

Your Security X- Ray:

**A Systematic Approach of Finding
Exploitable Spots with Big Data in Linux**

Source Seattle 2017

Ke Sun Ya Ou Li Shen

Chenyang Li Xiaoning Li



Agenda

- **Introduction**

- Motivation
- X86 Performance Monitor Feature

- **Analysis Approach**

- FE: Interrupt-based hardware-level instrumentation
- BE: big-data analysis
- Policy settings

- **Work in Windows**

- Control Flow Guard Bypass

- **Work in Linux**

- Framework enabling in Linux
- Preliminary Results

- **Summary & Future Work**

Agenda

- **Introduction**

- Motivation
- X86 Performance Monitor Feature

- **Analysis Approach**

- FE: Interrupt-based hardware-level instrumentation
- BE: big-data analysis
- Policy settings

- **Work in Windows**

- Control Flow Guard Bypass

- **Work in Linux**

- Framework enabling in Linux
- Preliminary Results

- **Summary & Future Work**

Introduction : Motivation

- A major focus of today's security mitigation is to cut the path between memory corruption and control flow hijacking

Mitigations:

- CFG (Microsoft)
- RAP (PaX)
- CET (Intel)

**Memory
Corruption**



**Control Flow
Hijacking**

Attack surfaces:

- Unprotected return address
 - Writable function pointer
 - RWX memory region
 - "Legal Gadgets"
-

Introduction : Motivation

- Therefore it is important to identify these attack surfaces with a systematic approach
 - To have an accurate and quantitative profiling of the potentially exploitable points
 - To understand what kind of mitigations are needed/effective for a given program
 - To evaluate the effectiveness of mitigation techniques that already implemented
-

Introduction : Motivation

- Static analysis may help but...



- *JITed Code*
- *Interpreted Code*
- *Self-modifying Code*
- *Obfuscation*
- ...

Introduction : Dynamic Analysis

- Dynamic Binary Instrumentation (DBI)
 - DBI is a powerful tool for runtime behavior analysis
 - DBI also has limitations for security-related usage
 - Non-zero impact to the native behavior of the target binary
 - Not-fully transparent (detectable) to the target binary
 - SMC handling
 - Risk of being pwned...

BREAK OUT OF THE TRUMAN SHOW

ACTIVE DETECTION AND ESCAPE OF DYNAMIC
BINARY INSTRUMENTATION

Ke Sun
Xiaoning Li
Ya Ou

wildsator@gmail.com
ldpatchguard@gmail.com
perfectno2015@gmail.com

Introduction : Dynamic Analysis

➤ Our solution:

Interrupt-based hardware-level instrumentation



Big data analysis

Introduction : Performance Monitoring Units




➤ Performance Monitor Units:

A hardware feature of IA processor for monitoring performance using counting or interrupt-based event sampling



**Intel® 64 and IA-32 Architectures
Software Developer's Manual**

**Volume 3 (3A, 3B, 3C & 3D):
System Programming Guide**

-  Chapter 17 Debug, Branch Profile, TSC, and Resource Monitoring Features
-  Chapter 18 Performance Monitoring
-  Chapter 19 Performance-Monitoring Events

Introduction : PMU MSRs

- Performance Monitor Units:

Implemented with a set of model-specific registers

- Control MSRs : *enable/disable event counting, LBR, etc*

- IA32_PERF_GLOBAL_CTRL

- IA32_DEBUGCTL

- Configuration MSRs: *event selection/filtering*

- IA32_PERFEVTSELx

- MSR_LBR_SELECT

- Counter MSRs: *events counting, triggers PMI when OF*

- IA32_PMCx

Introduction : PMU-related Researches in Security

- Previous works using PMU in security researches:
 - “Security Breaches as PMU Deviation: Detecting and Identifying Security Attacks Using Performance Counters”, Yuan et al., 2011
 - “CFIMon: Detecting Violation of Control Flow Integrity using Performance Counters”, Xia et al., 2012
 - “kBouncer: Efficient and Transparent ROP Mitigation”, Pappas, 2012
 - “Transparent ROP Detection using CPU Performance Counters”, Li & Crouse, 2014
 - “CPU Hardware Performance Counters for Security”, Herath et al., 2015
 - “Capturing 0Day Exploits with Perfectly Placed Hardware Traps”, Pierce et al., 2016
 - “Go Speed Tracer”, Richard Johnson, 2016
-

Agenda

- **Introduction**

- Motivation
- X86 Performance Monitor Feature

- **Analysis Approach**

- FE: Interrupt-based hardware-level instrumentation
- BE: big-data analysis
- Policy settings

- **Work in Windows**

- Control Flow Guard Bypass

- **Current Work in Linux**

- Framework enabling in Linux
- Preliminary Results

- **Summary & Future Work**

Analysis Approach

- Scope

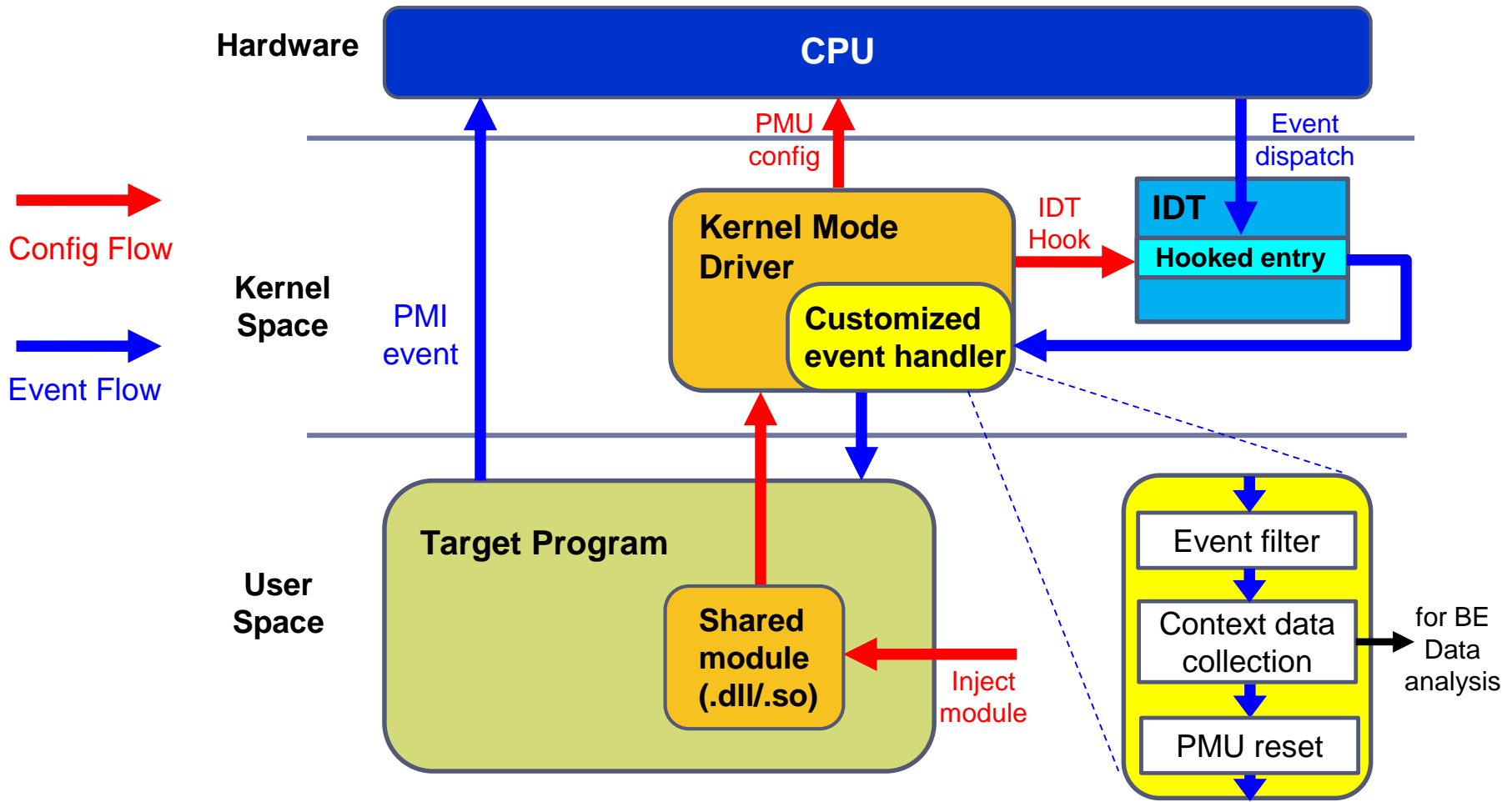
- Attack surface profiling for general applications
- Security mitigation effectiveness evaluation (e.g. bypass finding)

- Method

- Using PMU to setup hardware-level instrumentation
 - Event-based, interrupt-based
 - Runtime data collection (FE) + post-runtime data processing (BE)
 - Big data analysis
-

Analysis Approach : Front End

- FE: Interrupt-based hardware-level instrumentation



Analysis Approach : Event selection

➤ Hardware events:

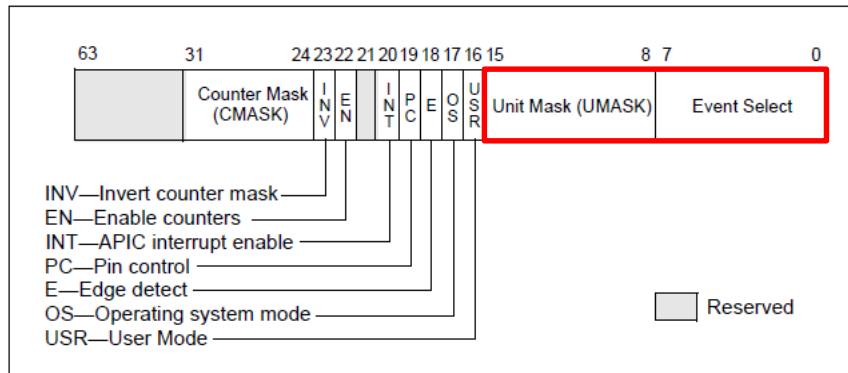


Figure 18-1. Layout of IA32_PERFVTSELx MSRs

Table 19-7. Non-Architectural Performance Events of the Processor Core Supported by Broadwell Microarchitecture (Contd.)

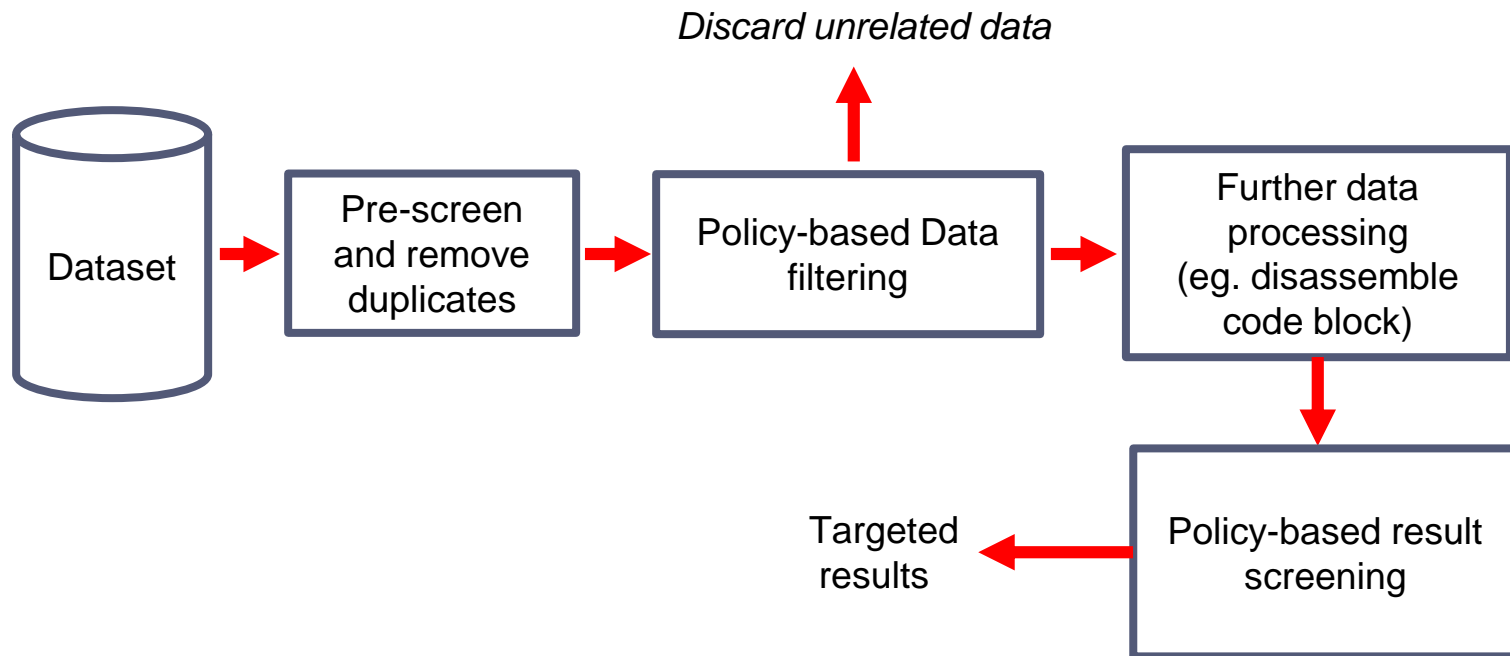
Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
80H	02H	ICACHE.MISSES	Number of Instruction Cache, Streaming Buffer and Victim Cache Misses. Includes UC accesses.	
85H	01H	ITLB_MISSES.MISS_CAUSES_A_WALK	Misses in ITLB that cause a page walk of any page size.	
85H	02H	ITLB_MISSES.WALK_COMPLETE_D_4K	Completed page walks due to misses in ITLB 4K page entries.	
85H	10H	ITLB_MISSES.WALK_DURATION	Cycle PMH is busy with a walk.	
85H	20H	ITLB_MISSES.STLB_HIT_4K	ITLB misses that hit STLB (4K).	
87H	01H	ILD_STALL.LCP	Stalls caused by changing prefix length of the instruction.	
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 or 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).	

