Intel Pentium with MMX Technology (P55C)
  - New CPU instruction: RDPMC

    Read Performance Monitoring Counter
  - Also, RDTSC – Read Time Stamp Counter

Performance Monitoring Counters

- Intel P6: Pentium Pro, Pentium II, Pentium II Xeon, Pentium III, Pentium III Xeon
  - 2 x 40-bit performance-monitoring counters
- Intel NetBurst: Pentium 4, Celeron, Pentium D, Celeron D, Pentium Extreme Edition, Pentium 4 Extreme Edition, Xeon (2001 – 2006)
  - 18 x 40-bit performance-monitoring counters
- Intel ® Atom and Intel Core (most)
  - 2 x programmable performance-monitoring counters
  - 3 x fixed-function performance-monitoring counters

Performance Monitoring Counters

- Programmable performance-monitoring counters
  - Occurrence
  - Duration
- Events
  - Some architectural
  - Lot more, model specific, in newer CPU cores
- For example, number of instructions decoded, number of interrupts received, number of cache loads, number of cache misses, number of mis-predicted branches and a lot more
- Individual counters can be set up to monitor different events
- Interrupt the CPU if required

Performance Monitoring Counters

- Relevant control registers (CR0 and CR4)
- In protected mode, MOV to/from control registers only in ring 0

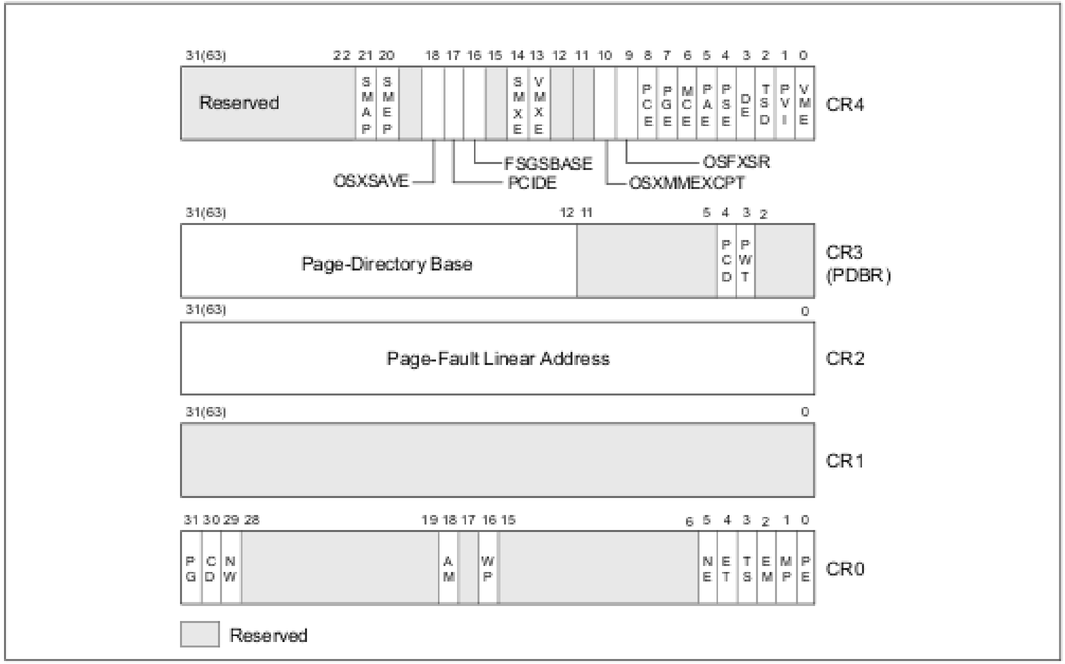


Figure 2-7. Control Registers

Performance Monitoring Counters

- CR0 - Contains system control flags that control operating mode and states of the processor
  - Protection Enable (bit 0 of CR0) enables protected mode when set; enables real-address mode when clear
- CR4 - Contains a group of flags that enable several architectural extensions, and indicate operating system or executive support for specific processor capabilities
  - Performance-Monitoring Counter Enable (bit 8 of CR4) enables execution of the RDPMC instruction for programs or procedures running at any protection level when set; RDPMC instruction can be executed only at protection level 0 when clear

Performance Monitoring Counters

- Time Stamp Disable (bit 2 of CR4) - restricts the execution of the RDTSC instruction to procedures running at privilege level 0 when set; allows RDTSC instruction to be executed at any privilege level when clear. This bit also applies to the RDTSCP instruction if supported

RDPMC

- RDPMC – Read Performance-Monitoring Counter
- Loads the contents of the performance-monitoring counter specified in the ECX register into registers EDX:EAX
- The Pentium 4 and Intel Xeon processors also support "fast" (32-bit) and "slow" (40-bit) reads of the performance counters, selected with bit 31 of the ECX register. If bit 31 is set, the RDPMC instruction reads only the low 32 bits of the selected performance counter; if bit 31 is clear, all 40 bits of the counter are read. A 32-bit read executes faster on a Pentium 4 or Intel Xeon processor than a full 40-bit read
- When in protected or virtual 8086 mode, the performance-monitoring counters enabled (PCE) flag in register CR4 restricts the use of the RDPMC instruction as follows. When the PCE flag is set, the RDPMC instruction can be executed at any privilege level; when the flag is clear, the instruction can only be executed at privilege level 0
- The RDPMC instruction is not a serializing instruction; that is, it does not imply that all the events caused by the preceding instructions have been completed or that events caused by subsequent instructions have not begun. If an exact event count is desired, software must insert a serializing instruction (such as the CPUID instruction) before and/or after the RDPCM instruction
- In the Pentium 4 and Intel Xeon processors, performing back-to-back fast reads are not guaranteed to be monotonic. To guarantee monotonicity on back-to-back reads, a serializing instruction must be placed between the tow RDPMC instructions

