Transparent ROP Detection using CPU Performance Counters

他山之石，可以攻玉
Stones from other hills may serve to polish jade

Xiaoning Li Intel Labs
Michael Crouse Harvard University

THREADS Conference 2014

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

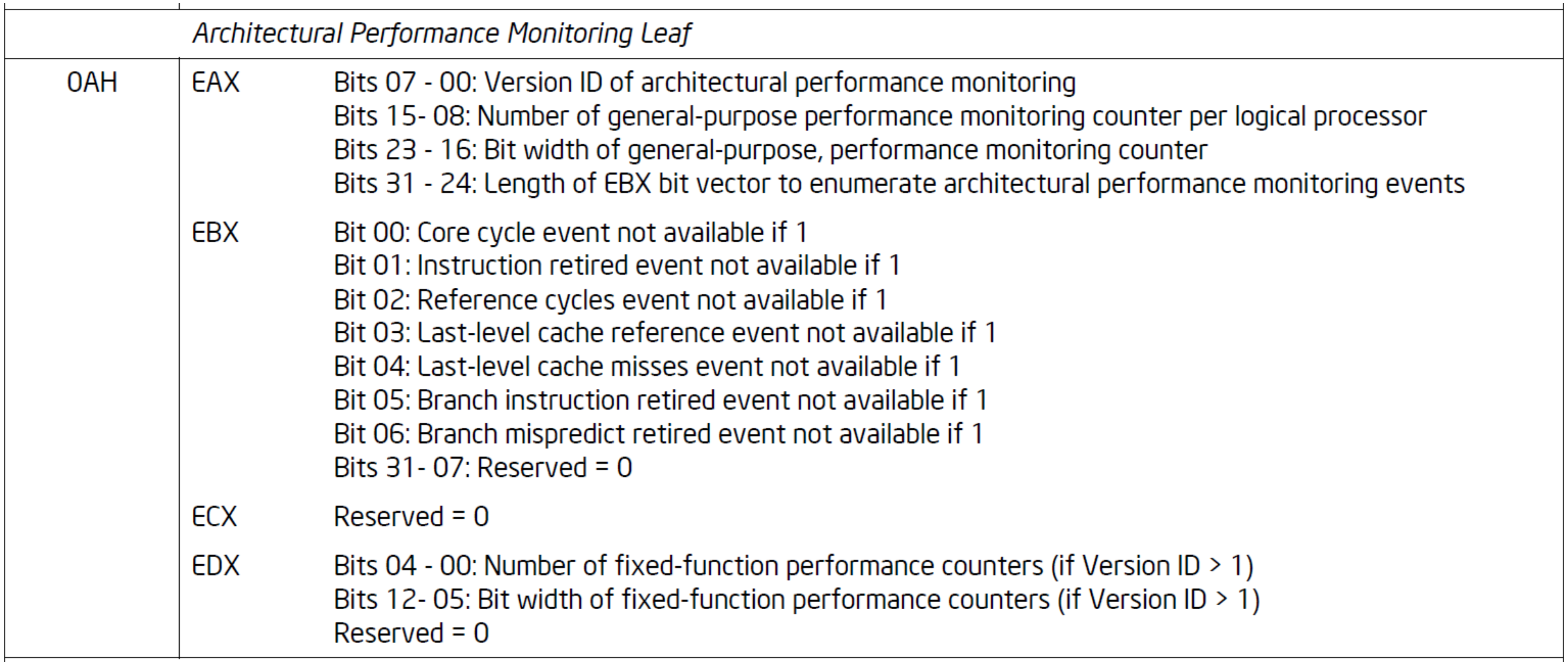
Architectural/Non-Architectural performance events Availability

- 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

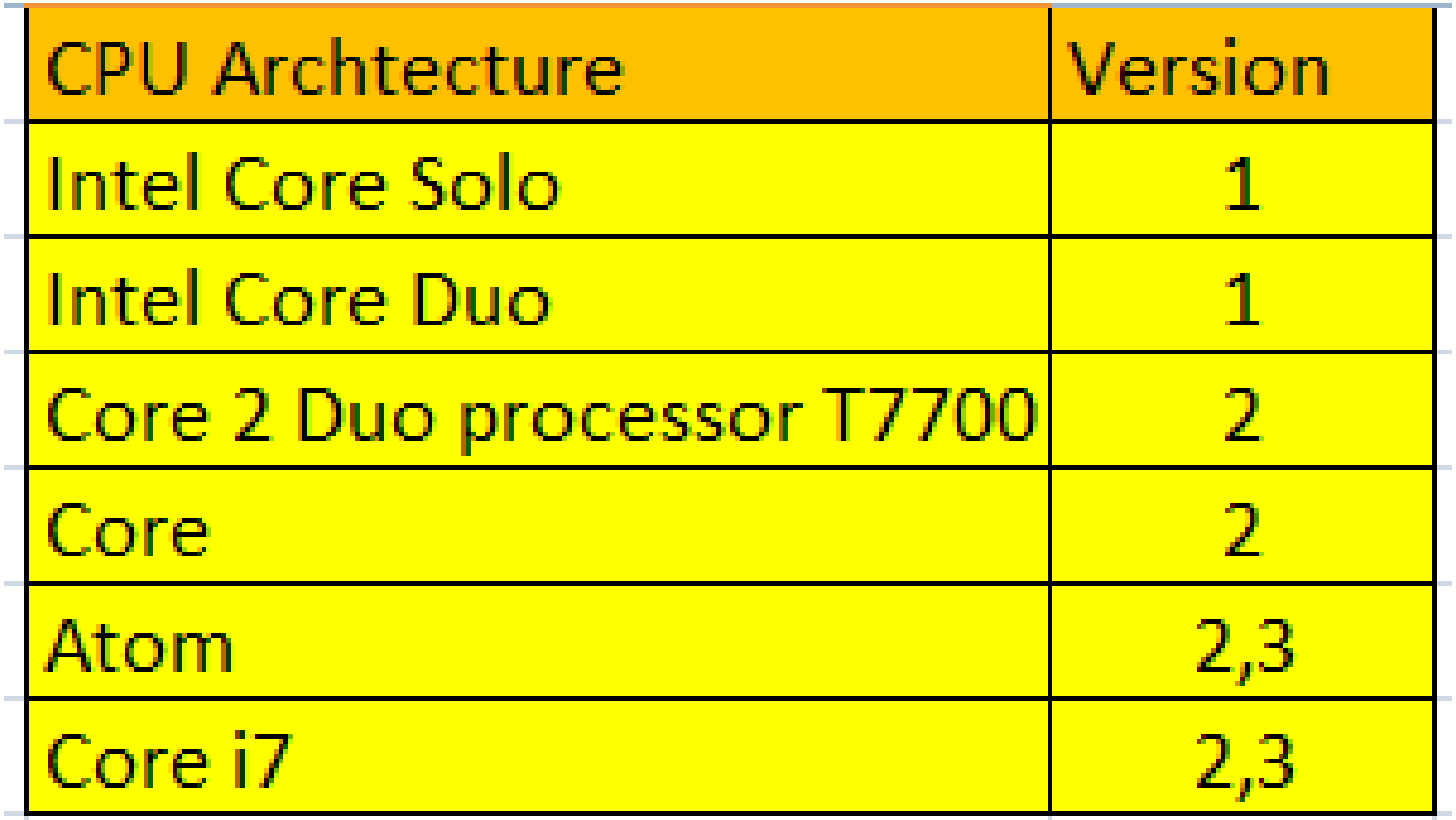
- Number of performance monitoring counters available in a logical processor
- Number of bits supported in each IA32_PMCx
- Number of architectural performance monitoring events supported in a logical processor
- Software can use CPUID to discover architectural performance monitoring availability (CPUID.0AH).

Architectural Performance Monitoring Leaf



	Architectural Performance Monitoring Leaf	
0AH	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

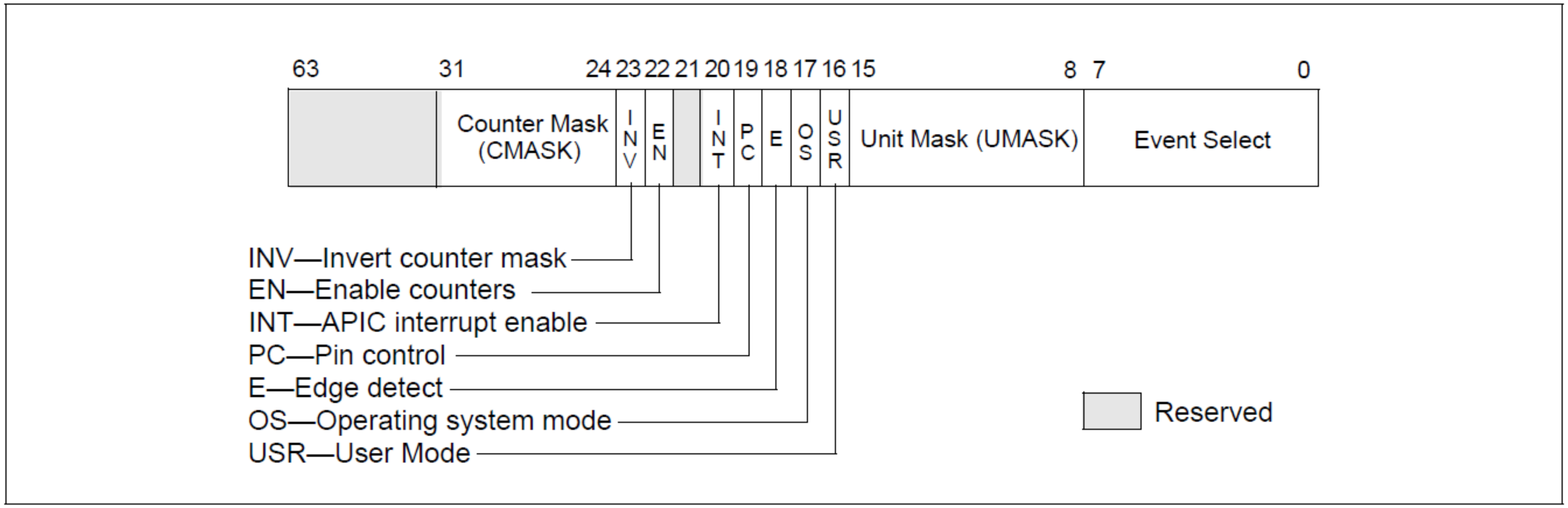


CPU Archtecture	Version
Intel Core Solo	1
Intel Core Duo	1
Core 2 Duo processor T7700	2
Core	2
Atom	2,3
Core i7	2,3

Architectural Performance Monitoring Version 1

- 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



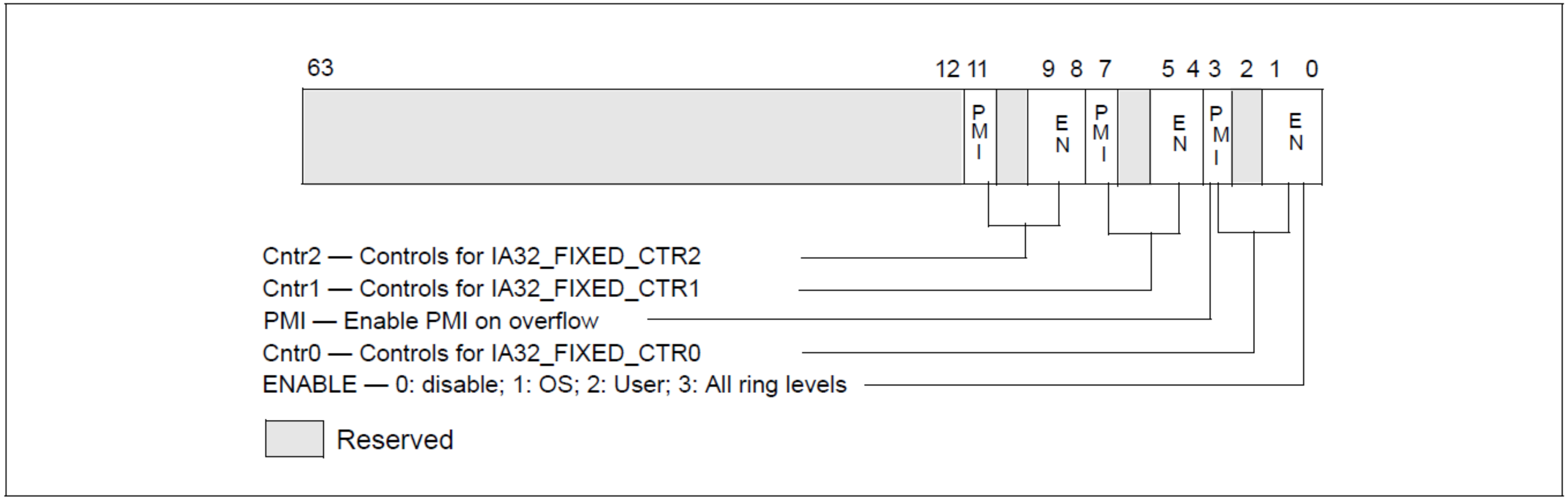
IA32_PERFEVTSELx MSRs

- 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)

Architectural Performance Monitoring Version 2

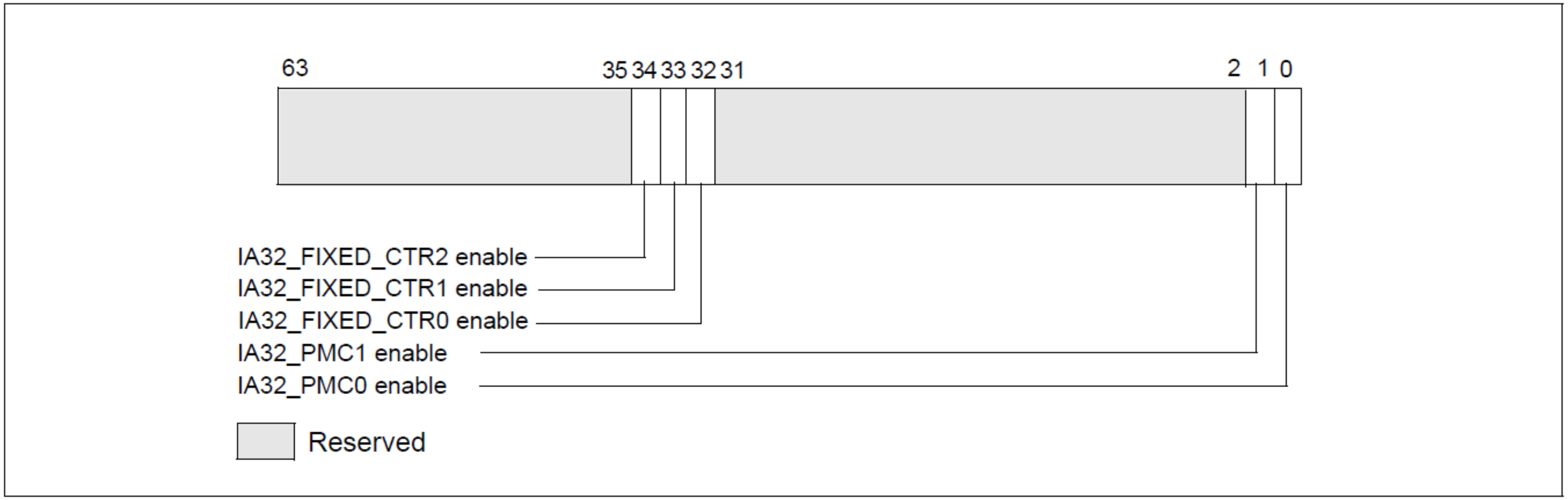
- Bits 0 through 4 of CPUID.0AH.EDX indicates the number of fixed-function performance counters available per core
- Bits 5 through 12 of CPUID.0AH.EDX indicates the bit-width of fixed-function performance counters. Bits beyond the width of the fixed-function counter are reserved and must be written as zeros.