Analysis Approach : PMU MSR Programming

- **IA32_DEBUGCTL**
 - Enable LBR on PMI
 - **MSR_LBR_SELECT**
 - LBR event filtering
 - **IA32_PERF_GLOBAL_CTRL**
 - Disable PMC before setting it
 - **IA32_PMCx**
 - Set PMC to -1 (*any event will overflow and trigger PMI*)
 - **IA32_PERFEVTSEL0**
 - Select hardware event
 - **IA32_PERF_GLOBAL_CTRL**
 - Enable PMC (good to go!)
-

Analysis Approach : Back End

- BE: post-time big data processing pipeline in Spark



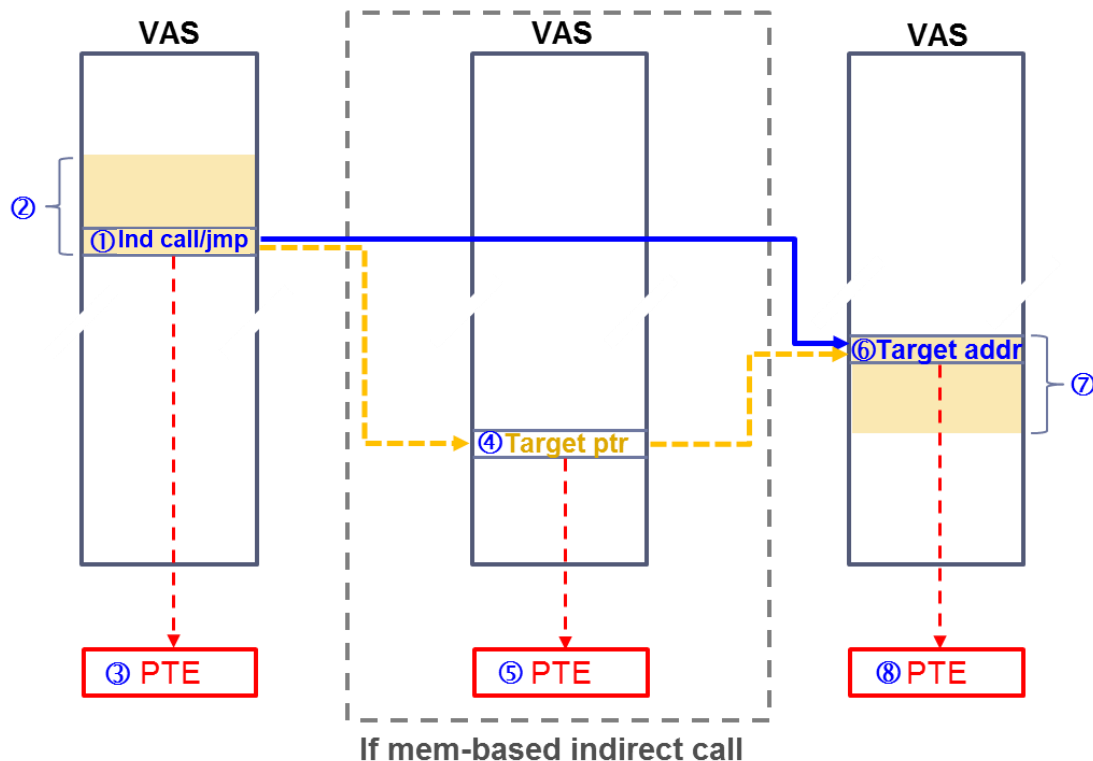
Analysis Approach : Policy Setting

- Policy setting defines the basis for the post-runtime data analysis
 - What data will be analyzed
 - How the data will be processed
 - What results are we looking for
 - Different policies can be applied to the same data set for different purpose / studies
-

Analysis Approach : Policy Setting

- Take indirect branch event as an example:

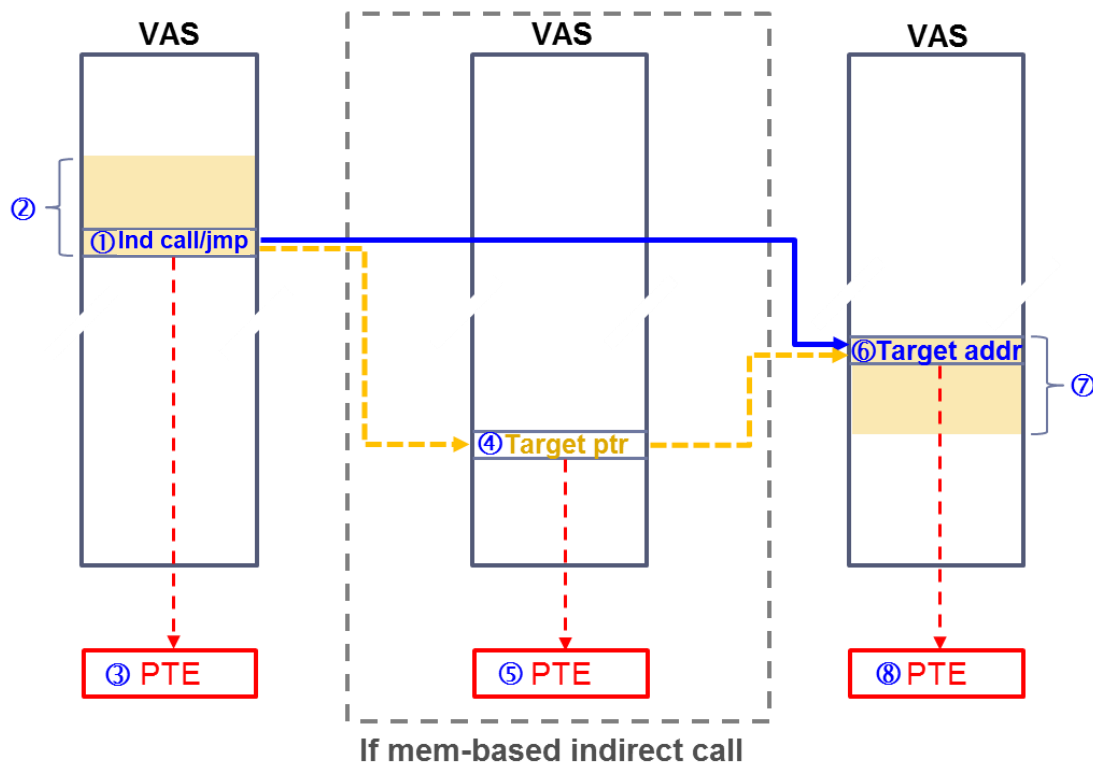
Context Data Collected



- ① “from” addr
- ② “from” code block
- ③ PTE of “from” addr
- ④ target ptr addr
- ⑤ PTE of target ptr addr
- ⑥ “to” addr
- ⑦ “to” code block
- ⑧ PTE of “to” addr

Analysis Approach : Policy Setting

- ②④⑤ can be used to find memory-based indirect calls with writable target pointer (CFG bypass)



① "from" addr

② "from" code block

③ PTE of "from" addr

④ target ptr addr

⑤ PTE of target ptr addr

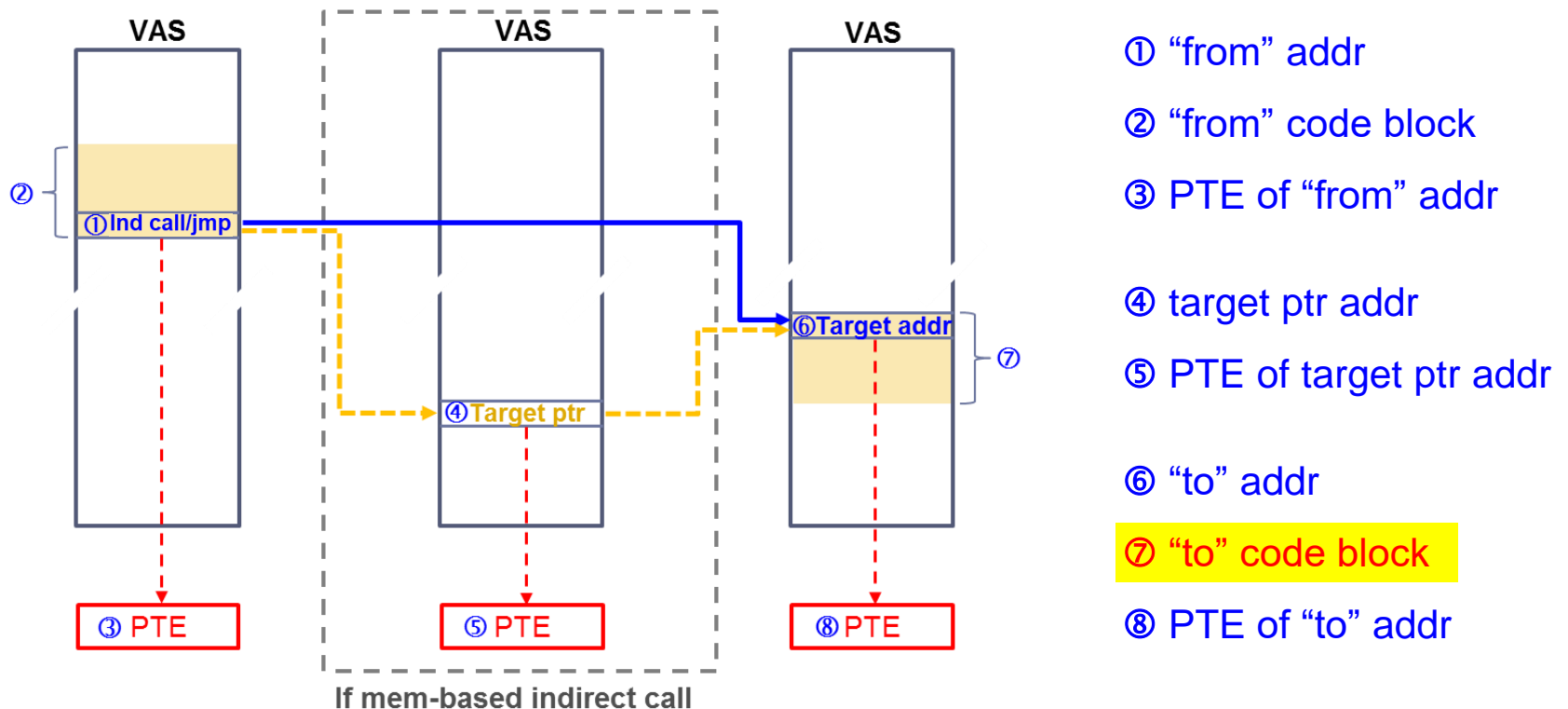
⑥ "to" addr

⑦ "to" code block

⑧ PTE of "to" addr

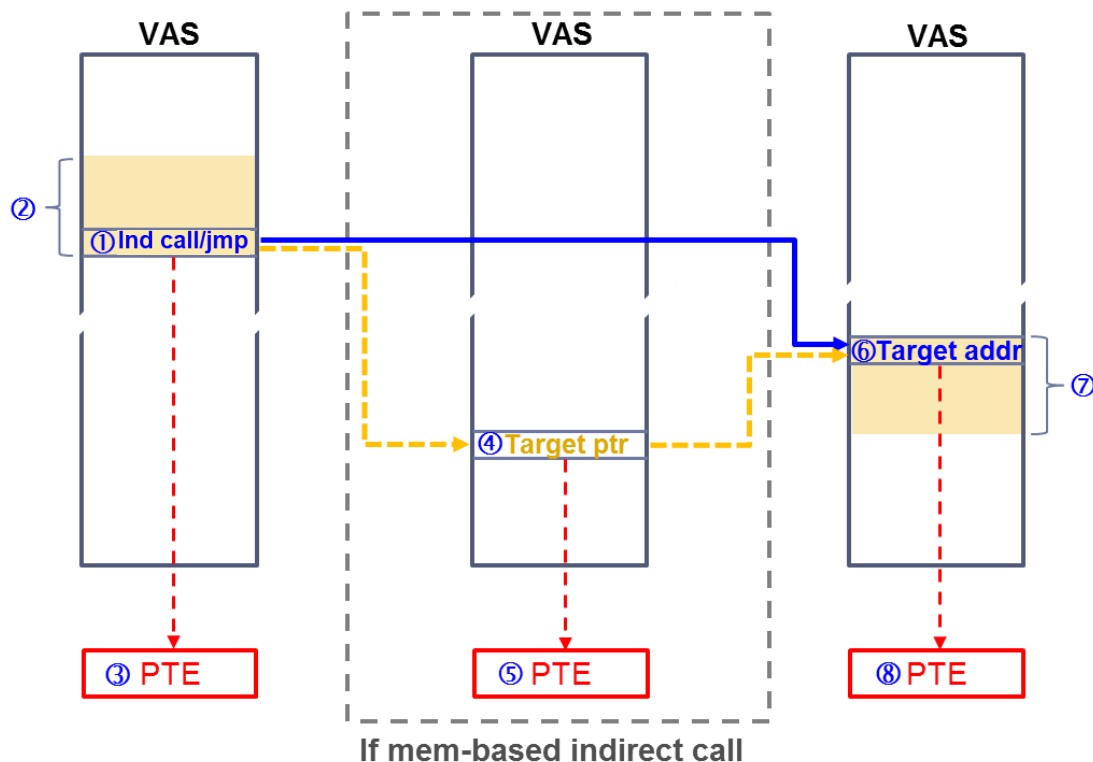
Analysis Approach : Policy Setting

- ⑦ can be used to find valid gadgets under control flow mitigations



Analysis Approach : Policy Setting

- ⑦ can also be used to find CFG bypass cases by looking for unguarded indirect jmp in the “to” code block



