CAPTURING ODAY EXPLOITS WITH PERFECTLY PLACED HARDWARE TRAPS

Cody Pierce . Matt Spisak . Kenneth Fitch

ENDGAME.

INTRODUCTION

EXPLOIT DETECTION IS A MOVING TARGET

- Exploitation is increasingly more sophisticated
- Creativity in exploitation is hard to plan for in the Security Development Lifecycle (SDL)
- A well financed attacker armed with 0days has the advantage

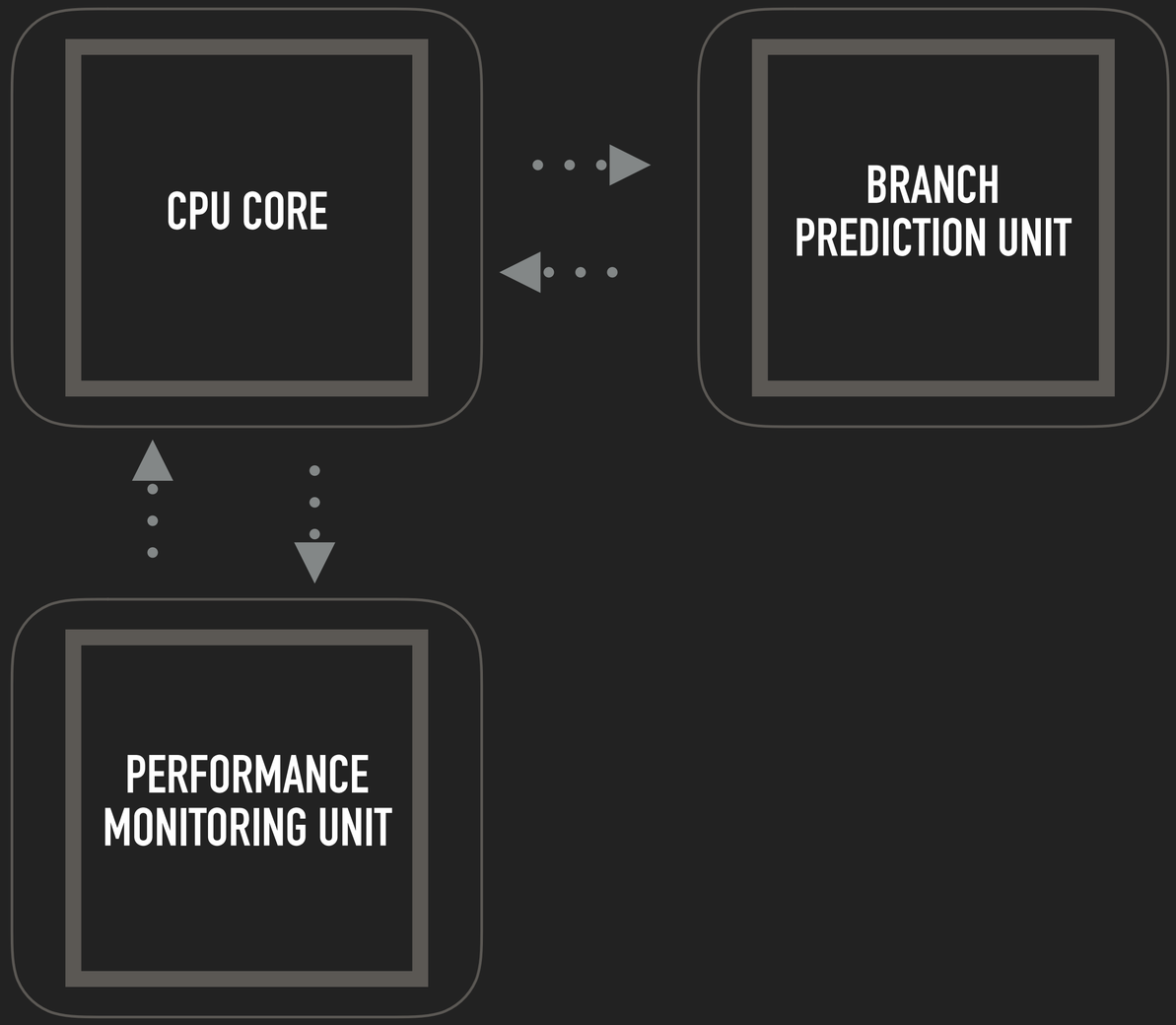
EARLY PREVENTION TO MAINTAIN THE ADVANTAGE

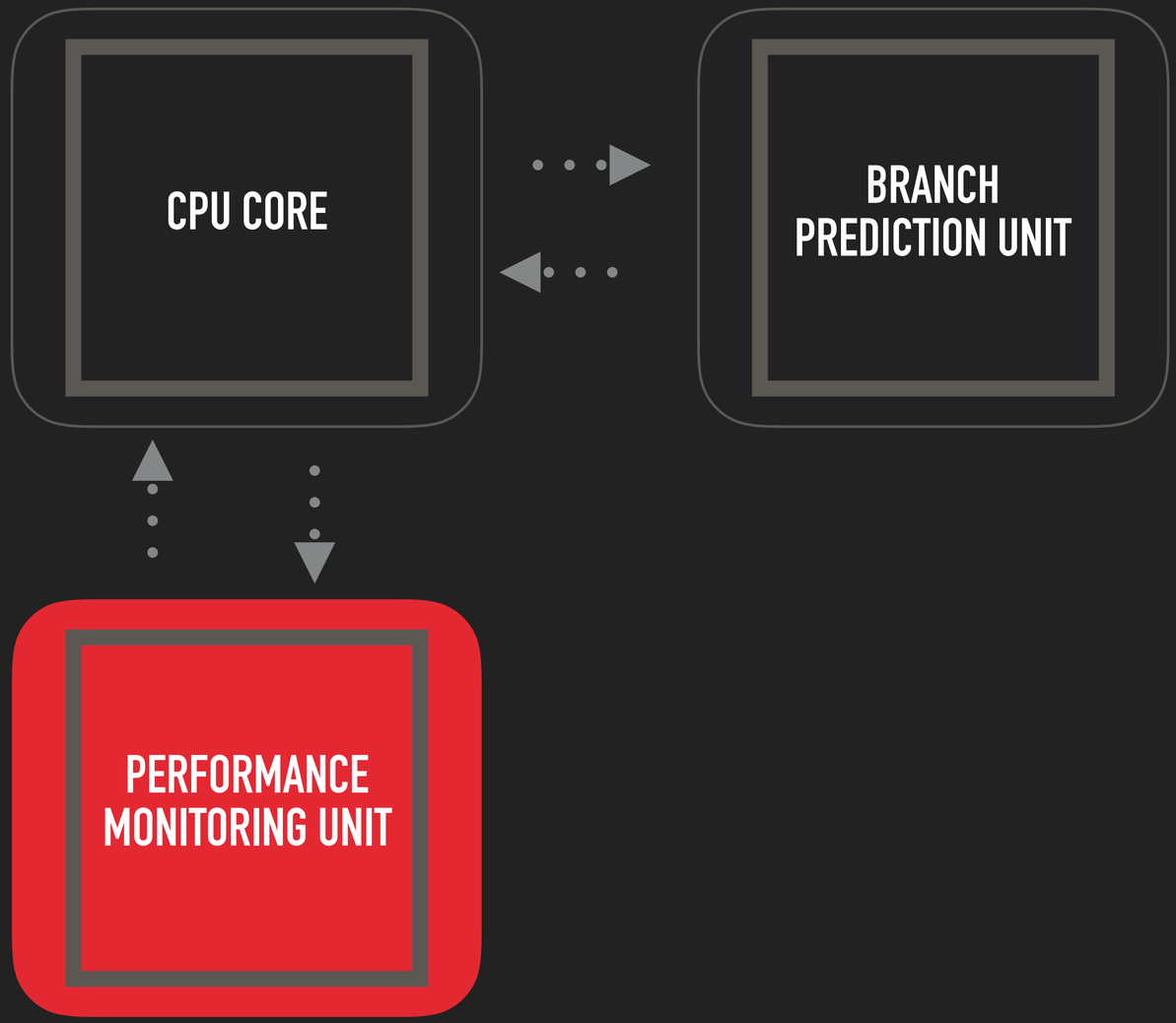
Pre-Exploitation	Exploitation	Post-Exploitation
Software Identification	Memory Organization	Return Oriented Programming
Version Identification	Vulnerability Preparation	Payload Execution
OS Identification	Vulnerability Trigger	COE
	Code Execution	Persistence

EARLY PREVENTION TO MAINTAIN THE ADVANTAGE

Software Identification	Memory Organization	Return Oriented Programming
Version Identification	Vulnerability Preparation	Payload Execution
OS Identification	Vulnerability Trigger	COE
	Code Execution	Persistence

INTRODUCING HARDWARE ASSISTANCE


CPU CORE
BRANCH PREDICTION UNIT
PERFORMANCE MONITORING UNIT


CPU CORE
BRANCH PREDICTION UNIT
PERFORMANCE MONITORING UNIT

PERFORMANCE MONITORING UNIT

- A special unit in microprocessor architectures to enable hardware level performance and system information. Often used to optimize hardware and software
- The PMU can be programed to record dozens of different hardware “events”
- Traditionally reserved for developers and system architects