IA32_FIXED_CTR_CTRL MSR

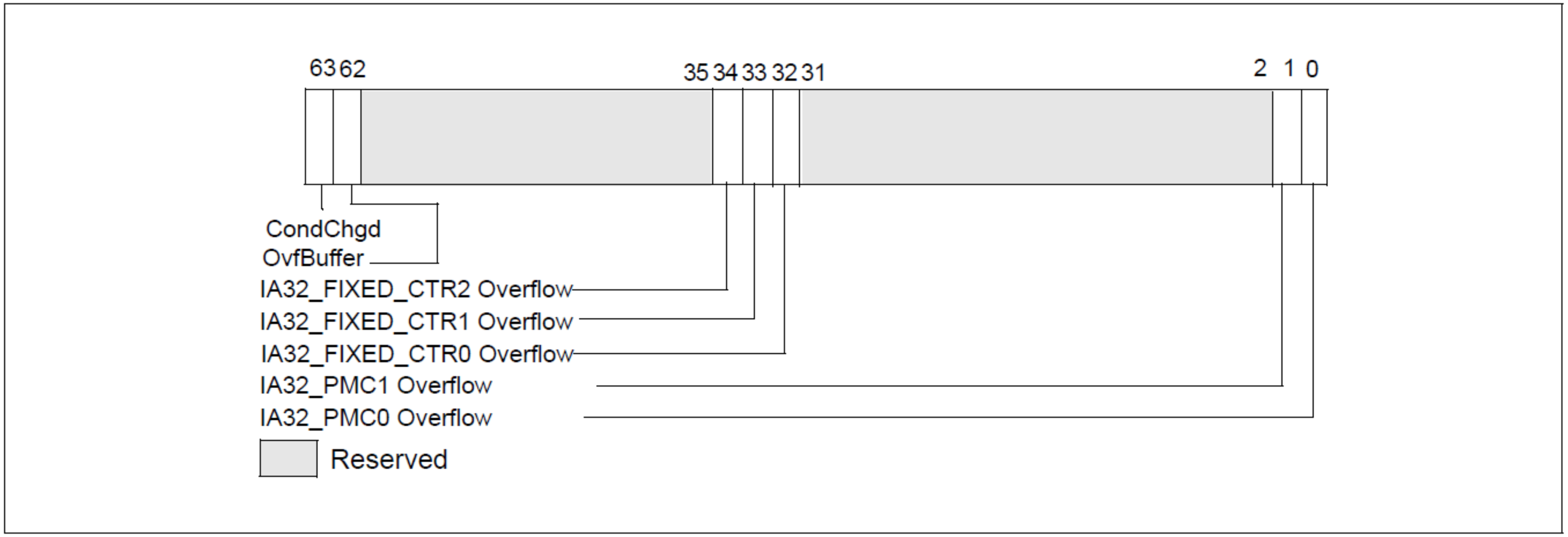

63
12 11
9 8 7
5 4 3 2 1 0
PMI
EN
PMI
EN
PMI
EN
Cntr2 — Controls for IA32_FIXED_CTR2
Cntr1 — Controls for IA32_FIXED_CTR1
PMI — Enable PMI on overflow
Cntr0 — Controls for IA32_FIXED_CTR0
ENABLE — 0: disable; 1: OS; 2: User; 3: All ring levels
Reserved

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

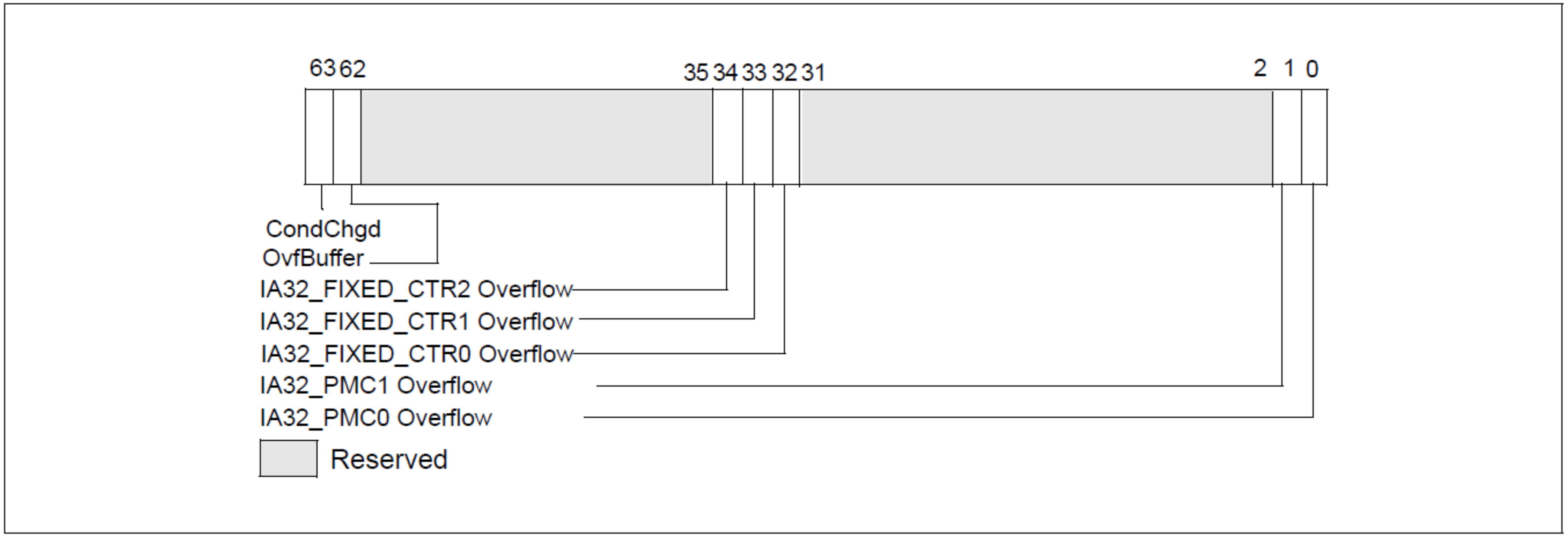
IA32_PERF_GLOBAL_CTRL MSR



IA32_PERF_GLOBAL_STATUS MSR



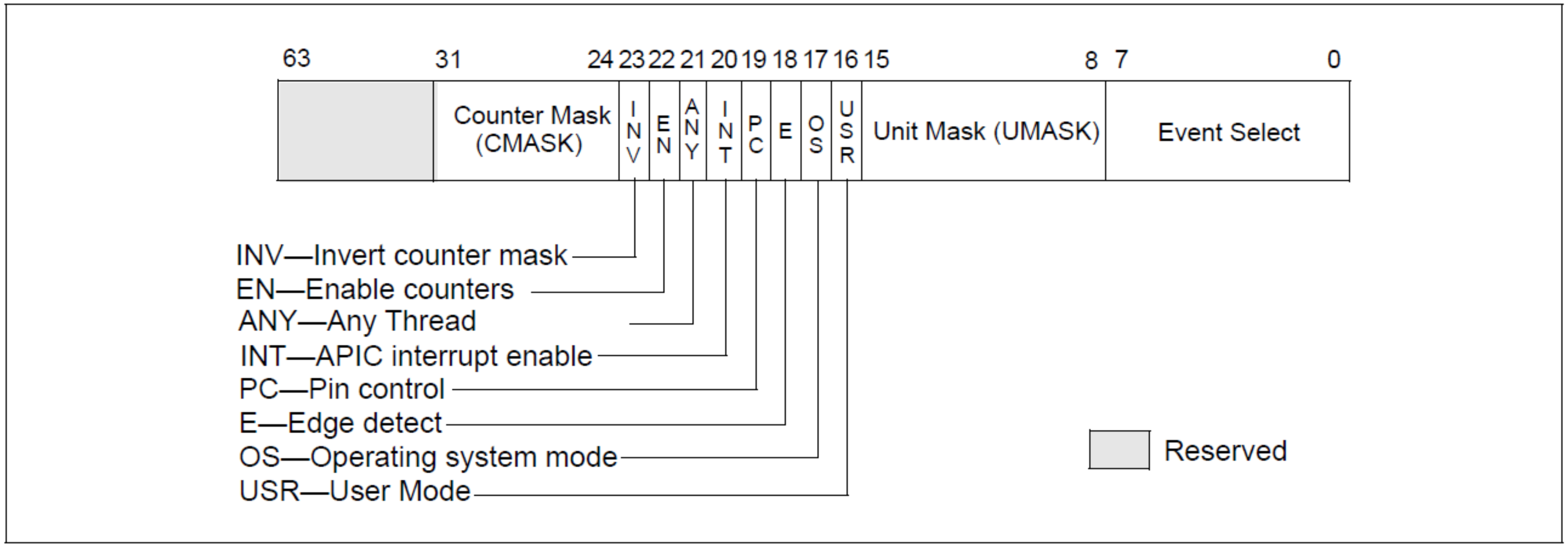
IA32_PERF_GLOBAL_OVF_CTRL MSR



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

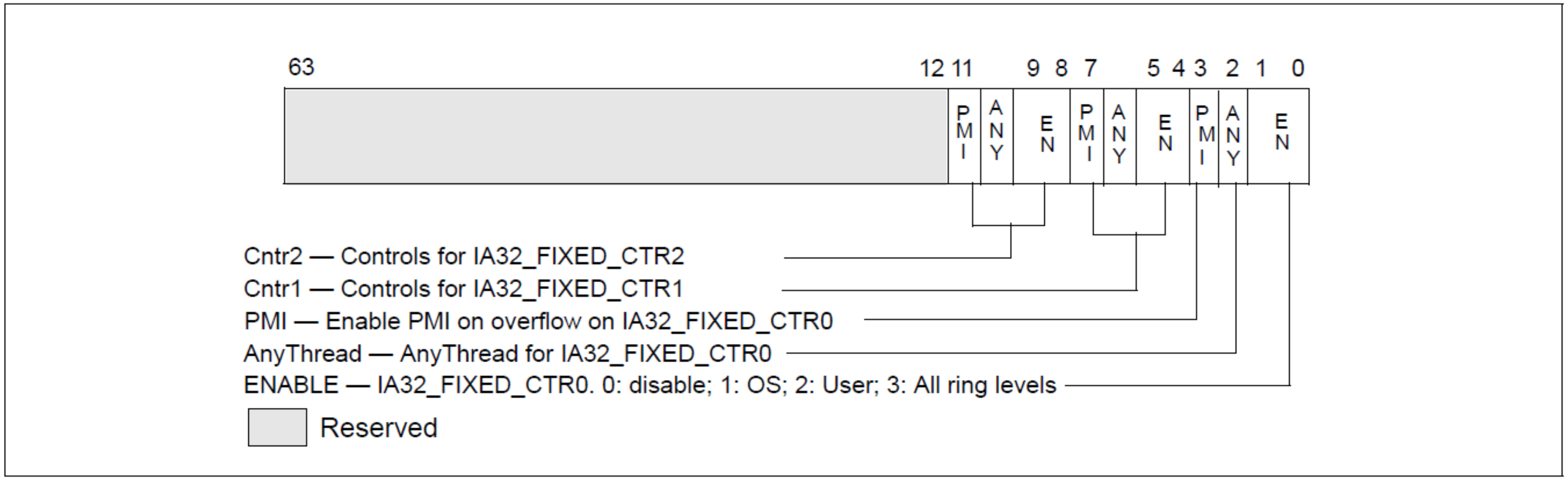
MSRs Supporting Architectural Performance Monitoring Version 3



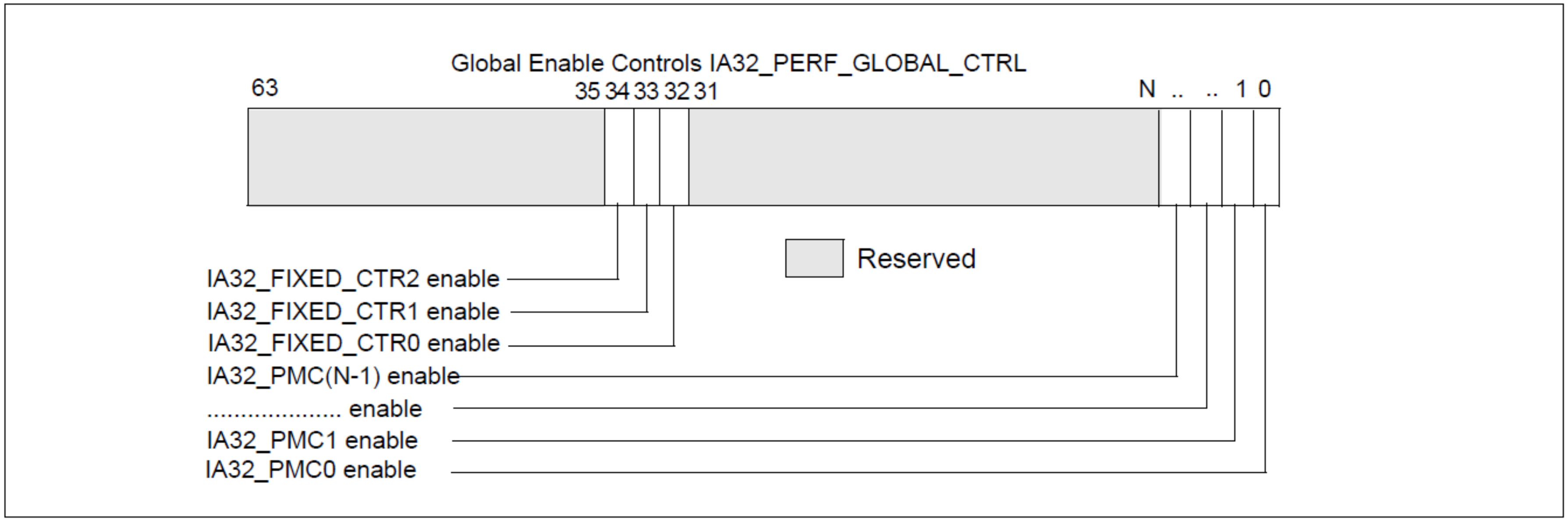
IA32_PERFEVTSELx MSRs Supporting Architectural Performance Monitoring Version 3

- 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.