① “from” addr

② “from” code block

③ PTE of “from” addr

④ target ptr addr

⑤ PTE of target ptr addr

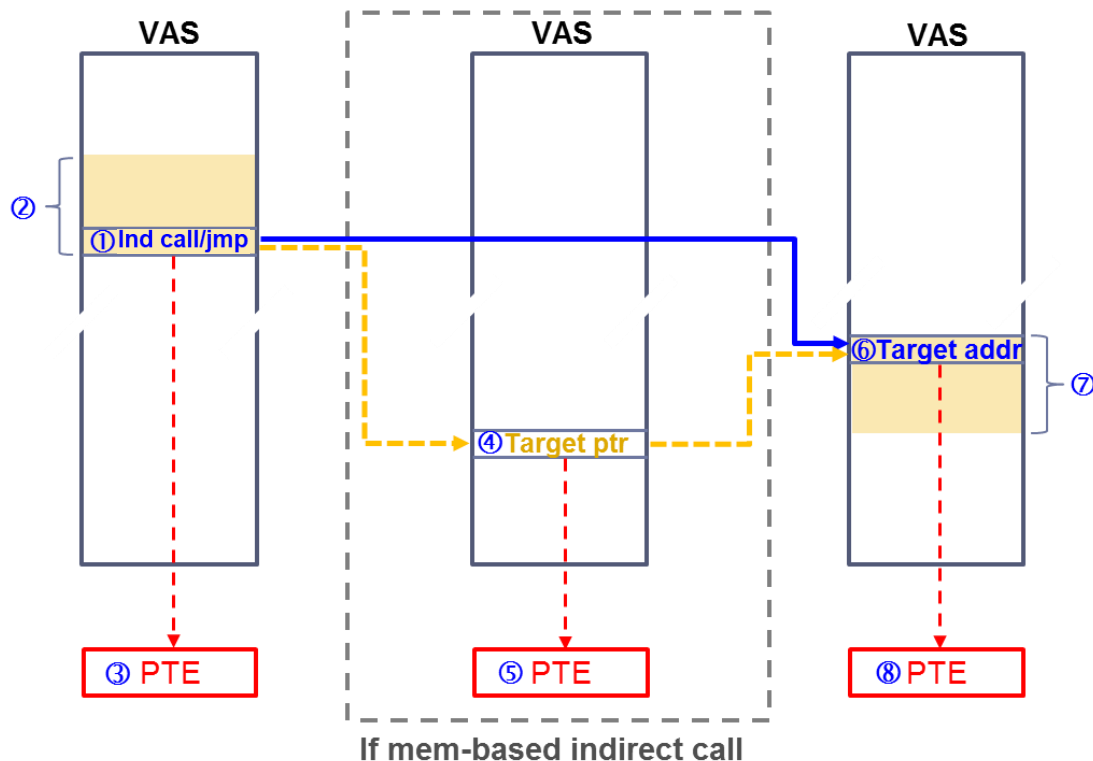
⑥ “to” addr

⑦ “to” code block

⑧ PTE of “to” addr

Analysis Approach : Policy Setting

- ③ and ⑧ can also be used to look for cases with writable “from” or “to” address (active RWX locations)



① “from” addr

② “from” code block

③ PTE of “from” addr

④ target ptr addr

⑤ PTE of target ptr addr

⑥ “to” addr

⑦ “to” code block

⑧ PTE of “to” addr

Analysis Approach : Advantages

- Reach deep into the execution flow



Analysis Approach : Advantages

- Totally transparent to target being analyzed, no impact to the native runtime behavior



To the targeted program, it's like a time freeze...

Analysis Approach : Advantages

- Scalable to collect large amount of data with automated data screening & analysis to find the **corner cases**

*In the end, what mess things up
is always the corner case...*



Agenda

- **Introduction**

- Motivation
- X86 Performance Monitor Feature

- **Analysis Approach**

- FE: Interrupt-based hardware-level instrumentation
- BE: big-data analysis
- Policy settings

- **Work in Windows**

- Control Flow Guard Bypass

- **Work in Linux**

- Framework enabling in Linux
- Preliminary Results

- **Summary & Future Work**

Work in Windows : Recognition

Mitigation Bypass

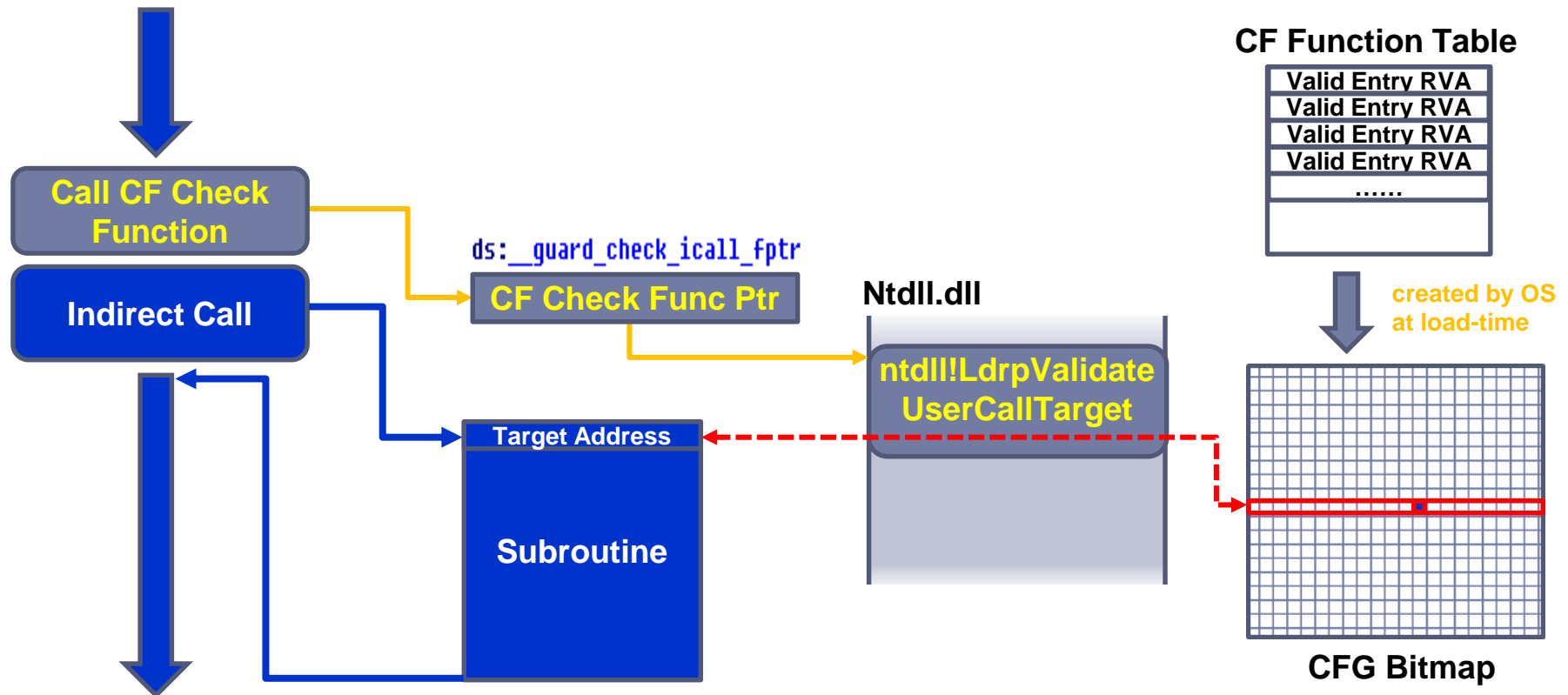
Name	Company	Amount	Year	Donation to Charity
Thomas Garnier (mxatone@)		\$5,000	2017	
Yang Junfeng (@bluerust)	FireEye, Inc.	\$15,000	2016	
Yanhui Zhao Ke Sun Ya Ou Xiaomin Song Xiaoning Li	Intel Labs	\$7,500	2016	
Liu Long	Qihoo360	\$10,000	2016	
Henry Li	TrendMicro	\$18,000	2016	
Bing Sun	Intel Security Group	\$13,000	2016	
Andrew Wesie (awesie)	Theori	\$10,000	2016	

Microsoft fixed these issues on March 2017

MS17-008	Defense-in-depth	Yanhui Zhao, Ke Sun of Intel SeCoE Ya Ou, Xiaomin Song, Xiaoning Li of Intel Labs
----------	------------------	--

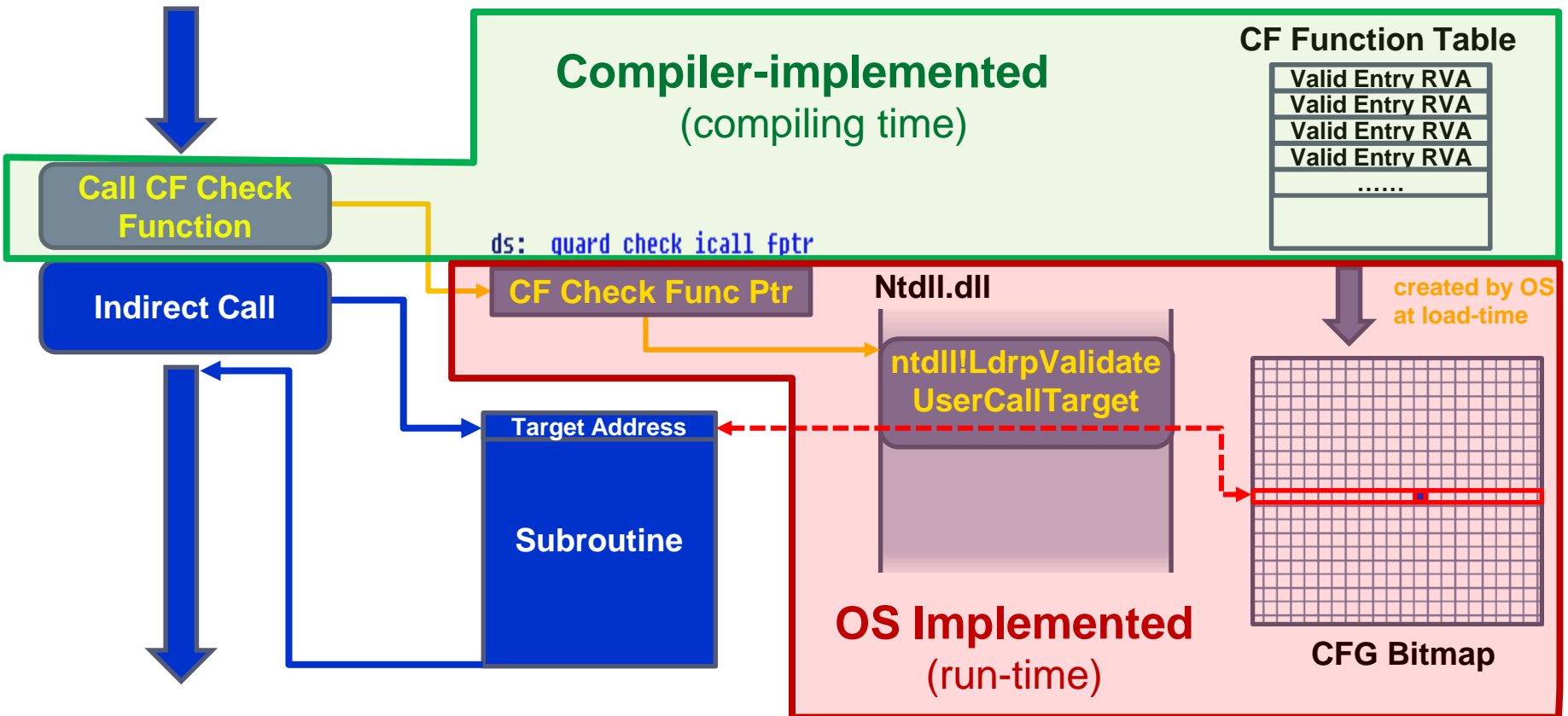
Work in Windows : CFG Overview

- Control Flow Guard (CFG) is a mitigation technology to prevent control flow being redirected to unintended locations, by validating the target address of an indirect branch before it takes place

