Transparent ROP Detection using CPU Performance Counters

Stones from other hills may serve to polish jade

Xiaoning Li
Intel Labs

Michael Crouse
Harvard University
Agenda

- Performance Monitor Architecture Overview
- Past Research
- Performance Events In Haswell/Silvermont
- Mispredicted Branch Transfer
- Stack Pivoting Detection Overview
- Defense with CPU Performance Counters
- APSA13-2 Case Study
- Misc
Performance Monitor Architecture Overview
Performance Monitor Overview

- Introduced in the Pentium processor
  - A set of model-specific performance-monitoring counter MSRs
- Enhanced to monitor a set of events in Intel P6 family of processors
- Pentium 4 and Intel Xeon processors introduced a new performance monitoring mechanism and new set of performance events
- Architectural/Non-Architectural performance events
Architectural/Non-Architectural performance events

- The performance monitoring mechanisms and performance events defined for the Pentium, P6 family, Pentium 4, and Intel Xeon processors are not architectural.
- Intel Core Solo and Intel Core Duo processors support a set of architectural performance events and a set of non-architectural performance events.
- Processors based on Intel Core/Intel® Atom™ micro architecture support enhanced architectural performance events and non-architectural performance events.
Availability of architectural performance monitoring capabilities can be enumerated using the CPUID.0AH

Non-architectural events for a given micro architecture can not be enumerated using CPUID
CPUID Mechanism

- Number of performance monitoring counters available in a logical processor
- Number of bits supported in each IA32_PMCx
- Number of architectural performance monitoring events supported in a logical processor
- Software can use CPUID to discover architectural performance monitoring availability (CPUID.0AH).
### Architectural Performance Monitoring Leaf

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OAH</td>
<td></td>
</tr>
</tbody>
</table>
| EAX   | Bits 07 - 00: Version ID of architectural performance monitoring  
       | Bits 15 - 08: Number of general-purpose performance monitoring counter per logical processor  
       | Bits 23 - 16: Bit width of general-purpose, performance monitoring counter  
       | Bits 31 - 24: Length of EBX bit vector to enumerate architectural performance monitoring events |
| EBX   | Bit 00: Core cycle event not available if 1  
       | Bit 01: Instruction retired event not available if 1  
       | Bit 02: Reference cycles event not available if 1  
       | Bit 03: Last-level cache reference event not available if 1  
       | Bit 04: Last-level cache misses event not available if 1  
       | Bit 05: Branch instruction retired event not available if 1  
       | Bit 06: Branch mispredict retired event not available if 1  
       | Bits 31 - 07: Reserved = 0 |
| ECX   | Reserved = 0 |
| EDX   | Bits 04 - 00: Number of fixed-function performance counters (if Version ID > 1)  
       | Bits 12 - 05: Bit width of fixed-function performance counters (if Version ID > 1)  
       | Reserved = 0 |
### Architectural Performance Monitoring Versions

<table>
<thead>
<tr>
<th>CPU Architecture</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel Core Solo</td>
<td>1</td>
</tr>
<tr>
<td>Intel Core Duo</td>
<td>1</td>
</tr>
<tr>
<td>Core 2 Duo processor T7700</td>
<td>2</td>
</tr>
<tr>
<td>Core</td>
<td>2</td>
</tr>
<tr>
<td>Atom</td>
<td>2, 3</td>
</tr>
<tr>
<td>Core i7</td>
<td>2, 3</td>
</tr>
</tbody>
</table>
Configure an architectural performance monitoring event with programming performance event select registers.

- Performance event select MSRs (IA32_PERFEVTSELx MSRs).
- Performance monitoring counter (IA32_PMCx MSR)
- Performance monitoring counters are paired with performance monitoring select registers.
Layout of IA32_PERFEVTSELx MSRs

- **Counter Mask (CMASK)**
- **Unit Mask (UMASK)**
- **Event Select**

- **INV**—Invert counter mask
- **EN**—Enable counters
- **INT**—APIC interrupt enable
- **PC**—Pin control
- **E**—Edge detect
- **OS**—Operating system mode
- **USR**—User Mode

Reserved
IA32_PERFEVTSELx MSRs

- Event select field (bits 0 through 7)
- Unit mask (UMASK) field (bits 8 through 15)
- USR (user mode) flag (bit 16)
- OS (operating system mode) flag (bit 17)
- E (edge detect) flag (bit 18)
- PC (pin control) flag (bit 19)
- INT (APIC interrupt enable) flag (bit 20)
- EN (Enable Counters) Flag (bit 22)
- INV (invert) flag (bit 23)
- Counter mask (CMASK) field (bits 24 through 31)
Bits 0 through 4 of CPUID.0AH.EDX indicates the number of fixed-function performance counters available per core

Bits 5 through 12 of CPUID.0AH.EDX indicates the bit-width of fixed-function performance counters. Bits beyond the width of the fixed-function counter are reserved and must be written as zeros.
IA32_FIXED_CTR_CTRL MSR

Event Name | Fixed-Function PMC | PMC Address
---|---|---
INST_RETIRED.ANY | MSR_PERF_FIXED_CTR0/IA32_FIXED_CTR0 | 309H
CPU_CLK_UNHALTED.CORE | MSR_PERF_FIXED_CTR1/IA32_FIXED_CTR1 | 30AH
CPU_CLK_UNHALTED.REF | MSR_PERF_FIXED_CTR2/IA32_FIXED_CTR2 | 30BH
IA32_PERF_GLOBAL_CTRL MSR

63  35  34  33  32  31  2  1  0

IA32_FIXED_CTR2 enable
IA32_FIXED_CTR1 enable
IA32_FIXED_CTR0 enable
IA32_PMC1 enable
IA32_PMC0 enable
Reserved
IA32_PERF_GLOBAL_STATUS MSR