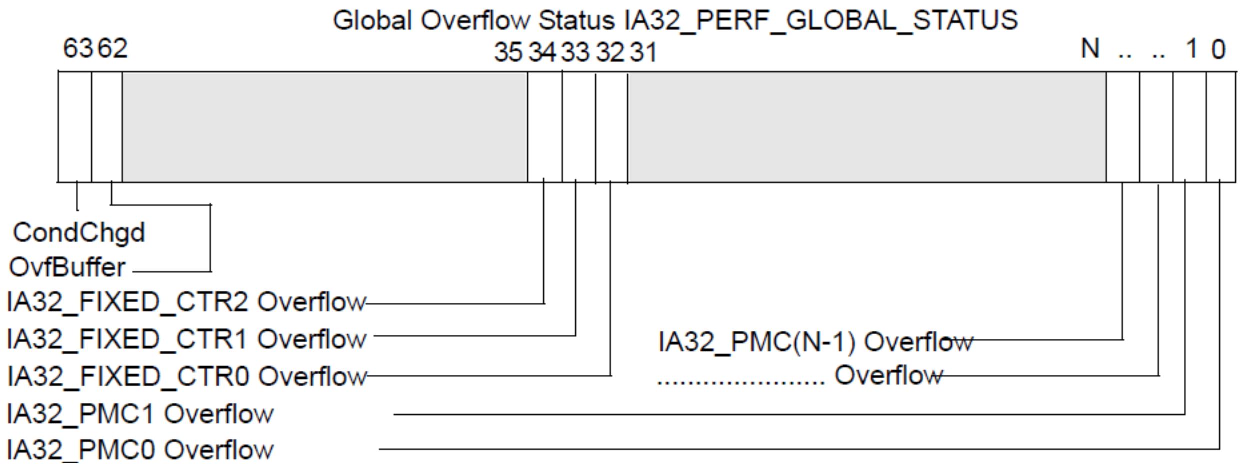
IA32_FIXED_CTR_CTRL MSR Supporting Architectural Performance Monitoring Version 3


63
12 11
9 8 7
5 4 3
2 1 0
PMI
ANY
EN
PMI
ANY
EN
PMI
ANY
EN
Cntr2 — Controls for IA32_FIXED_CTR2
Cntr1 — Controls for IA32_FIXED_CTR1
PMI — Enable PMI on overflow on IA32_FIXED_CTR0
AnyThread — AnyThread for IA32_FIXED_CTR0
ENABLE — IA32_FIXED_CTR0. 0: disable; 1: OS; 2: User; 3: All ring levels
Reserved

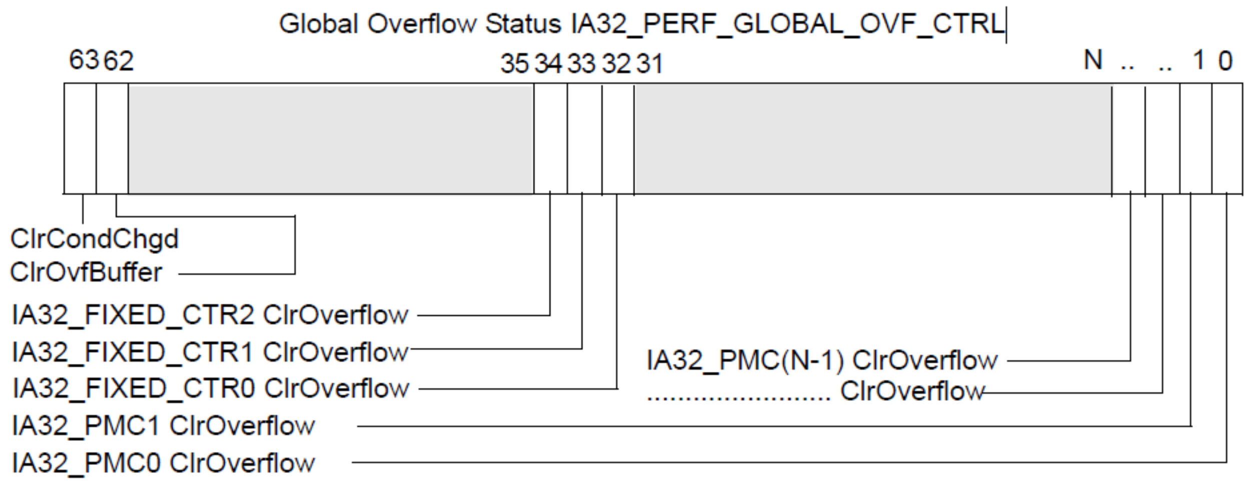
IA32_PERF_GLOBAL_CTRL MSR Supporting Architectural Performance Monitoring Version 3



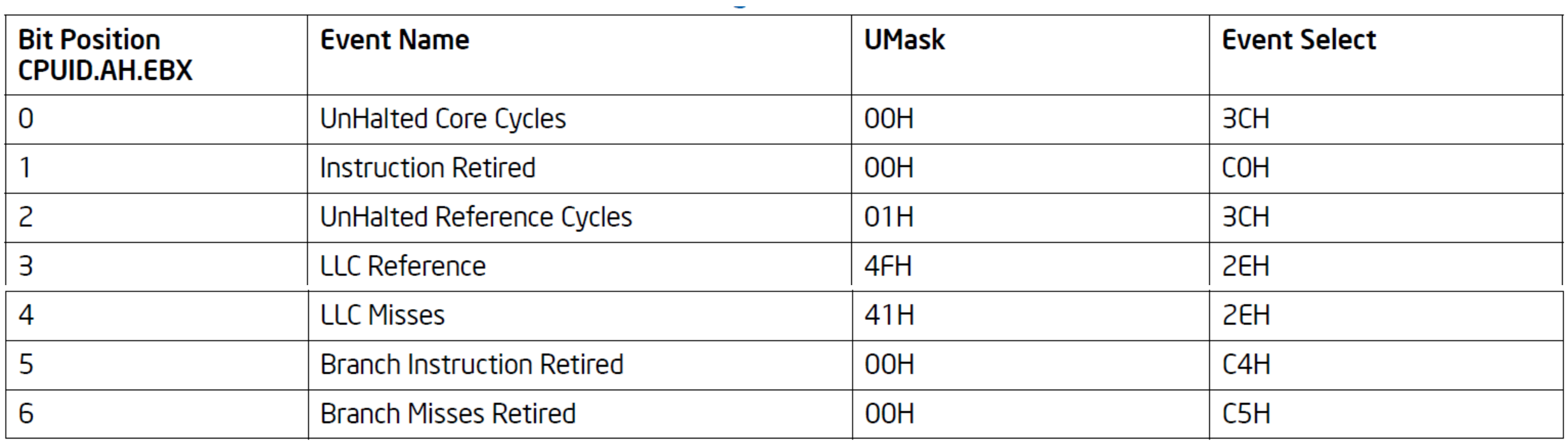
IA32_PERF_GLOBAL_STATUS MSR Supporting Architectural Performance Monitoring Version 3



IA32_PERF_GLOBAL_OVF_CTRL MSR Supporting Architectural Performance Monitoring Version 3


Global Overflow Status IA32_PERF_GLOBAL_OVF_CTRL
63
62
35
34
33
32
31
N 1 0
ClrCondChgd
ClrOvfBuffer
IA32_FIXED_CTR2 ClrOverflow
IA32_FIXED_CTR1 ClrOverflow
IA32_FIXED_CTR0 ClrOverflow
IA32_PMC1 ClrOverflow
IA32_PMC0 ClrOverflow
IA32_PMC(N-1) ClrOverflow
.... ClrOverflow

Pre-defined Architectural Performance Events



Bit Position CPUID.AH.EBX	Event Name	UMask	Event Select
0	UnHalted Core Cycles	00H	3CH
1	Instruction Retired	00H	C0H
2	UnHalted Reference Cycles	01H	3CH
3	LLC Reference	4FH	2EH
4	LLC Misses	41H	2EH
5	Branch Instruction Retired	00H	C4H
6	Branch Misses Retired	00H	C5H

Branch Instructions Retired Events

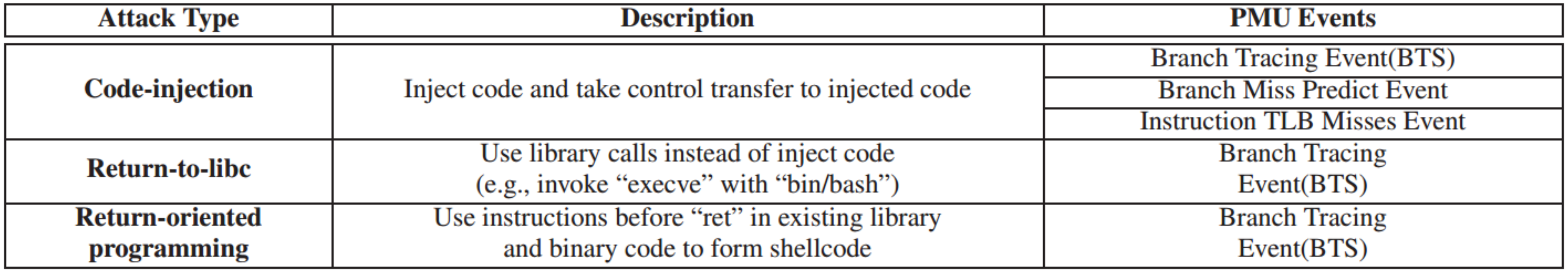
- Branch Instructions Retired — Event select C4H, Umask 00H This event counts branch instructions at retirement. It counts the retirement of the last micro-op of a branch instruction.
- All Branch Mispredict Retired — Event select C5H, Umask 00H

Past Research

Security Breaches as PMU Deviation: Detecting and Identifying Security Attacks Using Performance Counters in **APSys'11, 2011**

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

Source: http://ipads.se.sjtu.edu.cn/_media/publications:perf-apsys.pdf



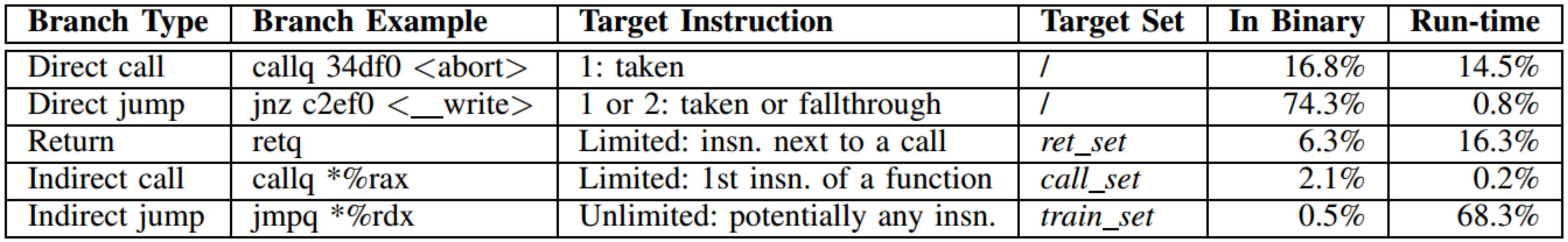
Attack Type	Description	PMU Events
Code-injection	Inject code and take control transfer to injected code	Branch Tracing Event(BTS)
		Branch Miss Predict Event
		Instruction TLB Misses Event
Return-to-libc	Use library calls instead of inject code (e.g., invoke "execve" with "bin/bash")	Branch Tracing Event(BTS)
Return-oriented programming	Use instructions before "ret" in existing library and binary code to form shellcode	Branch Tracing Event(BTS)

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



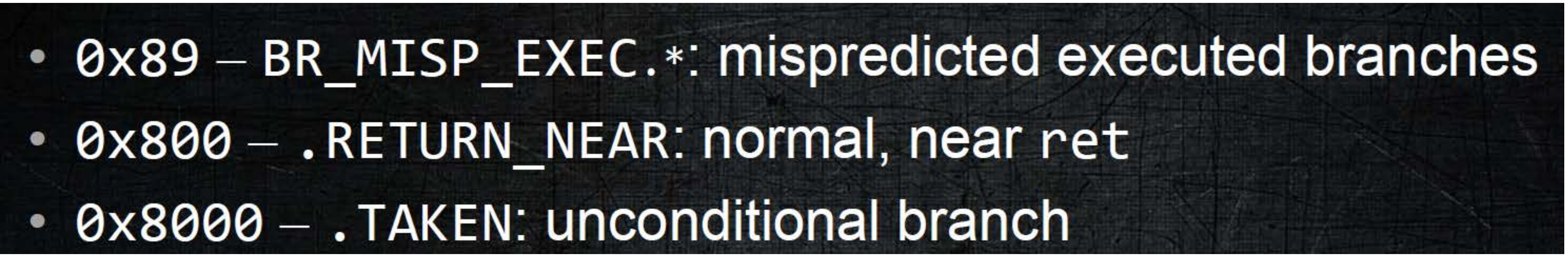
Branch Type	**Branch Example**	**Target Instruction**	**Target Set**	**In Binary**	**Run-time**
Direct call	callq 34df0 <abort>	1: taken	/	16.8%	14.5%
Direct jump	jnz c2ef0 <__write>	1 or 2: taken or fallthrough	/	74.3%	0.8%
Return	retq	Limited: insn. next to a call	*ret_set*	6.3%	16.3%
Indirect call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