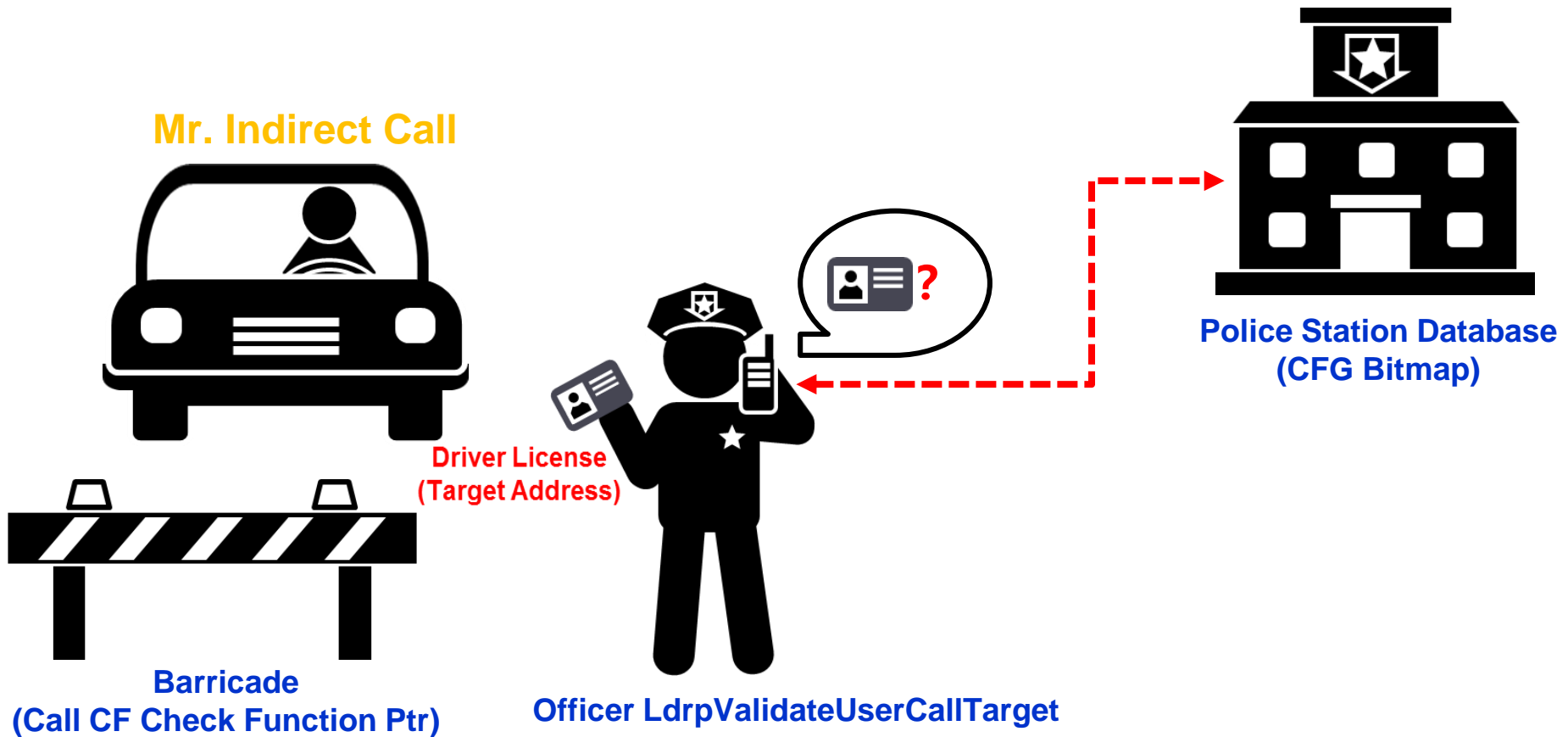


Work in Windows : CFG Overview

- Control Flow Guard (CFG) is a mitigation technology to prevent control flow being redirected to unintended locations, by validating the target address of an indirect branch before it takes place



Work in Windows : CFG Implementation



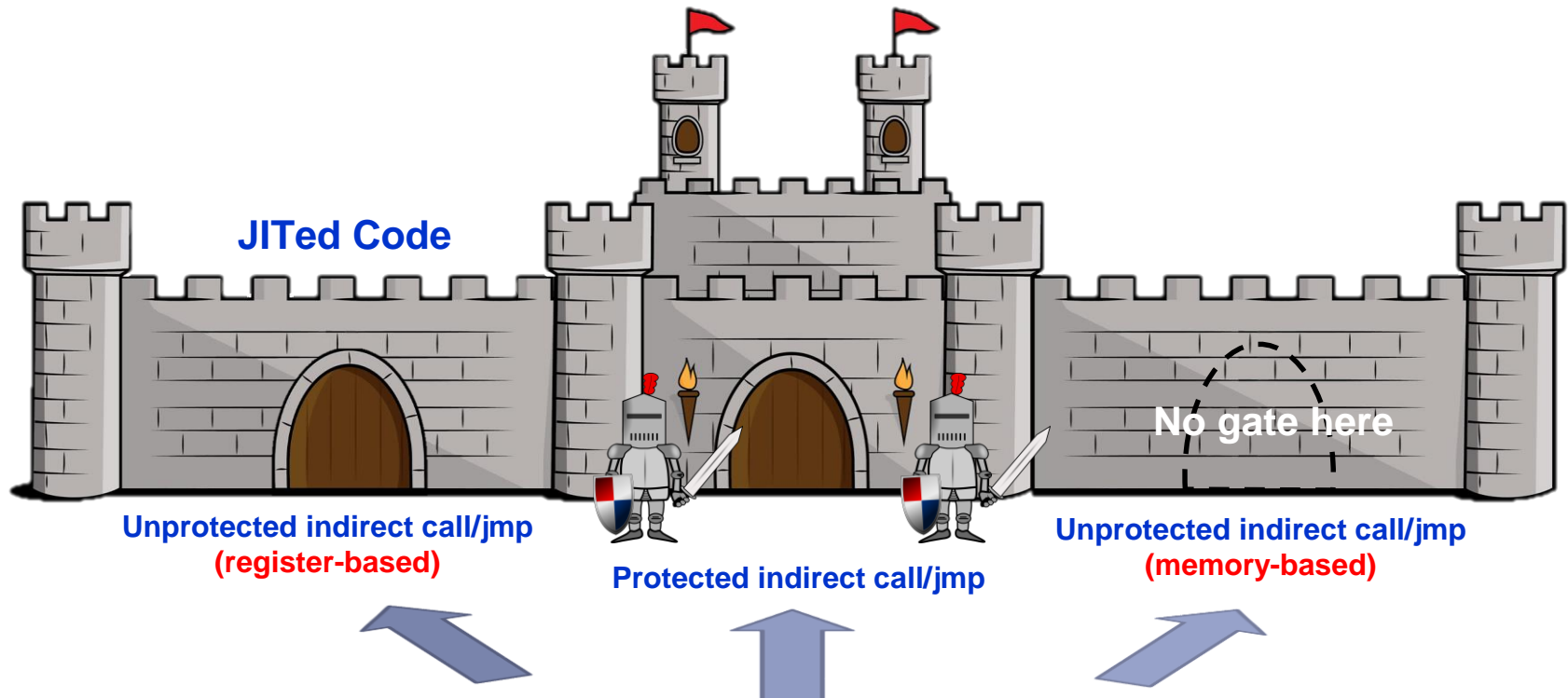
Work in Windows : Previous Research in CFG Bypass

- An incomplete list of previous CFG-bypass studies (most related to JIT)



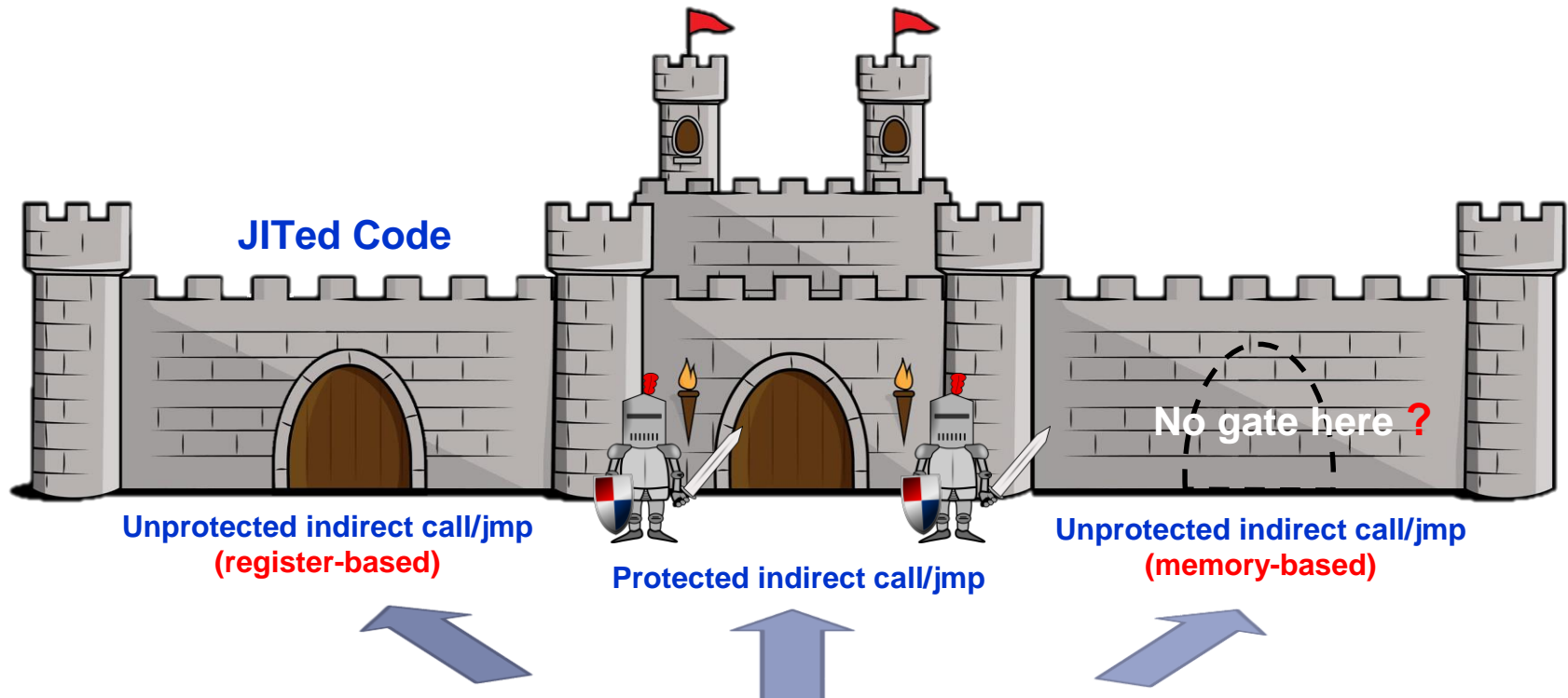
Work in Windows : Research Focus

- Memory-based indirect call (from READ_ONLY locations) is **NOT** CFG-protected due to it's being considered “safe”