0x03 LD_BLOCKS

0x0D INT_MISC

0x3C CPU_CLK_UNHALTED

0x48 L1D_PEND_MISS

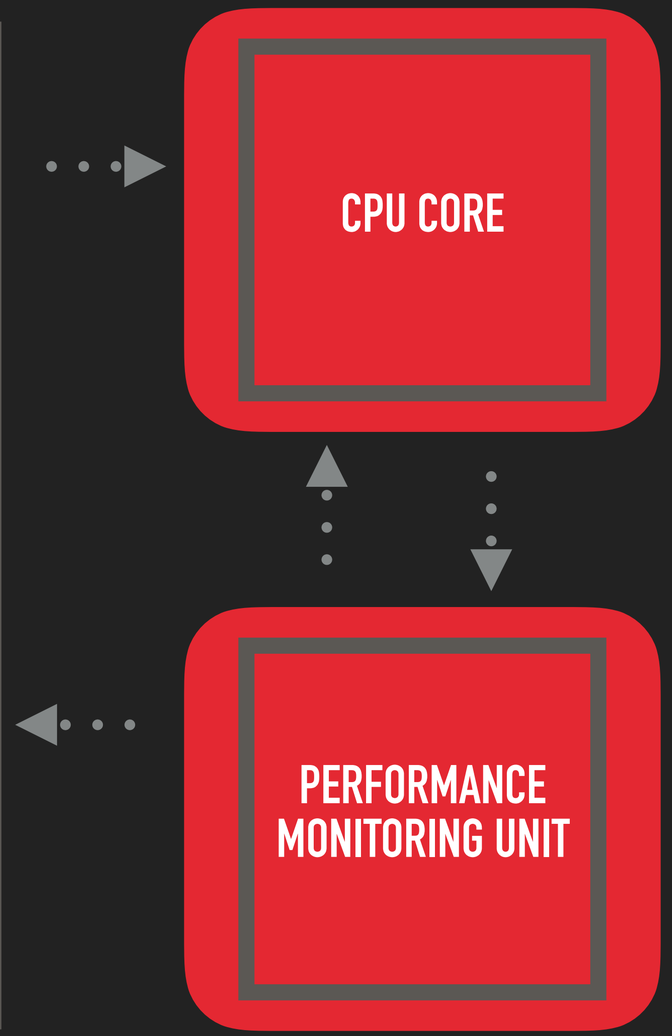
0x85 ITLB_MISSES

0x89 BR_MISP_EXEC

0xA2 RESOURCE_STALLS

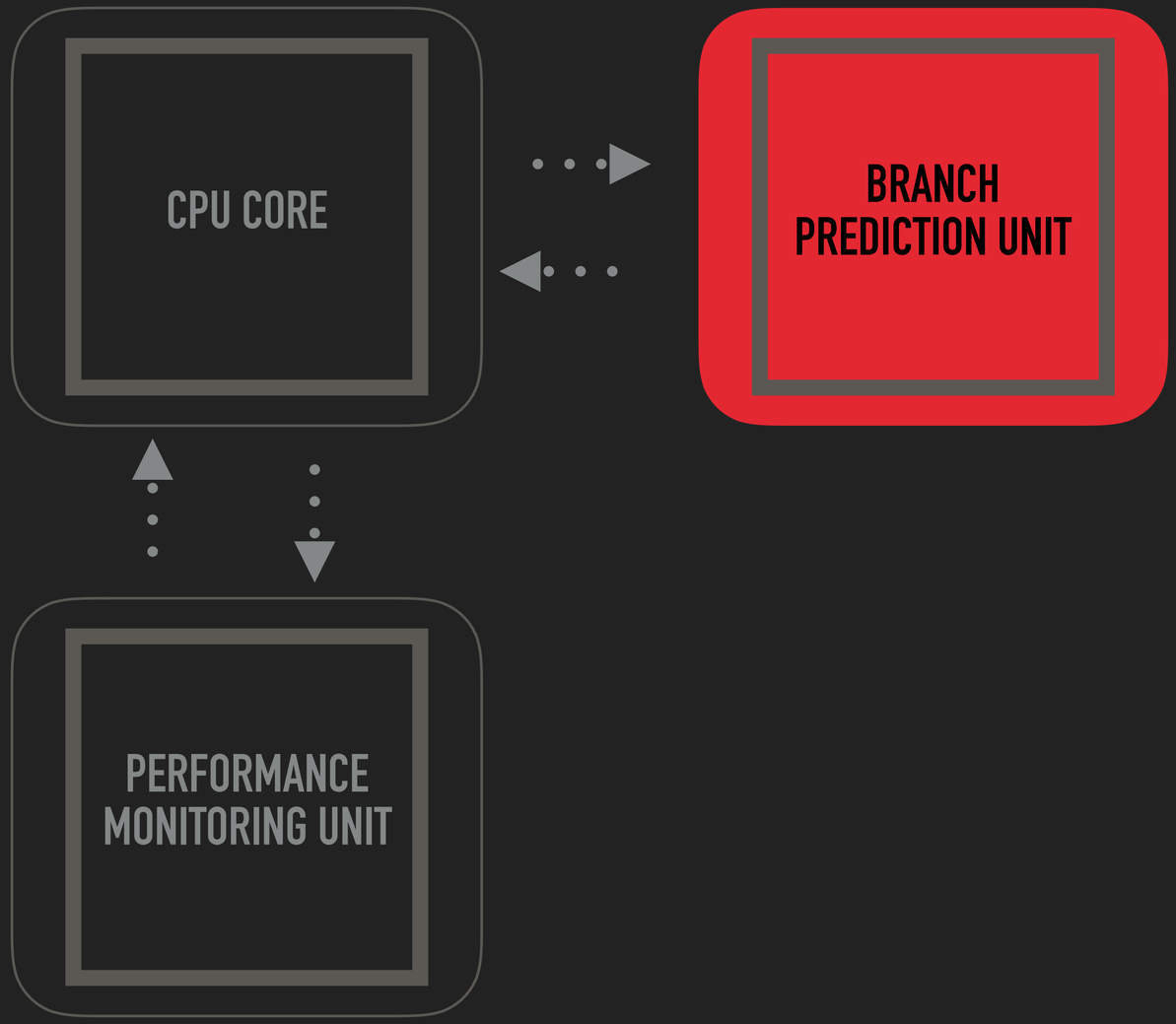
0xAE ITLB.ITLB_FLUSH

0xC0 INST_RETIRED



PERFORMANCE MONITORING UNIT FOR SECURITY

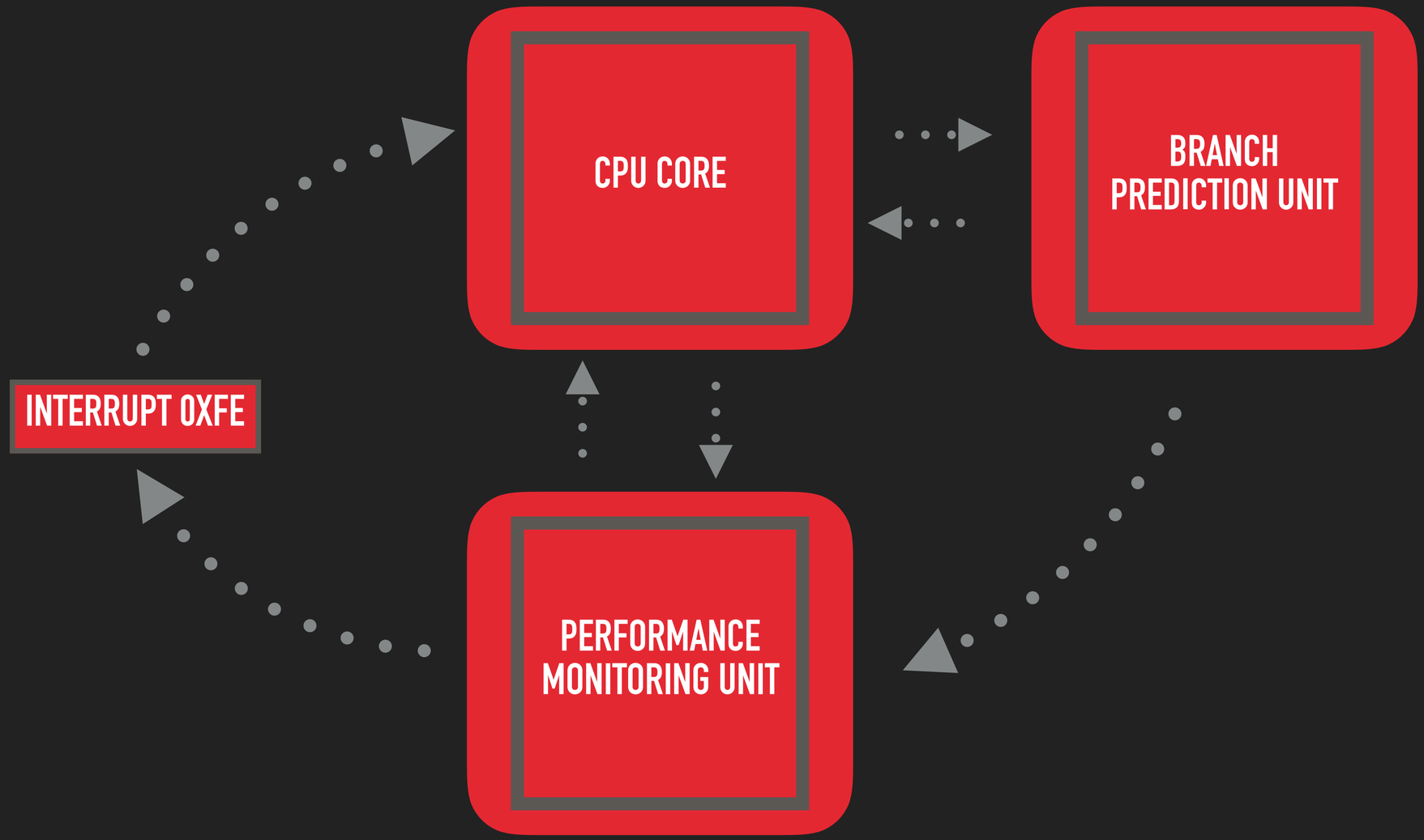
- “Security Breaches as PMU Deviation: Detecting and Identifying Security Attacks Using Performance Counters”, Yuan et al., 2011
- “CFIMon: Detecting Violation of Control Flow Integrity using Performance Counters”, Xia et al., 2012
- “kBouncer: Efficient and Transparent ROP Mitigation”, Pappas, 2012
- “Transparent ROP Detection using CPU Performance Counters”, Li & Crouse, 2014


CPU CORE
BRANCH
PREDICTION UNIT
PERFORMANCE
MONITORING UNIT

BRANCH PREDICTION UNIT

- A unit in microprocessor architectures dedicated to improving the prediction of branch destinations to increase instruction pipeline efficiency
- Better branch prediction can have a large effect on processor performance
- Misprediction penalties can be many clock cycles due to flushing and filling the correct branch into the instruction pipeline
- Indirect branches can be common in C++ applications and predicting them is crucial to performance