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



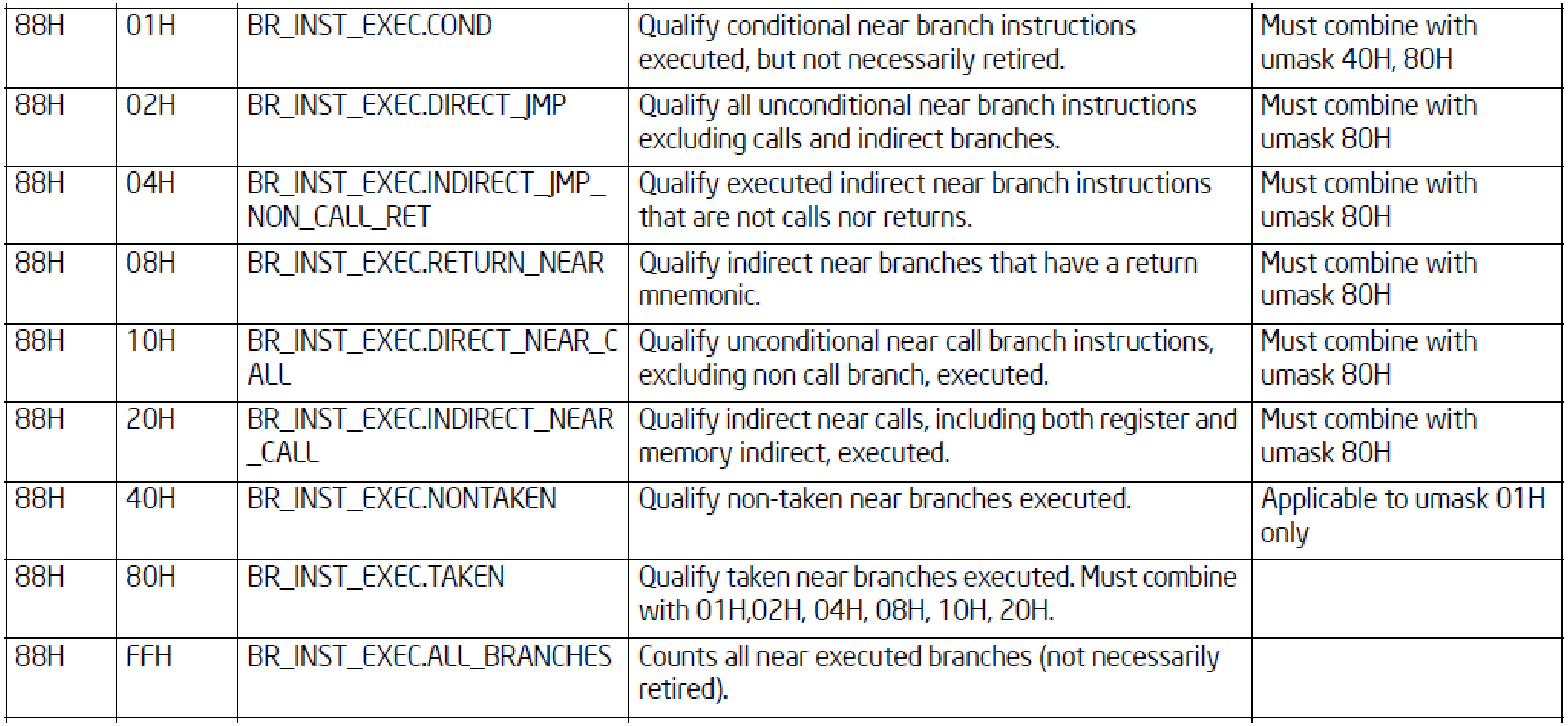
- 0x89 – BR_MISP_EXEC.*: mispredicted executed branches
- 0x800 – .RETURN_NEAR: normal, near ret
- 0x8000 – .TAKEN: unconditional branch

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

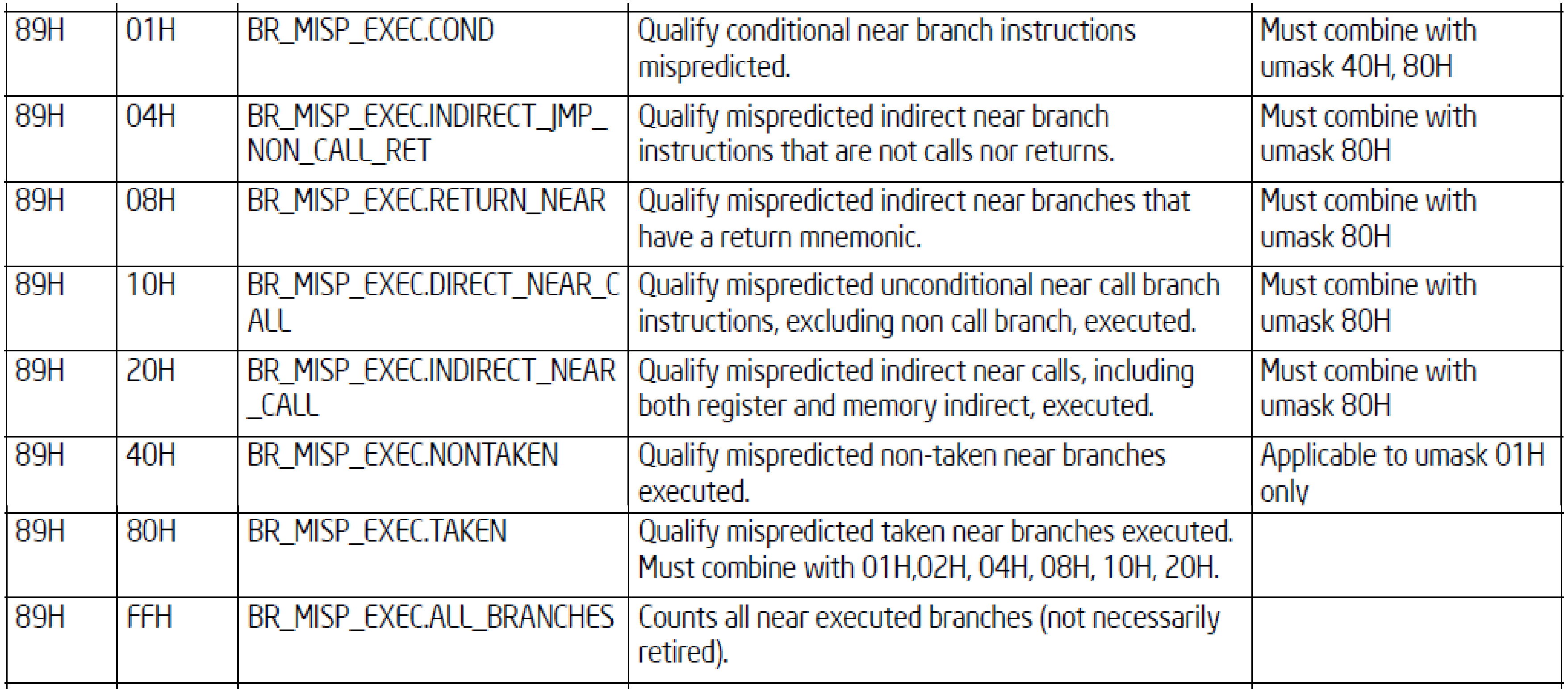
Performance Events in Haswell/Silvermont

BR_INST_EXEC Event and UMask



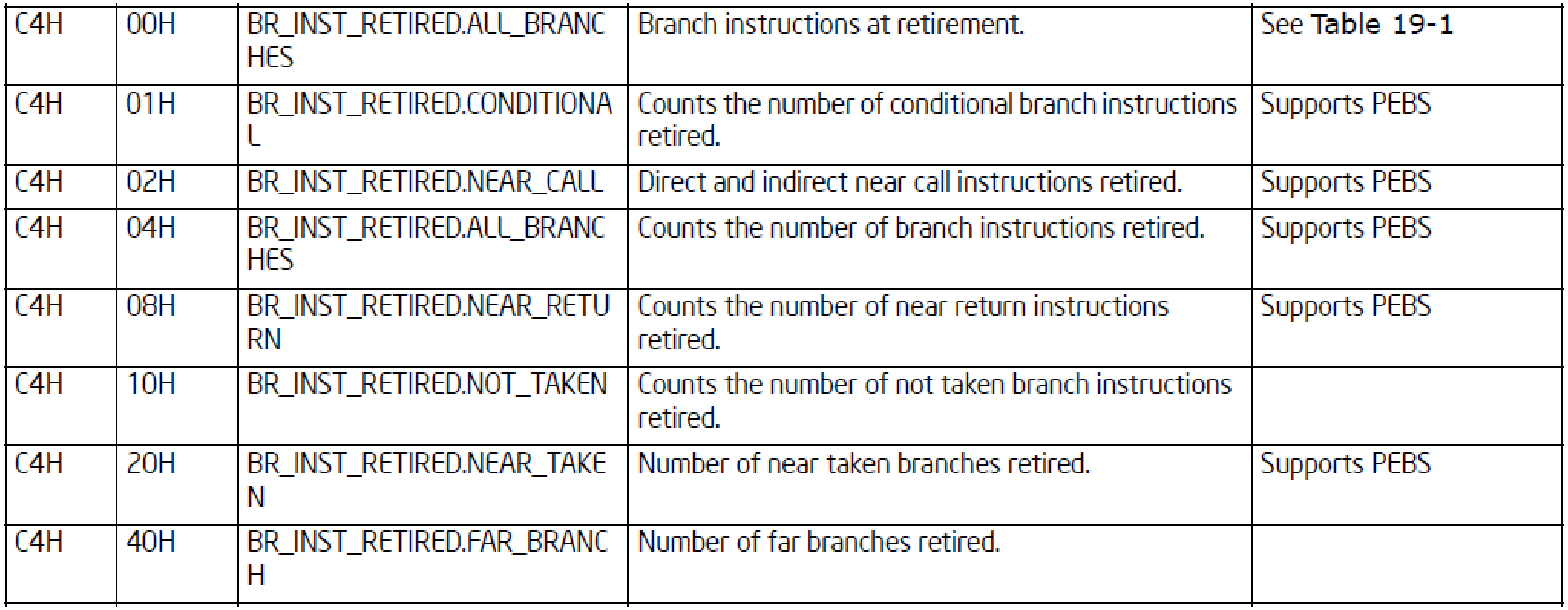
88H	01H	BR_INST_EXEC.COND	Qualify conditional near branch instructions executed, but not necessarily retired.	Must combine with umask 40H, 80H
88H	02H	BR_INST_EXEC.DIRECT_JMP	Qualify all unconditional near branch instructions excluding calls and indirect branches.	Must combine with umask 80H
88H	04H	BR_INST_EXEC.INDIRECT_JMP_NON_CALL_RET	Qualify executed indirect near branch instructions that are not calls nor returns.	Must combine with umask 80H
88H	08H	BR_INST_EXEC.RETURN_NEAR	Qualify indirect near branches that have a return mnemonic.	Must combine with umask 80H
88H	10H	BR_INST_EXEC.DIRECT_NEAR_CALL	Qualify unconditional near call branch instructions, excluding non call branch, executed.	Must combine with umask 80H
88H	20H	BR_INST_EXEC.INDIRECT_NEAR_CALL	Qualify indirect near calls, including both register and memory indirect, executed.	Must combine with umask 80H
88H	40H	BR_INST_EXEC.NONTAKEN	Qualify non-taken near branches executed.	Applicable to umask 01H only
88H	80H	BR_INST_EXEC.TAKEN	Qualify taken near branches executed. Must combine with 01H,02H, 04H, 08H, 10H, 20H.	
88H	FFH	BR_INST_EXEC.ALL_BRANCHES	Counts all near executed branches (not necessarily retired).	

BR_MISP_EXEC Event and UMask



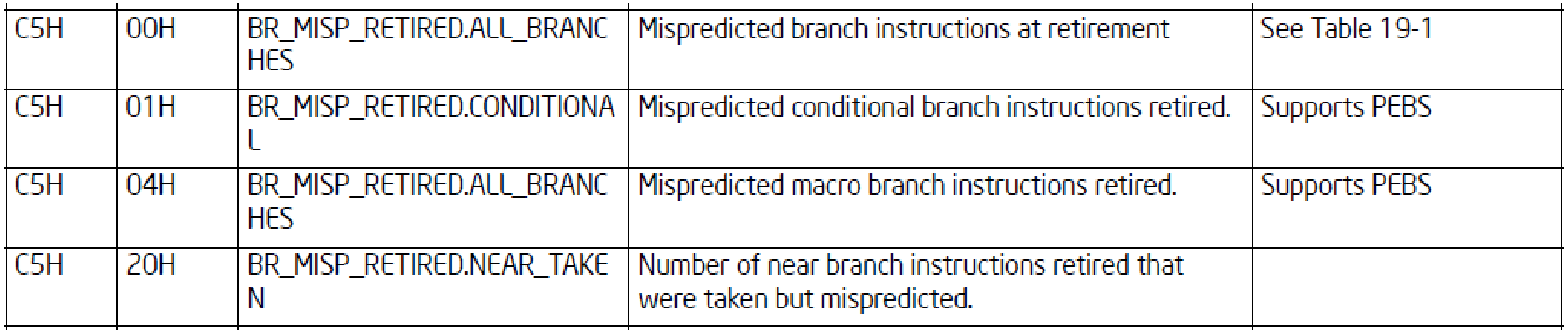
89H	01H	BR_MISP_EXEC.COND	Qualify conditional near branch instructions mispredicted.	Must combine with umask 40H, 80H
89H	04H	BR_MISP_EXEC.INDIRECT_JMP_NON_CALL_RET	Qualify mispredicted indirect near branch instructions that are not calls nor returns.	Must combine with umask 80H
89H	08H	BR_MISP_EXEC.RETURN_NEAR	Qualify mispredicted indirect near branches that have a return mnemonic.	Must combine with umask 80H
89H	10H	BR_MISP_EXEC.DIRECT_NEAR_CALL	Qualify mispredicted unconditional near call branch instructions, excluding non call branch, executed.	Must combine with umask 80H
89H	20H	BR_MISP_EXEC.INDIRECT_NEAR_CALL	Qualify mispredicted indirect near calls, including both register and memory indirect, executed.	Must combine with umask 80H
89H	40H	BR_MISP_EXEC.NONTAKEN	Qualify mispredicted non-taken near branches executed.	Applicable to umask 01H only
89H	80H	BR_MISP_EXEC.TAKEN	Qualify mispredicted taken near branches executed. Must combine with 01H,02H, 04H, 08H, 10H, 20H.	
89H	FFH	BR_MISP_EXEC.ALL_BRANCHES	Counts all near executed branches (not necessarily retired).	