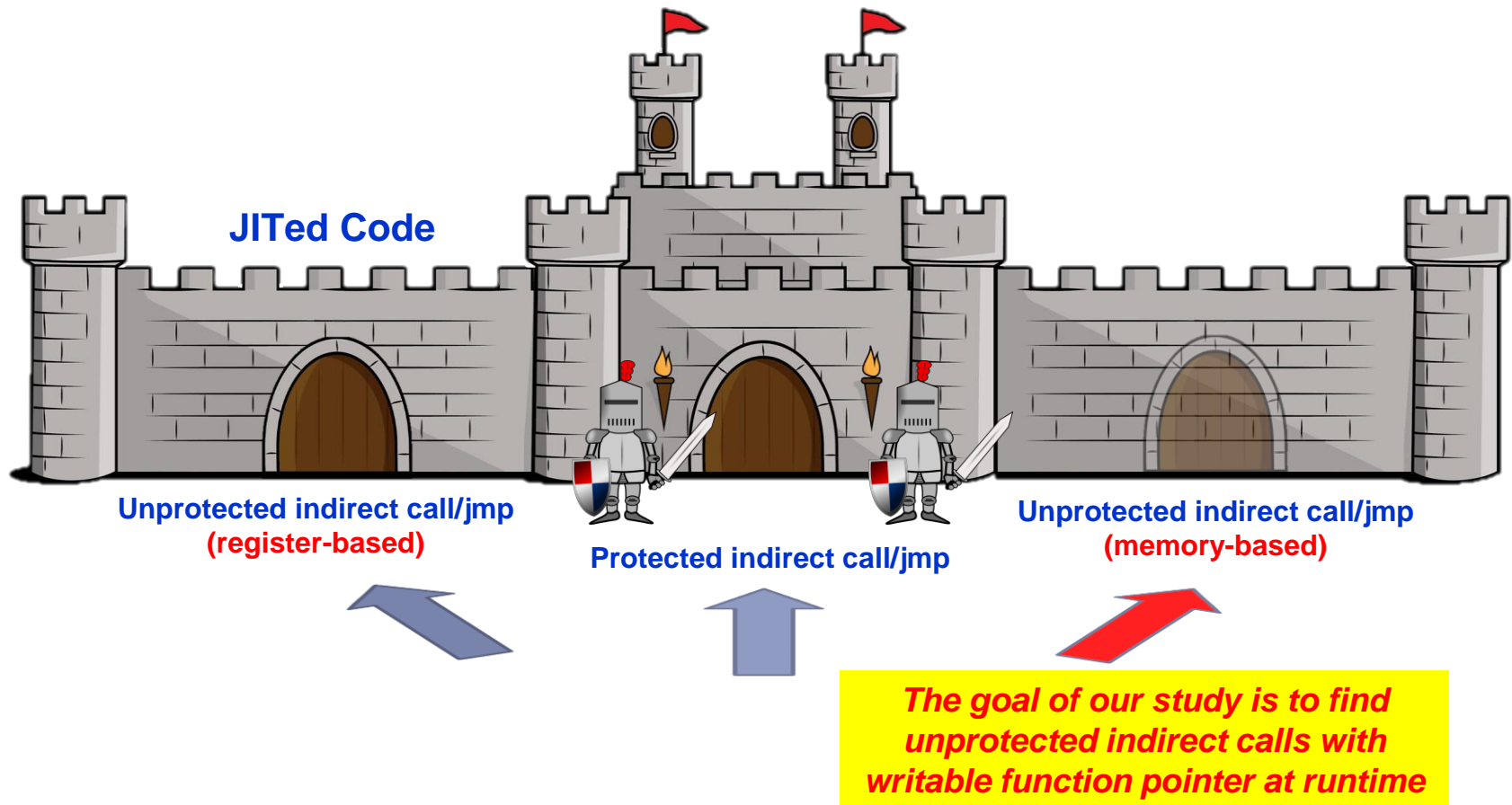
Work in Windows : Research Focus

- Memory-based indirect call (from READ_ONLY locations) is not CFG-protected due to it's considered “safe”, **is it?**



Work in Windows : Research Focus

- However, if for some reason, the target address pointer of an indirect call become writable, it will become an unguarded gate...



Work in Windows : Results & Discussion

- With the analysis method mentioned, we have
 - For Edge, collected data items: 69,341,184, data file size: 4.4G, unique combinations of “to” eip address and code block items: 20,611
 - For flash, collected data items: 9,949,184, data file size: 637M, unique combinations of “to” eip address and code block items: 688
 - 3 cases of memory-based indirect calls (all from edge), which were not protected by CFG per policy, had writable target address pointer:
 - 2 cases with the target address pointers located within the .data section, which is PAGE_READWRITE ([windows.storage.dll](#) and [ieapfltr.dll](#))
 - 1 case with the writable target address pointer in the IAT of .idata section of msctf.dll, which is very interesting...
-

Work in Windows : Results & Discussion

- 1st case of the 2 findings with memory-based indirect call's target address pointers in .data section (RW)

windows.storage.dll

```
747d26d9 47          inc     edi
747d26da eb0c        jmp     Windows_Storage!ATL::CSimpleArray<CLoadedItemVectorBase
747d26dc 03ff        add     edi,edi
747d26de 7844        js      Windows_Storage!ATL::CSimpleArray<CLoadedItemVectorBase
747d26e0 81fffffff0f cmp     edi,0FFFFFFFh
747d26e6 773c        ja      Windows_Storage!ATL::CSimpleArray<CLoadedItemVectorBase
747d26e8 6a08        push    8
747d26ea 57          push    edi
747d26eb ff36        push    dword ptr [esi]
747d26ed ff1500609c74 call    dword ptr [Windows_Storage!_imp__realloc (749c6000)]
```

```
.data:10506000 ; Segment type: Pure data
.data:10506000 ; Segment permissions: Read/Write
.data:10506000 _data          segment para public
.data:10506000          assume cs:_data
.data:10506000          ;org 10506000h
.data:10506000 _imp__realloc dd offset __realloc
.data:10506000
.data:10506004          align 8
```

0:013> !address 749c6000

Usage:	Image
Base Address:	749c6000
End Address:	749ca000
Region Size:	00004000 (16.000 kB)
State:	00001000 MEM_COMMIT
Protect:	00000004 PAGE_READWRITE
Type:	01000000 MEM_IMAGE
Allocation Base:	744c0000
Allocation Protect:	00000080 PAGE_EXECUTE_WRITECOPY
Image Path:	C:\Windows\System32\Windows.Storage.dll
Module Name:	Windows_Storage

Work in Windows : Results & Discussion

- 2nd case of the 2 findings with memory-based indirect call's target address pointers in .data section (RW)

ieapfltr.dll

```
5c078375 6a04      push     4
5c078377 40         inc      eax
5c078378 50         push     eax
5c078379 ff7604     push     dword ptr [esi+4]
5c07837c ff1500600f5c call     dword ptr [ieapfltr!_imp_realloc (5c0f6000)]
5c078382 83c40c     add      esp,0Ch
```

```
0:013> !address 5c0f6000
```

```
Mapping file section regions...
Mapping module regions...
Mapping PEB regions...
Mapping TEB and stack regions...
Mapping heap regions...
Mapping page heap regions...
Mapping other regions...
Mapping stack trace database regions...
Mapping activation context regions...
```

```
Usage:
Base Address:      5c0f6000
End Address:        5c0f9000
Region Size:        00003000 ( 12.000 kB)
State:              00001000      MEM_COMMIT
Protect:            00000004      PAGE_READWRITE
Type:               01000000      MEM_IMAGE
Allocation Base:    5c040000
Allocation Protect: 00000080      PAGE_EXECUTE_WRITECOPY
Image Path:         C:\Windows\SYSTEM32\ieapfltr.dll
Module Name:        ieapfltr
```

```
.data:72F56000 ; Segment type: Pure data
.data:72F56000 ; Segment permissions: Read/Write
.data:72F56000 _data      segment para public 'D'
.data:72F56000          assume cs:_data
.data:72F56000          ;org 72F56000h
.data:72F56000 [_imp_realloc] dd offset _realloc
.data:72F56000
```

Work in Windows : Results & Discussion

- The one case found with indirect call's target address pointer writable and located in the IAT of .idata section

msctf.dll

```
74cd9fab 8945e8      mov     dword ptr [ebp-18h],eax
74cd9fae 33c9         xor     ecx,ecx
74cd9fb0 85c0         test    eax,eax
74cd9fb2 7413         je      MSCTF!CtfImeDispatchDefImeMessage+0x1a7 (74cd9fc7)
74cd9fb4 50           push    eax
74cd9fb5 ff15a430da74 call    dword ptr [MSCTF!_imp__ImmLockIMC (74da30a4)]
74cd9fbb 8bd8         mov     ebx,eax
74cd9fbd 85db         test    ebx,ebx
74cd9fbf 0f848fe40300 je      MSCTF!CtfImeDispatchDefImeMessage+0x3e634 (74d1845)
74cd9fc5 33c9         xor     ecx,ecx
74cd9fc7 85ff         test    edi,edi
```

```
0:024> !address 74da30a4
```








```
Usage:                               Image
Base Address:                        74da3000
End Address:                         74da4000
Region Size:                         00001000 ( 4.000 kB)
State:                               00001000      MEM_COMMIT
Protect:                             00000004      PAGE_READWRITE
Type:                                01000000      MEM_IMAGE
Allocation Base:                     74cc0000
Allocation Protect:                   00000080      PAGE_EXECUTE_WRITECOPY
Image Path:                          C:\Windows\System32\MSCTF.dll
Module Name:                         MSCTF
Loaded Image Name:                   C:\Windows\System32\MSCTF.dll
```

so we "CATCH THE FLAG"! 😊

Work in Windows : Results & Discussion

- The reason for this case:

the whole .idata segment was RW for this dll !!

Name	Start	End	R	W	X	D	L	Align	Base	Type	Class
 HEADER	10000000	10001000	?	?	?	.	L	page	0005	public	DATA
 .text	10001000	100DF000	R	.	X	.	L	para	0001	public	CODE
 .data	100DF000	100E1418	R	W	.	.	L	para	0002	public	DATA
 .idata	100E2000	100E26E0	R	W	.	.	L	para	0003	public	DATA
 .idata	100E26E0	100E5400	R	W	.	.	L	para	0003	public	DATA
 .idata	100E6000	100E6028	R	W	.	.	L	para	0004	public	DATA
 .didat	100E6028	100E6200	R	W	.	.	L	para	0004	public	DATA

Work in Windows : Results & Discussion

- **Bonus finding:** remember the `__guard_check_icall_fptr` is also in the IAT of `.idata` section...

```
0a238 85c9      test     ecx,ecx
0a23a 7423      je      MSCTF!CFunctionProviderBase::QueryInterface+0x6f (74cd.
0a23c 8b01      mov     eax,dword ptr [ecx]
0a23e 51        push    ecx
0a23f 8b7004    mov     esi,dword ptr [eax+4]
0a242 8bce     mov     ecx,esi
0a244 ff15e036da74 call    dword ptr [MSCTF!__guard_check_icall_fptr (74da36e0)]
0a24a ffd6     call    esi
0a24c 33c0     xor     eax,eax
```

0:024> !address 74da36e0

```
Usage:          Image
Base Address:   74da3000
End Address:    74da4000
Region Size:    00001000 ( 4.000 kB)
State:          00001000      MEM_COMMIT
Protect:        00000004      PAGE_READWRITE
Type:           01000000      MEM_IMAGE
Allocation Base: 74cc0000
Allocation Protect: 00000080      PAGE_EXECUTE_WRITECOPY
Image Path:     C:\Windows\System32\MSCTF.dll
Module Name:    MSCTF
Loaded Image Name: C:\Windows\System32\MSCTF.dll
```

All CFG checks in msctf.dll can be bypassed!!

Work in Windows : Results & Discussion

- **4093** Windows dll files under Windows 10 Home 32bit system (Version 1607, OS Build 14393.477) have been screened and 4 more dlls with RW .idata sections are found

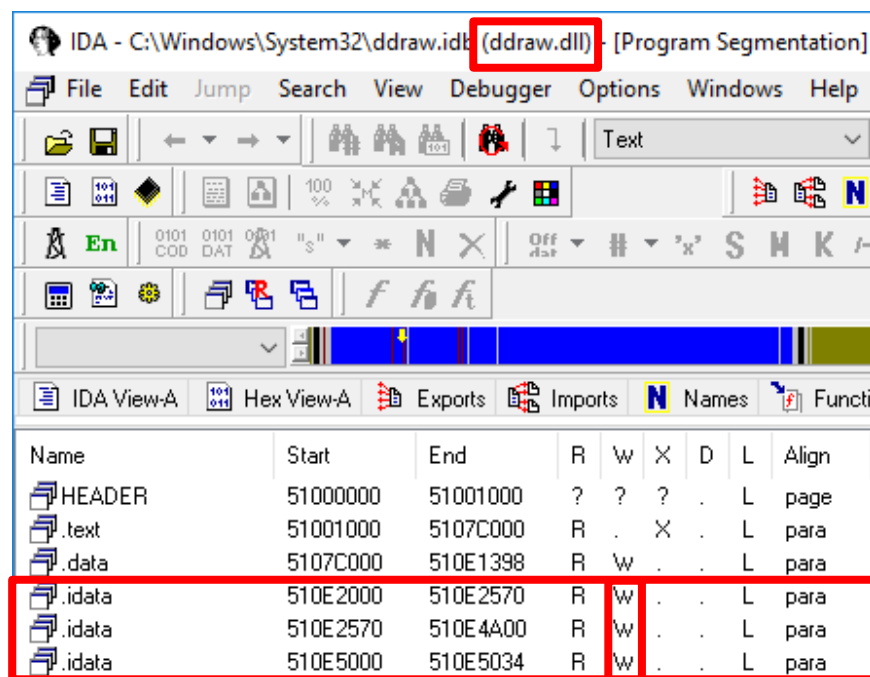
- ddraw.dll

- ddrawex.dll

- msutb.dll

- tapi32.dll

- Scan in Windows 10 Pro 64-bit system (Version 1607 OS Build 14393.953) shows the same results



Work in Windows : Fix for the Issues

- Microsoft fixed these issues on March 2017.
- Example: after the fix, in msctf.dll, the CFG function ptr is not Writable anymore.

```
Usage: Image
Base Address: 76ba2000
End Address: 76bf5000
Region Size: 00053000 ( 332.000 kB)
State: 00001000 MEM_COMMIT
Protect: 00000002 PAGE_READONLY
Type: 01000000 MEM_IMAGE
Allocation Base: 76ac0000
Allocation Protect: 00000080 PAGE_EXECUTE_WRITECOPY
Image Path: C:\Windows\System32\MSCTF.dll
Module Name: MSCTF
Loaded Image Name: C:\Windows\System32\MSCTF.dll
Mapped Image Name:
More info: lmv m MSCTF
More info: !lmi MSCTF
More info: !n 0x76ba26e0
More info: !dh 0x76ac0000
```

Before Fix **msctf.dll**

Name	Start	End	R	W	X
HEADER	10000000	10001000	?	?	?
.text	10001000	100DF000	R	.	X
.data	100DF000	100E1418	R	W	.
.idata	100E2000	100E26E0	R	W	.
.idata	100E26E0	100E5400	R	W	.
.idata	100E6000	100E6028	R	W	.
.didat	100E6028	100E6200	R	W	.