63-62
CondChgd
OvfBuffer
IA32_FIXED_CTR2 Overflow
IA32_FIXED_CTR1 Overflow
IA32_FIXED_CTR0 Overflow
IA32_PMC1 Overflow
IA32_PMC0 Overflow
Reserved
35-34 33 32 31
2-1 0
IA32_PERF_GLOBAL_OVF_CTRL MSR
Architectural Performance Monitoring Version 3

- The number of general-purpose performance counters (IA32_PMCx) is reported in CPUID.0AH:EAX[15:8], the bit width of general-purpose performance counters is reported in CPUID.0AH:EAX[23:16]
- The bit vector representing the set of architectural performance monitoring events supported
MSRs Supporting Architectural Performance Monitoring Version 3
IA32_PERFEVTSELx MSRs Supporting Architectural Performance Monitoring Version 3

- Bit 21 (AnyThread) of IA32_PERFEVTSELx is supported in architectural performance monitoring version 3.
  - When set to 1, it enables counting the associated event conditions (including matching the thread’s CPL with the OS/USR setting of IA32_PERFEVTSELx) occurring across all logical processors sharing a processor core.
  - When bit 21 is 0, the counter only increments the associated event conditions (including matching the thread’s CPL with the OS/USR setting of IA32_PERFEVTSELx) occurring in the logical processor which programmed the IA32_PERFEVTSELx MSR.
IA32_PERF_GLOBAL_CTRL MSR Supporting Architectural Performance Monitoring Version 3
IA32_PERF_GLOBAL_STATUS MSR Supporting Architectural Performance Monitoring Version 3
IA32_PERF_GLOBAL_OVF_CTRL MSR Supporting Architectural Performance Monitoring Version 3
## Pre-defined Architectural Performance Events

<table>
<thead>
<tr>
<th>Bit Position CPUID.AH.EBX</th>
<th>Event Name</th>
<th>UMask</th>
<th>Event Select</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>UnHalted Core Cycles</td>
<td>00H</td>
<td>3CH</td>
</tr>
<tr>
<td>1</td>
<td>Instruction Retired</td>
<td>00H</td>
<td>C0H</td>
</tr>
<tr>
<td>2</td>
<td>UnHalted Reference Cycles</td>
<td>01H</td>
<td>3CH</td>
</tr>
<tr>
<td>3</td>
<td>LLC Reference</td>
<td>4FH</td>
<td>2EH</td>
</tr>
<tr>
<td>4</td>
<td>LLC Misses</td>
<td>41H</td>
<td>2EH</td>
</tr>
<tr>
<td>5</td>
<td>Branch Instruction Retired</td>
<td>00H</td>
<td>C4H</td>
</tr>
<tr>
<td>6</td>
<td>Branch Misses Retired</td>
<td>00H</td>
<td>C5H</td>
</tr>
</tbody>
</table>
Branch Instructions Retired Events

- Branch Instructions Retired — Event select C4H, Umask 00H This event counts branch instructions at retirement. It counts the retirement of the last micro-op of a branch instruction.
- All Branch Mispredict Retired — Event select C5H, Umask 00H
Past Research

- On Linux 2.6.34
- Using Machine Learning with performance counters
- Using PEBS/BTS for branch record
- Mentioned Branch Miss Predict Event


<table>
<thead>
<tr>
<th>Attack Type</th>
<th>Description</th>
<th>PMU Events</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code-injection</td>
<td>Inject code and take control transfer to injected code</td>
<td>Branch Tracing Event(BTS)</td>
</tr>
<tr>
<td>Return-to-libc</td>
<td>Use library calls instead of inject code (e.g., invoke “execve” with “bin/bash”)</td>
<td>Branch Tracing Event(BTS)</td>
</tr>
<tr>
<td>Return-oriented programming</td>
<td>Use instructions before “ret” in existing library and binary code to form shellcode</td>
<td>Branch Tracing Event(BTS)</td>
</tr>
</tbody>
</table>

Table 1: Deviation in performance characteristics of common attacks.
**CFIMon:** Detecting violation of control flow integrity using performance counters in Dependable Systems and Networks (DSN), 2012

- On Linux
- CFI defense with PMU events
- Using PEBS/BTS for branch record
- Target all branch
- Using call_set/Ret_set policy

Source: http://ipads.se.sjtu.edu.cn/_media/publications:cfimon.pdf

<table>
<thead>
<tr>
<th>Branch Type</th>
<th>Branch Example</th>
<th>Target Instruction</th>
<th>Target Set</th>
<th>In Binary</th>
<th>Run-time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct call</td>
<td>callq 34df0 &lt;abort&gt;</td>
<td>1: taken</td>
<td>/</td>
<td>16.8%</td>
<td>14.5%</td>
</tr>
<tr>
<td>Direct jump</td>
<td>jnz c2ef0 &lt;__write&gt;</td>
<td>1 or 2: taken or fallthrough</td>
<td>/</td>
<td>74.3%</td>
<td>0.8%</td>
</tr>
<tr>
<td>Return</td>
<td>retq</td>
<td>Limited: insn. next to a call</td>
<td>ret_set</td>
<td>6.3%</td>
<td>16.3%</td>
</tr>
<tr>
<td>Indirect call</td>
<td>callq *%rax</td>
<td>Limited: 1st insn. of a function</td>
<td>call_set</td>
<td>2.1%</td>
<td>0.2%</td>
</tr>
<tr>
<td>Indirect jump</td>
<td>jmpq *%rdx</td>
<td>Unlimited: potentially any insn.</td>
<td>train_set</td>
<td>0.5%</td>
<td>68.3%</td>
</tr>
</tbody>
</table>

**TABLE 1.** Branch Classification. The distribution is from Apache and libraries it uses.
Taming the ROPe on Sandy Bridge in SYSCAN 2013

- On Linux to detect Ring 0 ROP on SandyBridge
- Use BR_MISP_EXEC to catch RET misprediction event to detect ROP
- Using preceding call policy