BR_INST_RETIRED Event and UMask



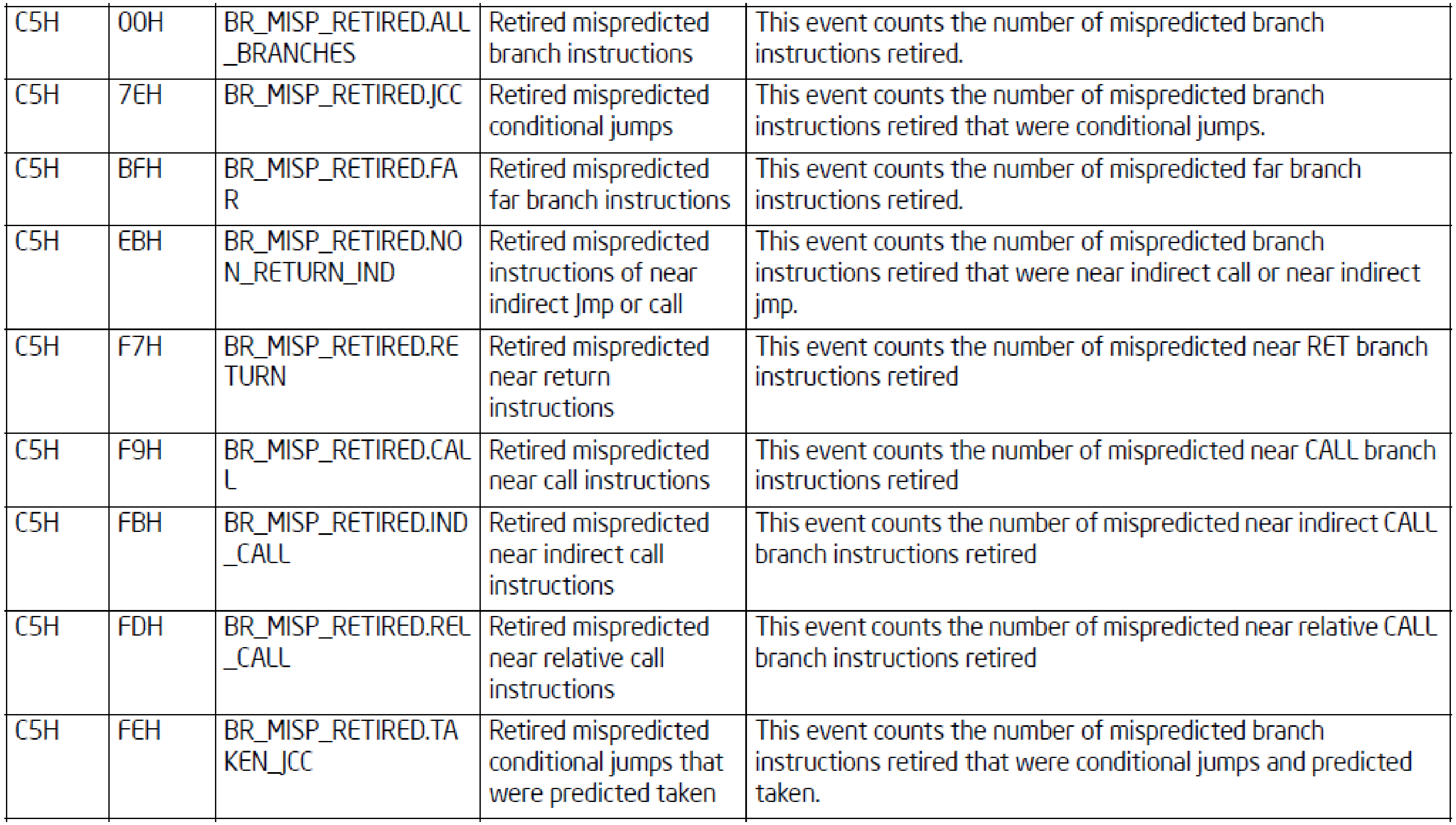
C4H	00H	BR_INST_RETIRED.ALL_BRANCHES	Branch instructions at retirement.	See **Table 19-1**
C4H	01H	BR_INST_RETIRED.CONDITIONAL	Counts the number of conditional branch instructions retired.	Supports PEBS
C4H	02H	BR_INST_RETIRED.NEAR_CALL	Direct and indirect near call instructions retired.	Supports PEBS
C4H	04H	BR_INST_RETIRED.ALL_BRANCHES	Counts the number of branch instructions retired.	Supports PEBS
C4H	08H	BR_INST_RETIRED.NEAR_RETURN	Counts the number of near return instructions retired.	Supports PEBS
C4H	10H	BR_INST_RETIRED.NOT_TAKEN	Counts the number of not taken branch instructions retired.	
C4H	20H	BR_INST_RETIRED.NEAR_TAKEN	Number of near taken branches retired.	Supports PEBS
C4H	40H	BR_INST_RETIRED.FAR_BRANCH	Number of far branches retired.	

BR_MISP_RETIRED Event and UMask



C5H	00H	BR_MISP_RETIRED.ALL_BRANCHES	Mispredicted branch instructions at retirement	See Table 19-1
C5H	01H	BR_MISP_RETIRED.CONDITIONAL	Mispredicted conditional branch instructions retired.	Supports PEBS
C5H	04H	BR_MISP_RETIRED.ALL_BRANCHES	Mispredicted macro branch instructions retired.	Supports PEBS
C5H	20H	BR_MISP_RETIRED.NEAR_TAKEN	Number of near branch instructions retired that were taken but mispredicted.	

BR_MISP_RETIRED In Silvermont



C5H	00H	BR_MISP_RETIRED.ALL_BRANCHES	Retired mispredicted branch instructions	This event counts the number of mispredicted branch instructions retired.
C5H	7EH	BR_MISP_RETIRED.JCC	Retired mispredicted conditional jumps	This event counts the number of mispredicted branch instructions retired that were conditional jumps.
C5H	BFH	BR_MISP_RETIRED.FAR	Retired mispredicted far branch instructions	This event counts the number of mispredicted far branch instructions retired.
C5H	EBH	BR_MISP_RETIRED.NON_RETURN_IND	Retired mispredicted instructions of near indirect Jmp or call	This event counts the number of mispredicted branch instructions retired that were near indirect call or near indirect jmp.
C5H	F7H	BR_MISP_RETIRED.RETURN	Retired mispredicted near return instructions	This event counts the number of mispredicted near RET branch instructions retired
C5H	F9H	BR_MISP_RETIRED.CALL	Retired mispredicted near call instructions	This event counts the number of mispredicted near CALL branch instructions retired
C5H	FBH	BR_MISP_RETIRED.IND_CALL	Retired mispredicted near indirect call instructions	This event counts the number of mispredicted near indirect CALL branch instructions retired
C5H	FDH	BR_MISP_RETIRED.REL_CALL	Retired mispredicted near relative call instructions	This event counts the number of mispredicted near relative CALL branch instructions retired
C5H	FEH	BR_MISP_RETIRED.TAKEN_JCC	Retired mispredicted conditional jumps that were predicted taken	This event counts the number of mispredicted branch instructions retired that were conditional jumps and predicted taken.

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**	Indirect CALL misprediction
First unintended/intended exploit code trigged by **indirect JMP**	Indirect JMP misprediction
First unintended/intended exploit code trigged by **RET**	RET misprediction
Following unintended/intended ROP gadgets	Indirect CALL/JMP, RET misprediction

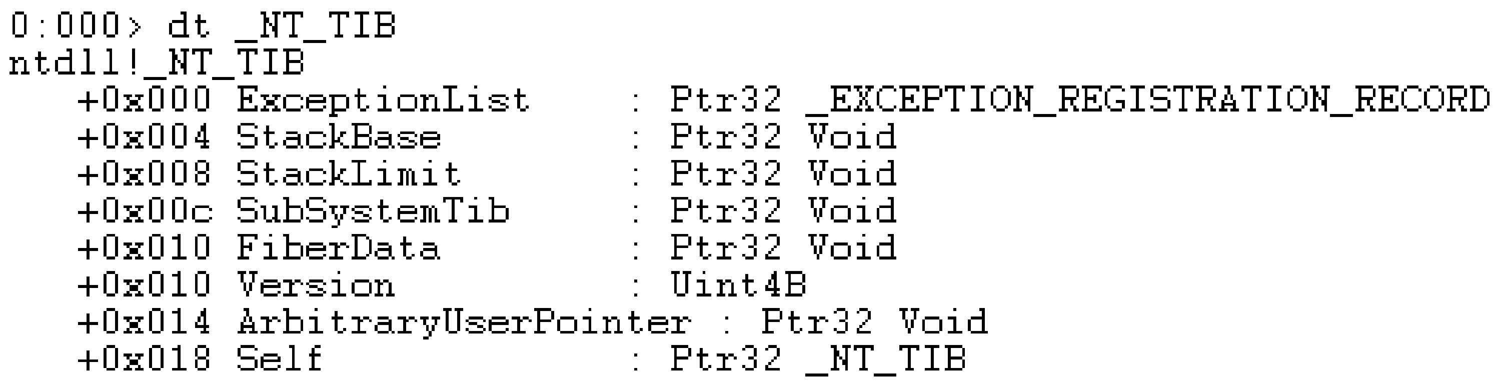
Stack Pivoting Detection Overview

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

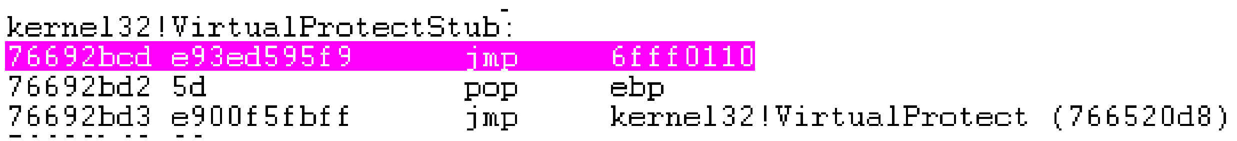


```
0:000> dt _NT_TIB
ntdll!_NT_TIB
   +0x000 ExceptionList    : Ptr32 _EXCEPTION_REGISTRATION_RECORD
   +0x004 StackBase        : Ptr32 Void
   +0x008 StackLimit       : Ptr32 Void
   +0x00c SubSystemTib     : Ptr32 Void
   +0x010 FiberData        : Ptr32 Void
   +0x010 Version          : Uint4B
  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

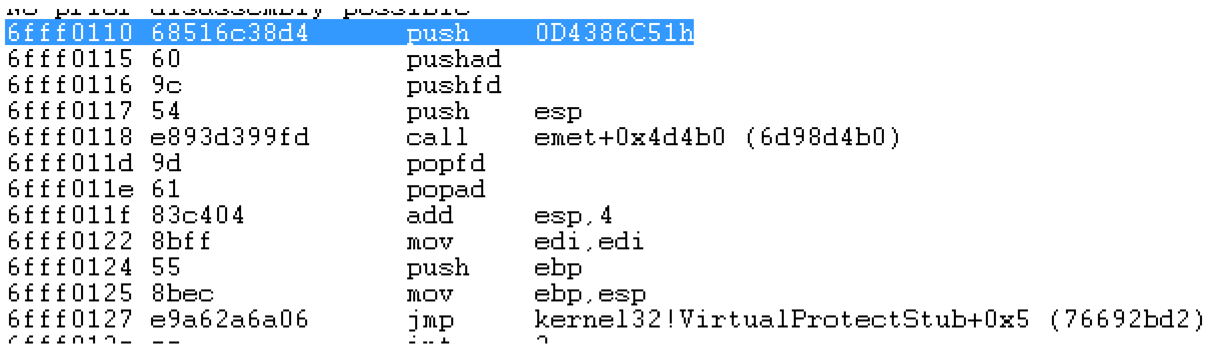
Hooked API Examples IN EMET

- 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



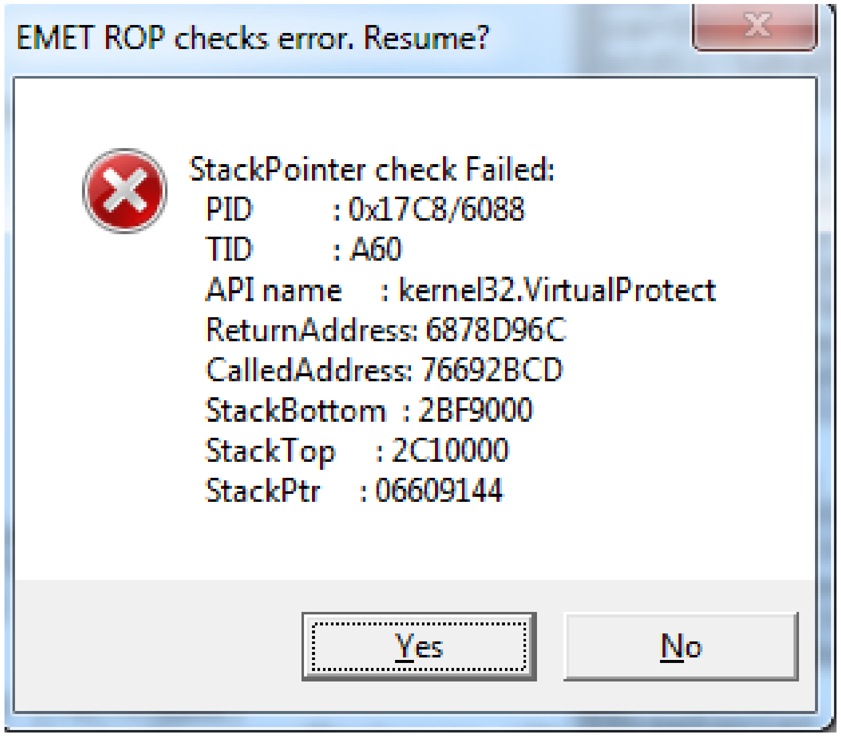
```
kernel32!VirtualProtectStub:
76692bcd e93ed595f9       jmp     6fff0110
76692bd2 5d               pop     ebp
76692bd3 e900f5fbff       jmp     kernel32!VirtualProtect (766520d8)
```



```
6fff0110 68516c38d4       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)
```

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
```