RDPMC: x64 (64-bit)

- RDPMC requires an index to specify the offset of the performance-monitoring counter
- In 64-bit mode for Pentium 4 or Intel Xeon processor families, the index is specified in ECX[30:0]
- Count of the performance-monitoring counter is stored in EDX:EAX
  - i.e. RDX[31:0]:RAX[31:0] with RDX[63:32]:RAX[63:32] cleared

RDTSC

- RDTSC – Read Time Stamp Counter
- Count in the time-stamp counter is stored in EDX:EAX
- The time-stamp counter is contained in a 64-bit MSR. The high-order 32 bits of the MSR are loaded into the EDX register, and the low-order 32 bits are loaded into the EAX register. The processor monotonically increments the time-stamp counter MSR every clock cycle and resets it to 0 whenever the processor is reset
- The time-stamp counter can also be read with the RDMSR instruction, when executing at privilege level 0
- The RDTSC instruction is not a serializing instruction. Thus, it does not necessarily wait until all previous instructions have been executed before reading the counter. Similarly, subsequent instructions may begin execution before the read operation is performed

RDTSC: x64 (64-bit)

- Count in the time-stamp counter is stored in EDX:EAX
  - i.e. RDX[31:0]:RAX[31:0] with RDX[63:32]:RAX[63:32] cleared

PMCs: What is Available?

- Use CPUID to verify that the CPU supports any PMC capabilities we wish to use
- CPUID leaf 0x0A (EAX = 0x0A)
  - If the performance monitoring version identifier, EAX[7:0] is greater than zero, then architectural performance monitoring is supported
  - EAX, EBX returns facilities available
    - EAX[15:8] indicates the number of PMCs on the logical CPU
      - For each IA32_PERFEVTSELx MSR, corresponding IA32_PMCx MSR
    - EAX[23:16] indicates the size of the general purpose PMCs in bits
    - EAX[31:24] indicates the length of the EBX (architectural events) bit vector
    - EBX indicates the supported architectural events
    - EDX[4:0] indicates the number of fixed function performance counters per thread
    - EDX[12:5] indicates the size of the fixed function PMCs in bits

PMCs: What is Available?

- For model specific performance events: Intel® 64 and IA-32 Architectures Software Developer's Manual
  - Volume 3 (3A, 3B & 3C): System Programming Guide
    - Chapter 19 – Performance-monitoring Events

PMCs: Setting Up

- After CPU reset:
  - In IA32_PERF_GLOBAL_CTRL MSR
    - All programmable counter global enable bits are reset to 1
  - All other counters and control registers are disabled and cleared/reset to 0
- Setup the PMCs to record events
  - Disable all counting by setting the IA32_PERF_GLOBAL_CTRL to 0, or;

    selectively, reset IA32_PERFEVTSELx.ENABLE bit for any relevant PMCs
  - Then reset the relevant IA32_PMCx counter(s), or;

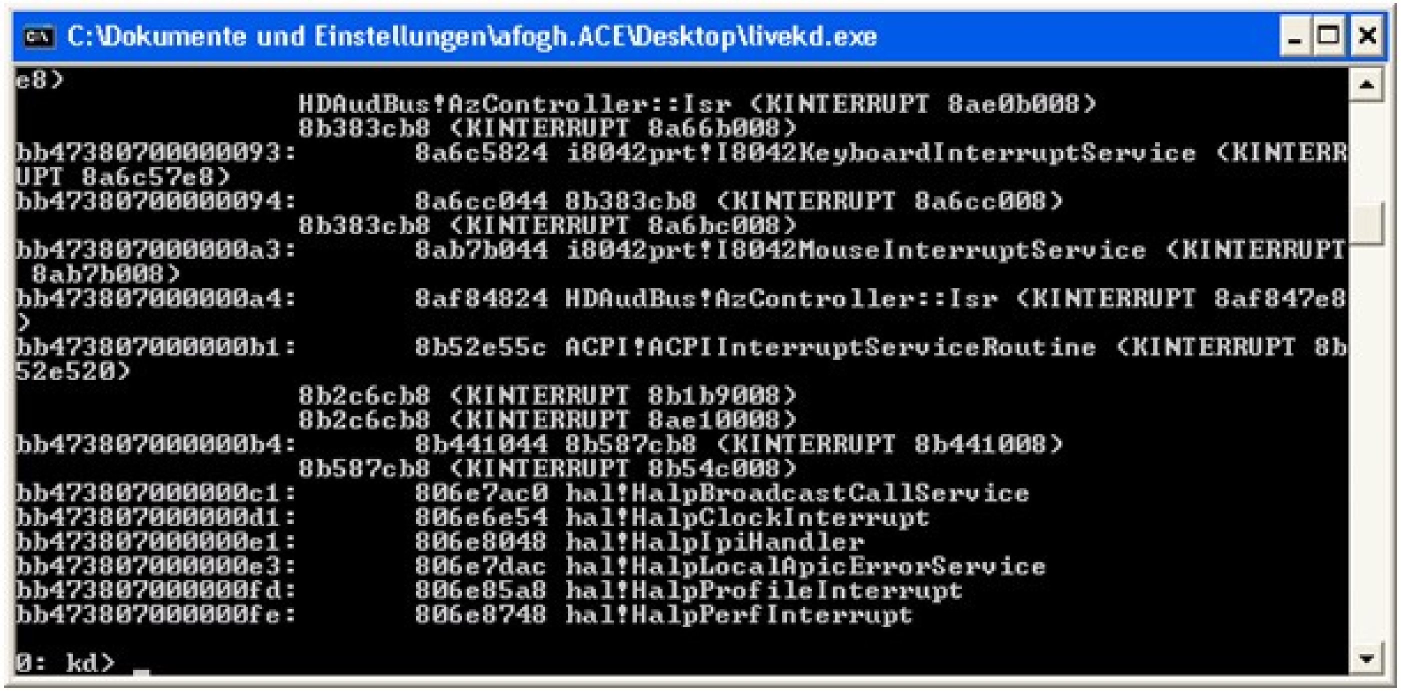
    If using PEBS, set the relevant IA32_PMCx counter(s) to negative y to trigger PMI after y number of events
    - When setting a negative value, setting the lower 32 bits is sufficient. The upper bits will sign extend automatically