Source: Taming the ROPe on Sandy Bridge in SYSCAN 2013

- 0x89 — BR_MISP_EXEC.*: mispredicted executed branches
- 0x800 — .RETURN_NEAR: normal, near ret
- 0x8000 — .TAKEN: unconditional branch
HDROP: Detecting ROP Attacks Using Performance Monitoring Counters in ISPEC’14

- On Linux
- Use BR_RET_MISSP_EXEC Event, which is replaced by 0x89
- Expect BR_RET_MISSP_RETIRED, but not existed
Performance Events in Haswell/Silvermont
<table>
<thead>
<tr>
<th>Code</th>
<th>Value</th>
<th>Description</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>88H</td>
<td>01H</td>
<td>BR_INST_EXEC.COND</td>
<td>Qualify conditional near branch instructions executed, but not necessarily retired.</td>
</tr>
<tr>
<td>88H</td>
<td>02H</td>
<td>BR_INST_EXEC.DIRECT_JMP</td>
<td>Qualify all unconditional near branch instructions excluding calls and indirect branches.</td>
</tr>
<tr>
<td>88H</td>
<td>04H</td>
<td>BR_INST_EXEC INDIRECT_JMP NON_CALL_RET</td>
<td>Qualify executed indirect near branch instructions that are not calls nor returns.</td>
</tr>
<tr>
<td>88H</td>
<td>08H</td>
<td>BR_INST_EXEC.RETURN_NEAR</td>
<td>Qualify indirect near branches that have a return mnemonic.</td>
</tr>
<tr>
<td>88H</td>
<td>10H</td>
<td>BR_INST_EXEC.DIRECT_NEAR_CALL</td>
<td>Qualify unconditional near call branch instructions, excluding non call branch, executed.</td>
</tr>
<tr>
<td>88H</td>
<td>20H</td>
<td>BR_INST_EXEC INDIRECT_NEAR_CALL</td>
<td>Qualify indirect near calls, including both register and memory indirect, executed.</td>
</tr>
<tr>
<td>88H</td>
<td>40H</td>
<td>BR_INST_EXEC.NONTAKEN</td>
<td>Qualify non-taken near branches executed.</td>
</tr>
<tr>
<td>88H</td>
<td>80H</td>
<td>BR_INST_EXEC.TAKEN</td>
<td>Qualify taken near branches executed. Must combine with 01H, 02H, 04H, 08H, 10H, 20H.</td>
</tr>
<tr>
<td>88H</td>
<td>FFH</td>
<td>BR_INST_EXEC.ALL_BRANCHES</td>
<td>Counts all near executed branches (not necessarily retired).</td>
</tr>
</tbody>
</table>
## BR_MISP_EXEC Event and UMask

<table>
<thead>
<tr>
<th>Code</th>
<th>Mask</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>89H 01H</td>
<td>BR_MISP_EXEC.COND</td>
<td>Qualify conditional near branch instructions mispredicted.</td>
<td>Must combine with umask 40H, 80H</td>
</tr>
<tr>
<td>89H 04H</td>
<td>BR_MISP_EXEC INDIRECT_IMP NON_CALL RET</td>
<td>Qualify mispredicted indirect near branch instructions that are not calls nor returns.</td>
<td>Must combine with umask 80H</td>
</tr>
<tr>
<td>89H 08H</td>
<td>BR_MISP_EXEC RETURN_NEAR</td>
<td>Qualify mispredicted indirect near branches that have a return mnemonic.</td>
<td>Must combine with umask 80H</td>
</tr>
<tr>
<td>89H 10H</td>
<td>BR_MISP_EXEC DIRECT_NEAR_CALL</td>
<td>Qualify mispredicted unconditional near call branch instructions, excluding non call branch, executed.</td>
<td>Must combine with umask 80H</td>
</tr>
<tr>
<td>89H 20H</td>
<td>BR_MISP_EXEC INDIRECT_NEAR_CALL</td>
<td>Qualify mispredicted indirect near calls, including both register and memory indirect, executed.</td>
<td>Must combine with umask 80H</td>
</tr>
<tr>
<td>89H 40H</td>
<td>BR_MISP_EXEC NONTAKEN</td>
<td>Qualify mispredicted non-taken near branches executed.</td>
<td>Applicable to umask 01H only</td>
</tr>
<tr>
<td>89H 80H</td>
<td>BR_MISP_EXEC TAKEN</td>
<td>Qualify mispredicted taken near branches executed. Must combine with 01H, 02H, 04H, 08H, 10H, 20H.</td>
<td></td>
</tr>
<tr>
<td>89H FFH</td>
<td>BR_MISP_EXEC ALL_BRANCHES</td>
<td>Counts all near executed branches (not necessarily retired).</td>
<td></td>
</tr>
</tbody>
</table>
# BR_INST_RETIRED Event and UMask

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>C4H</td>
<td>00H</td>
<td>BR_INST_RETIRED.ALL_BRANCHES</td>
<td>Branch instructions at retirement.</td>
</tr>
<tr>
<td>C4H</td>
<td>01H</td>
<td>BR_INST_RETIRED.CONDITIONAL</td>
<td>Counts the number of conditional branch instructions retired.</td>
</tr>
<tr>
<td>C4H</td>
<td>02H</td>
<td>BR_INST_RETIRED.NEAR_CALL</td>
<td>Direct and indirect near call instructions retired.</td>
</tr>
<tr>
<td>C4H</td>
<td>04H</td>
<td>BR_INST_RETIRED.ALL_BRANCHES</td>
<td>Counts the number of branch instructions retired.</td>
</tr>
<tr>
<td>C4H</td>
<td>08H</td>
<td>BR_INST_RETIRED.NEAR_RETURN</td>
<td>Counts the number of near return instructions retired.</td>
</tr>
<tr>
<td>C4H</td>
<td>10H</td>
<td>BR_INST_RETIRED.NOT_TAKEN</td>
<td>Counts the number of not taken branch instructions retired.</td>
</tr>
<tr>
<td>C4H</td>
<td>20H</td>
<td>BR_INST_RETIRED.NEAR_TAKEN</td>
<td>Number of near taken branches retired.</td>
</tr>
<tr>
<td>C4H</td>
<td>40H</td>
<td>BR_INST_RETIRED.FAR_BRANCH</td>
<td>Number of far branches retired.</td>
</tr>
</tbody>
</table>
## BR_MISP_RETIRED Event and UMask