HOLD FOR VIDEO

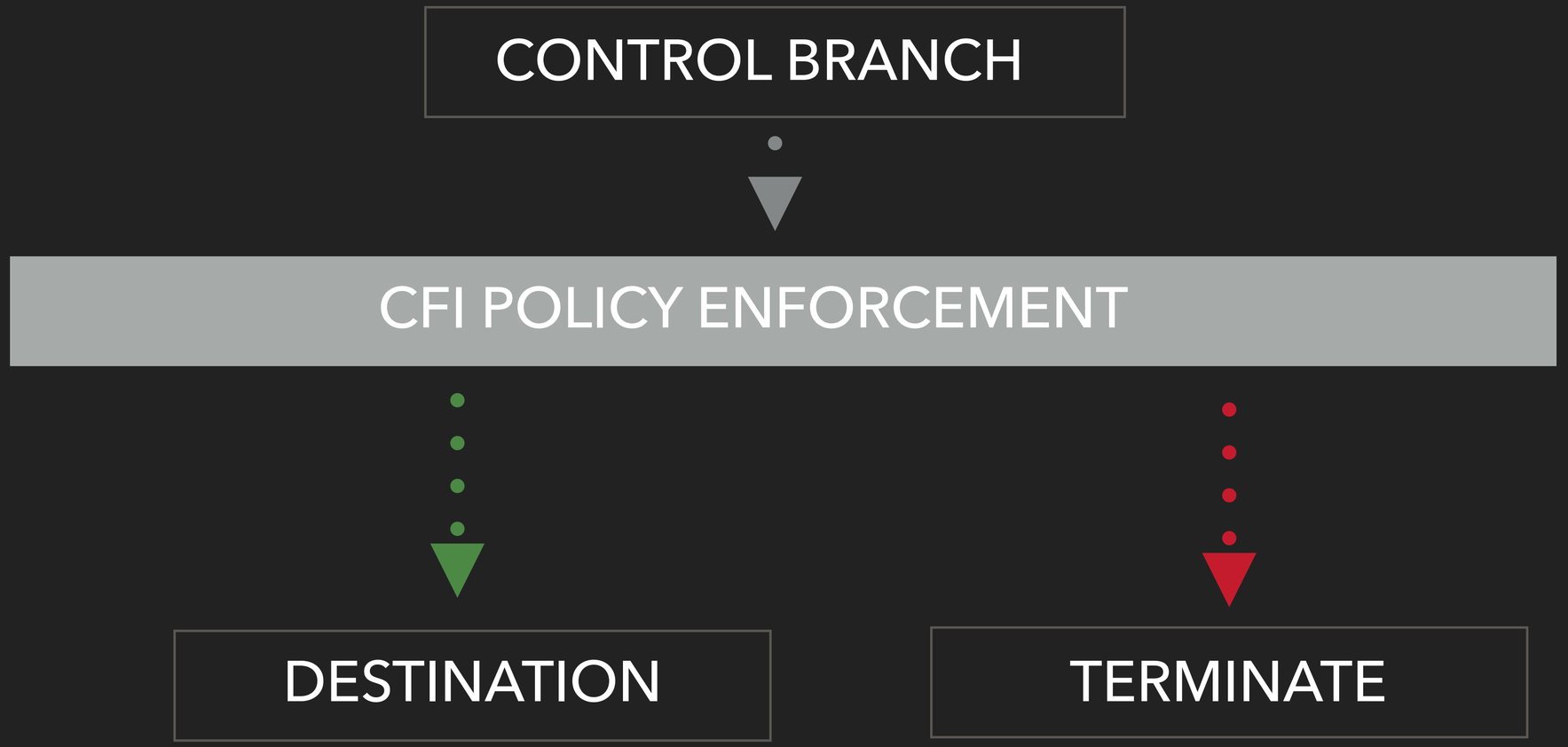

CPU CORE
BRANCH PREDICTION UNIT
INTERRUPT 0XFE
PERFORMANCE MONITORING UNIT

INTRODUCING CONTROL FLOW INTEGRITY

CONTROL FLOW INTEGRITY

- Enforcement of legitimate control flow in a program
- Traditionally done with compiler generated instrumentation
- Many different implementation of policy enforcement exist but the basic idea is to validate each indirect control flow transfer against a static list of trusted functions


CONTROL TRANSFER
DESTINATION
DESTINATION
DESTINATION


CONTROL BRANCH
CFI POLICY ENFORCEMENT
DESTINATION
TERMINATE

ALTERNATIVE CFI IMPLEMENTATIONS

- Control Flow Guard (CFG), Microsoft, 2014
- Control-flow Enforcement Technology (CET) , Intel, TBD?
- Return Address Protection/Indirect Control Transfer Protection (RAP/ICTP), PaX Team, 2015

ALTERNATIVE CFI IMPLEMENTATIONS

- Control Flow Guard (CFG), Microsoft, 2014
- Control-flow Enforcement Technology (CET) , Intel, TBD?
- Return Address Protection/Indirect Control Transfer Protection (RAP/ICTP), PaX Team, 2015
- While these are very strong implementations they require recompilation, updated software/kernel/OS, or aren’t cross platform

SCOPING OUR RESEARCH TO FILL THE GAP

- No source code access
- Cross-Platform OS support
- 32 and 64 bit support
- No pre-processing of binaries or CFG reconstruction
- Not specific to a single bug-class or exploit technique such as Use-After-Free (UAF) or Return-oriented Programming (ROP)
- Overhead must be acceptable in benchmarks and subjective user experience

REAL-WORLD VERIFICATION

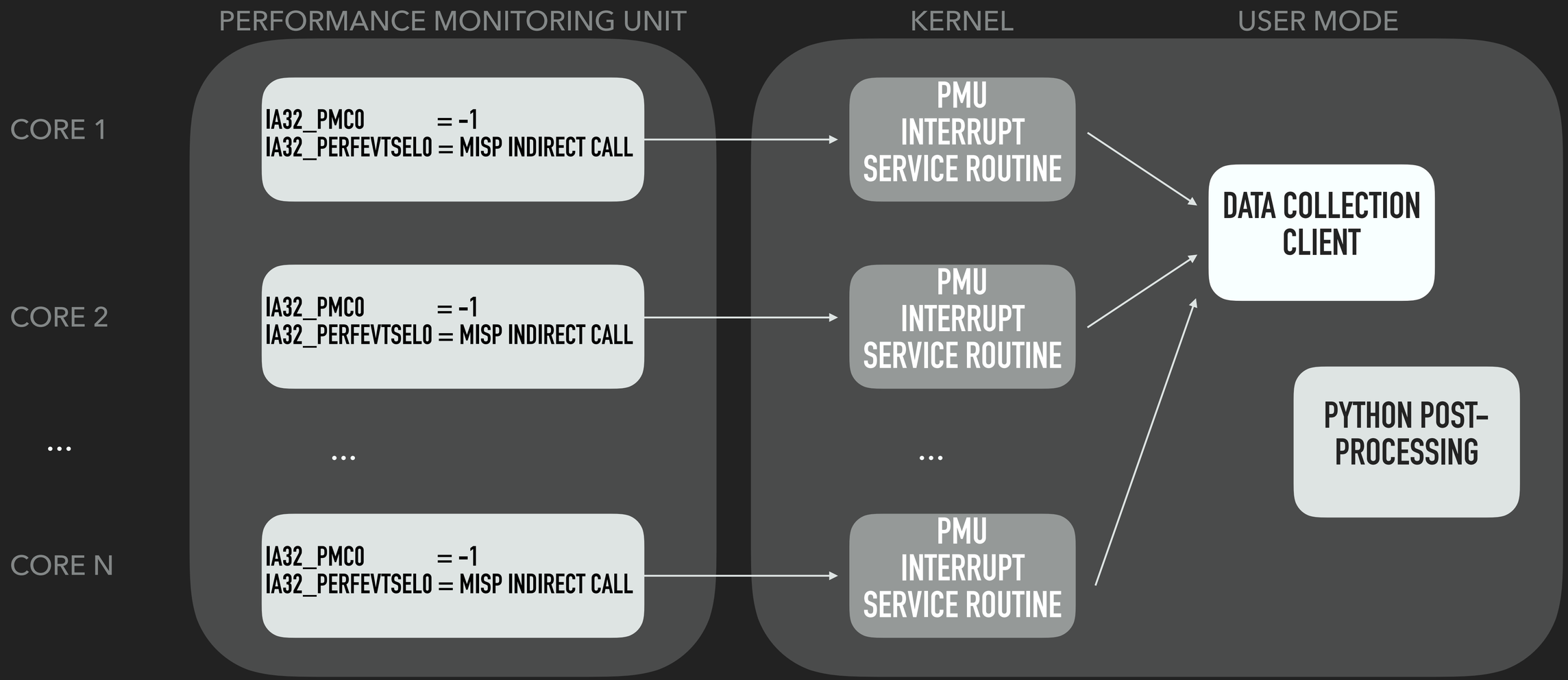
- Approach must be verified using “real” exploits and “real” software
- Cyber Grand Challenge samples
- Research community PoCs
- Metasploit modules
- Exploit Kit samples including previous 0days
- Internally developed exploits

CFI APPROACH

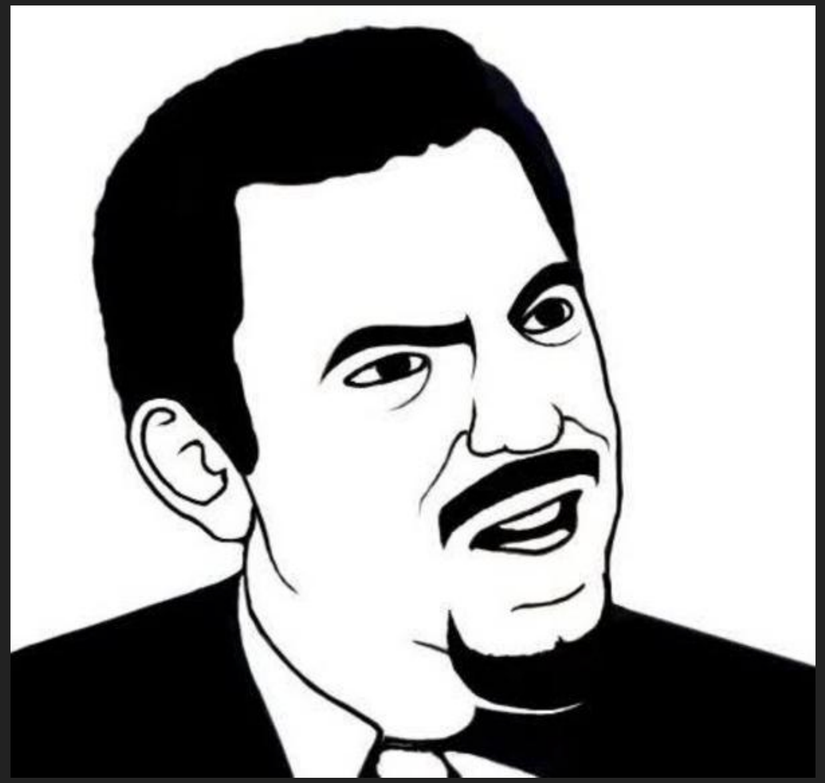
HARDWARE-ASSISTED CONTROL FLOW INTEGRITY (HA-CFI)