PMCs: Setting Up for PEBS

- PEBS (Precise Event Based Sampling) is a special counting mode where PMCs:
  - Can be configured to trigger on PMC overflow
  - Interrupt the processor (PMI – Performance Monitoring Interrupt)
- To use PEBS we to need to hook PMI and setup our own ISR
  - Most operating systems provide functionality for this
    - Undocumented HAL function on Windows _HalpSetSystemInformation()
  - Alternatively:
    - Setup the appropriate Local Vector Table entry of the APIC
    - Hook the IDT entry of each CPU accordingly

PMCs: Setting Up for PEBS

- For PMI in Windows: Vector 0xfe, ISR hal!HalpPerfInterrupt
- Brute force?



C:\Dokumente und Einstellungen\afogh.ACE\Desktop\livekd.exe

```
e8)
                    HDAudBus!AzController::Isr (KINTERRUPT 8ae0b008)
                    8b383cb8 (KINTERRUPT 8a66b008)
bb47380700000093:        8a6c5824 i8042prt!I8042KeyboardInterruptService (KINTERR
UPT 8a6c57e8)
bb47380700000094:        8a6cc044 8b383cb8 (KINTERRUPT 8a6cc008)
                    8b383cb8 (KINTERRUPT 8a6bc008)
bb473807000000a3:        8ab7b044 i8042prt!I8042MouseInterruptService (KINTERRUPT
 8ab7b008)
bb473807000000a4:        8af84824 HDAudBus!AzController::Isr (KINTERRUPT 8af847e8
)
bb473807000000b1:        8b52e55c ACPI!ACPIInterruptServiceRoutine (KINTERRUPT 8b
52e520)
                    8b2c6cb8 (KINTERRUPT 8b1b9008)
                    8b2c6cb8 (KINTERRUPT 8ae10008)
bb473807000000b4:        8b441044 8b587cb8 (KINTERRUPT 8b441008)
                    8b587cb8 (KINTERRUPT 8b54c008)
bb473807000000c1:        806e7ac0 hal!HalpBroadcastCallService
bb473807000000d1:        806e6e54 hal!HalpClockInterrupt
bb473807000000e1:        806e8048 hal!HalpIpiHandler
bb473807000000e3:        806e7dac hal!HalpLocalApicErrorService
bb473807000000fd:        806e85a8 hal!HalpProfileInterrupt
bb473807000000fe:        806e8748 hal!HalpPerfInterrupt

0: kd> _
```

PMCs: Setting Up

- Setup the PMCs to record events
  - Setup the relevant IA32_PERFEVTSELx register(s)
  - Event select and unit mask can be found in the Intel documentation

PMCs: Recording Events

- Start recording events:
  - by setting IA32_PERFEVTSELx.ENABLE bit for any relevant PMCs, and/or;
  - by setting the status in IA32_PERF_GLOBAL_CTRL

PMCs: Shutting Down

- When done recording events
  - Set IA32_PERF_GLOBAL_CTRL to 0
  - Then all relevant IA32_PMCx counters and associated IA32_PERFEVTSELx registers to 0
- Clean up the PMI interrupt hook if PEBS was used

Performance of PMCs

- Monitoring without interrupts is almost free!
  - In our tests, around 0.3% performance penalty
- With interrupts:
  - Incidence count
  - Duration of the interrupt
- Basic cost for usermode interrupt is around 530ns (upper limit)
  - Event processing cost is in addition

Performance of PMCs

- Incidence counts for events differ widely. For approximately 30 seconds:
  - UOPS_ALL: billions
  - RET_MISS: ~400000 (bad case scenario)
  - LLC_MISS:
    - Rowhammering: ~1,000,000,000
- There is no such thing as a free lunch!