<table>
<thead>
<tr>
<th>C5H</th>
<th>00H</th>
<th>BR_MISP RETIRED.ALL BRANCHES</th>
<th>Mispredicted branch instructions at retirement</th>
<th>See Table 19-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>C5H</td>
<td>01H</td>
<td>BR_MISP RETIRED.CONDITIONAL</td>
<td>Mispredicted conditional branch instructions retired</td>
<td>Supports PEBS</td>
</tr>
<tr>
<td>C5H</td>
<td>04H</td>
<td>BR_MISP RETIRED.ALL BRANCHES</td>
<td>Mispredicted macro branch instructions retired.</td>
<td>Supports PEBS</td>
</tr>
<tr>
<td>C5H</td>
<td>20H</td>
<td>BR_MISP RETIRED.NEAR_TAKEN</td>
<td>Number of near branch instructions retired that were taken but mispredicted.</td>
<td></td>
</tr>
<tr>
<td>Address</td>
<td>Code</td>
<td>Description</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
<td>--------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>C5H 00H</td>
<td>BR_MISP_RETIRERD.ALL_BRANCHES</td>
<td>Retired mispredicted branch instructions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C5H 07H</td>
<td>BR_MISP_RETIRERD.ICC</td>
<td>Retired mispredicted conditional jumps</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C5H 0B H</td>
<td>BR_MISP_RETIRERD.FAR</td>
<td>Retired mispredicted far branch instructions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C5H 0E H</td>
<td>BR_MISP_RETIRERD.NON_RETURN_IND</td>
<td>Retired mispredicted instructions of near indirect jmp or call</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C5H 0F H</td>
<td>BR_MISP_RETIRERD.RETURN</td>
<td>Retired mispredicted near return instructions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C5H 19 H</td>
<td>BR_MISP_RETIRERD.CALL</td>
<td>Retired mispredicted near call instructions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C5H 1B H</td>
<td>BR_MISP_RETIRERD.IND_CALL</td>
<td>Retired mispredicted near indirect call instructions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C5H 1D H</td>
<td>BR_MISP_RETIRERD.REL_CALL</td>
<td>Retired mispredicted near relative call instructions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C5H 1F H</td>
<td>BR_MISP_RETIRERD.TAKEN_1CC</td>
<td>Retired mispredicted conditional jumps that were predicted taken</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Mispredicted Branch Transfer and Exploits
Most non-logic vulnerabilities/exploits will cause unexpected code flows!
Branch Misprediction Unit is designed very well
Branch Misprediction Events based approach will catch enough events caused by exploits with reasonable performance impact
Exploit Code And Misprediction

- First unintended/intended exploit code trigged by indirect CALL
- First unintended/intended exploit code trigged by indirect JMP
- First unintended/intended exploit code trigged by RET
- Following unintended/intended ROP gadgets

- Indirect CALL misprediction
- Indirect JMP misprediction
- RET misprediction
- Indirect CALL/JMP, RET misprediction
Stack Pivoting Detection
Overview
Stack Pivoting Detection Solution

- Stack Pivoting needs to point stack pointer to customized data buffer, usually it’s from heap
- Current detection solution
  - Critical APIs check
Stack Pivoting Detection

- Validate stack limitation
- From FS:18h
- Use _NT_TIB or _TEB to get stack limitation

```
0:000> dt _NT_TIB
ntdll!_NT_TIB
  +0x000 ExceptionList : Ptr32 _EXCEPTION_REGISTRATION_RECORD
  +0x004 StackBase : Ptr32 Void
  +0x008 StackLimit : Ptr32 Void
  +0x00c SubSystemTib : Ptr32 Void
  +0x010 FiberData : Ptr32 Void
  +0x010 Version : Uint4B
  +0x014 ArbitraryUserPointer : Ptr32 Void
  +0x018 Self : Ptr32 _NT_TIB
```
Hooked API Examples IN EMET

- `kernel32![API]Stub`
- **Hooked APIs**
  - `kernel32.MapViewOfFileEx`
  - `kernel32.MapViewOfFile`
  - `kernel32.CreateFileMappingW`
  - `kernel32.CreateFileMappingA`
  - `kernel32.CreateFileW`
  - `kernel32.CreateFileA`
  - `kernel32.WinExec`
  - `kernel32.WriteProcessMemory`
  - `kernel32.CreateRemoteThread`
  - `kernel32.CreateProcessInternalW`
  - `kernel32.CreateProcessInternalA`
  - `kernel32.CreateProcessW`
  - `kernel32.CreateProcessA`
  - `kernel32.HeapCreate`
  - `kernel32.VirtualAllocEx`
  - `kernel32.VirtualAlloc`
  - `kernel32.LoadLibraryExW`
  - `kernel32.LoadLibraryExA`
  - `kernel32.LoadLibraryW`
  - `kernel32.LoadLibraryA`
  - `kernel32.VirtualProtectEx`
  - `kernel32.VirtualProtect`
Hooks Code Example

kernel32!VirtualProtectStub:
76692bcd e93ed595f9      jmp  6fff0110
76692bd2  5d            pop   ebp
76692bd3 e900f5fbff     jmp  kernel32!VirtualProtect (766520d8)

No prior disassembly possible
6fff0110 6851cd38d4     push  0D4386C51h
6fff0115  60           pushad
6fff0116  9c           pushfd
6fff0117  54           push  esp
6fff0118 e893d399fd    call  emet+0x4d4b0 (6d98d4b0)
6fff011d  9d           popfd
6fff011e  61           popad
6fff011f  83c404       add   esp,4
6fff0122  8bff         mov   edi,edi
6fff0124  55           push  ebp
6fff0125  8bec         mov   ebp,esp
6fff0127 e9a62a6a06     jmp  kernel32!VirtualProtectStub+0x5 (76692bd2)
6fff012f  83c40c       add   esp,8
StackPivot in EMET

GetStack_Limits(&stacklimit, &stackbase);
if ( caller_esp < stacklimit || caller_esp > stackbase )
{

EMET ROP checks error. Resume?