- Hijacked indirect branches almost always mispredicted by BPU
- HA-CFI Approach:
  - Use Intel PMU to trap all mispredicted indirect branches
  - Requires setting counter to -1
  - Use ISR for CFI policy: validate indirect branch destinations in real-time
  - Initial prototype in Linux

OUR INITIAL APPROACH



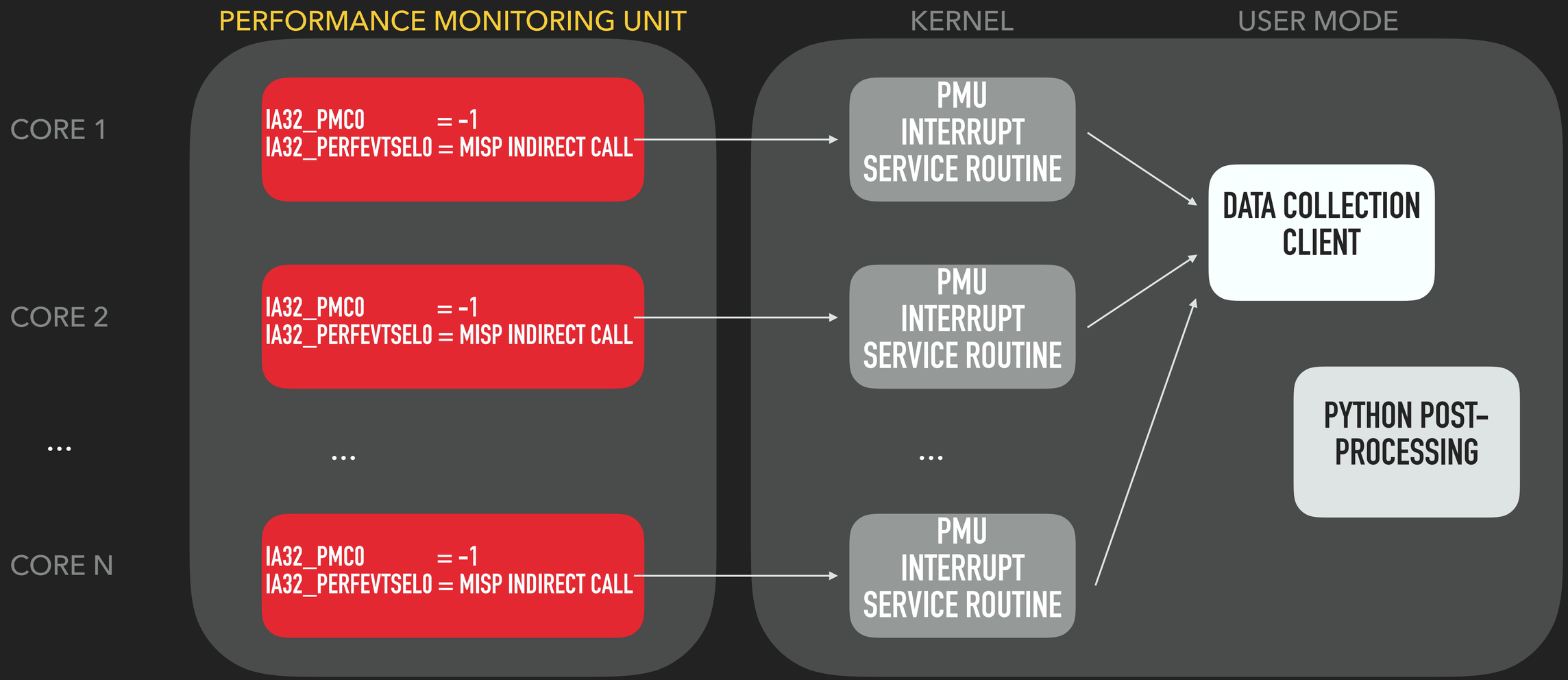
BUT WONT ALL THOSE INTERRUPTS BE EXPENSIVE?



INDIRECT BRANCH CFI COMPARISON

	Source Code Required	Patching Required	Overhead	CFI Logic Frequency
Binary Rewriting	NO	YES	LOW	100% FOR PROTECTED CALLS
Compiler Transformation	YES	NO	LOW	100% FOR PROTECTED CALLS
PMU-Assisted	NO	NO	MEDIUM	ONLY WHEN MISPREDICTED 1%-20%

PROGRAMMING THE PMU



PROGRAMMING THE PMU

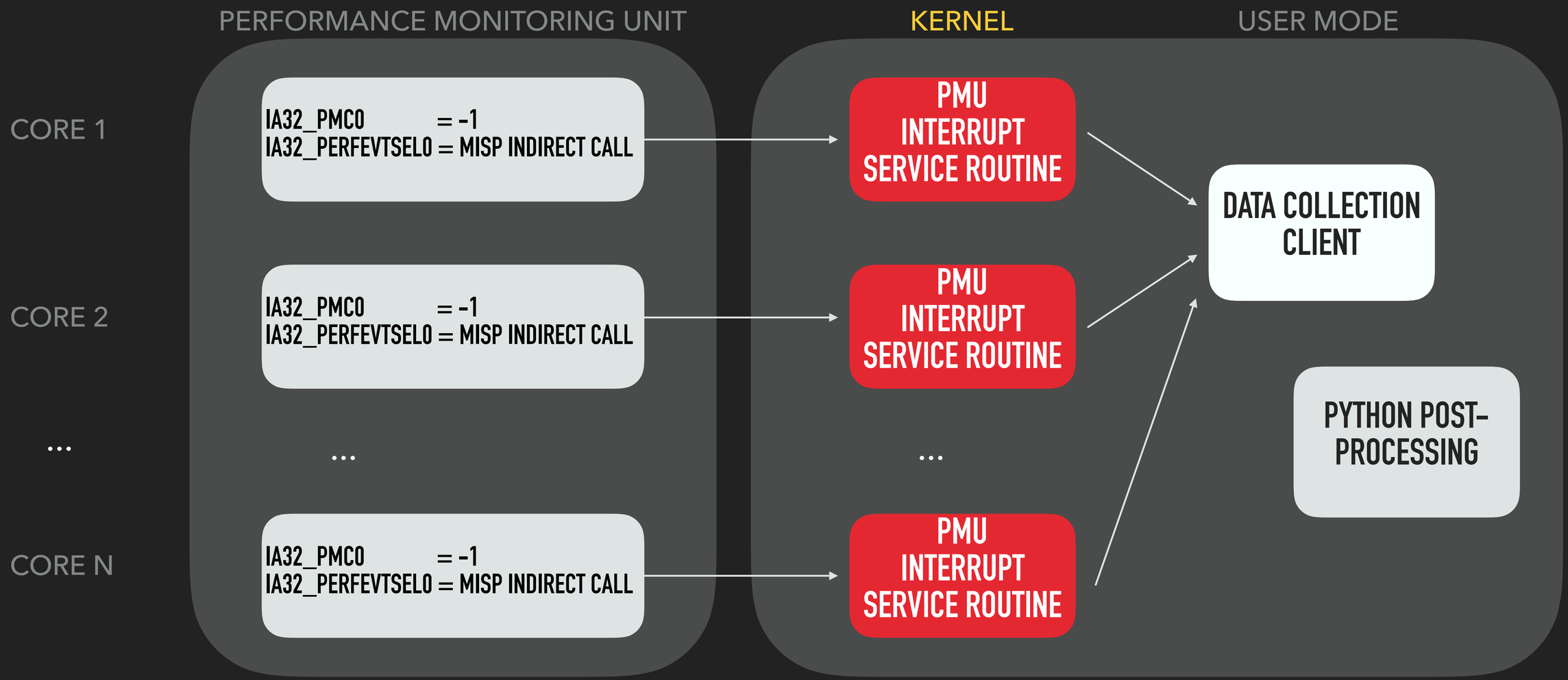
- Controlled by several Model Specific Registers (MSRs)
  - IA32_PERF_GLOBAL_CTRL : global enable/disable of counters
  - IA32_PERFEVTSELx : event to count, mode inclusion bits, interrupt bit
  - IA32_PMCx : counter value
  - IA32_PERF_GLOBAL_STATUS / IA32_PERF_GLOBAL_OVF_CTRL
    - counter overflow status and clear registers
- Additional references: Threads 2014 [Li et al], BH USA 2015 [Herath, Fogh]

INDIRECT BRANCH – INTEL PMU EVENTS

EVENT NAME	UMASK	CODE	DESCRIPTION
BR_MISP_RETIRED.NEAR_CALL	0x02	0xC5	Direct and indirect mispredicted near call instructions retired
BR_MISP_EXEC.TAKEN_INDIRECT_NEAR_CALL	0xA0	0x89	Taken speculative and retired mispredicted indirect calls

- BR_MISP_RETIRED (PEBS) counts retired only, includes direct and indirect
- BR_MISP_EXEC includes speculative events == branches falsely labeled as mispredicted
- Opted to use BR_MISP_RETIRED.NEAR_CALL since more precise and fewer Interrupts

THE INTERRUPT SERVICE ROUTINE



PMU TRAPS "ON PAPER"

IA32_PMC0: 0xFFFFFFFF (-1) Event: 0x5102C5

PMC	INSTRUCTION
-1	0x1000: MOV rax, [rsi]
-1	0x1003: MOV rdi, [rax+0x78]
-1	0x1007: CALL rdi
foo:	
0	0xB890: MOV rax, rsp
	0xB893: MOV [rax+0x20], r9d

RIP: 0xB890

PMI

PMU ISR

```
#ifdef WINDOWS
  ip = KTRAP_FRAME.RIP;
#else
  ip = pt_regs.rip;

//Apply CFI policy to RIP
do_cfi(ip);
```

PMU TRAPS IN THE REAL WORLD

IA32_PMC0: 0xFFFFFFFF (-1) Event: 0x5102C5

PMC	INSTRUCTION
-1	0x1000: MOV rax, [rsi]
-1	0x1003: MOV rdi, [rax+0x78]
-1	0x1007: CALL rdi
	foo:
0	0xB890: MOV rax, rsp
0	0xB893: MOV [rax+0x20], r9d

RIP: 0xB893

Skid = 1 Instruction

PMI

PMU ISR

?