Performance of PMCs

- Minimize performance penalties by limiting scope!
  - Probabilistic detection/mitigation (trigger count on interrupt)

    Sometimes called heuristics ;-)
  - Can be limited to a specific "ring" level – ex: kernel mode only
  - Can be limited to specific processes – ex: process sandboxing
  - Can be triggered at key points by system/process instrumentation
    - Compile time
    - Runtime through existing callbacks, detours and/or indirection hooking
  - Consider using rare PMC events to trigger more costly analysis

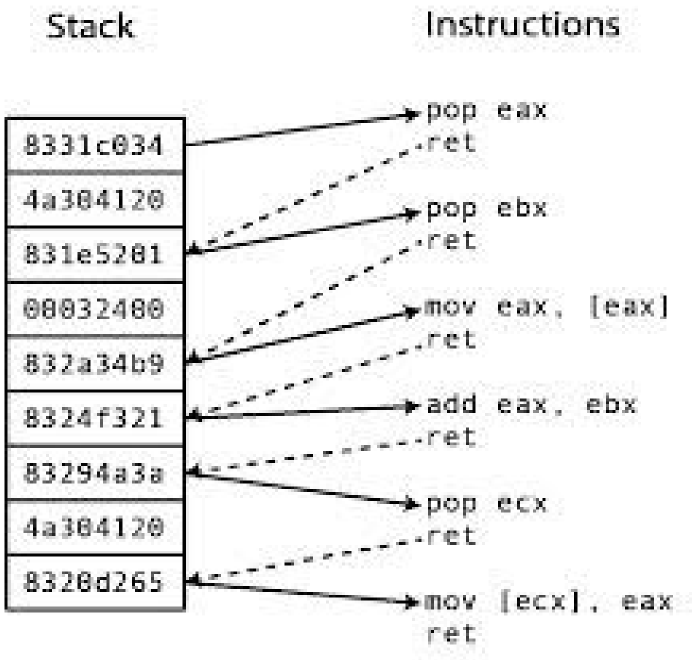
Return Oriented Programming

Are You ROPping In There?

Return Oriented Programming

- Generalized return-to-libc
- Circumvents DEP (Data Execution Prevention) by using existing code
- First line of defense is ASLR (Address Space Layout Randomization)
- ASLR can be defeated in many ways (ex: infoleak bugs)
- Related:
  - JOP – Jump Oriented Programming
  - COP – Call Oriented Programming (indirect CALLs)
  - COOP – Counterfeit Object Oriented Programming

Return Oriented Programming



Stack

8331c034
4a304120
831e5201
08032480
832a34b9
8324f321
83294a3a
4a304120
8320d265

Instructions

```
pop  eax
ret

pop  ebx
ret

mov  eax, [eax]
ret

add  eax, ebx
ret

pop  ecx
ret

mov  [ecx], eax
ret
```

ROP Detection: kBouncer

- *“kBouncer: Efficient and Transparent ROP Mitigation”*
  - Vasilis Pappas (Columbia University)
  - Winner of $200,000 Microsoft BlueHat Prize (2012)
  - http://www.cs.columbia.edu/~vpappas/papers/kbouncer.pdf

ROP Detection: kBouncer

- PoC on Windows 7 Professional SP1
- Use the Last Branch Recording (LBR) branch trace mechanism
- Per process data
- No Windows native system call interception due to PatchGuard
  - Hook key Windows API calls in usermode using Detours
  - Dispatch IOCTL to kernel driver to read LBR

ROP Detection: kBouncer

- *“Transparent ROP Exploit Mitigation using Indirect Branch Tracing”*
  - Vasilis Pappas, Michalis Polychronakis, Angelos D. Keromytis (Columbia University)
  - http://www.cs.columbia.edu/~vpappas/papers/kbouncer.sec13.pdf

ROP Detection

- US patent application: US 20140075556 A1 - *"Threat Detection for Return Oriented Programming"*
  - Georg Wicherski (Crowdstrike Inc.)
  - http://www.google.com/patents/US20140075556
- Ring 0 detection of PMC RET misses

Rowhammer

But Can We Nohammer?

Rowhammer

- [RH1] *“Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors”*
  - Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, Onur Mutlu
  - Carnegie Mellon University, Intel Labs
  - *https://users.ece.cmu.edu/~yoonguk/papers/kim-isca14.pdf*