StackPointer check Failed:
PID : 0x17C8/6088
TID : A60
API name : kernel32.VirtualProtect
ReturnAddress: 6878D96C
CalledAddress: 76692BCD
StackBottom : 2BF9000
StackTop : 2C10000
StackPtr : 06609144

Yes  No

}
}
Improved Stack Pivoting Detection

- Problems in API based approach
  - Hook hopping could bypass the check
  - Valid stack pointer before API calling will bypass the check
- Improvement
  - Check on more APIs
  - Check on instruction
  - Check on branch instruction
  - Check on CALL/JMP branch instruction
  - Check on indirect CALL/JMP branch instruction
  - Check on mispredicted Indirect CALL/JMP branch instruction
Defense with 
CPU Performance Counters
Defense with CPU Performance Counters
Defense on Windows 7 32bits

- MSR Programming
- Counter configuration
- Register Event Handler
- Interrupt Context on Stack
- Interrupt Event Handler
Write MSR

- Compiler Intrinsics
  ```c
  void __writemsr( unsigned long Register, unsigned __int64 Value );
  ```

- Inline ASM
  ```asm
  mov ecx, MSRID
  mov edx, HIGH32b
  mov eax, LOW32b
  ;wrmsr
  __emit 00Fh
  __emit 030h
  ```

### WRMSR—Write to Model Specific Register

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Instruction</th>
<th>Op/En</th>
<th>64-Bit Mode</th>
<th>Compal/Leg Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0F 30</td>
<td>WRMSR</td>
<td>NP</td>
<td>Valid</td>
<td>Valid</td>
<td>Write the value in EDX:EAX to MSR specified by ECX.</td>
</tr>
</tbody>
</table>
MSR Programming

- Use PMC0 to trigger PMI
  1. IA32_DEBUGCTL
  2. LBR_SELECT
  3. MSR_PERF_GLOBAL_CTRL
  4. PMC0
  5. IA32_PERFEVTSEL0
  6. MSR_PERF_GLOBAL_CTRL
Counter configuration

- Trigger Interrupt for every event
  - Set PMC0 = 0xffffffffffffffff
- Interrupt will happen for every event cross whole system
Register Event Handler

- Interrupt Vector - 0xFE
- Set up event handler
  - IDT Hook
    - Replace vector 0xFE handler
  - API Hook
    - Hal!HalpPerfInterrupt
  - Callback Hook
    ```assembly
    mov    ecx, ebp
    mov    eax, ds:_HalpPerfInterruptHandler
    or     eax, eax
    jz     short loc_8002B354
    call   eax
    ```
- Register via API
Interrupt Context on Stack

- IDT Hook needs to maintain interrupt context as usual
- Callback Hook needs to address undocumented interrupt context from stack

```c
typedef struct {
    UINT32 stacktop;
    UINT32 intno;
    UINT32 esp;
    UINT32 eip;
    UINT32 Data[24];
    UINT32 int_eip;
    UINT32 int_cs;
    UINT32 int_eflag;
    UINT32 int_sp;
    UINT32 int_ss;
} IA32_PERFINTERRUPT_PARAMETER;
```
Interrupt Event Handler

- Check IA32_PERF_GLOBAL_STATUS and clear MSR_PERF_GLOBAL_OVF_CTL if multiple PMCs are used
- Check CS in interrupt frame to filter out all ring 0 events
- Check current CR3 is targeted process
- Carefully deal with pagable memory to get stack range, code@From, code@To
  - APC, IRQL changes
- Compare stack in interrupt frame with TIB stack range
- Get last branch transfer record from LBR TOS
- Clear IA32_PERF_GLOBAL_OVF_CTRL if need
APSA13-2 Case Study
APSA13-02 PDF 0-day

- Reported by FireEye in February 2013
- Best Client-Side Bug - CVE-2013-0641
- Sophisticated ROP only without shellcode
- First public in the wild exploit Adobe Sandbox Bypassing

Malicious PDF

<table>
<thead>
<tr>
<th>Acro32</th>
<th>Trigger 1st Vul</th>
<th>StackPivoting</th>
<th>D.T</th>
<th>2 Threads</th>
<th>IPC</th>
<th>StackPivoting</th>
<th>L2P.T</th>
<th>L2P.T</th>
<th>L2P.T</th>
<th>Acro32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open PDF file</td>
<td>Heap Overflow</td>
<td>Run ROP Stackpivoting</td>
<td>Create and Load D.T Library</td>
<td>1: Show Error</td>
<td>IPC to trigger 2nd vul, then quit</td>
<td>2: Create L2P.T</td>
<td>Create Visaform Turkey.pdf</td>
<td>New Process shows Visaform Turkey.pdf</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sandbox Reader</td>
<td>Broker Reader</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## CVE-2013-0640 Exploit

<table>
<thead>
<tr>
<th>Step</th>
<th>Action Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acro32</td>
<td>Open PDF file</td>
</tr>
<tr>
<td>Trigger 1(^{nd}) Vul</td>
<td>Heap Overflow</td>
</tr>
<tr>
<td>StackPivoting</td>
<td>Run ROP Stackpivoting</td>
</tr>
<tr>
<td>D.T</td>
<td>Create and Load D.T Library</td>
</tr>
<tr>
<td>2 Threads</td>
<td>1: Show Error 2: Create L2P.T</td>
</tr>
<tr>
<td>IPC</td>
<td>IPC to trigger 2nd vul, then quit</td>
</tr>
<tr>
<td>Broker Reader</td>
<td>Heap Overflow</td>
</tr>
</tbody>
</table>