After Fix

Name	Start	End	R	W	X
HEADER	10000000	10001000	?	?	?
.text	10001000	100DE400	R	.	X
.data	100DF000	100E1418	R	W	.
.idata	100E2000	100E26E0	R	.	.
.idata	100E26E0	100E5400	R	.	.
.idata	100E6000	100E6028	R	W	.
.didat	100E6028	100E6200	R	W	.

Agenda

- **Introduction**

- Motivation
- X86 Performance Monitor Feature

- **Analysis Approach**

- FE: Interrupt-based hardware-level instrumentation
- BE: big-data analysis
- Policy settings

- **Work in Windows**

- Control Flow Guard Bypass

- **Work in Linux**

- Framework enabling in Linux
- Preliminary Results

- **Summary & Future Work**



Work in Linux : Existing Framework

- Existing performance monitor interface: **Perf (perf_events)**, which provides generalized abstractions for hardware performance monitor feature:
- Support certain hardware events
- Support certain measurements with event-based record sampling (instruction pointer, stack sample)

```
# perf list | grep Hardware
cpu-cycles OR cycles          [Hardware event]
instructions                  [Hardware event]
cache-references              [Hardware event]
cache-misses                  [Hardware event]
branch-instructions OR branches [Hardware event]
branch-misses                  [Hardware event]
bus-cycles                     [Hardware event]
stalled-cycles-frontend OR idle-cycles-frontend [Hardware event]
stalled-cycles-backend OR idle-cycles-backend  [Hardware event]
ref-cycles                     [Hardware event]
L1-dcache-loads                [Hardware cache event]
L1-dcache-load-misses          [Hardware cache event]
L1-dcache-stores                [Hardware cache event]
L1-dcache-store-misses         [Hardware cache event]
[...]
```

But we want more flexibilities in event selection, interrupt handling and context data collection...

Work in Linux : PMI Management

- To intercept a Performance Monitor Interrupt (PMI)
 - PMI is managed by the APIC
 - PMI delivery mode is defined in LVT Performance Counter Register

LVT Performance Counter Register (FEE0 0340H) — Specifies interrupt delivery when a performance counter generates an interrupt on overflow (see Section 18.6.3.5.8, "Generating an Interrupt on Overflow"). This LVT entry is implementation specific, not architectural. If implemented, it is not guaranteed to be at base address FEE0 0340H.

LVT Performance Counter Register address = APIC base physical address + offset 0x340H (MMIO)

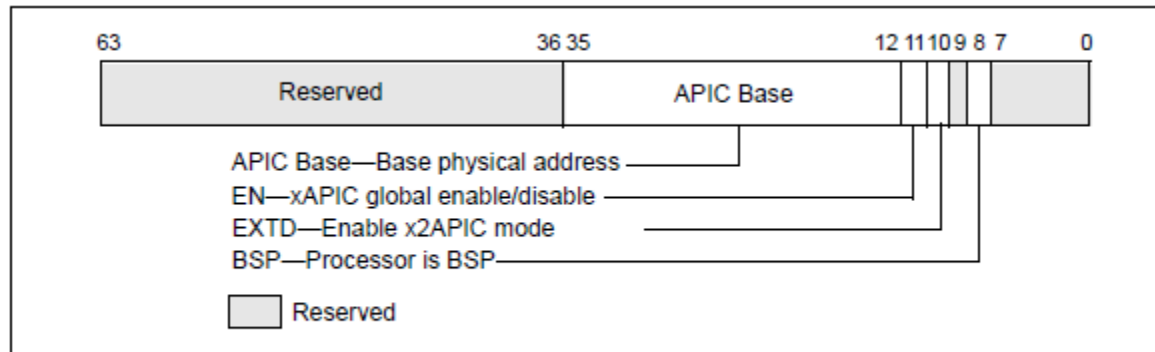


Figure 10-26. IA32_APIC_BASE MSR Supporting x2APIC

Work in Linux : PMI Management

- The interrupt vector for PMI is defined in Vector field LVT register

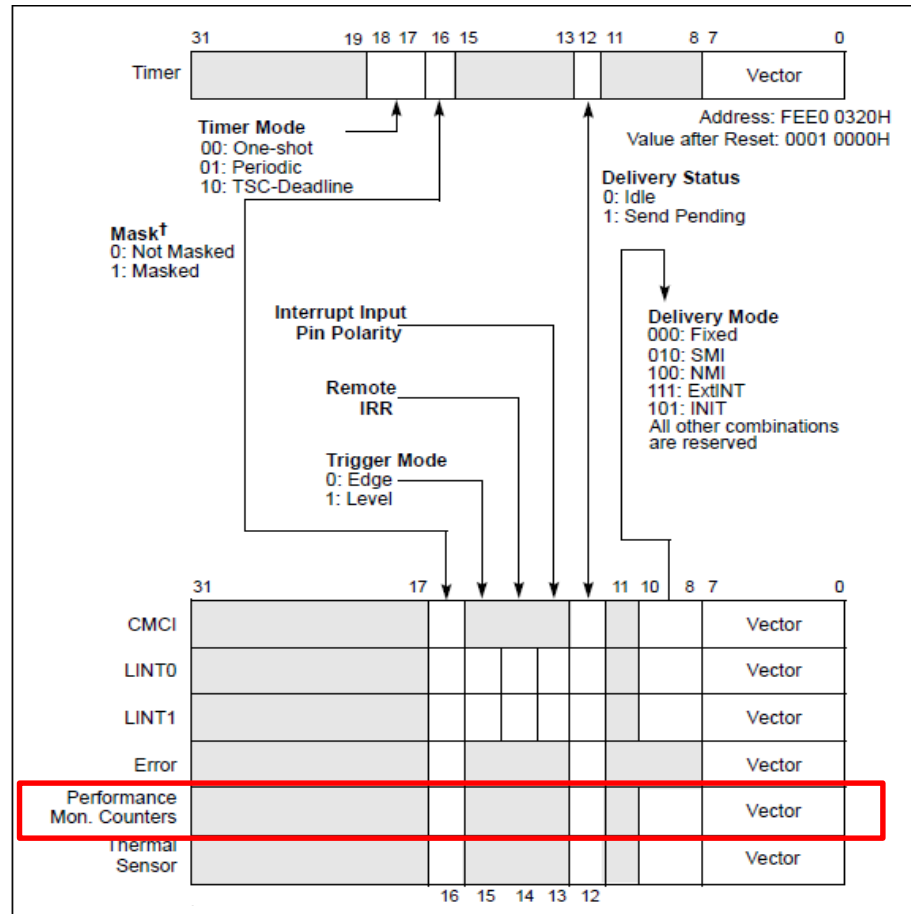


Figure 10-8. Local Vector Table (LVT)

Work in Linux : PMI Vector

- In Windows, kernel uses Fixed Delivery Mode and PMI's IDT vector is 0xfe *

```
kd> !dd [uc] fee00340
#fee00340 000000fe 00000000 000000fe 00000000
#fee00350 0001001f 00000000 0001001f 00000000
#fee00360 000004ff 00000000 000004ff 00000000
#fee00370 000000e3 00000000 000000e3 00000000
#fee00380 00000000 00000000 00000000 00000000
#fee00390 00000000 00000000 00000000 00000000
```

- However, in Linux...

```
00:09:04 2017] PMU: LVT PMR physical address = fee00340
00:09:04 2017] PMU: Cur LVT PMR value = 400
```

* <http://hypervsir.blogspot.com/2014/11/anybody-knows-how-to-legitimately.html>

Work in Linux : PMI Vector

- Linux treat PMI as an NMI...

```
00:09:04 2017] PMU: LVT PMR physical address = fee00340
00:09:04 2017] PMU: Cur LVT PMR value = 400
```

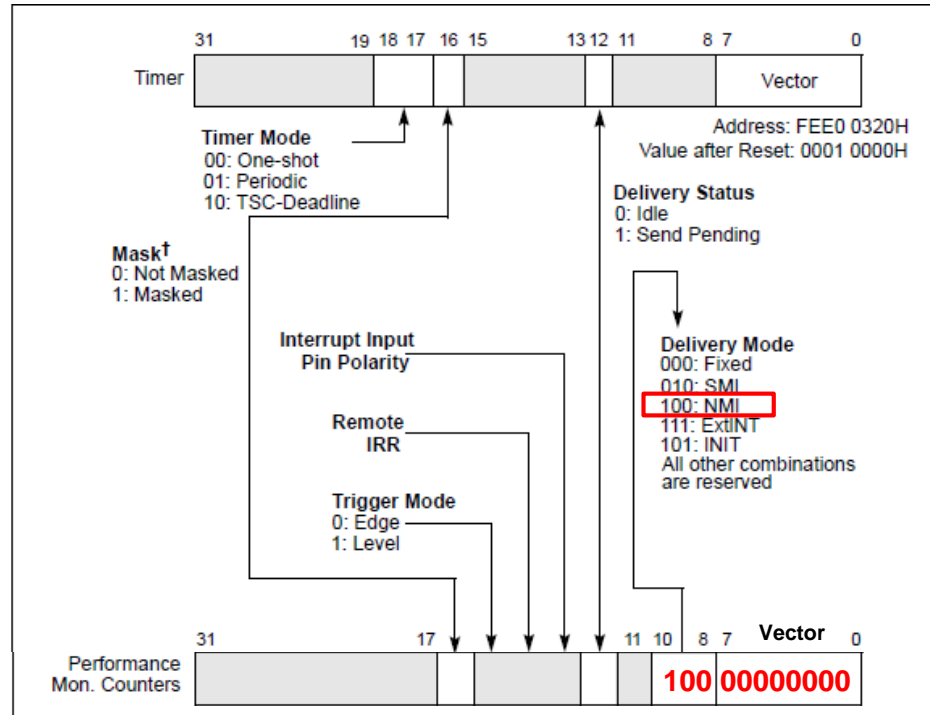


Figure 10-8. Local Vector Table (LVT)

Work in Linux : PMI Vector

- No problem, we can just overwrite


Table 10-1 Local APIC Register Address Map

Address	Register Name	Software Read/Write
FEE0 0000H	Reserved	
FEE0 0010H	Reserved	
FEE0 0020H	Local APIC ID Register	Read/Write.
FEE0 0330H	LVT Thermal Sensor Register ²	Read/Write.
FEE0 0340H	LVT Performance Monitoring Counters Register ³	Read/Write.
FEE0 0350H	LVT LINT0 Register	Read/Write.

Work in Linux : PMI Handler Hooking

- Overwrite PMI vector to a IDT entry that is not currently in use
- It seems vector 0x14 is not actually used

```
2017] PMU: IDTR Base: ffffffff57c000, Limit: 0fff
2017] PMU: Vector[14] Type: 0x8e Selector: 0x10
Address: 0xfffffffffaebd0b4
```



```
0xfffffffffaebd0b4: int3
0xfffffffffaebd0b5: int3
0xfffffffffaebd0b6: int3
0xfffffffffaebd0b7: int3
0xfffffffffaebd0b8: int3
0xfffffffffaebd0b9: int3
0xfffffffffaebd0ba: int3
0xfffffffffaebd0bb: int3
0xfffffffffaebd0bc: int3
0xfffffffffaebd0bd: int3
0xfffffffffaebd0be: int3
0xfffffffffaebd0bf: int3
0xfffffffffaebd0c0: int3
0xfffffffffaebd0c1: int3
0xfffffffffaebd0c2: int3
0xfffffffffaebd0c3: int3
0xfffffffffaebd0c4: int3
0xfffffffffaebd0c5: int3
0xfffffffffaebd0c6: int3
0xfffffffffaebd0c7: int3
0xfffffffffaebd0c8: int3
0xfffffffffaebd0c9: int3
0xfffffffffaebd0ca: int3
```

Read-only IDT?