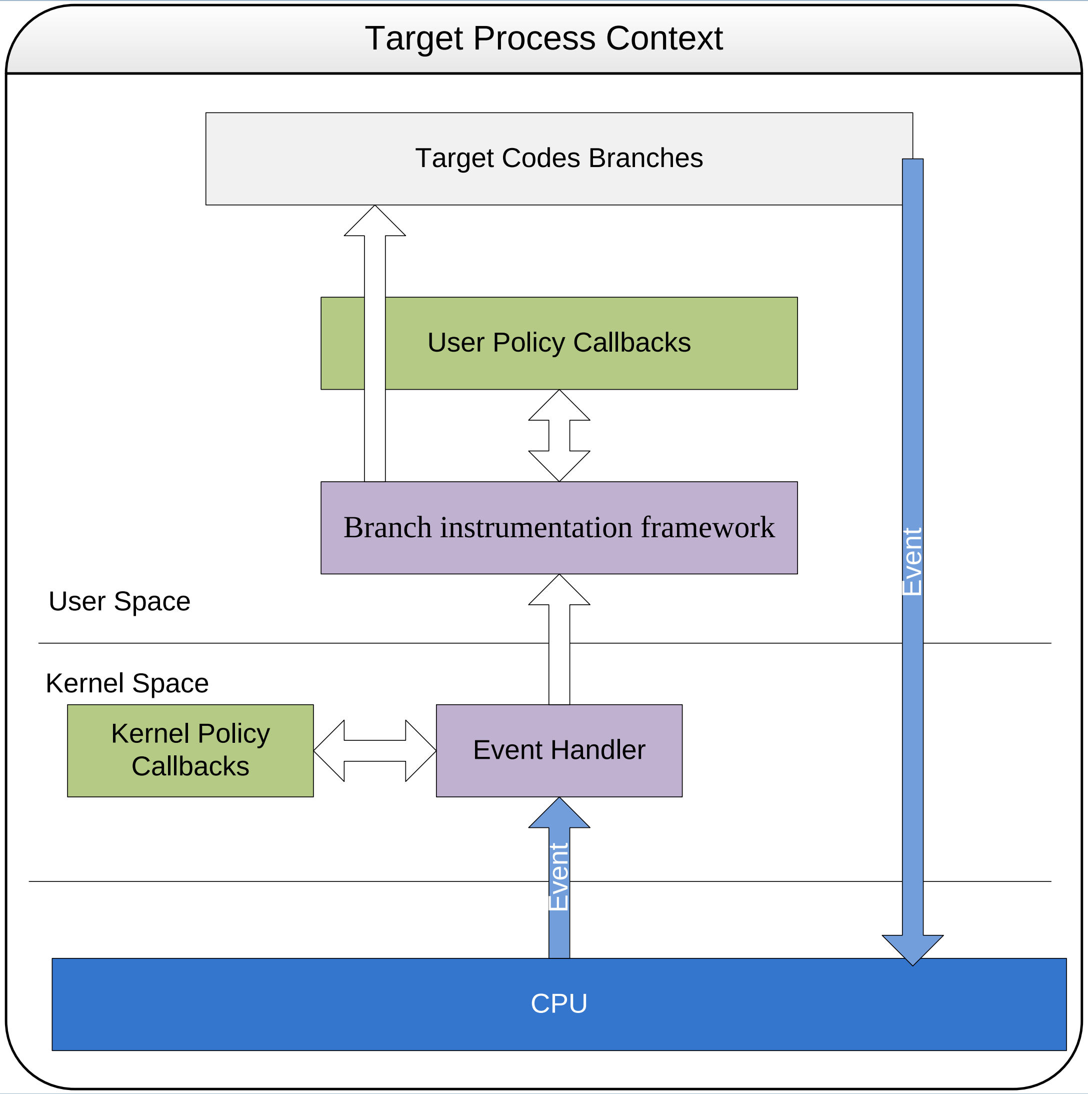
```
}
```

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

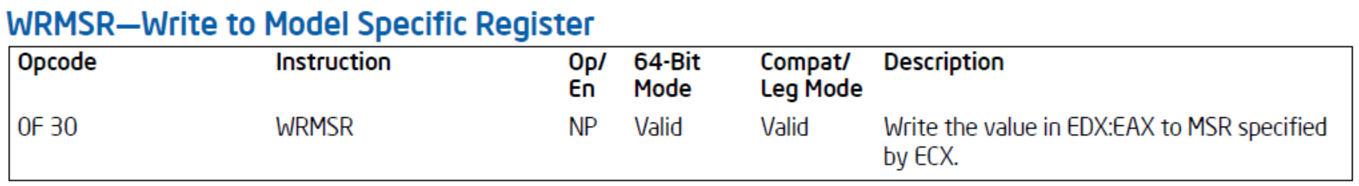
- MSR Programming
- Counter configuration
- Register Event Handler
- Interrupt Context on Stack
- Interrupt Event Handler

Write MSR

- Compiler Intrinsics

  void __writemsr(unsigned long Register, unsigned __int64 Value);
- Inline ASM

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



WRMSR—Write to Model Specific Register

Opcode	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
0F 30	WRMSR	NP	Valid	Valid	Write the value in EDX:EAX to MSR specified by ECX.

MSR Programming

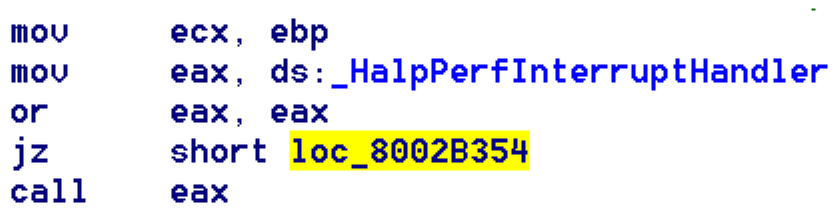
- Use PMC0 to trigger PMI
  1. IA32_DEBUGCTL
  2. LBR_SELECT
  3. MSR_PERF_GLOBAL_CTRL
  4. PMC0
  5. IA32_PERFEVTSEL0
  6. MSR_PERF_GLOBAL_CTRL

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



```
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

```
typedef struct {
        UINT32  stacktop;
        UINT32  intno;
        UINT32  esp;
        UINT32  eip;
        UINT32  Data[24];
        UINT32  int_eip;
        UINT32  int_cs;
        UINT32  int_eflag;
        UINT32  int_esp;
        UINT32  int_ss;
} IA32_PERFINTERRUPT_PARAMETER;
```

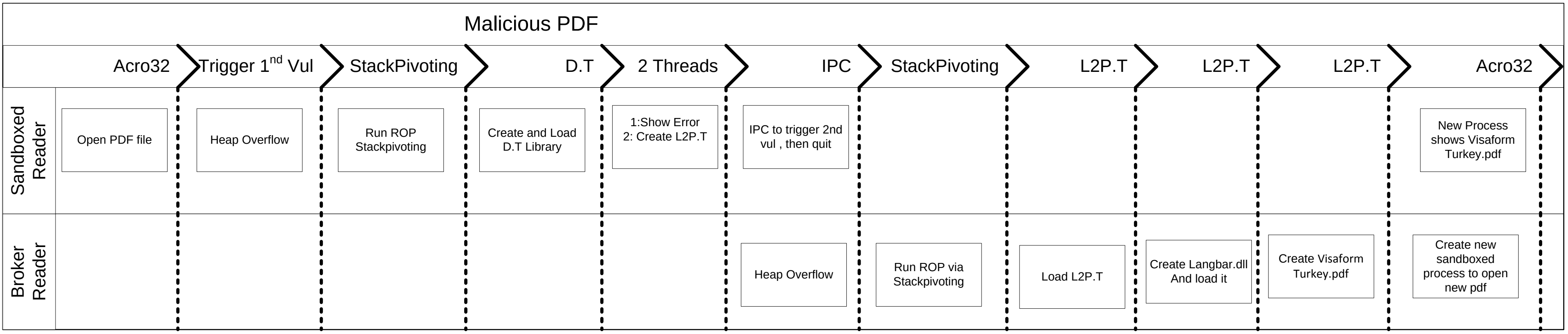
Interrupt Event Handler

- 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

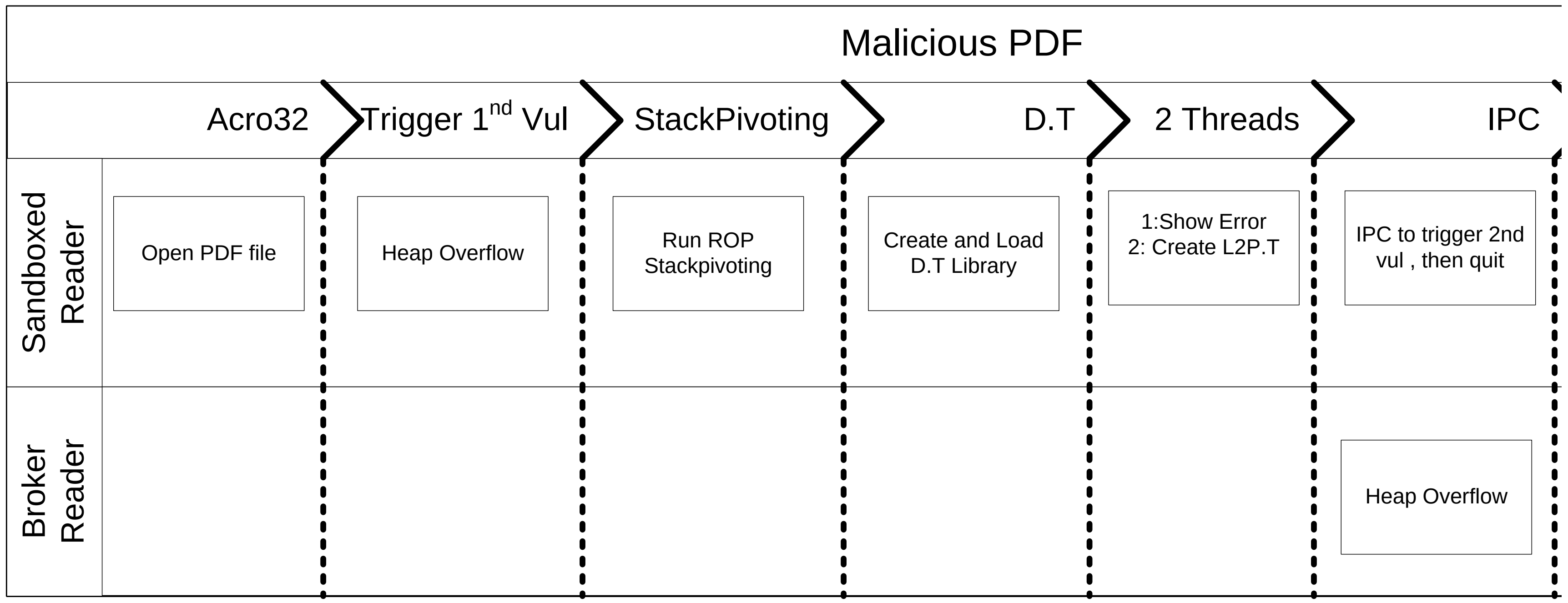
APSA13-02 PDF 0-day

- 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



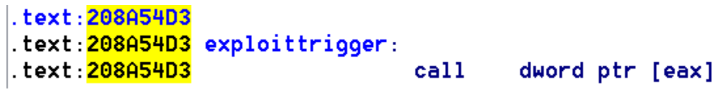
Malicious PDF
Acro32
Trigger 1nd Vul
StackPivoting
D.T
2 Threads
IPC
StackPivoting
L2P.T
L2P.T
L2P.T
Acro32
Sandboxed Reader
Broker Reader
Open PDF file
Heap Overflow
Run ROP Stackpivoting
Create and Load D.T Library
1:Show Error 2: Create L2P.T
IPC to trigger 2nd vul , then quit
New Process shows Visaform Turkey.pdf
Heap Overflow
Run ROP via Stackpivoting
Load L2P.T
Create Langbar.dll And load it
Create Visaform Turkey.pdf
Create new sandboxed process to open new pdf

CVE-2013-0640 Exploit



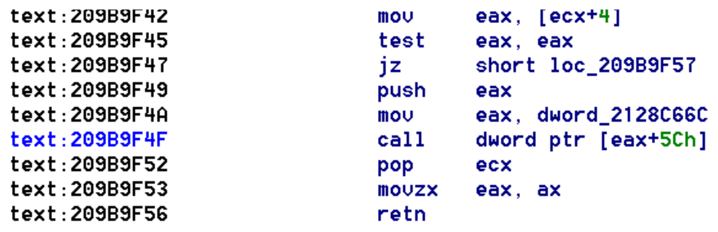
CVE-2013-0640 PDF 0-day analysis (1)

- Trigger Point in AcroForm.api

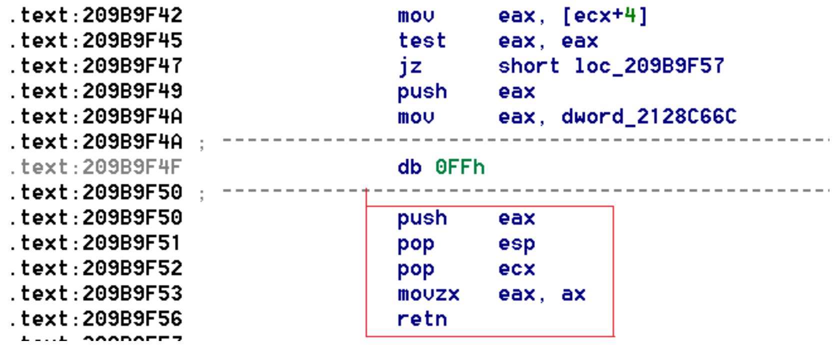


```
.text:208A54D3
.text:208A54D3 exploittrigger:
.text:208A54D3                         call    dword ptr [eax]
```

- Stack Pivoting



```
text:209B9F42                 mov     eax, [ecx+4]
text:209B9F45                 test    eax, eax
text:209B9F47                 jz      short loc_209B9F57
text:209B9F49                 push    eax
text:209B9F4A                 mov     eax, dword_2128C66C
text:209B9F4F                 call    dword ptr [eax+5Ch]
text:209B9F52                 pop     ecx
text:209B9F53                 movzx   eax, ax
text:209B9F56                 retn
```

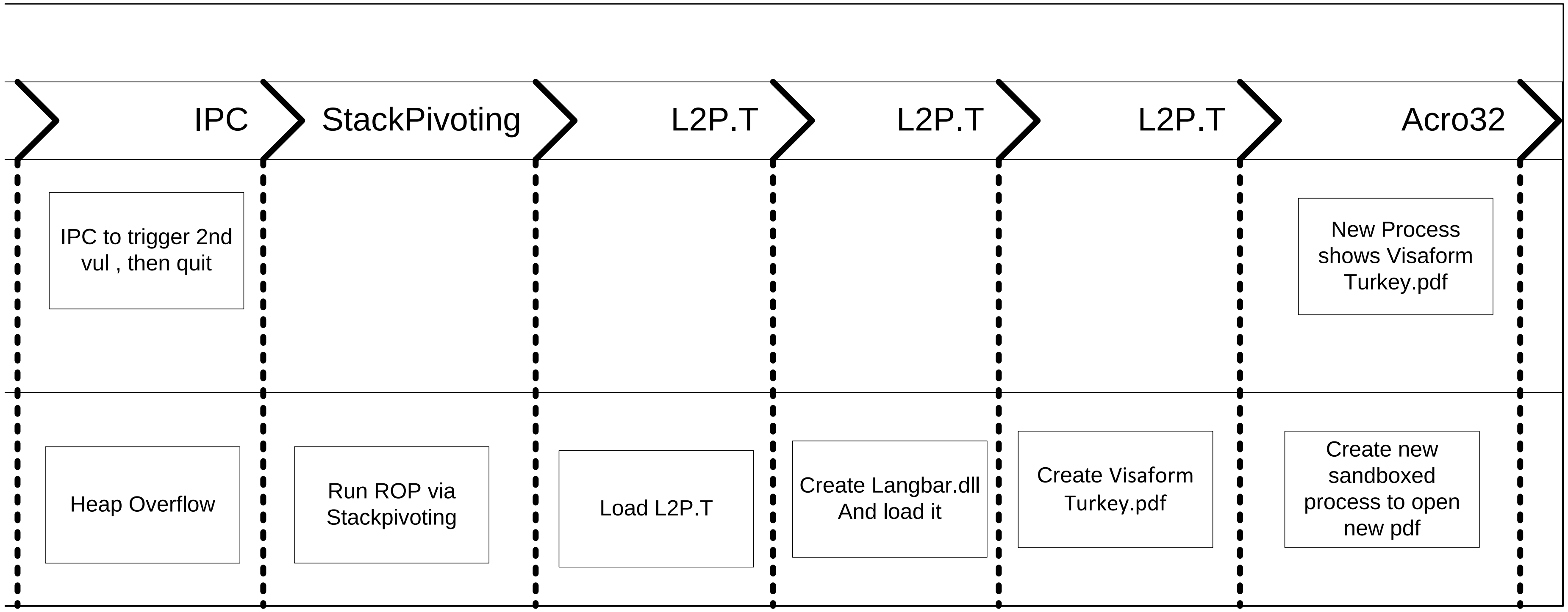