---

### Sandbox Reader
- Open PDF file
- Heap Overflow
- Run ROP Stackpivoting
- Create and Load D.T Library
- 1: Show Error
- 2: Create L2P.T
- IPC to trigger 2nd vul, then quit

### Broker Reader
- Heap Overflow
CVE-2013-0640 PDF 0-day analysis (1)

触发点在 AcroForm.api

```
.exploittrigger:
call dword ptr [eax]
```

堆栈偏移

```
.text:209B9F42
mov eax, [ecx+4]
test eax, eax
jz short loc_209B9F57
push eax
mov eax, dword_2128C66C
call dword ptr [eax+5Ch]
pop ecx
movzx eax, ax
ret
```

```
.text:209B9F48
mov eax, [ecx+4]
test eax, eax
jz short loc_209B9F57
push eax
mov eax, dword_2128C66C
-------------------------
db 0FFh
-------------------------
push eax
pop esp
pop ecx
movzx eax, ax
ret
```
Lots of ROP gadgets

- Get ntdll.dll and get related API address
  - for example, RtlDecompressBuffer and CryptStringToBinaryA
- Call CryptStringToBinaryA to convert one string to binary
- Call RtlDecompressBuffer to decompress binary to real D.T binary code in memory
- Finally, the ROP gadget will call GetTempPathA to get current temp path, it’s sandboxed path, create D.T under this path and call LoadLibraryA to run D.T.

D.T dll will trigger 2nd 0-day to load L2P.T in broker process
CVE-2013-0641 Exploit

- IPC: IPC to trigger 2nd vul, then quit
- StackPivoting: Heap Overflow, Run ROP via Stackpivoting
- L2P.T: Load L2P.T, Create Langbar.dll And load it
- L2P.T: Create Visaform Turkey.pdf
- Acro32: Create new sandboxed process to open new pdf
CVE-2013-0641 PDF 0-day analysis (1)

- Trigger Point in acrord32.exe

```plaintext
.text:0049728A secondu1_triggerpoint:
text:0049728A call eax
```

- Stack Pivoting

```plaintext
.text:6F881564 ;----------------- db 0FFh
.text:6F881565 push edx
.text:6F881566 pop esp
.text:6F881567 pop ebp
.text:6F881568         retn  8
```

- ROP Gadget will load L2P.T and kick off malware infection
Demo
Misc

- Interrupted EIP
  - Sometimes the Interrupted EIP is different from target EIP
- Evasion
  - Dummy exploit to trigger code to avoid detection before real exploit code happen
- Global Events
  - PMU Events are global
- Missed Event
  - PMU is designed for profiling, not for security
- LBR Record
- User Mode Scheduler
Interrupt EIP and Event EIP

The Interrupt could happen on different EIP from Event EIP

<table>
<thead>
<tr>
<th>FromAddr=0x5792d826</th>
<th>IntAddr=0x57932ba3</th>
<th>ToAddr=0x57932b80</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>57932ba8 83ac08</th>
<th>push ebp</th>
</tr>
</thead>
<tbody>
<tr>
<td>57932ba3 53</td>
<td>push ebx</td>
</tr>
<tr>
<td>57932b88 88</td>
<td>mov ebx,dword ptr [esp+14h]</td>
</tr>
<tr>
<td>57932b89 55</td>
<td>push ebp</td>
</tr>
<tr>
<td>57932b8a 8b5c2414</td>
<td>mov ebp,dword ptr [esp+14h]</td>
</tr>
<tr>
<td>57932ba0 8b7d50</td>
<td>push esi</td>
</tr>
<tr>
<td>57932ba1 8b7d50</td>
<td>mov esi,dword ptr [esp+14h]</td>
</tr>
<tr>
<td>57932ba2 8b7d50</td>
<td>push edi</td>
</tr>
<tr>
<td>57932ba3 8b7d50</td>
<td>mov edi,dword ptr [esp+14h]</td>
</tr>
<tr>
<td>57932ba4 8b7d50</td>
<td>push edx</td>
</tr>
<tr>
<td>57932ba5 8b7d50</td>
<td>mov edx,dword ptr [esp+14h]</td>
</tr>
<tr>
<td>57932ba6 89442410</td>
<td>mov dword ptr [esp+10h],eax</td>
</tr>
<tr>
<td>57932ba7 8b5140</td>
<td>mov edx,dword ptr [ecx+40h]</td>
</tr>
<tr>
<td>57932ba8 8d442410</td>
<td>lea eax,[esp+10h]</td>
</tr>
<tr>
<td>57932ba9 8d442410</td>
<td>lea edx,[esp+10h]</td>
</tr>
<tr>
<td>57932baa 50</td>
<td>push eax</td>
</tr>
<tr>
<td>57932bab 8bc8</td>
<td>mov ecx,eax</td>
</tr>
<tr>
<td>57932bb0 8b9542410</td>
<td>mov dword ptr [esp+18h],edx</td>
</tr>
<tr>
<td>57932bb1 51</td>
<td>push ecx</td>
</tr>
<tr>
<td>57932bb2 8c8</td>
<td>lea edx,[edi+2dh]</td>
</tr>
<tr>
<td>57932bb3 8d7524</td>
<td>lea edx,[edi+2dh]</td>
</tr>
<tr>
<td>57932bb4 8b7d50</td>
<td>mov edi,dword ptr [ebp+50h]</td>
</tr>
</tbody>
</table>