Not a problem, disable
Write Protection in CR0 *

Work in Linux : PMI Handler Hooking

➤ Yet things are never that simple...

```
2017] PMU: LVT PMR physical address = fee00340
2017] PMU: Cur LVT PMR value = 400
2017] PMU: New LVT PMR value = 14
2017] MSR 1D9H (IA32_DEBUGCTL) = 4801
2017] MSR 1C8H (MSR_LBR_SELECT) = 1ed
2017] MSR 38FH (IA32_PERF_GLOBAL_CTRL)= 0
2017] MSR 186H (IA32_PERFVTSEL0) = 51a088
2017] MSR 38FH (IA32_PERF_GLOBAL_CTRL)= 1

2017] PMU: LVT PMR physical address = fee00340
2017] PMU: Cur LVT PMR value = 400
2017] PMU: New LVT PMR value = 400
2017] PMU: PMU trace stopped
```

```
2017] PMU: LVT PMR physical address = fee00340
2017] PMU: Cur LVT PMR value = 400
2017] PMU: New LVT PMR value = 14
2017] MSR 1D9H (IA32_DEBUGCTL) = 4801
2017] MSR 1C8H (MSR_LBR_SELECT) = 1ed
2017] MSR 38FH (IA32_PERF_GLOBAL_CTRL)= 0
2017] MSR 186H (IA32_PERFVTSEL0) = 51a088
2017] MSR 38FH (IA32_PERF_GLOBAL_CTRL)= 1
```

```
2017] PMU: LVT PMR physical address = fee00340
2017] PMU: Cur LVT PMR value = 10014
2017] PMU: New LVT PMR value = 400
2017] PMU: PMU trace stopped
```

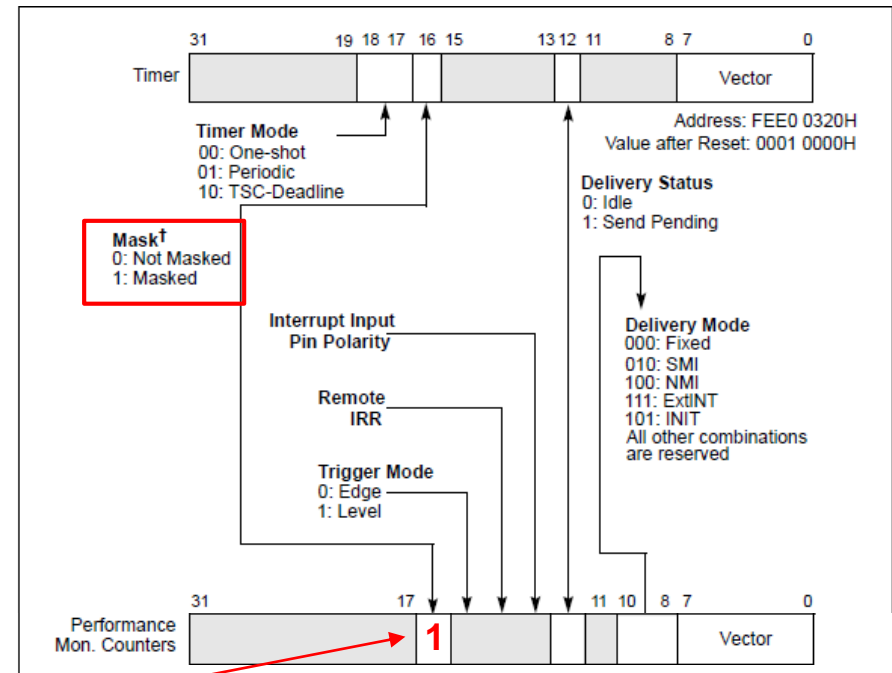


Figure 10-8. Local Vector Table (LVT)

Work in Linux : Linux PMI Management

- We are conflicting with Linux kernel's PMI management...

```
torvalds/linux:v4.12:arch/x86/include/asm/apicdef.h
```

```
92- #define      SET_APIC_DEST_FIELD(x) ((x) << 24)
93- #define APIC_LVTT      0x320
94- #define APIC_LVTTTHR    0x330
95- #define APIC_LVTPC      0x340
96- #define APIC_LVT0       0x350
97- #define      APIC_LVT_TIMER_BASE_MASK      (0x3 << 18)
98- #define      GET_APIC_TIMER_BASE(x)        (((x) >> 18) & 0x3)
```

```
torvalds/linux:v4.12:arch/x86/kernel/apic/apic.c
```

```
1034-         v = apic_read(APIC_LVT1);
1035-         apic_write(APIC_LVT1, v | APIC_LVT_MASKED);
1036-         if (maxlvt >= 4) {
1037-             v = apic_read(APIC_LVTPC);
1038-             apic_write(APIC_LVTPC, v | APIC_LVT_MASKED);
1039-         }
1040-
1041-         /* lets not touch this if we didn't frob it */

1062-         if (maxlvt >= 3)
1063-             apic_write(APIC_LVTERR, APIC_LVT_MASKED);
1064-         if (maxlvt >= 4)
1065-             apic_write(APIC_LVTPC, APIC_LVT_MASKED);
1066-
1067-         /* Integrated APIC (!82489DX) ? */
1068-         if (lapic_is_integrated()) {

2377-         apic_pm_state.apic_spiv = apic_read(APIC_SPIV);
2378-         apic_pm_state.apic_lvtt = apic_read(APIC_LVTT);
2379-         if (maxlvt >= 4)
2380-             apic_pm_state.apic_lvtpc = apic_read(APIC_LVTPC);
2381-         apic_pm_state.apic_lvt0 = apic_read(APIC_LVT0);
2382-         apic_pm_state.apic_lvt1 = apic_read(APIC_LVT1);
```

Work in Linux : PMI Handler Register

- So we gave up the rude way of intercepting PMI...

```
fffffffafe09b80 t perf_ibs_nmi_handler
fffffffafe30f20 t perf_trace_nmi_handler
fffffffafe31020 t trace_event_raw_event_nmi_handler
fffffffafe310d0 t trace_raw_output_nmi_handler
fffffffafe31260 T __register_nmi_handler
fffffffafe31460 T unregister_nmi_handler
fffffffafe64050 t kgdb_nmi_handler
fffffffafeb59478 r __tracepoint_ptr_nmi_handler
fffffffafeb5aa88 r __tpstrtab_nmi_handler
fffffffafeb6aaf0 r __ksymtab__register_nmi_handler
fffffffafeb8c0e0 r __ksymtab_unregister_nmi_handler
fffffffafeb8e2d1 r __kstrtab_unregister_nmi_handler
fffffffafeb8e2e8 r __kstrtab__register_nmi_handler
fffffffafec13a20 d perf_event_nmi_handler_na.39253
fffffffafec14560 d perf_ibs_nmi_handler_na.39284
fffffffafec274c0 d print_fmt_nmi_handler
fffffffafec27520 d trace_event_type_funcs_nmi_handler
fffffffafec27540 d event_nmi_handler
fffffffafec35200 d kgdb_nmi_handler_na.34270
fffffffafec35260 d kgdb_nmi_handler_na.34267
fffffffafed3a4a0 d event_class_nmi_handler
fffffffafed48a40 D __tracepoint_nmi_handler
fffffffafedc8eb3 t trace_event_define_fields_nmi_handler
fffffffafe0250 t __event_nmi_handler
```

Work in Linux : PMI Handler Register

➤ Defined in linux/arch/x86/include/asm/nmi.h

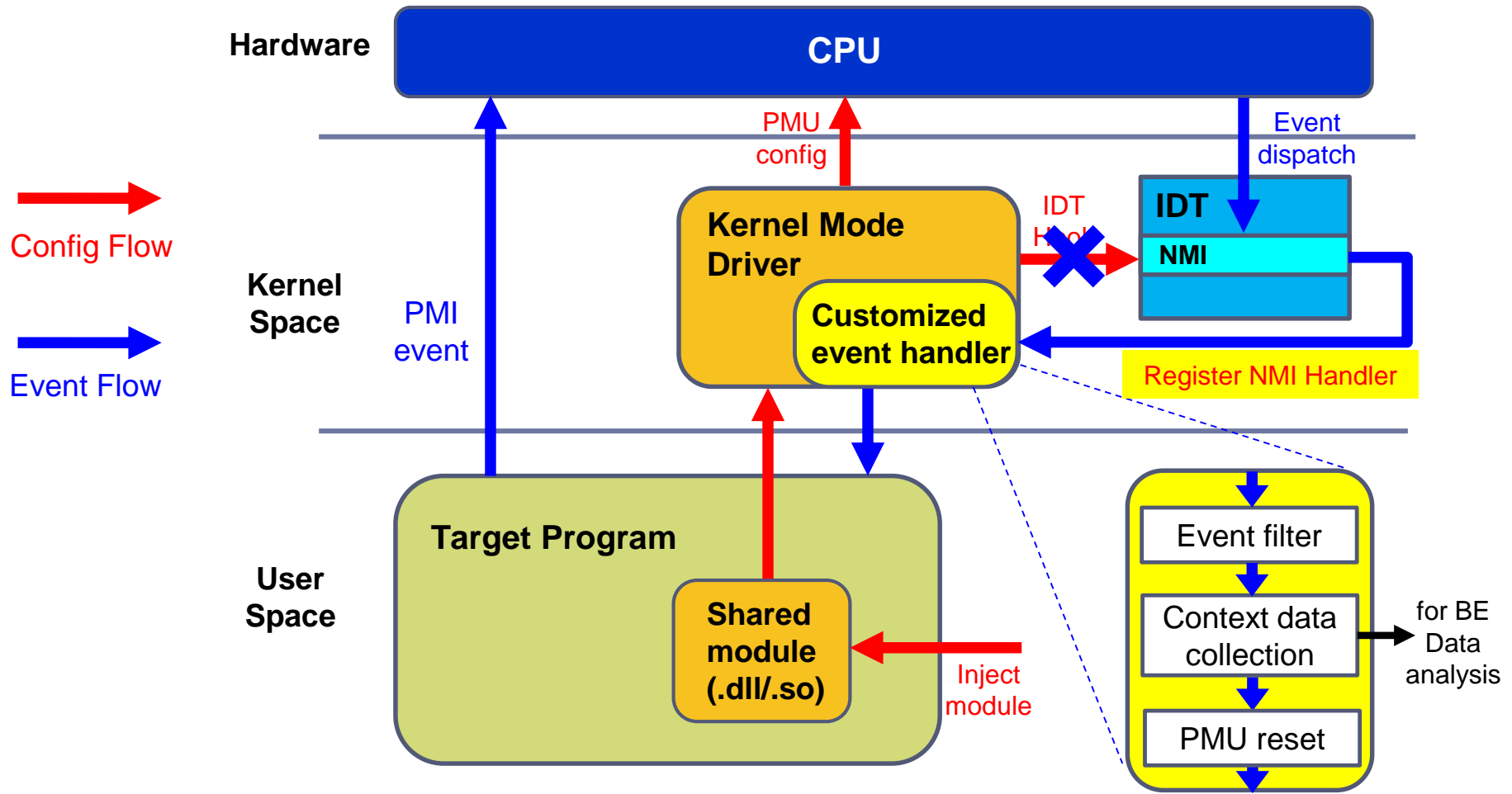
```
34 #define NMI_DONE      0
35 #define NMI_HANDLED   1

36
37 typedef int (*nmi_handler_t)(unsigned int, struct pt_regs *);
38
39 struct nmiaction {
40     struct list_head    list;
41     nmi_handler_t       handler;
42     u64                  max_duration;
43     struct irq_work      irq_work;
44     unsigned long        flags;
45     const char           *name;
46 };
47
48 #define register_nmi_handler(t, fn, fg, n, init...) \
49 ({ \
50     static struct nmiaction init fn##_na = { \
51         .handler = (fn), \
52         .name = (n), \
53         .flags = (fg), \
54     }; \
55     __register_nmi_handler((t), &fn##_na); \
56 })
```

Need to return NMI_HANDLED to continuously invoke PMI

Work in Linux : Updated Front End Scheme

- FE: Interrupt-based hardware-level instrumentation



Work in Linux : Preliminary Results

- Using RWX policy for indirect calls to find active RWX spots in common applications under Linux
 - Active RWX spots: (1) code executed; (2) code writable at execution time
 - Not protected by most of the current mitigation technologies



- Environment: Ubuntu16.04.1 (kernel 4.10.0)
 - Testing Load: SunSpider 1.0.2 JavaScript Benchmark
-

Work in Linux : Preliminary Results



- 121 active RWX locations found out of 621,950,646
- 41 unique locations out of 121 hits, in 21 memory regions
- Address distribution:

Memory Range	RWX Locations	RWX @ from addr	RWX @ to addr	Occurance	Library
0x3545f7138000 - 0x3545f7198000	3	2	1	4	anonymous mapped region
0x3545f7198000 - 0x3545f71a8000	3	0	3	4	anonymous mapped region
0x3545f71b8000 - 0x3545f71c8000	1	0	1	1	anonymous mapped region
0x3545f71c8000 - 0x3545f71f8000	3	1	2	7	anonymous mapped region
0x3545f71f8000 - 0x3545f7208000	2	0	2	2	anonymous mapped region
0x3545f7208000 - 0x3545f7228000	2	1	1	2	anonymous mapped region
0x3545f7228000 - 0x3545f7238000	1	1	0	1	anonymous mapped region
0x3545f7238000 - 0x3545f7248000	1	1	0	1	anonymous mapped region
0x3545f7248000 - 0x3545f7258000	1	0	1	1	anonymous mapped region
0x3545f7268000 - 0x3545f7288000	1	0	1	1	anonymous mapped region
0x3545f7298000 - 0x3545f72a8000	1	1	0	1	anonymous mapped region
0x3545f72a8000 - 0x3545f72c8000	1	1	0	1	anonymous mapped region
0x3545f72c8000 - 0x3545f72f8000	3	3	0	4	anonymous mapped region
0x3545f7308000 - 0x3545f7328000	1	0	1	1	anonymous mapped region
0x3545f7358000 - 0x3545f7388000	4	3	1	4	anonymous mapped region
0x3545f73b8000 - 0x3545f73e8000	1	1	0	1	anonymous mapped region
0x3545f73e8000 - 0x3545f7428000	1	1	0	1	anonymous mapped region
0x3545f74a8000 - 0x3545f7518000	2	1	1	2	anonymous mapped region
0x5594e7db9000 - 0x5594e7de0000	6	5	1	59	/usr/lib/firefox/firefox
0x7f93fb73a000 - 0x7f94008d6000	2	2	0	2	/usr/lib/firefox/libxul.so
0x7f940c094000 - 0x7f940c254000	1	1	0	21	/lib/x86_64-linux-gnu/libc-2.23.so

Work in Linux : Preliminary Results



- RWX location in module: runtime code sample can match static code
 - Runtime results: (in /usr/lib/firefox/firefox)

```
firefox-2623 PMU: === RWE From memory found at 0x5594e7dbfada ===  
firefox-2623 PMU: From_addr code block: 0xff501084c089c50f8459ffffff660f1f  
firefox-2623 PMI_NMI_count=bc205c
```

+ offset
0x6ada

```
5594e7db9000-5594e7de0000 r-xp 00000000 08:15 6689840 /usr/lib/firefox/firefox  
5594e7fdf000-5594e7fe0000 r--p 00026000 08:15 6689840 /usr/lib/firefox/firefox  
5594e7fe0000-5594e7fe1000 rw-p 00027000 08:15 6689840 /usr/lib/firefox/firefox
```

- Static code

```
/usr/lib/firefox/firefox: file format elf64-x86-64  
  
Disassembly of section .text:  
  
0000000000006ada <_ZN7mozilla6detail9MutexImpl6unlockEv@@Base+0x29a>:  
6ada: ff 50 10 callq *0x10(%rax)  
6add: 84 c0 test %al,%al  
6adf: 89 c5 mov %eax,%ebp  
6ae1: 0f 84 59 ff ff ff je 6a40 <_ZN7mozilla6detail9MutexImpl6unlockEv@@Base+0x200>  
6ae7: 66 0f 1f 84 00 00 00 nopw 0x0(%rax,%rax,1)  
6aee: 00 00  
6af0: 83 eb 01 sub $0x1,%ebx  
6af3: 83 fb ff cmp $0xffffffff,%ebx  
6af6: 0f 84 46 ff ff ff je 6a42 <_ZN7mozilla6detail9MutexImpl6unlockEv@@Base+0x202>  
6afc: 49 8b 07 mov (%r15),%rax  
6aff: 49 83 47 08 01 addq $0x1,0x8(%r15)  
6b04: ba 01 00 00 00 mov $0x1,%edx
```

Work in Linux : Preliminary Results



- RWX location in module: runtime code sample can match static code
- Runtime results: (in /lib/x86_64-linux-gnu/libc-2.23.so)

```
firefox-2623 PMU: === RWE From memory found at 0x7f940c1a9834 ===  
firefox-2623 PMU: From_addr code block: 0xffd0483d00f0ffff76174883f8da741c  
firefox-2623 PMI_NMI_count=b5c34f3
```

+ offset
0x115834

```
7f940c094000-7f940c254000 r-xp 00000000 08:15 7213528 /lib/x86_64-linux-gnu/libc-2.23.so  
7f940c254000-7f940c454000 ---p 001c0000 08:15 7213528 /lib/x86_64-linux-gnu/libc-2.23.so  
7f940c454000-7f940c458000 r--p 001c0000 08:15 7213528 /lib/x86_64-linux-gnu/libc-2.23.so  
7f940c458000-7f940c45a000 rw-p 001c4000 08:15 7213528 /lib/x86_64-linux-gnu/libc-2.23.so
```

- Static code

```
/lib/x86_64-linux-gnu/libc-2.23.so: file format elf64-x86-64  
  
Disassembly of section .text:  
  
0000000000115834 < __clock_gettime@@GLIBC_PRIVATE+0x24>:  
115834: ff d0 callq *%rax  
115836: 48 3d 00 f0 ff ff cmp $0xffffffffffffff00,%rax  
11583c: 76 17 jbe 115855 < __clock_gettime@@GLIBC_PRIVATE+0x45>  
11583e: 48 83 f8 da cmp $0xffffffffffffda,%rax  
115842: 74 1c je 115860 < __clock_gettime@@GLIBC_PRIVATE+0x50>  
115844: 48 8b 15 2d e6 2a 00 mov 0x2ae62d(%rip),%rdx # 3c3e78 <_IO_file_jumps@@GLIBC_2.2.5+0x  
11584b: f7 d8 neg %eax  
11584d: 64 89 02 mov %eax,%fs:(%rdx)  
115850: b8 ff ff ff ff mov $0xffffffff,%eax  
115855: 48 83 c4 08 add $0x8,%rsp
```

Work in Linux : Preliminary Results



- The memory range permissions in /proc/pid/maps are all non-writable

address	perm	offset	dev	inode	pathname
3545f7138000-3545f7198000	--p	00000000	00:00	0	
3545f7198000-3545f71a8000	r-xp	00000000	00:00	0	
3545f71b8000-3545f71c8000	r-xp	00000000	00:00	0	
3545f71c8000-3545f71f8000	r-xp	00000000	00:00	0	
3545f71f8000-3545f7208000	--p	00000000	00:00	0	
3545f7208000-3545f7228000	r-xp	00000000	00:00	0	
3545f7228000-3545f7238000	--p	00000000	00:00	0	
3545f7238000-3545f7248000	r-xp	00000000	00:00	0	
3545f7248000-3545f7258000	r-xp	00000000	00:00	0	
3545f7268000-3545f7288000	r-xp	00000000	00:00	0	
3545f7298000-3545f72a8000	--p	00000000	00:00	0	
3545f72a8000-3545f72c8000	r-xp	00000000	00:00	0	
3545f72c8000-3545f72f8000	r-xp	00000000	00:00	0	
3545f7308000-3545f7328000	r-xp	00000000	00:00	0	
3545f7358000-3545f7388000	r-xp	00000000	00:00	0	
3545f73b8000-3545f73e8000	r-xp	00000000	00:00	0	
3545f73e8000-3545f7428000	r-xp	00000000	00:00	0	
3545f74a8000-3545f7518000	r-xp	00000000	00:00	0	
5594e7db9000-5594e7de0000	r-xp	00000000	08:15	6689840	/usr/lib/firefox/firefox
5594e7fdf000-5594e7fe0000	r--p	00026000	08:15	6689840	/usr/lib/firefox/firefox
5594e7fe0000-5594e7fe1000	rw-p	00027000	08:15	6689840	/usr/lib/firefox/firefox
7f93fb73a000-7f94008d6000	r-xp	00000000	08:15	6685360	/usr/lib/firefox/libxul.so
7f94008d6000-7f9400ad5000	--p	0519c000	08:15	6685360	/usr/lib/firefox/libxul.so
7f9400ad5000-7f9400efd000	r--p	0519b000	08:15	6685360	/usr/lib/firefox/libxul.so
7f9400efd000-7f9400f41000	rw-p	055c3000	08:15	6685360	/usr/lib/firefox/libxul.so
7f940c094000-7f940c254000	r-xp	00000000	08:15	7213528	/lib/x86_64-linux-gnu/libc-2.23.so
7f940c254000-7f940c454000	--p	001c0000	08:15	7213528	/lib/x86_64-linux-gnu/libc-2.23.so
7f940c454000-7f940c458000	r--p	001c0000	08:15	7213528	/lib/x86_64-linux-gnu/libc-2.23.so
7f940c458000-7f940c45a000	rw-p	001c4000	08:15	7213528	/lib/x86_64-linux-gnu/libc-2.23.so

Work in Linux : Preliminary Results



- 78 active RWX locations found out of 262,659,664
- 33 unique locations out of 78 hits, in 4 memory regions
- Address distribution:

Memory Range	Occurance	Library
0x35e328f04000 - 0x35e328f7f000	8	anonymous mapped region
0x35e328f84000 - 0x35e328fff000	23	anonymous mapped region
0x35e329004000 - 0x35e32907f000	22	anonymous mapped region
0x35e329084000 - 0x35e3290ff000	23	anonymous mapped region

- Memory range permission:

```
address          perm offset  dev  inode  pathname
-----
35e328f04000-35e328f7f000 rwxp 00000000 00:00 0
35e328f84000-35e328fff000 rwxp 00000000 00:00 0
35e329004000-35e32907f000 rwxp 00000000 00:00 0
35e329084000-35e3290ff000 rwxp 00000000 00:00 0
```

Summary & Future Work

➤ Summary

- Hardware-level instrumentation using PMU feature + post-runtime big data analysis can be an effective approach for systematic profiling of exploitable points across platforms
- In Windows, using this approach we found multiple CFG bypass cases
- In Linux, we enabled the framework and obtained preliminary results finding active RWX locations within common applications

➤ Future Work

- Continue the work under Linux
 - More application profiling with more policy settings
 - PaX-enabled binary screening



Thank You!



Acknowledgement:

We would like to thank *Rodrigo Branco* for good advice and review!

Reference

- Control Flow Guard, [https://msdn.microsoft.com/en-us/library/windows/desktop/mt637065\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/mt637065(v=vs.85).aspx)
 - Exploring Control Flow Guard in Windows 10. Jack Tang. Trend Micro Threat Solution Team, 2015
 - Windows 10 Control Flow Guard Internals. MJ0011, POC 2014
 - “Security Breaches as PMU Deviation: Detecting and Identifying Security Attacks Using Performance Counters”, Yuan et al., 2011
 - “CFIMon: Detecting Violation of Control Flow Integrity using Performance Counters”, Xia et al., 2012
 - “kBouncer: Efficient and Transparent ROP Mitigation”, Pappas, 2012
 - “Transparent ROP Detection using CPU Performance Counters”, Li & Crouse, 2014
 - “CPU Hardware Performance Counters for Security”, Herath et al., 2015
 - “Capturing 0Day Exploits with Perfectly Placed Hardware Traps”, Pierce et al., 2016
 - https://perf.wiki.kernel.org/index.php/Main_Page
 - Bypass Control Flow Guard Comprehensively, Yunhai Zhang, Blackhat 2015
 - Exploiting CVE-2015-0311, Part II: Bypassing Control Flow Guard on Windows 8.1 Update 3, Francisco Falcón, Mar 2015
 - <https://www.blackhat.com/docs/eu-15/materials/eu-15-Falcon-Exploiting-Adobe-Flash-Player-In-The-Era-Of-Control-Flow-Guard.pdf>
 - <https://securingtomorrow.mcafee.com/mcafee-labs/microsofts-june-patch-kills-potential-cfg-bypass/>
 - <https://www.yumpu.com/en/document/view/55963117/jit-spraying-never-dies>
 - <http://xlab.tencent.com/en/2015/12/09/bypass-dep-and-cfg-using-jit-compiler-in-chakra-engine/>
 - <http://xlab.tencent.com/en/2016/01/04/use-chakra-engine-again-to-bypass-cfg/>
 - <http://theori.io/research/chakra-jit-cfg-bypass>
-

Reference

- <http://hypervsir.blogspot.com/2014/11/anybody-knows-how-to-legitimately.html>
 - Intel® 64 and IA-32 Architectures Software Developer Manuals, <http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html>
 - Security Breaches as PMU Deviation: Detecting and Identifying Security Attacks Using Performance Counters. Liwei Yuan, Weichao Xing, Haibo Chen, Binyu Zang. APSYS 2011
 - Loop-Oriented Programming: A New Code Reuse Attack to Bypass Modern Defenses, B Lan et al, 2015 IEEE Trustcom/BigDataSE/ISPA
 - Capturing 0day Exploit With Perfectly Placed Hardware Traps, C. Pierce, M. Spisak, K. Fitch, Blackhat usa 2016
 - IROP – interesting ROP gadgets, Xiaoning Li/Nicholas Carlini, Source Boston 2015
 - Apache Spark, <http://spark.apache.org/>
 - Capstone, <http://www.capstone-engine.org/>
-