PMU TRAPS IN THE REAL WORLD

- Due to instruction skid after overflow, no guarantee saved IP is address of branch destination
- AMD docs state skid could be up to 72 instructions
- We found 1 instruction skid (or none) to be most common on Intel
- Need a more precise way to get branch target address on PMU overflow

LBR TO THE RESCUE

- Intel Last Branch Record (LBR) can provide us precise branch addresses
- Configured and accessed via MSRs:
  - IA32_DEBUGCTL : Enable/Disable bit, Freeze on PMI bit
  - LBR_SELECT : filter types of branches
  - LASTBRANCH_x_FROM_IP / LASTBRANCH_x_TO_IP : LBR stack entries
  - LBR_TOS : Offset that points to current top of LBR stack

LBR TO THE RESCUE

Table 17-13. 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

LBR_SELECT = 0x1ED (Indirect Calls in ring > 0)

easy first check in ISR

Table 17-8. MSR_LASTBRANCH_x_FROM_IP

Bit Field	Bit Offset	Access	Description
Data	47:0	R/O	This is the "branch from" address. See Section 17.4.8.1 for address format.
SIGN_EXt	62:48	R/O	Signed extension of bit 47 of this register.
MISPRED	63	R/O	When set, indicates either the target of the branch was mispredicted and/or the direction (taken/non-taken) was mispredicted; otherwise, the target branch was predicted.

PMU TRAPS WITH LBR PRECISION

IA32_PMC0: 0xFFFFFFFF (-1) Event: 0x5102C5

PMC	INSTRUCTION
-1	0x1000: MOV rax, [rsi]
-1	0x1003: MOV rdi, [rax+0x78]
-1	0x1007: CALL rdi
	foo:
0	0xB890: MOV rax, rsp
0	0xB893: MOV [rax+0x20], r9d

RIP: 0xB893

TOS →

LBR FROM	LBR TO
0x59BC5CE5	0x75DFC3FB
0x1234	0x1000
0x8000000000001007	0xB890
0x59BC61C3	0x75DFC452

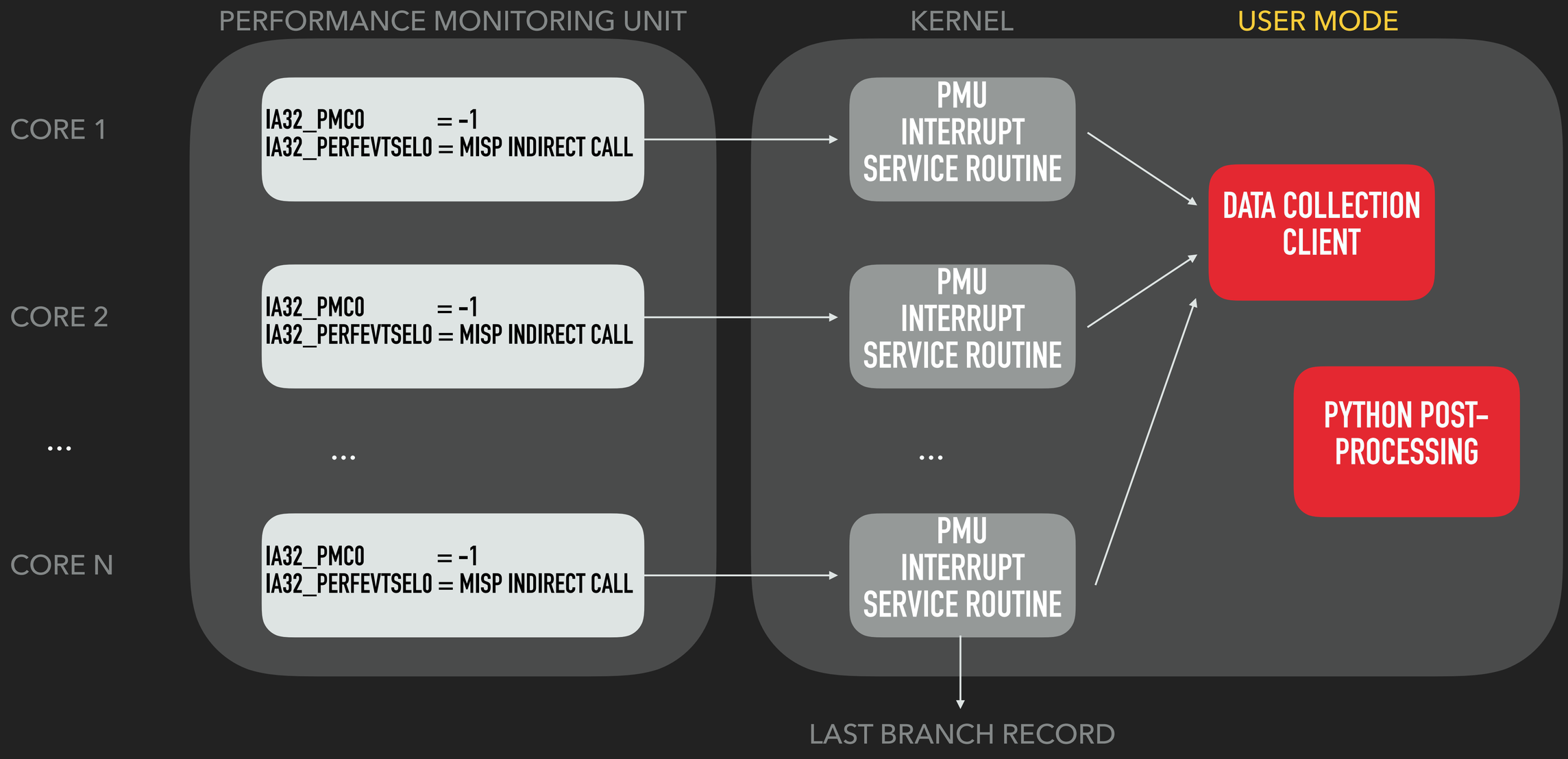
PMI →

PMU ISR

```
//Get LBR TO
tos = rdmsr(LBR_TOS);
lbr_to = rdmsr(LBR_TO + tos)

//Apply CFI policy to LBR to
do_cfi(lbr_to);
```

COLLECTING ALL THE DATA



VALIDATING APPROACH W/ CYBER GRAND CHALLENGE SAMPLES

```
vrp@ubuntu:~$ miniperf -p 8491 -i 1 -e 0x51a089
     Monitoring process: CROMU_00044 (8491)

     80007F51FCF6DDBD 7F51FCF9D5F0
     80007F51FCF9F62B 7F51FCF9E570
     80007F51FCF9E734 7F51FCF9E7C0
     80007F51FCF9E4D9 7F51FCF9CFC0
     80007F51FCF9E69D 7F51FCF9D5D0
     80007F51FCF6DDBD 7F51FCF9D5F0
     80007F51FCF9F62B 7F51FCF9E570
     80007F51FCF9E734 7F51FCF9E7C0


     …
     80007F51FCF9D69E 7F51FCF9E7C0
     80007F51FCF9E4D9 7F51FCF9CFC0
     800000000004032B2 41414141
```

```
403294:   callq  <_ZN10CUserEntry20GetLastUnreadMessageEv>
403299:   mov    %rax,-0x30(%rbp)
40329d:   mov    -0x30(%rbp),%rax
4032a1:   mov    (%rax),%rax
4032a4:   add    $0x10,%rax
4032a8:   mov    (%rax),%rax
4032ab:   mov    -0x30(%rbp),%rdx
4032af:   mov    %rdx,%rdi
4032b2:   callq  *%rax
```

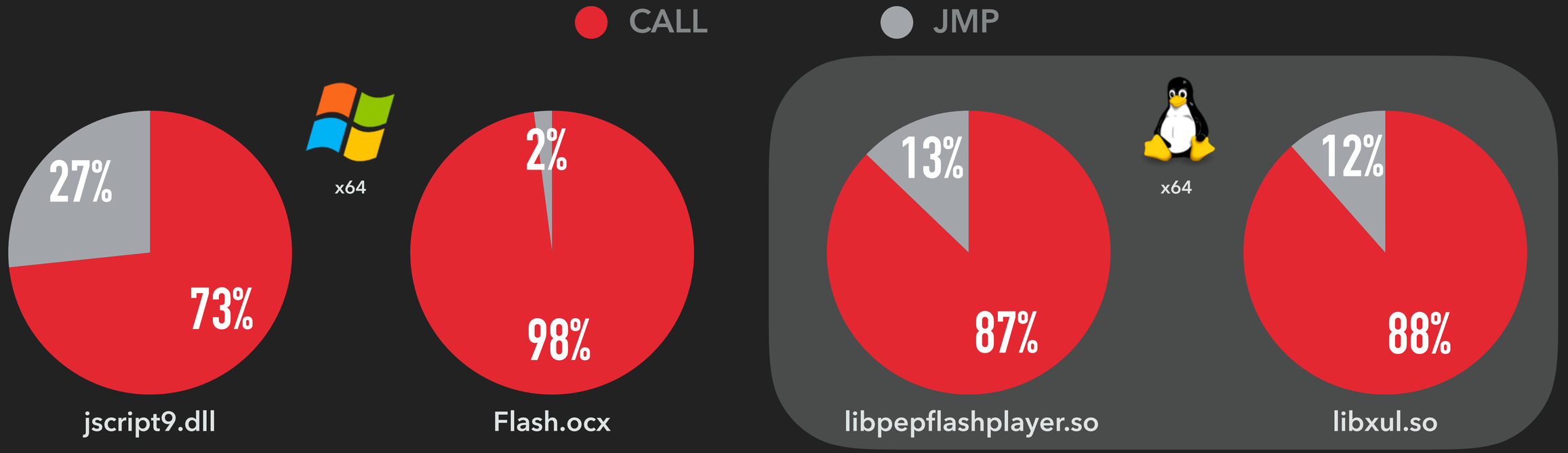
```
// Display last unread message
pCur = pUser->GetLastUnreadMessage();
printf( "From: @s\n", pCur->GetFrom().c_str() );
```

INDIRECT BRANCH ANALYSIS – CVE-2014-0556

- Ubuntu 14.04.3 LTS x64 Pepper Flash 14.0.0.177
- Moved to real-world Linux x64 POC, but missed hijack due to JMP
- Tweaked ActionScript POC from Chris Evans to generate additional data:
  - 16 unique hijack points
  - Call / JMP Analysis