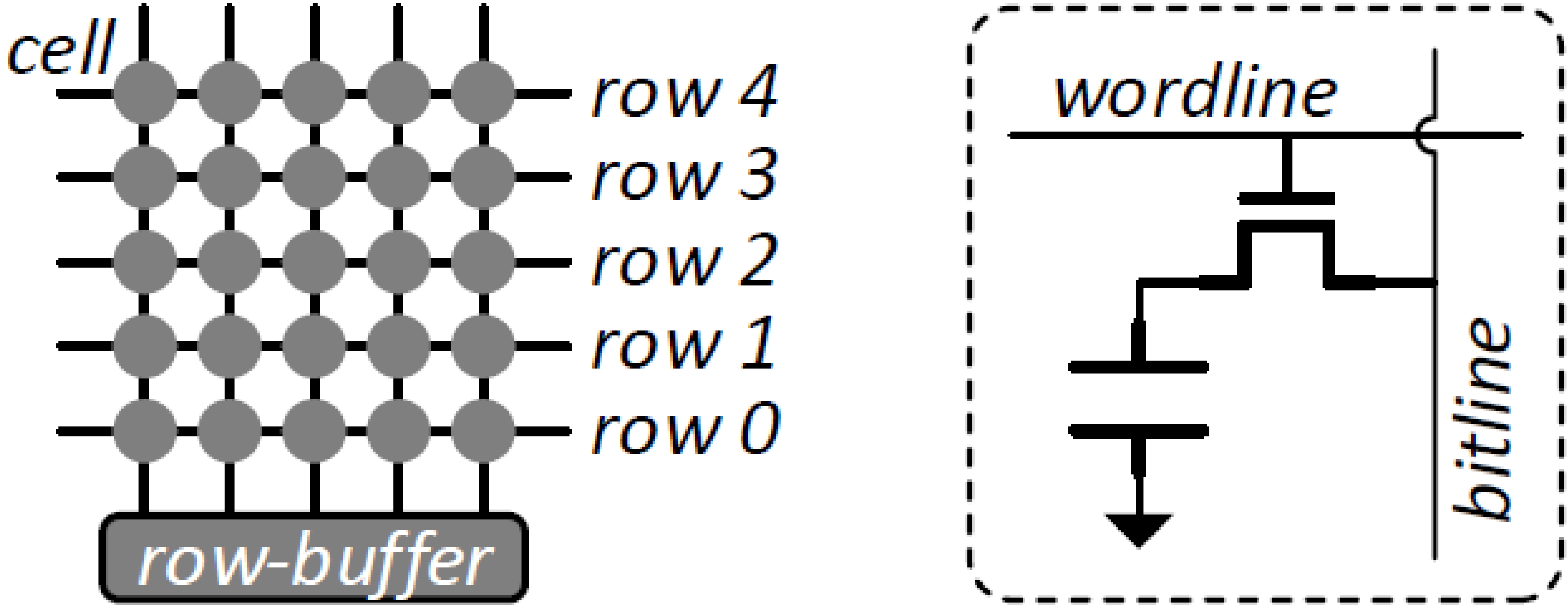
Rowhammer

- [RH2] *“Exploiting the DRAM rowhammer bug to gain kernel privileges”*
  - Mark Seaborn, Thomas Dullien (Google Inc.)
  - http://googleprojectzero.blogspot.com/2015/03/exploiting-dram-rowhammer-bug-to-gain.html

Rowhammer: DDR3 Cells

- On some DDR3 DRAM modules, by repeatedly accessing the same row of memory, it can cause bit flips in adjacent rows - [RH1]
- DRAM: two-dimensional array of DRAM cells
- Each cell
  - Capacitor (stores 1 bit of data)
  - Access transistor (read capacitor state when the wordline is set high)
- When wordline is high, data in the row is transferred to the "row-buffer" and the cells are discharged

Rowhammer: DDR3 Cells



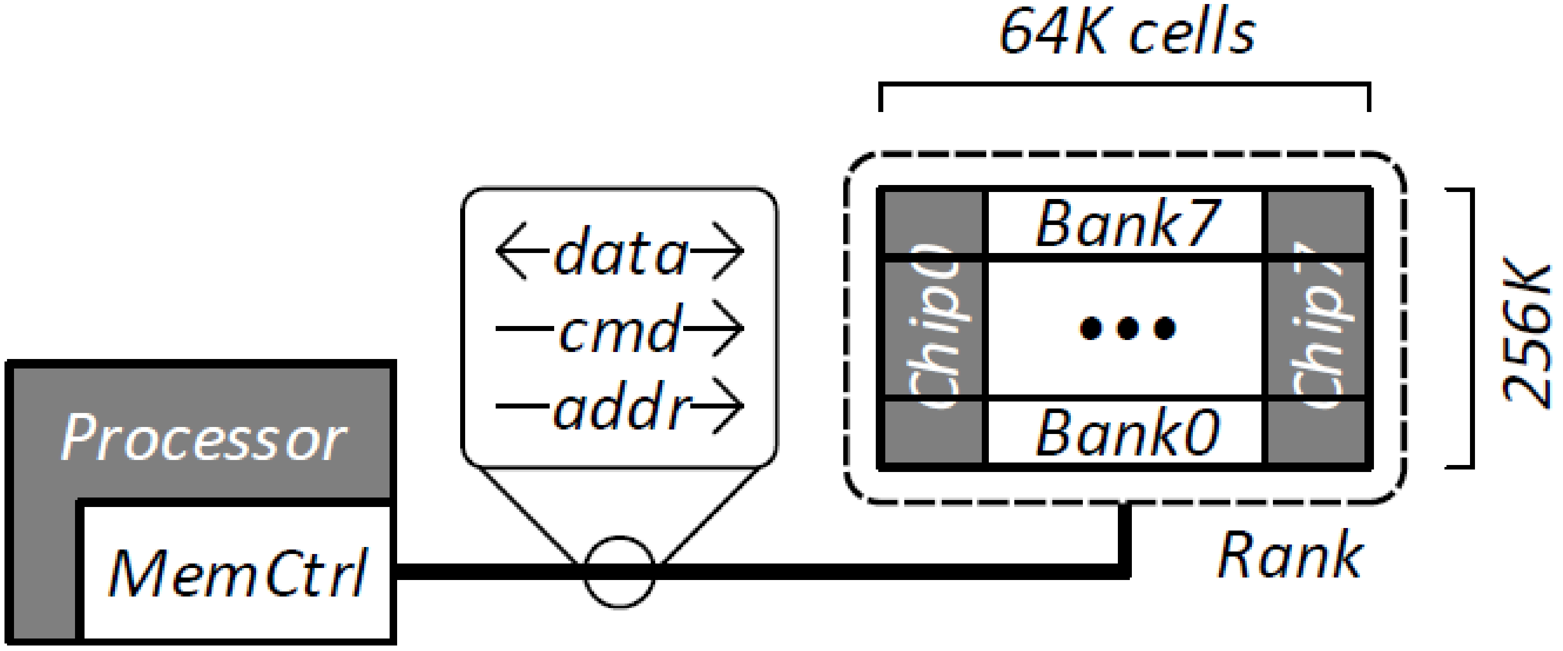
a. Rows of cells

b. A single cell

Rowhammer: DDR3 Organization

- 2GB per rank
- Each rank is 256K rows, vertically partitioned
- 8 banks of 32K rows
  - *"Banks are groups of DRAM chips whose rows are activated in lockstep"* - [RH2]
- 8KB or 64Kb per row
- Row-buffer per bank