57932ba0 8b7d50 mov edi,dword ptr [ebp+50h]
57932ba3 8bc3 mov eax,ebx
57932ba5 c1e010 shl eax,10h
57932ba8 89442410 mov dword ptr [esp+18h],edx
57932bb1 51 push ecx
57932bb2 8c8 lea edx,[edi+2dh]
## Repeated Indirect Call Misprediction

<table>
<thead>
<tr>
<th>Time</th>
<th>FromAddr</th>
<th>IntAddr</th>
<th>ToAddr</th>
</tr>
</thead>
<tbody>
<tr>
<td>4:41:40.238 PM</td>
<td>0x7612c171</td>
<td>0x759a68ef</td>
<td>0x762bf8d</td>
</tr>
<tr>
<td>4:41:40.238 PM</td>
<td>0x7611cc50</td>
<td>0x759a68ef</td>
<td>0x762bf8d</td>
</tr>
<tr>
<td>4:41:40.238 PM</td>
<td>0x7612c847</td>
<td>0x759a68f2</td>
<td>0x762bf8d</td>
</tr>
<tr>
<td>4:41:40.238 PM</td>
<td>0x7612db08</td>
<td>0x759a68f5</td>
<td>0x762bf8d</td>
</tr>
<tr>
<td>4:41:40.238 PM</td>
<td>0x761bc453</td>
<td>0x759a68f5</td>
<td>0x762bf8d</td>
</tr>
<tr>
<td>4:41:40.238 PM</td>
<td>0x7610d89e7</td>
<td>0x759a68f5</td>
<td>0x762bf8d</td>
</tr>
<tr>
<td>4:41:40.238 PM</td>
<td>0x7610e0a1</td>
<td>0x759a68f5</td>
<td>0x762bf8d</td>
</tr>
<tr>
<td>4:41:40.238 PM</td>
<td>0x761465f3</td>
<td>0x759a68f2</td>
<td>0x762bf8d</td>
</tr>
<tr>
<td>4:41:40.238 PM</td>
<td>0x7610d1c5</td>
<td>0x759a68f2</td>
<td>0x762bf8d</td>
</tr>
<tr>
<td>4:41:40.238 PM</td>
<td>0x7612c7d4</td>
<td>0x759a68f2</td>
<td>0x762bf8d</td>
</tr>
<tr>
<td>4:41:40.238 PM</td>
<td>0x7612c1c2</td>
<td>0x759a68f2</td>
<td>0x762bf8d</td>
</tr>
<tr>
<td>4:41:40.238 PM</td>
<td>0x7612c1f8</td>
<td>0x759a68f2</td>
<td>0x762bf8d</td>
</tr>
<tr>
<td>4:41:40.238 PM</td>
<td>0x7612c192</td>
<td>0x759a68f2</td>
<td>0x762bf8d</td>
</tr>
<tr>
<td>4:41:40.238 PM</td>
<td>0x7610db94</td>
<td>0x759a68f2</td>
<td>0x762bf8d</td>
</tr>
<tr>
<td>4:41:40.238 PM</td>
<td>0x7612e22b</td>
<td>0x759a68f5</td>
<td>0x762bf8d</td>
</tr>
<tr>
<td>4:41:40.238 PM</td>
<td>0x7612c259</td>
<td>0x759a68f5</td>
<td>0x762bf8d</td>
</tr>
<tr>
<td>4:41:40.238 PM</td>
<td>0x76115672</td>
<td>0x759a68f5</td>
<td>0x762bf8d</td>
</tr>
<tr>
<td>4:41:40.238 PM</td>
<td>0x7612c043</td>
<td>0x759a68f5</td>
<td>0x762bf8d</td>
</tr>
<tr>
<td>4:41:40.238 PM</td>
<td>0x7610de8b</td>
<td>0x759a68f5</td>
<td>0x762bf8d</td>
</tr>
<tr>
<td>4:41:40.238 PM</td>
<td>0x7610d248</td>
<td>0x759a68f5</td>
<td>0x762bf8d</td>
</tr>
<tr>
<td>4:41:40.238 PM</td>
<td>0x761bc9f2</td>
<td>0x759a68f5</td>
<td>0x762bf8d</td>
</tr>
<tr>
<td>4:41:40.238 PM</td>
<td>0x761bc426</td>
<td>0x759a68f5</td>
<td>0x762bf8d</td>
</tr>
<tr>
<td>4:41:40.238 PM</td>
<td>0x7610d9af</td>
<td>0x759a68f5</td>
<td>0x762bf8d</td>
</tr>
<tr>
<td>4:41:40.238 PM</td>
<td>0x761bc854</td>
<td>0x759a68f5</td>
<td>0x762bf8d</td>
</tr>
<tr>
<td>4:41:40.238 PM</td>
<td>0x7612c7d4</td>
<td>0x759a68f5</td>
<td>0x762bf8d</td>
</tr>
<tr>
<td>4:41:40.238 PM</td>
<td>0x7612c1c2</td>
<td>0x759a68f5</td>
<td>0x762bf8d</td>
</tr>
<tr>
<td>4:41:40.238 PM</td>
<td>0x76115120</td>
<td>0x759a68f5</td>
<td>0x762bf8d</td>
</tr>
<tr>
<td>4:41:40.238 PM</td>
<td>0x761150c6</td>
<td>0x759a68f5</td>
<td>0x762bf8d</td>
</tr>
<tr>
<td>4:41:40.238 PM</td>
<td>0x761bc426</td>
<td>0x759a68f5</td>
<td>0x762bf8d</td>
</tr>
<tr>
<td>4:41:40.238 PM</td>
<td>0x7612c1f8</td>
<td>0x759a68f5</td>
<td>0x762bf8d</td>
</tr>
<tr>
<td>4:41:40.238 PM</td>
<td>0x7612c192</td>
<td>0x759a68f5</td>
<td>0x762bf8d</td>
</tr>
<tr>
<td>4:41:40.238 PM</td>
<td>0x7610db94</td>
<td>0x759a68f5</td>
<td>0x762bf8d</td>
</tr>
<tr>
<td>4:41:40.238 PM</td>
<td>0x761bc9f2</td>
<td>0x759a68f5</td>
<td>0x762bf8d</td>
</tr>
<tr>
<td>4:41:40.238 PM</td>
<td>0x761bc854</td>
<td>0x759a68f5</td>
<td>0x762bf8d</td>
</tr>
</tbody>
</table>
Global Events