```
.text:209B9F42                mov     eax, [ecx+4]
.text:209B9F45                test    eax, eax
.text:209B9F47                jz      short loc_209B9F57
.text:209B9F49                push    eax
.text:209B9F4A                mov     eax, dword_2128C66C
.text:209B9F4A ; ---------------------------------------------------
.text:209B9F4F                db 0FFh
.text:209B9F50 ; ---------------------------------------------------
.text:209B9F50                push    eax
.text:209B9F51                pop     esp
.text:209B9F52                pop     ecx
.text:209B9F53                movzx   eax, ax
.text:209B9F56                retn
```

CVE-2013-0640 PDF 0-day analysis (2)

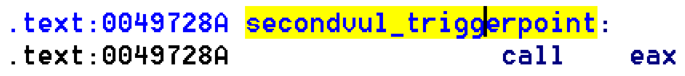
- Lots of ROP gadgets
  - Get ntdll.dll and get related API address
    - for example, RtlDecompressBuffer and CryptStringToBinaryA
  - Call CryptStringToBinaryA to convert one string to binrary
  - Call RtlDecompressBuffer to decompress binary to real D.T binary code in memory
  - Finally, the ROP gadget 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
StackPivoting
L2P.T
L2P.T
L2P.T
Acro32
IPC to trigger 2nd vul , then quit
New Process shows Visaform Turkey.pdf
Heap Overflow
Run ROP via Stackpivoting
Load L2P.T
Create Langbar.dll And load it
Create Visaform Turkey.pdf
Create new sandboxed process to open new pdf

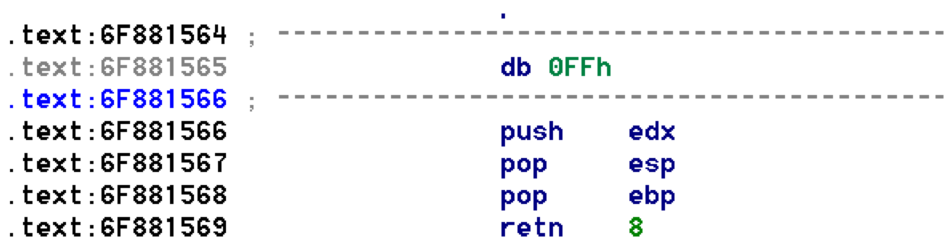
CVE-2013-0641 PDF 0-day analysis (1)

- Trigger Point in acrord32.exe



```
.text:0049728A secondvul_triggerpoint:
.text:0049728A                 call    eax
```

- Stack Pivoting



```
.text:6F881564 ; ---------------------------------------
.text:6F881565                 db 0FFh
.text:6F881566 ; ---------------------------------------
.text:6F881566                 push    edx
.text:6F881567                 pop     esp
.text:6F881568                 pop     ebp
.text:6F881569                 retn    8
```

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

Demo

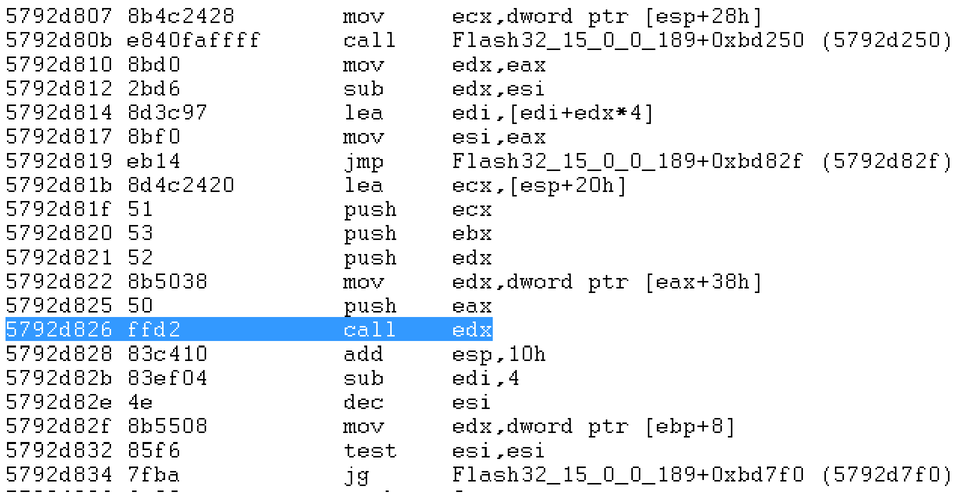
Misc

- 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

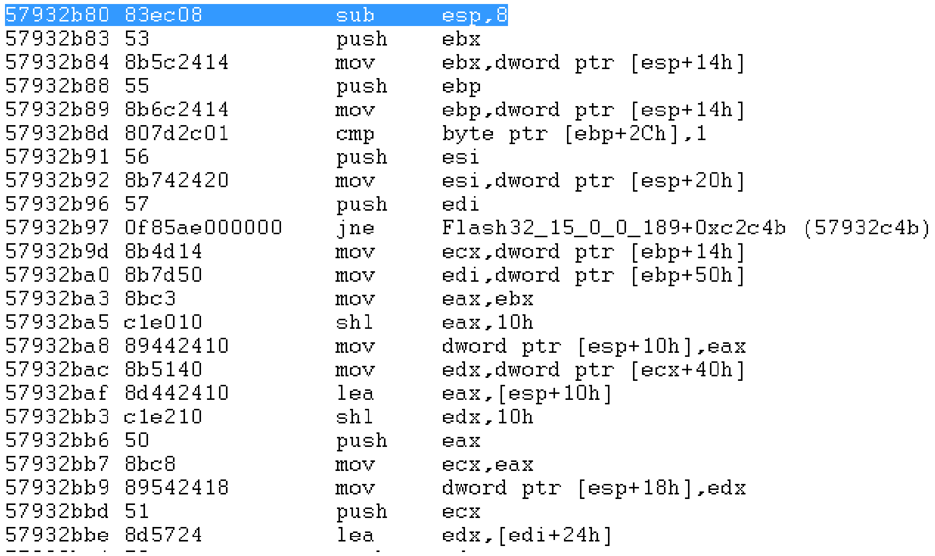
Interrupt EIP and Event EIP

- The Interrupt could happen on different EIP from Event EIP

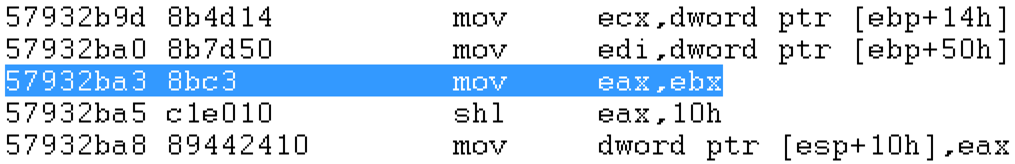
FromAddr=0x5792d826 **IntAddr=0x57932ba3** **ToAddr=0x57932b80**



```
5792d807 8b4c2428        mov     ecx,dword ptr [esp+28h]
5792d80b e840faffff      call    Flash32_15_0_0_189+0xbd250 (5792d250)
5792d810 8bd0            mov     edx,eax
5792d812 2bd6            sub     edx,esi
5792d814 8d3c97          lea     edi,[edi+edx*4]
5792d817 8bf0            mov     esi,eax
5792d819 eb14            jmp     Flash32_15_0_0_189+0xbd82f (5792d82f)
5792d81b 8d4c2420        lea     ecx,[esp+20h]
5792d81f 51              push    ecx
5792d820 53              push    ebx
5792d821 52              push    edx
5792d822 8b5038          mov     edx,dword ptr [eax+38h]
5792d825 50              push    eax
5792d826 ffd2            call    edx
5792d828 83c410          add     esp,10h
5792d82b 83ef04          sub     edi,4
5792d82e 4e              dec     esi
5792d82f 8b5508          mov     edx,dword ptr [ebp+8]
5792d832 85f6            test    esi,esi
5792d834 7fba            jg      Flash32_15_0_0_189+0xbd7f0 (5792d7f0)
```

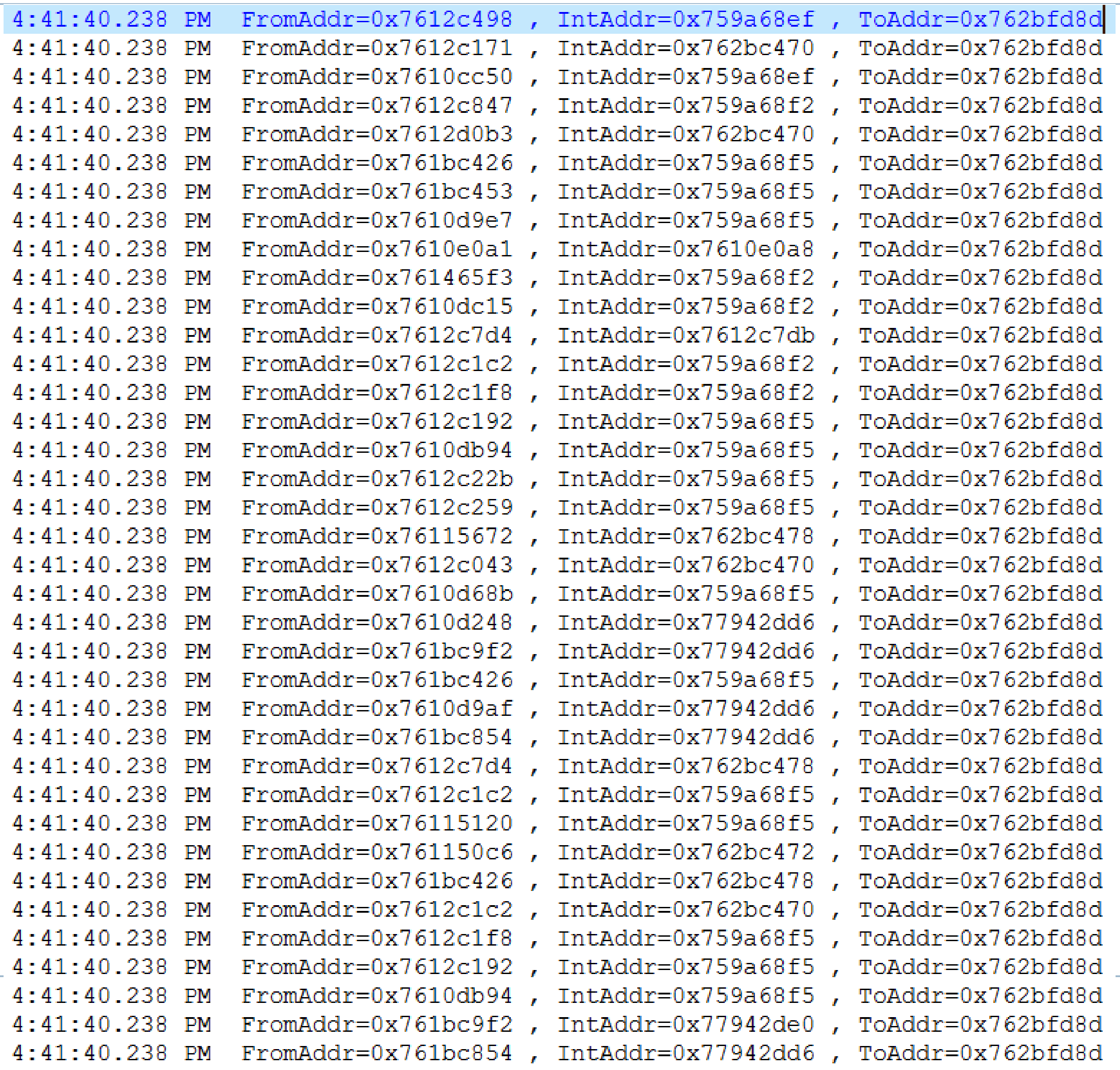


```
57932b80 83ec08          sub     esp,8
57932b83 53              push    ebx
57932b84 8b5c2414        mov     ebx,dword ptr [esp+14h]
57932b88 55              push    ebp
57932b89 8b6c2414        mov     ebp,dword ptr [esp+14h]
57932b8d 807d2c01        cmp     byte ptr [ebp+2Ch],1
57932b91 56              push    esi
57932b92 8b742420        mov     esi,dword ptr [esp+20h]
57932b96 57              push    edi
57932b97 0f85ae000000    jne     Flash32_15_0_0_189+0xc2c4b (57932c4b)
57932b9d 8b4d14          mov     ecx,dword ptr [ebp+14h]
57932ba0 8b7d50          mov     edi,dword ptr [ebp+50h]
57932ba3 8bc3            mov     eax,ebx
57932ba5 c1e010          shl     eax,10h
57932ba8 89442410        mov     dword ptr [esp+10h],eax
57932bac 8b5140          mov     edx,dword ptr [ecx+40h]
57932baf 8d442410        lea     eax,[esp+10h]
57932bb3 c1e210          shl     edx,10h
57932bb6 50              push    eax
57932bb7 8bc8            mov     ecx,eax
57932bb9 89542418        mov     dword ptr [esp+18h],edx
57932bbd 51              push    ecx
57932bbe 8d5724          lea     edx,[edi+24h]
```