Rowhammer: DDR3 Organization



Rowhammer: DDR3 Access

- Memory access:
  - Transfer data from the relevant row into the row-buffer
    - Discharging the cells of the row
  - Access (read or write) row-buffer contents
  - Copy row-buffer contents back into the original cells
    - Recharges the cells of the row

Rowhammer: DDR3

```
code1a:
  mov (X), %eax  // Read from address X
  mov (Y), %ebx  // Read from address Y
  clflush (X)    // Flush cache for address X
  clflush (Y)    // Flush cache for address Y
  jmp code1a
```

- To ensure repeated cell discharge/recharge of a row (rowhammering):
  - Instead of only accessing the row buffer: *“Address selection: For code1a to cause bit flips, addresses X and Y must map to different rows of DRAM in the same bank”* - [RH2]
  - Instead of accessing the same row buffers in 2 banks: *“So if addresses X and Y point to different banks, code1a will just read from those banks’ row buffers without activating rows repeatedly.”* - [RH2]
- Such accessing of a row, thus discharging it and recharging, repeatedly can disturb the charges of cells in adjacent rows - [RH1]
- If done enough times, within the (usually) every 64ms refreshes of the adjacent rows, it can cause bit flips in the adjacent rows.

To Rowhammer…

- Read a LOT from physical memory (DRAM chips)
  - Bypass memory cache(s)
- Read fast!
  - Before DRAM refresh recharges the capacitors
- Know which physical memory bits to flip
  - Knowledge of a security relevant physical memory location
- Two memory locations in the physical vicinity of the above memory
  - *“We have found that we can increase the chances of getting bit flips in row N by row-hammering both of its neighbours (rows N-1 and N+1), rather than by hammering one neighbour and a more-distant row. We dub this 'double-sided hammering'. For many machines, double-sided hammering is the only way of producing bit flips in reasonable time.”* - [RH2]
- Vulnerable memory (DRAM) modules

Check Out Rowhammer!

- *“We tested a selection of x86 laptops that were readily available to us (all with non-ECC memory) using CLFLUSH with the 'random address selection' approach above. We found that a large subset of these machines exhibited rowhammer-induced bit flips.”* - [RH2]
  - https://github.com/google/rowhammer-test
- Memory tester for Rowhammer (built on top of Memtest86+ v5.01)
  - https://github.com/CMU-SAFARI/rowhammer

Rowhammer → Nohammer

- *"This vulnerability exists within hardware and cannot be mitigated by just upgrading software."*
  - *http://blogs.cisco.com/security/mitigations-available-for-the-dram-row-hammer-vulnerability*
- But we respectfully disagree!
  - *"It might be possible to detect row hammering attempts using CPUs' performance counters."* - [RH2]

Nohammer: More Refresh?

- *"Two times (2x) refresh – is a mitigation that has been commonly implemented on server based chipsets from Intel since the introduction of Sandy Bridge and is the suggested default. This reduces the row refresh time by the memory controller from 64ms to 32ms and shrinks the potential window for a row hammer, or other gate pass type memory error to be introduced."*
  - *http://blogs.cisco.com/security/mitigations-available-for-the-dram-row-hammer-vulnerability*

Nohammer: More Refresh?

- *"However, frequent refreshes also degrade performance and energy-efficiency. Today's modules already spend 1.4–4.5% of their time just performing refreshes [34]. This number would increase to 11.0–35.0% if the refresh interval is shortened to 8.2ms, which is required… "* - [RH1]
- *"Such a high overhead is unlikely to be acceptable for many systems."*

  - [RH1]
- *"…which shows that, for some DRAM modules, a refresh period of 32ms is not short enough to reduce the error rate to zero."* - [RH2]
- But 32ms might be good enough to rule out javascript rowhammer attacks?

Nohammer: More Refresh?

- Increased power consumption
- More frequent refreshing may reduce memory throughput
  - Refreshed rows are not accessible during the refresh operation
  - Guesstimate: ~2% performance loss on memory going from current 64ms to 32ms.

Nohammer: "*PARA*" - [RH1]

- *"Our main proposal to prevent DRAM disturbance errors is a low-overhead mechanism called PARA (probabilistic adjacent row activation)."*
- *"The key idea of PARA is simple: every time a row is opened and closed, one of its adjacent rows is also opened (i.e., refreshed) with some low probability. If one particular row happens to be opened and closed repeatedly, then it is statistically certain that the row's adjacent rows will eventually be opened as well."*
- *"The main advantage of PARA is that it is stateless."*
- *"PARA does not require expensive hardware data-structures to count the number of times that rows have been opened or to store the addresses of the aggressor/victim rows."*

Nohammer: pTRR

- *"Pseudo Target Row Refresh (pTRR) – available in modern memory and chipsets. pTRR does not introduce any performance and power impact."*
  - http://blogs.cisco.com/security/mitigations-available-for-the-dram-row-hammer-vulnerability
- *"Server-based chipsets starting with the Intel Ivy Bridge (IVB) chipset provide support for pTRR."*
  - http://blogs.cisco.com/security/mitigations-available-for-the-dram-row-hammer-vulnerability

Nohammer: LPDDR4

- JEDEC’s LPDDR4 standard for DRAM (JEDEC JESD209-4)
  - Targeted Row Refresh (TRR) mode - memory controller can tell a DRAM device to refresh adjacent rows
  - Maximum Activation Count (MAC) – number of activations a row can sustain before adjacent rows require refresh
  - http://www.jedec.org/standards-documents/results/jesd209-4
- *“Subsequently, Haswell (HSW) and Broadwell (BSW) server chipsets from Intel also included support for the Joint Electron Design Engineering Council (JEDEC) Targeted Row Refresh (TRR) algorithm. The TRR is an improved version of the previously implemented pTRR algorithm.”*
  - http://blogs.cisco.com/security/mitigations-available-for-the-dram-row-hammer-vulnerability

Nohammer: Hammerproof!