- Without MSR enabling/disabling during OS Scheduler
  - The event could happen on every code on logical processor whatever the process
- With proper MSR enabling/disabling during context switch, performance will be improved
- For example, 5-6 events on target CR3 per 10K

<table>
<thead>
<tr>
<th>Events</th>
<th>All</th>
<th>R3</th>
<th>R3 Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>Events</td>
<td>27638657</td>
<td>27044900</td>
<td>15345</td>
</tr>
</tbody>
</table>
LBR Record

- Must freeze LBRs on PMI
- MSR
  - MSR_LASTBRANCH_TOS
  - MSR_LASTBRANCH_X_FROM_IP
  - MSR_LASTBRANCH_X_TO_IP

Figure 17-3. IA32_DEBUGCTL MSR for Processors based on Intel Core microarchitecture
MSR_LASTBRANCH_X_FROM_IP/MSR_LASTBRANCH_X_TO_IP

- Silvermont
  - MSR_LASTBRANCH_0_FROM_IP (address 40H)
  - MSR_LASTBRANCH_0_TO_IP (address 60H)
- Haswell
  - MSR_LASTBRANCH_0_FROM_IP (address 680H)
  - MSR_LASTBRANCH_0_TO_IP (address 6C0H)
## Bonus

- Improved LIFO filter to capture Call Stack
- Block simple repeated LBR FLUSH

### Table 17-11. MSR_LBR_SELECT for Intel® microarchitecture code name Haswell

<table>
<thead>
<tr>
<th>Bit Field</th>
<th>Bit Offset</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPL_EQ_0</td>
<td>0</td>
<td>R/W</td>
<td>When set, do not capture branches occurring in ring 0</td>
</tr>
<tr>
<td>CPL_NEQ_0</td>
<td>1</td>
<td>R/W</td>
<td>When set, do not capture branches occurring in ring &gt;0</td>
</tr>
<tr>
<td>JCC</td>
<td>2</td>
<td>R/W</td>
<td>When set, do not capture conditional branches</td>
</tr>
<tr>
<td>NEAR_REL_CALL</td>
<td>3</td>
<td>R/W</td>
<td>When set, do not capture near relative calls</td>
</tr>
<tr>
<td>NEAR_IND_CALL</td>
<td>4</td>
<td>R/W</td>
<td>When set, do not capture near indirect calls</td>
</tr>
<tr>
<td>NEAR_RET</td>
<td>5</td>
<td>R/W</td>
<td>When set, do not capture near returns</td>
</tr>
<tr>
<td>NEAR_IND_JMP</td>
<td>6</td>
<td>R/W</td>
<td>When set, do not capture near indirect jumps except near indirect calls and near returns</td>
</tr>
<tr>
<td>NEAR_REL_JMP</td>
<td>7</td>
<td>R/W</td>
<td>When set, do not capture near relative jumps except near relative calls.</td>
</tr>
<tr>
<td>FAR_BRANCH</td>
<td>8</td>
<td>R/W</td>
<td>When set, do not capture far branches</td>
</tr>
<tr>
<td>EN_CALLSTACK(^1)</td>
<td>9</td>
<td></td>
<td>Enable LBR stack to use LIFO filtering to capture Call stack profile</td>
</tr>
<tr>
<td>Reserved</td>
<td>63:10</td>
<td></td>
<td>Must be zero</td>
</tr>
</tbody>
</table>
Summary

- Most Exploits will cause branch misprediction with unintended/intended code
- Branch mispredicted events are useful to detect exploits with minimized performance impacts
- APSA13-2 exploit was successfully detected by branch mispredicted based approach
Thanks!

Xiaoning.li@intel.com
mcrouse@seas.harvard.edu

Acknowledge Haifei Li and Dave Marcus’ review!
Reference

[3] Smashing the Heap with Vector: Advanced Exploitation Technique in Recent Flash Zero-day Attack, Haifei Li
[4] Interpreter Exploitation: Pointer Inference and JIT Spraying, Dion Blazakis, BlackHat DC 2010
[8] kBouncer: Efficient and Transparent ROP Mitigation, Vasilis Pappas
[9] Taming the ROPe on Sandy Bridge, Georg Wicherski, SysCan 2013
[12] PRACTICAL RETURN-ORIENTED PROGRAMMING, Dino Dai Zovi, RSA 2010
[14] return oriented exploitation, Dino Dai Zovi, Blackhat 2010
Backup
Architectural Performance Monitoring Version 1

- Bit field layout of IA32_PERFEVTSELx is consistent across micro architectures
- Addresses of IA32_PERFEVTSELx MSRs remain the same across micro architectures
- Addresses of IA32_PMC MSRs remain the same across micro architectures
- Each logical processor has its own set of IA32_PERFEVTSELx and IA32_PMCx MSRs. Configuration facilities and counters are not shared between logical processors sharing a processor core.
Architectural Performance Monitoring Version 1

Facilities

- IA32_PMCx MSRs start at address 0C1H and occupy a contiguous block of MSR address space the number of MSRs per logical processor is reported using CPUID.0AH:EAX[15:8]
- IA32_PERFEVTSELx MSRs start at address 186H and occupy a contiguous block of MSR address space.
- The bit width of an IA32_PMCx MSR is reported using the CPUID.0AH:EAX[23:16]. This the number of valid bits for read operation. On write operations, the lower-order 32 bits of the MSR may be written with any value, and the high-order bits are sign-extended from the value of bit 31
- Bit field layout of IA32_PERFEVTSELx MSRs is defined architecturally