ACTIONSCRIPT TRIGGER	HIJACKED CALL SITE
ByteArray.readBytes()	0x33D438: jmp rax
ByteArray.readBytes()	0x33D3BC: call qword ptr [rax]
ByteArray.readMultiByte()	0x33D1D6: call qword ptr [rax]
ByteArray.readMultiByte()	0x33D343: call qword ptr [rax+0x10]
ByteArray.readMultiByte()	0x33D1A7: call qword ptr [rax+0x10]
ByteArray.readMultiByte()	0x405358: call qword ptr [rax+0x8]
ByteArray.writeBytes()	0x33D4A8: jmp rax
ByteArray.writeBytes()	0x33D0E7: call qword ptr [rax+0x10]
ByteArray.writeMultiByte()	0x33CFFB: call qword ptr [rax+0x10]
ByteArray.writeMultiByte()	0x40805A: call qword ptr [rcx]
ByteArray.writeUTF()	0x33CE48: call qword ptr [rax]
ByteArray.writeUTFBytes()	0x33D0B8: call qword ptr [rax]
ByteArray.writeObject()	0x33D05E: call qword ptr [rax+0x10]
ByteArray.writeObject()	0x40477A: jmp rax
ByteArray.readObject()	0x33CDCE: call qword ptr [rax+0x10]
ByteArray.readObject()	0x40482B: jmp rax

INDIRECT BRANCH – CALL VS JMP

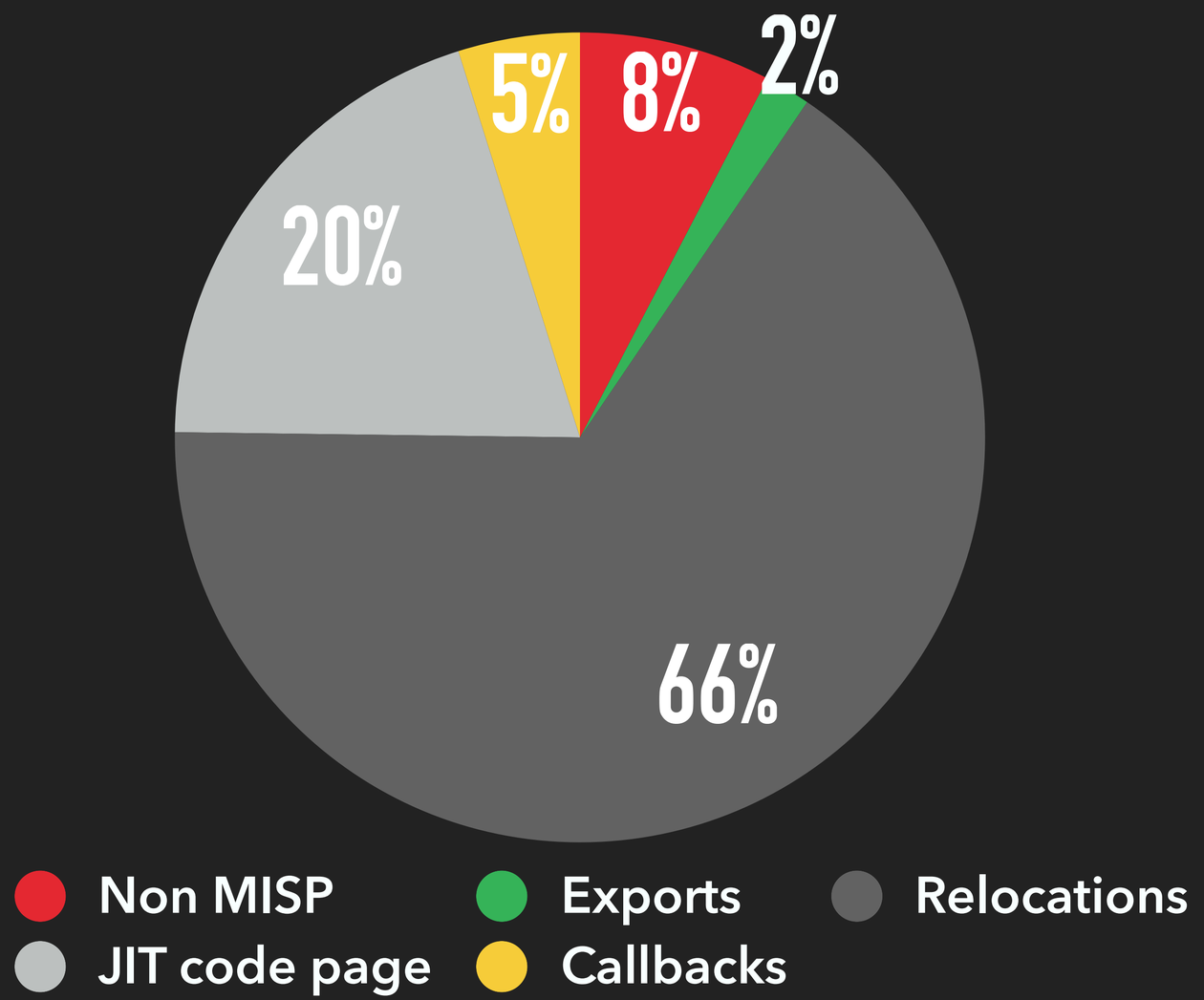


- Hijackable indirect JMP slightly more common in Linux binaries
- Indirect JMPs often used for switch statements
- For this talk we will focus exclusively on indirect CALLs

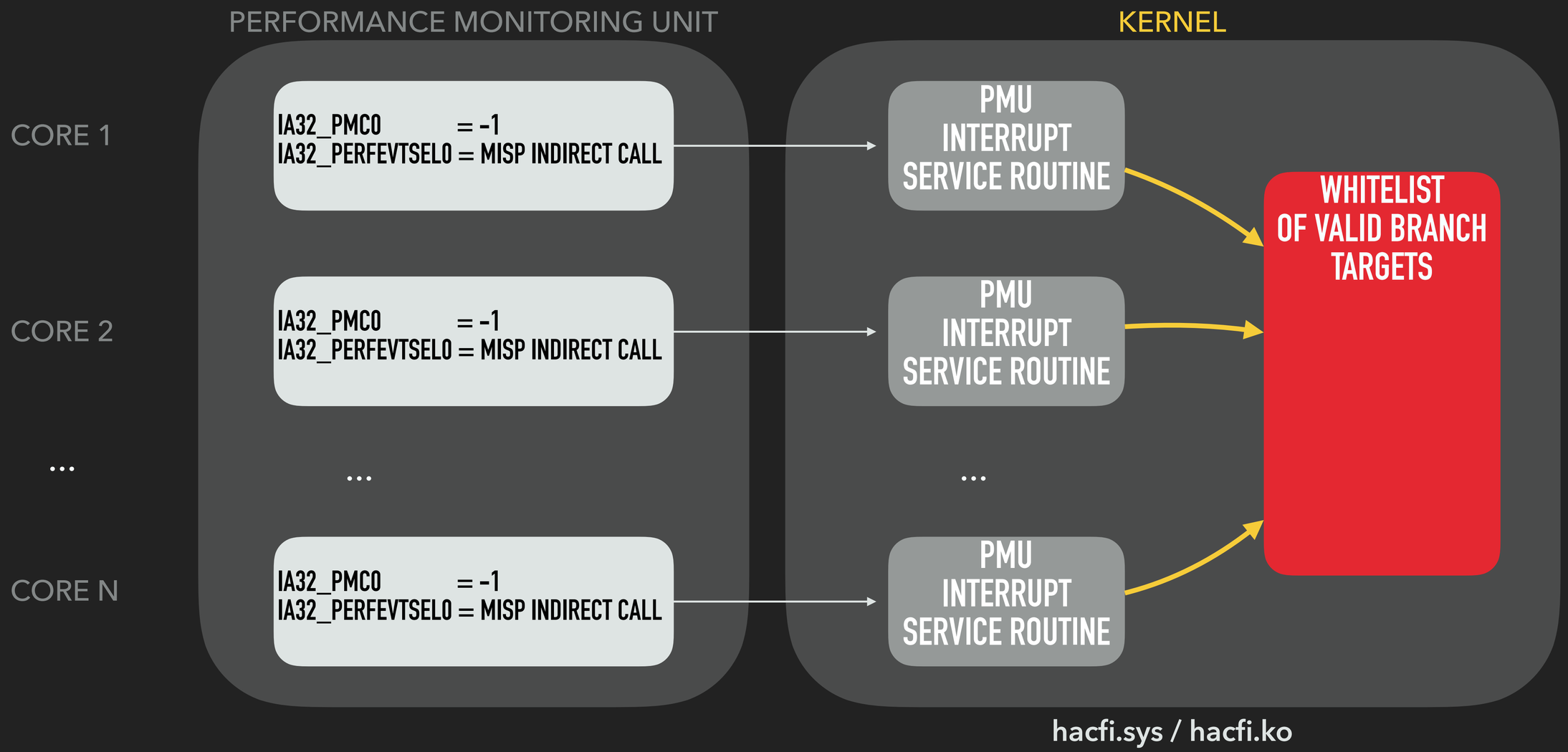
WHAT IS A VALID INDIRECT BRANCH?