- *"We found that at least one DRAM vendor indicates, in their public data sheets, that they implement rowhammer mitigations internally within a DRAM device, requiring no special memory controller support."* - [RH2]

Nohammer: No CLFLUSH?

- *"Our proof-of-concept exploits use the x86 CLFLUSH instruction, because it's the easiest way to force memory accesses to be sent to the underlying DRAM and thus cause row hammering."* - [RH2]
- *"We have changed NaCl's x86 validator to disallow CLFLUSH."* - [RH2]
- Requires (dynamic) binary inspection

Nohammer: No CLFLUSH?

- Today, an OS can't feasibly enforce a CLFLUSH ban
- *"The fact that CLFLUSH is usable from unprivileged code is surprising, because the number of legitimate uses for it outside of a kernel or device driver is probably very small."* - [RH2]
  - Maybe Intel should make CLFLUSH ring 0 only?
- Maybe Intel could make CLFLUSH cause a VMEXIT, so that a VMM can deal with it?

Nohammer: No CLFLUSH?

- *"However, there might be ways of causing row hammering without CLFLUSH, which might work on non-x86 architectures too:"* - [RH2]
  - Possible to cause enough cache misses in multi-level caches in the right pattern without CLFLUSH? *"If this is possible, it would be a serious problem, because it might be possible to generate bit flips from JavaScript code on the open web, perhaps via JavaScript typed arrays."* - [RH2]
  - *"Non-temporal memory accesses: On x86, these include non-temporal stores (MOVNTI, MOVNTQ, MOVNTDQ(A), MOVNTPD, MOVNTSD and MOVNTSS) and non-temporals reads (via prefetches — PREFETCHNTA)."* - [RH2]
  - Abuse of OS supported, uncached memory page mappings

Nohammer: No CLFLUSH?

- Other ways to flush the cache
- Probably fast enough for Rowhammering!
  - Ex: *“The Spy in the Sandbox -- Practical Cache Attacks in Javascript”*
    - http://arxiv.org/abs/1502.07373

Nohammer: ECC DRAM?

- ECC DRAM does not detect all multiple bit errors :-(

Nohammer: No Mappings For You!

- Make no information available from usermode about where virtual memory is physically mapped to
- We like this one!
- We know from the KASLR story, infoleaks do happen!
- PTEs are not the only attack surface by a long shot

Nohammer: PMCs FTW!

- *"In order to hammer an area of DRAM effectively, an attacker must generate a large number of accesses to the underlying DRAM in a short amount of time. Whether this is done using CLFLUSH or using only normal memory accesses, it will generate a large number of cache misses."* - [RH2]
- We can detect rowhammer using PMCs by observing:
  - High numbers of LLC miss (architectural) events within short periods of time (refresh size intervals)

Nohammer: Detection PMC Way

- Can running without the interrupt set (for low overhead)
  - Look for peaks of LLC misses within refresh intervals (64ms)
- Can be applied locally (to minimize performance costs)
  - Sandbox only; or,
  - Per relevant processes; or,
  - User mode only
- Potential false positives
  - Ex: Motion search in video encoding prone to make lots of cache misses
- Exploit might be successful
  - We can only signal user
  - Possibly shut down the attempt

Nohammer: Mitigation PMC Way

- Can also be applied locally (to minimize performance costs)
  - Sandbox only; or,
  - Per relevant processes; or,
  - User mode only
- Two main mitigation methods
  - Blind mitigation
  - Mitigation through identification

Nohammer: Mitigation PMC Way

- Blind mitigation (through delay)
- Interrupt on PMC LLC misses.
  - If too many LLC misses too fast, cause a delay to delay the next hammer beyond the current refresh interval
- False positives remain, but only causes a slowdown
- Exploit cannot be successful, if implemented right

Nohammer: Mitigation PMC Way

- Mitigation through identification
- Interrupt on PMC LLC misses
- Analyze opcodes before interrupt for memory address
  - If too many on same memory addresses we got row hammer
  - Or if we see CLFLUSH
- Difficult to implement due to variability in attack code considering CLFLUSH may not be needed

Nohammer: Mitigation PMC Way

- Disrupt :
  - Read data at the victim memory address, bypassing any caching
    - to refresh the victim row
  - Delay (same as with blind mitigation)
  - Terminate the offending process
- False positives quite unlikely

Rootkits

Got Rootkit?