```
57932b9d 8b4d14          mov     ecx,dword ptr [ebp+14h]
57932ba0 8b7d50          mov     edi,dword ptr [ebp+50h]
57932ba3 8bc3            mov     eax,ebx
57932ba5 c1e010          shl     eax,10h
57932ba8 89442410        mov     dword ptr [esp+10h],eax
```

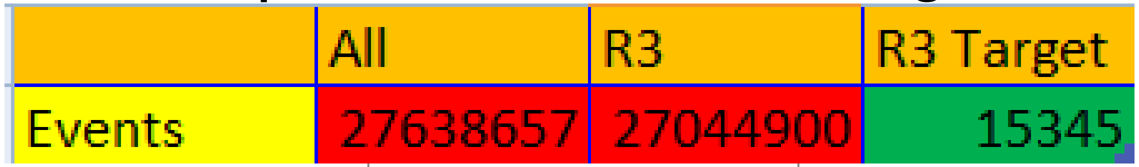
Repeated Indirect Call Misprediction



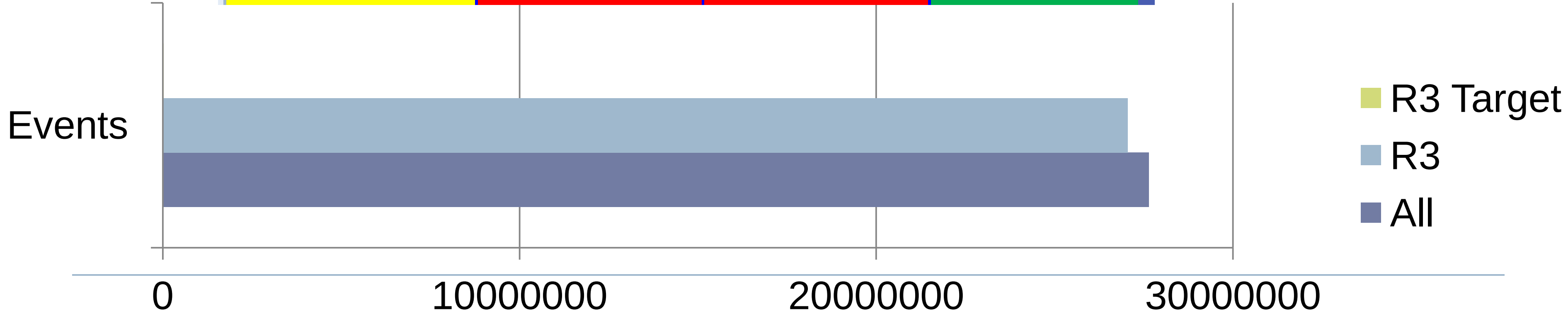
```
4:41:40.238 PM  FromAddr=0x7612c498 , IntAddr=0x759a68ef , ToAddr=0x762bfd8d
4:41:40.238 PM  FromAddr=0x7612c171 , IntAddr=0x762bc470 , ToAddr=0x762bfd8d
4:41:40.238 PM  FromAddr=0x7610cc50 , IntAddr=0x759a68ef , ToAddr=0x762bfd8d
4:41:40.238 PM  FromAddr=0x7612c847 , IntAddr=0x759a68f2 , ToAddr=0x762bfd8d
4:41:40.238 PM  FromAddr=0x7612d0b3 , IntAddr=0x762bc470 , ToAddr=0x762bfd8d
4:41:40.238 PM  FromAddr=0x761bc426 , IntAddr=0x759a68f5 , ToAddr=0x762bfd8d
4:41:40.238 PM  FromAddr=0x761bc453 , IntAddr=0x759a68f5 , ToAddr=0x762bfd8d
4:41:40.238 PM  FromAddr=0x7610d9e7 , IntAddr=0x759a68f5 , ToAddr=0x762bfd8d
4:41:40.238 PM  FromAddr=0x7610e0a1 , IntAddr=0x7610e0a8 , ToAddr=0x762bfd8d
4:41:40.238 PM  FromAddr=0x761465f3 , IntAddr=0x759a68f2 , ToAddr=0x762bfd8d
4:41:40.238 PM  FromAddr=0x7610dc15 , IntAddr=0x759a68f2 , ToAddr=0x762bfd8d
4:41:40.238 PM  FromAddr=0x7612c7d4 , IntAddr=0x7612c7db , ToAddr=0x762bfd8d
4:41:40.238 PM  FromAddr=0x7612c1c2 , IntAddr=0x759a68f2 , ToAddr=0x762bfd8d
4:41:40.238 PM  FromAddr=0x7612c1f8 , IntAddr=0x759a68f2 , ToAddr=0x762bfd8d
4:41:40.238 PM  FromAddr=0x7612c192 , IntAddr=0x759a68f5 , ToAddr=0x762bfd8d
4:41:40.238 PM  FromAddr=0x7610db94 , IntAddr=0x759a68f5 , ToAddr=0x762bfd8d
4:41:40.238 PM  FromAddr=0x7612c22b , IntAddr=0x759a68f5 , ToAddr=0x762bfd8d
4:41:40.238 PM  FromAddr=0x7612c259 , IntAddr=0x759a68f5 , ToAddr=0x762bfd8d
4:41:40.238 PM  FromAddr=0x76115672 , IntAddr=0x762bc478 , ToAddr=0x762bfd8d
4:41:40.238 PM  FromAddr=0x7612c043 , IntAddr=0x762bc470 , ToAddr=0x762bfd8d
4:41:40.238 PM  FromAddr=0x7610d68b , IntAddr=0x759a68f5 , ToAddr=0x762bfd8d
4:41:40.238 PM  FromAddr=0x7610d248 , IntAddr=0x77942dd6 , ToAddr=0x762bfd8d
4:41:40.238 PM  FromAddr=0x761bc9f2 , IntAddr=0x77942dd6 , ToAddr=0x762bfd8d
4:41:40.238 PM  FromAddr=0x761bc426 , IntAddr=0x759a68f5 , ToAddr=0x762bfd8d
4:41:40.238 PM  FromAddr=0x7610d9af , IntAddr=0x77942dd6 , ToAddr=0x762bfd8d
4:41:40.238 PM  FromAddr=0x761bc854 , IntAddr=0x77942dd6 , ToAddr=0x762bfd8d
4:41:40.238 PM  FromAddr=0x7612c7d4 , IntAddr=0x762bc478 , ToAddr=0x762bfd8d
4:41:40.238 PM  FromAddr=0x7612c1c2 , IntAddr=0x759a68f5 , ToAddr=0x762bfd8d
4:41:40.238 PM  FromAddr=0x76115120 , IntAddr=0x759a68f5 , ToAddr=0x762bfd8d
4:41:40.238 PM  FromAddr=0x761150c6 , IntAddr=0x762bc472 , ToAddr=0x762bfd8d
4:41:40.238 PM  FromAddr=0x761bc426 , IntAddr=0x762bc478 , ToAddr=0x762bfd8d
4:41:40.238 PM  FromAddr=0x7612c1c2 , IntAddr=0x762bc470 , ToAddr=0x762bfd8d
4:41:40.238 PM  FromAddr=0x7612c1f8 , IntAddr=0x759a68f5 , ToAddr=0x762bfd8d
4:41:40.238 PM  FromAddr=0x7612c192 , IntAddr=0x759a68f5 , ToAddr=0x762bfd8d
4:41:40.238 PM  FromAddr=0x7610db94 , IntAddr=0x759a68f5 , ToAddr=0x762bfd8d
4:41:40.238 PM  FromAddr=0x761bc9f2 , IntAddr=0x77942de0 , ToAddr=0x762bfd8d
4:41:40.238 PM  FromAddr=0x761bc854 , IntAddr=0x77942dd6 , ToAddr=0x762bfd8d
```

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



	All	R3	R3 Target
Events	27638657	27044900	15345



LBR Record

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

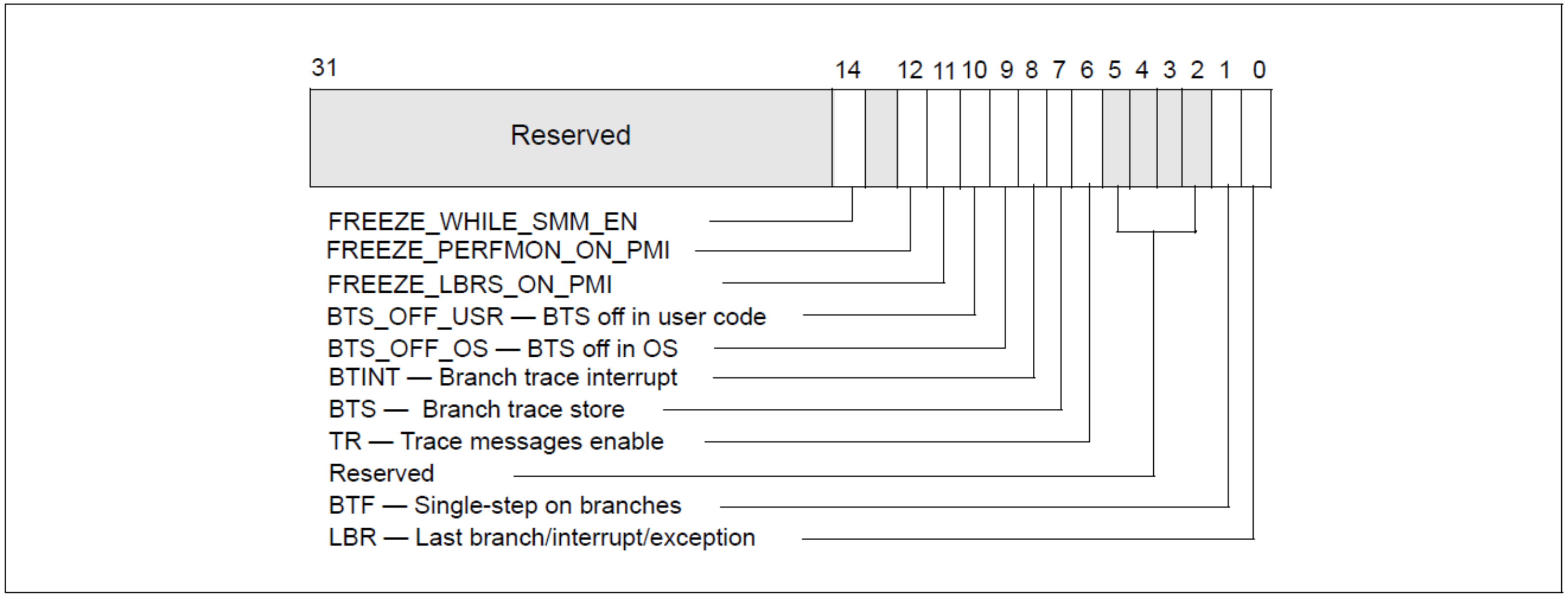

31
14
12
11
10
9
8
7
6
5
4
3
2
1
0
Reserved
FREEZE_WHILE_SMM_EN
FREEZE_PERFMON_ON_PMI
FREEZE_LBRS_ON_PMI
BTS_OFF_USR — BTS off in user code
BTS_OFF_OS — BTS off in OS
BTINT — Branch trace interrupt
BTS — Branch trace store
TR — Trace messages enable
Reserved
BTF — Single-step on branches
LBR — Last branch/interrupt/exception

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

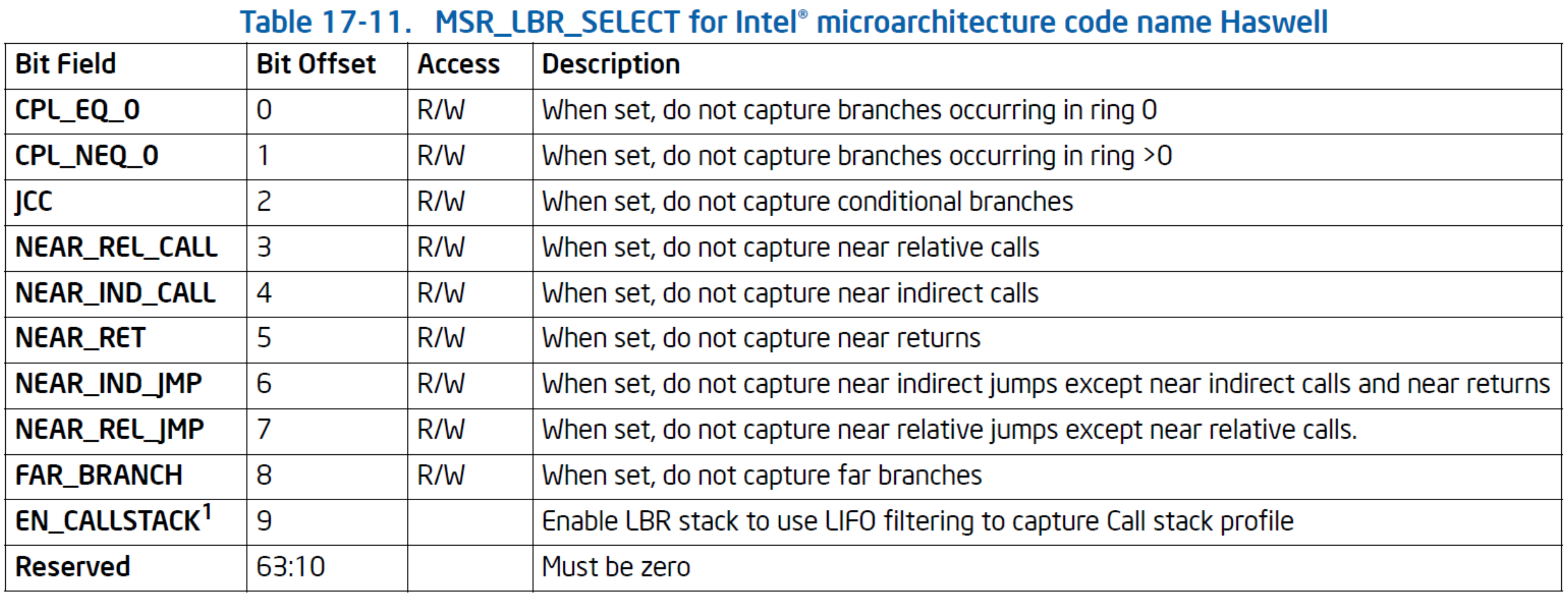


Table 17-11. MSR_LBR_SELECT for Intel® microarchitecture code name Haswell

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

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

[1] White Phosphorus Exploit Pack Sayonara ASLR DEP Bypass Technique, http://www.whitephosphorus.org/sayonara.txt

[2] Active Zero-Day Exploit Targets Internet Explorer Flaw, http://blogs.mcafee.com/mcafee-labs/active-zero-day-exploit-targets-internet-explorer-flaw

[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

[5] Attacking the Windows 7/8 Address Space Randomization, http://kingcope.wordpress.com/2013/01/24/attacking-the-windows-78-address-space-randomization

[6] DEP/ASLR bypass without ROP/JIT, Yang Yu, CanSecWest 2013

[7] The BlueHat Prize, http://www.microsoft.com/security/bluehatprize

[8] kBouncer: Efficient and Transparent ROP Mitigation, Vasilis Pappas

[9] Taming the ROPe on Sandy Bridge,Georg Wicherski, SysCan 2013

[10] Security Breaches as PMU Deviation: Detecting and Identifying

Security Attacks Using Performance Counters, Liwei Yuan etc, APSYS 2011

[11] Intel® 64 and IA-32 Architectures Software Developer’s Manual Combined Volumes:1, 2A, 2B, 2C, 3A, 3B, and 3C, www.intel.com

[12] PRACTICAL RETURN-ORIENTED PROGRAMMING, Dino Dai Zovi, RSA 2010

[13] The Past, Present, and Future of Flash Security, Haifei Li, SysCan360 2012

[14] return oriented exploitation, Dino Dai Zovi, Blackhat 2010

[15] Security Mitigations for Return-Oriented Programming Attacks, Piotr Bania, http://piotrbania.com/all/articles/pbania_rop_mitigations2010.pdf

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