Firefox
167,755,264
Branches collected

Dromaeo JavaScript Benchmark
dromaeo.com



OUR FINAL DESIGN



WHITELIST GENERATION

- Only after we were able to validate all 160M branches from ELF did we even explore real-time whitelist query
- Generate list on each image load in protected processes
- Overall approach is same on ELF and PE:
  - Find all code pointer addresses present in loaded image
  - Code pointer considered if relative or absolute address points to .text
  - Primarily focus on Exports, Relocations, and “Callbacks”

IMPLEMENTATION CHALLENGES

KEY CHALLENGES

- Receiving PMU Interrupts
- Clearing PMU Interrupts
- Thread Tracking

RECEIVING PMU INTERRUPTS ON WINDOWS

- Modifying the Interrupt Descriptor Table (IDT) for the PMU interrupt will not work for x64 due to PatchGuard
- While investigating how Windows handles PMI, we discovered a non-exported kernel routine in hal.dll
  - **`HalpSetSystemInformation()`**
    - **`InformationClass`** of **`HalProfileSourceInterruptHandler`**
    - Reachable through **`HalDispatchTable`** export

RECEIVING PMU INTERRUPTS ON WINDOWS

```
NT_STATUS _HalpSetSystemInformation(HAL_SET_INFORMATION_CLASS InformationClass, ULONG BufferSize,
PVOID *Buffer) {
  // ...

  if(InformationClass == HalProfileSourceInterruptHandler) {
    if(BufferSize != 4)
      return STATUS_INFO_LENGTH_MISMATCH;

    if(HalpFeatureBits & 1 == 0)
      return STATUS_INVALID_DEVICE_REQUEST;

    if(ProfilingProcessId == 0) {
      _HalpPerfInterruptHandler = Buffer[0];
      if(Buffer[0] != NULL)
        ProfilingProcessId = PsGetCurrentProcessId();

    } else {
      if(PsGetCurrentProcessId() != ProfilingProcessId)
        return STATUS_INVALID_DEVICE_REQUEST;
      _HalpPerfInterruptHandler = Buffer[0];
      ProfilingProcessId = (Buffer[0] ? ProfilingProcessId : 0);
    }

    return STATUS_SUCCESS;
  }

  // ...
```

RECEIVING PMU INTERRUPTS ON WINDOWS

- Pass in the interrupt handler function and it will be called when a PMI occurs

```
NTSTATUS status;
PVOID buffer[1];

buffer[0] = profileSourceInterruptHandler;
status = HalpSetSystemInformation(HalProfileSourceInterruptHandler,
                                  sizeof(PVOID),
                                  buffer);
```

- Calling (from the same process) with a NULL pointer deregisters the handler

```
NTSTATUS status;
PVOID buffer[1];

buffer[0] = NULL;
status = HalpSetSystemInformation(HalProfileSourceInterruptHandler,
                                  sizeof(PVOID),
                                  buffer);
```

CLEARING PMU INTERRUPTS ON WINDOWS

- Another issue encountered involved unmasking PMU interrupts from the handler
- PMU interrupts are delivered by the APIC
- In order to acknowledge an interrupt has been handled and to receive future interrupts, a register in the APIC needs to be written
- How this is accomplished depends on the APIC interface used, which differs between Windows versions

CLEARING PMU INTERRUPTS ON WINDOWS

xAPIC

- Existed since Pentium 4
- Windows 7
- APIC Registers are accessed through mapped physical memory
- Register access accomplished using physical memory mapped into kernel virtual memory via `MmMapIoSpace`

x2APIC

- Introduced in Nehalem microarch
- Windows 8/8.1
- APIC Registers are accessed via MSRs
- Interface can be accessed with a single __writemsr intrinsic

```
__writemsr(LVT_x2APIC_PMI, 0xFE)
```

RECEIVING PMU INTERRUPTS ON LINUX

- Setting this up on Linux is even simpler
- Register for a Non-Maskable Interrupt (NMI) handler

```
register_nmi_handler(NMI_LOCAL,
      our_nmi_handler,
      NMI_FLAG_FIRST,
      “hacfi_pmi");


unregister_nmi_handler(NMI_LOCAL, "hacfi_pmi");
```

THREAD TRACKING

- We don’t want to monitor the entire system
- Monitoring can be restricted to a few “high threat” executables
- The PMU doesn’t know anything about thread or process context

THREAD TRACKING ON WINDOWS

- Not so straightforward
- Windows has no (explicit) mechanism for executing arbitrary code at thread context switches
- Without some sort of callback when a thread quantum starts execution, we don’t know when to turn on the PMU counters
- This is a problem

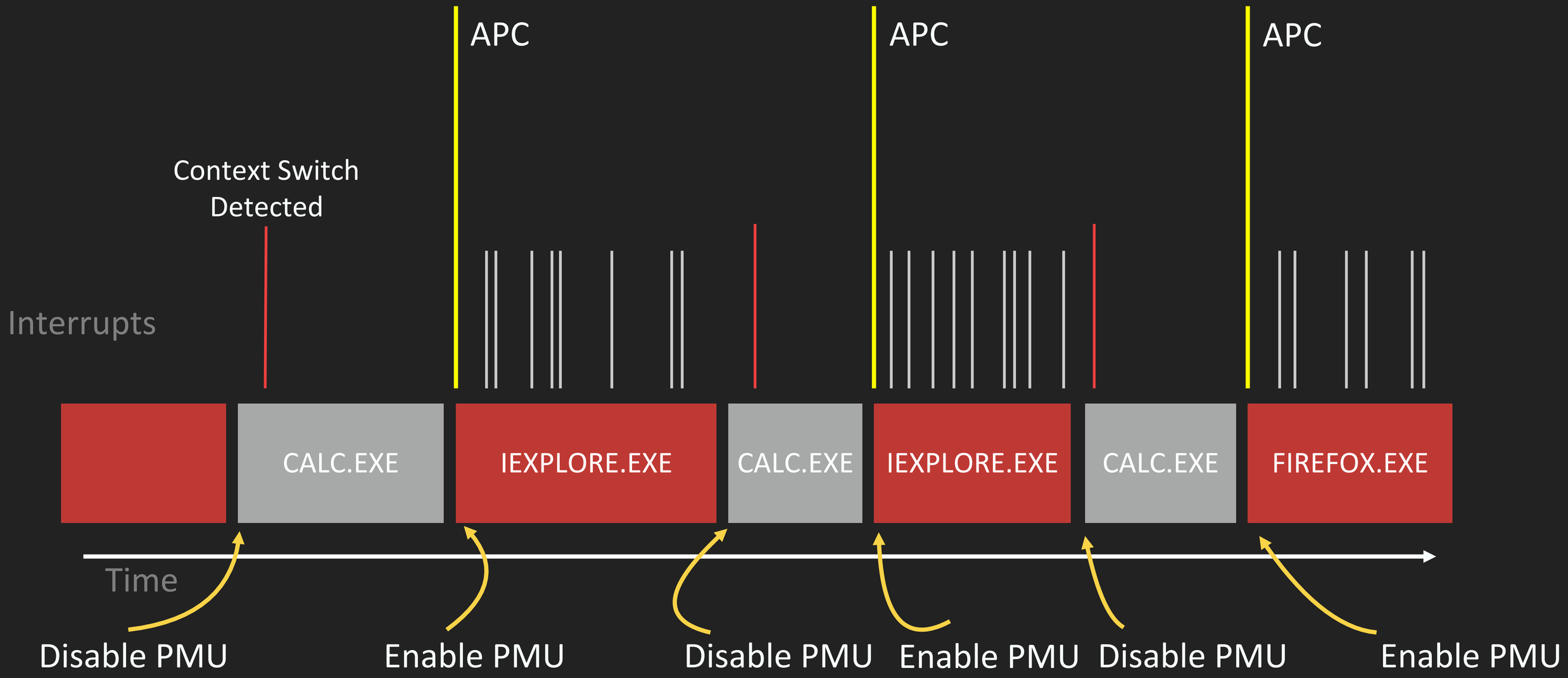
ASYNCHRONOUS PROCEDURE CALLS TO THE RESCUE

- “When an APC is queued to a thread, the system issues a software interrupt. The next time the thread is scheduled, it will run the APC function.” - Microsoft
- Perfect! We could just use APCs to get callbacks, and re-queue a new on whenever we finish the previous
- Not quite that simple, since we don’t track all threads and don’w know when a monitored quantum has ended
- Also, scheduling an APC for the current thread, from an APC handler, leads to an endless APC loop due to the software interrupt

OUR APC SOLUTION

1. Schedule a kernel APC for every thread we want to track
2. Configure PMU to trap all mispredicted branches
3. When we see an interrupt for the wrong thread, schedule a new APC for the previous thread on the processor (or all tracked threads that don’t have one currently queued)
4. Repeat

OUR APC SOLUTION



THREAD TRACKING ON LINUX

- Very straightforward
- preempt_notifier_init gives us a simple callback registration for when a thread is preempted

```
static struct preempt_notifier notifier;
static struct preempt_ops hacfi_preempt_ops = {
    .sched_in = hacfi_notifier_sched_in,
    .sched_out = hacfi_notifier_sched_out
};

static void hacfi_notifier_sched_in(struct preempt_notifier *notifier, int cpu);

static void hacfi_notifier_sched_out(struct preempt_notifier *notifier,
          struct task_struct *next);

preempt_notifier_init(&notifier, &hacfi_preempt_ops);
```