Rootkit Detection Using PMCs

- Any code that executes triggers PMCs
  - Rootkits execute code
- With interrupt on PMC overflow we get EIP/RIP
- If EIP/RIP not in a known module we got a suspect
- We solve the problem of differentiating between data and code since we get EIP/RIP
- Suspect can be dealt with by using standard methods

Rootkit Detection Using PMCs

- PMC that triggers on any code - Ex: UOPS_ALL
- Consider L2 Cache miss
  - Much less L2 Cache misses – especially since repeat interrupts in the same loop less likely
  - Virtually impossible to code anything without incurring L2 Misses
  - Relative performance penalty is smaller on L2 misses than on other the average instruction
  - To minimize performance penalty we only record EIP/RIP and do further processing in a separate thread

Rootkit Detection Using PMCs

- Problems with false negatives
  - We could identify any code execution outside of code sections in any privilege level but for performance reasons ring 0 seems like a good limit
  - Regardless we must tune between performance and false negatives using the counter
  - Only probabilistic detection is possible
    - However, unlikely code will survive a year without detection using this method
    - Critical areas (ex: TrueType font processing) could be instrumented to be more rigorously checked, without too much fall out on entire system
    - Example: Equation group Grayfish uses a hash 1000 times to find a key before decrypting more code. We think that might trigger a PMI with the right PMC events PEBS setup

Rootkit Detection Using PMCs

- Problems with false positives
  - A complete known good list of drivers, true type fonts, etc., is strictly required.
    - Feasible only in kernel mode
  - Sometimes developers purposely execute outside code pages (ex: JIT compilers, etc.)
    - Probably rare in kernel mode
  - Traditional analysis can help since we now know EIP/RIP is code:
    - Does code use stealth technology?
    - Does code use "delta offset" offset technology?
    - Known good signatures

Rootkit Detection Using PMCs

- Other problems
  - We are not the only client using PMCs
  - Only takes a few instructions to turn off PMCs
    - However: RDTSC might statistically tell on attackers turning off PMCs.
    - We might get a chance to actually check the MSRs
    - PMC virtualization by a VMM

Cache Side Channel Attacks

Spies!

Cache Side Channel Attacks

- [CS1] *"Cache Template Attacks:Automating Attacks on Inclusive Last-Level Caches"*
  - Daniel Gruss, Raphael Spreitzer and Stefan Mangard
  - Graz University of Technology, Austria
  - https://online.tugraz.at/tug_online/voe_main2.getvolltext?pCurrPk=85809

Cache Side Channel Attacks

- Flush + reload attack:
  - Flush the shared memory with CLFLUSH (or equivalent)
  - Wait
  - Measure the time taken for accessing that memory
    - If the access time is fast enough, then victim placed memory in cache (used the memory)
    - If not, the victim did not use the memory
  - Works well because L3 cache is shared between all cores and hardware threads

Cache Side Channel Attacks

- Library code (ex: user32.dll, msvcrtxx.dll, etc.) is commonly shared between processes
- Page deduplication - pages with the same content can be shared across processes and privilege levels

Cache Side Channel Attacks

- Template attacks
  - Generate a template in a controlled environment: Use flush + reload in controlled environment to map cache activity to information of interest
  - Collect flush + reload cache activity from the victim
  - Compare victim data to template to identify information

Cache Side Channel Attacks

- Template attacks can under the right circumstances
  - Identify keypress by victim: *"We also found leakage of accurate keypress timings in other libraries, such as the ncurses library"* [CS1]
  - AES keys: "*We launched an efficient and automated access-driven attack against the AES T-table implementation of OpenSSL 1.0.2*" [CS1]
  - And probably many many other things
- And there are other offensive uses for cache side channels
  - Ex: Covert channel for information through sandbox:
    - *"The Spy in the Sandbox: Practical Cache Attacks in Javascript"*
    - Yossef Oren, Vasileios P. Kemerlis, Simha Sethumadhavan, Angelos D. Keromytis
    - http://arxiv.org/abs/1502.07373

Cache Side Channel Attack Mitigation

- Remove CLFLUSH instruction (same problems discussed before, in relation to rowhammer mitigation)
- Disable cache line sharing:
  - Avoid shared memory
  - Requires:
    - OS changes and/or;
    - Hardware changes and/or;
    - Significant programming effort
- Using the cache side channel attacks on possible attack vectors to detect attacks
- Improve prefetcher
  - Requires hardware changes

Cache Side Channel Attack Mitigation: PMCs Again!

- High frequency (fine grained) attacks (attacks with a high rate "polling")
  - Detection same as rowhammer
  - If too many L3 cache misses we got a side channel attack
  - For less risk of false positives, limit to:
    - Important process(es)
    - Potential aggressor(s)
    - Ring 3
  - Optionally, Instrument potential attack vectors

Cache Side Channel Attack Mitigation: PMCs Again!

- Low frequency (coarse grained) attacks can be mitigated using L3 cache miss with interrupts. Look for RDTSC or other fine grained timers at EIP/RIP on interrupt in potential attacker.
  - If multiple interrupts near the same EIP/RIP in potential aggressor, we are probably looking at the aggressor probe with negative result
  - Interrupts in victim/sender causes noise in the channel
    - Attacker can mitigate this if he knows he is being monitored
  - Interrupts in the victim can possibly be used to trigger Copy-on-Write.
    - Works only for victim code, and maybe tricky to implement
  - Performance going to be a problem if potential victims are doing lot's of cache misses

Resources

Useful Stuff!

Resources: Manuals

- Intel® 64 and IA-32 Architectures Software Developer Manuals
  - http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html

Resources: Code

- Performance Application Programming Interface
  - http://icl.cs.utk.edu/papi/
- Perfmon2
  - http://perfmon2.sourceforge.net/
- Andi Kleen's pmu-tools
  - https://github.com/andikleen/pmu-tools
- LikwidPerfCtr
  - https://github.com/rrze-likwid/likwid

Questions

Yes, No, Maybe?

Black Hat Sound Bytes

- Unintended(?) security consequences of CPU hardware Performance (Monitoring) Counters!
- Performance (Monitoring) Counters against all R's - ROP, rootkits, Rowhammer!
- Performance Monitoring Interrupt maybe the sweetest of all interrupts!

nherath@qualys.com
anders@protect-software.com