zakaazis@gmail.com

RESULTS

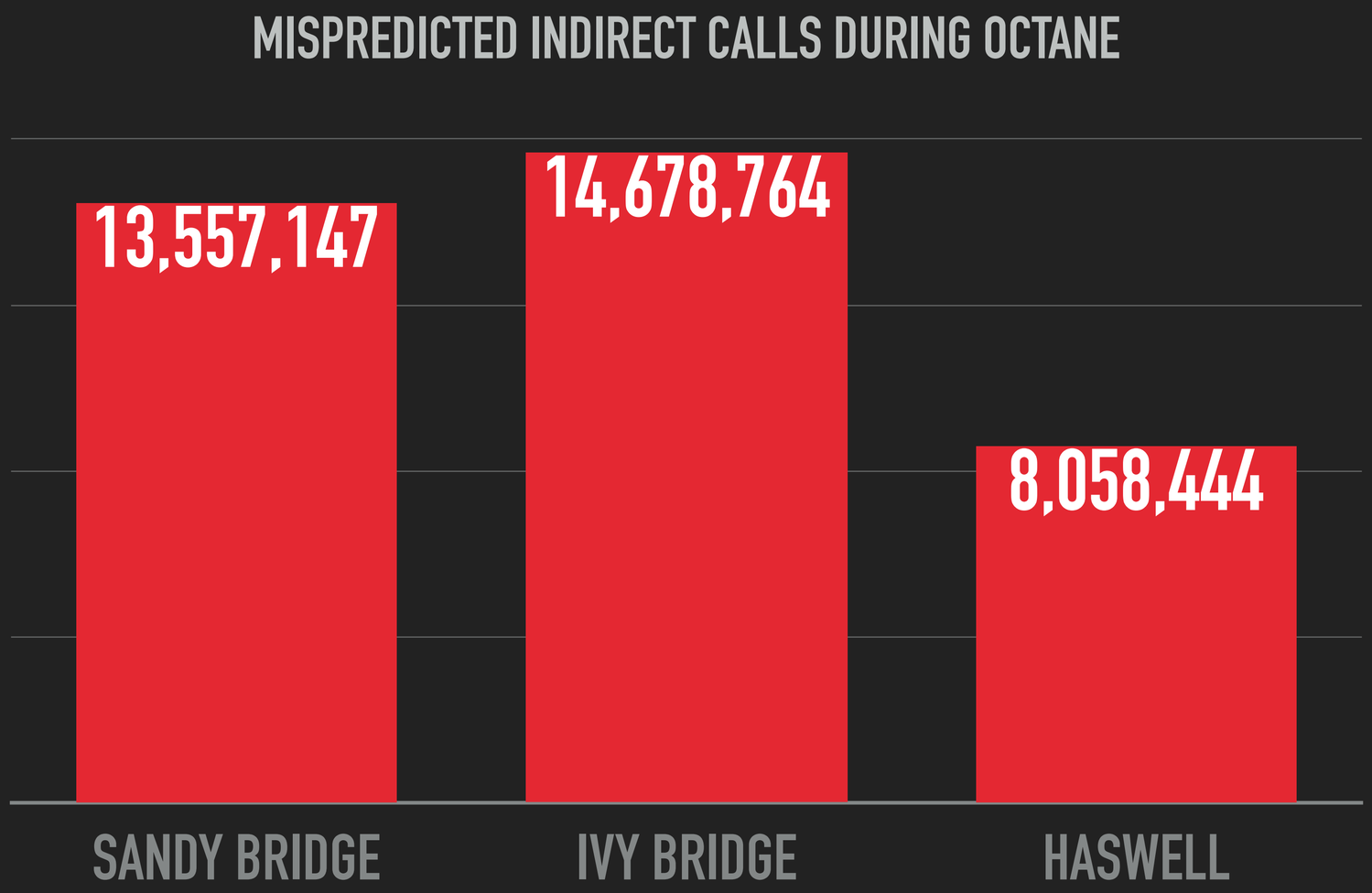
ANALYSIS OF RESULTS

- Performance Overhead
- Exploit Detection efficacy testing

PERFORMANCE

- We expect there to be a lot of mispredicted branches leading to excessive interrupts
- There is also a minor fixed overhead for each quantum from the APC
- We need to test and see if this is feasible…

HOW MANY PMU INTERRUPTS ARE WE TALKING ABOUT?



PERFORMANCE OVERHEAD

Benchmark	Baseline	HA-CFI	EMET
PassMark PerformanceTest	score: 940	9% score: 855	3% score: 910
Dromaeo JavaScript w/ Internet Explorer	325 runs/s	22% 253 runs/s	32% 220 runs/s

*TESTING PERFORMED ON AN INTEL HASWELL CPU

EXPLOIT DETECTION TESTING

- We needed exploits to test….
- We wanted exploits of recent CVEs for Adobe Flash, Internet Explorer, and Microsoft Office
- To Metasploit!

EXPLOIT DETECTION TESTING – METASPLOIT

VULNERABILITY	TARGET	DETECTION RATE
CVE-2014-0497	Flash Player 11.7.700.202	100%
CVE-2014-0515	Flash Player 11.7.700.275	100%
CVE-2014-0556	Flash Player 14.0.0.145	100%
CVE-2014-0569	Flash Player 15.0.0.167	100%
CVE-2014-8440	Flash Player 15.0.0.189	100%
CVE-2015-0311	Flash Player 16.0.0.235	100%
CVE-2015-0313	Flash Player 16.0.0.296	100%
CVE-2015-0359	Flash Player 17.0.0.134	100%
CVE-2015-3090	Flash Player 17.0.0.169	90%
CVE-2015-3105	Flash Player 17.0.0.188	100%
CVE-2015-3113	Flash Player 18.0.0.160	100%
CVE-2015-5119	Flash Player 15.0.0.189	100%
CVE-2015-5122	Flash Player 18.0.0.194	100%
CVE-2014-1761	Microsoft Word 2010	100%

EXPLOIT DETECTION TESTING

- Metasploit results were great, but what about the bad guys?
- The techniques used in an exploit matter as much or more than the actual vulnerability itself
- We don't think Metasploit is a great testbed for HA-CFI, due to lack of diversity in exploitation approach
- So we turned to VirusTotal and Exploit Kit samples collected in the wild

EXPLOIT DETECTION TESTING – VIRUSTOTAL

- VirusTotal enabled us to test on real-world malware including previously 0day exploits
- Decided that samples from some of the more popular exploit kits would be a good basis for testing
- Using actual exploits from ‘the wild’ should provide a good sample of exploitation techniques
- We chose 48 unique samples for our testbed

7
Exploit Kits

48
Samples

20
Unique CVEs

EXPLOIT DETECTION TESTING – VIRUSTOTAL

- We analyzed each sample and bucketed them into three separate categories according to exploitation technique
- **ROP Technique** - Uses standard Return Oriented Programming techniques
- **ROPless Technique A** - Flash exploitation technique invoking a wrapper routine of VirtualProtect to make shellcode executable
- **ROPless Technique B** - Similar to A, but via hijacking Method.apply() in ActionScript to find and invoke VirtualProtect directly (Vitaly Toropov)

EXPLOIT KIT DETECTION – HA-CFI VS EMET

CODE EXECUTION TECHNIQUE	# SAMPLES	HA-CFI DETECTION RATE	EMET DETECTION RATE
ROP	37	95%	100%
ROPless Technique A	1	100%	0%
ROPless Technique B	10	100%	0%

EXPLOIT KIT DETECTION – BY BUG CLASS

BUG CLASS	# CVE'S	# SAMPLES	HA-CFI DETECTION RATE
Out-of-bounds Write	3	6	83.3%
Buffer Overflow	3	6	83.3%
Integer Overflow	2	6	100%
Use-After-Free	4	14	100%
Double Free	2	4	100%
Type Confusion	3	6	100%
Race Condition	1	4	100%
Uninitialized Memory	1	1	100%

CASE STUDIES

CLASSIC ROP TECHNIQUE

- CVE-2015-2419 : Double-free in jscript9 (MS15-065)
- Magnitude EK Sample

HA-CFI blocks the initial hijack.
RIP in ISR = jscript9 + 3BE32

StackPivot detected on VirtualProtect

```
jscript9 + A7541 ; JavascriptOperators::OP_SetElementI
   call edi

jscript9 + 3BE32
   xchg eax,esp  ; stack pivot gadget
   retn

jscript9 + 4B0B5
   mov [ecx+0xC],ax  ; CoE help
   retn

jscript9 + 3BE33
   retn

kernel32 + 42C15
 VirtualProtectStub  ; mark shellcode +X
```

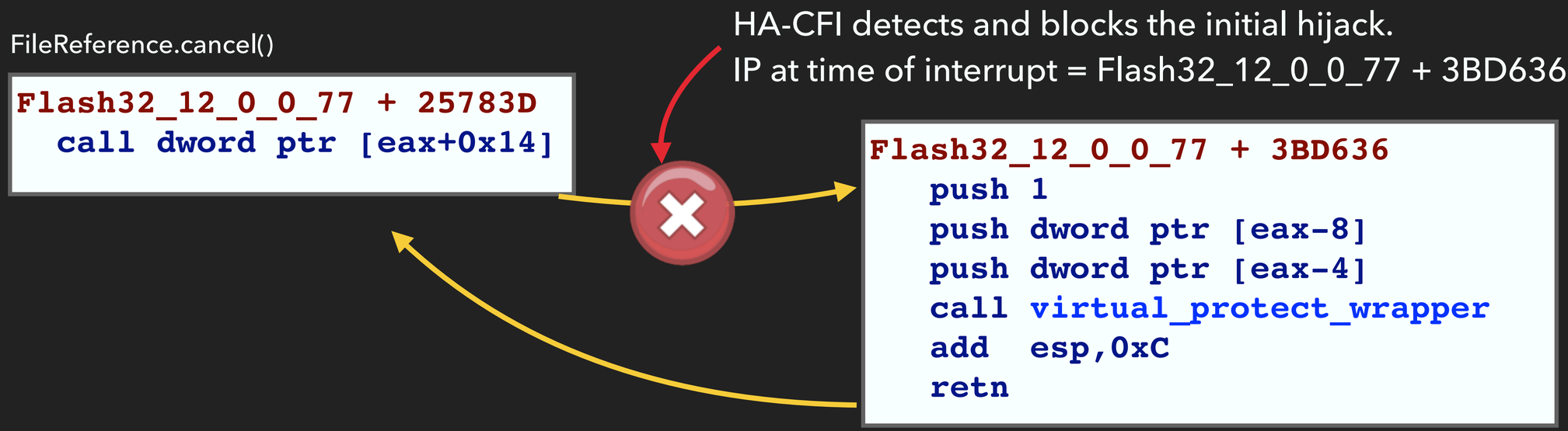

EMET
HA-CFI
HICKSON
21
SPURS
24
PARKER

ROPLESS TECHNIQUE #1

- CVE-2014-0515 : Heap overflow in Adobe Flash (patch in 13.0.0.206)
- Found in many Exploit kits and watering hole attacks
- ROPless technique re-uses VirtualProtect wrapper function in Flash image
- 2 control flow hijacks: one to VP wrapper, second one to shellcode
- Bypasses anti-ROP checks since VP invoked somewhat legitimately

ROPLESS TECHNIQUE #1

- CVE-2014-0515 : Heap overflow in Adobe Flash (patch in 13.0.0.206)



HA-CFI

FUTURE WORK

- Hypervisor support to enable hardware features in virtual machines
  - Last Branch Record (LBR) is not fully supported in popular hypervisors
  - Performance Monitoring Interrupts (PMI) on overflow is supported in many hypervisors
  - We wrote a patch for Xen to enable HA-CFI but it crashes randomly, anyone want to help?
- Just-In-Time code pages are hard to validate with our current whitelist approach

CONCLUSION

EXPLOIT DEFENSE

- Exploit defense needs to detect and prevent exploitation at the earliest phase
- Compile-time solutions are powerful, but there is room for run-time defense too
- Defenses focused exclusively on techniques such as Return-oriented Programming can be easily circumvented as new methods get adopted
- Exploits will continue to “look normal” to bypass prevention checks

HARDWARE ASSISTED CONTROL FLOW INTEGRITY

- CFI is a powerful first step in ensuring only trusted code paths can be executed
- Many vulnerabilities must hijack control-flow to achieve code execution
- Hardware can be leveraged for strong CFI policy enforcement of applications at run-time
- Many new hardware features are emerging that can be used for exploit defense

CFI ENFORCEMENT

- CFI policies can be more complex
- Powerful features of the PMU interrupt on branches for prevention are the high IRQL and complete access to context information
- We have more ideas in the works to detect additional events and apply policies to detect abnormal read, writes, and cases where attackers stay within our whitelist

SPECIAL THANKS

- Aaron Lamb, Endgame
- Gabriel Landau, Endgame
- Andrea Limbago, Endgame
- Kafeine, malware.dontneedcoffee.com
- Fellow researchers and vendors working on exploit defense

QUESTIONS?