black hat
EUROPE 2020
DECEMBER 9-10
BRIEFINGS

My Ticks Don't Lie:
New Timing Attacks for Hypervisor Detection

Daniele Cono D'Elia
@dcdelia

Who Am I

- Post-doc @ Sapienza University of Rome
- Software and systems security
  (malware, code reuse attacks, obfuscation, testing)
- BHEU’19: **BluePill** system for evasive malware
  (Neutralizing Anti-Analysis Behavior in Malware Dissection)

Turning sides this year: red pills ☺

OUTLINE

- Discrepancies of virtualization
- Building a covert time source
- Retrofitting and testing red pills
- LLC prime+probe evasion
- Outlook

Malware Analysis Today

- Hypervisors cannot be avoided
    - sandboxes run VMs on servers
    - VM introspection to implement sandbox components
    - analysts use VMs as well
- *Truly* bare-metal proposals are expensive

VIRTUALIZATION 101

VMX operation enables CPU support for virtualization

- **Virtual Machine Monitor** (VMM) acts as host: retains selective control of hw resources and offers virtual processors to guests
- VMM runs in VMX root mode, guest in non-root mode

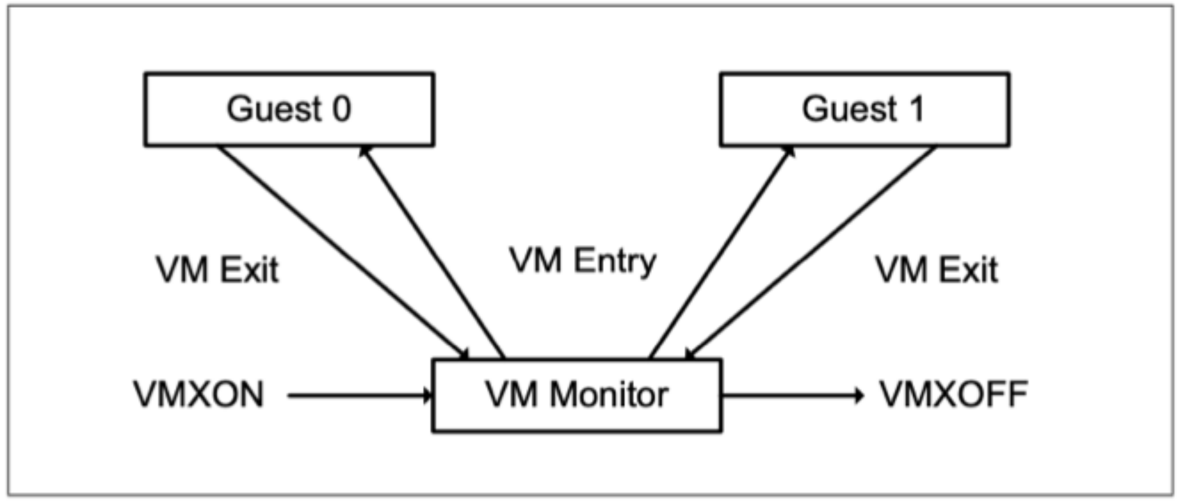


Figure 23-1. Interaction of a Virtual-Machine Monitor and Guests
(from: *Intel 64 and IA-32 Architectures SDM)*

VMCS (VM Control Structure) regulates VMX transitions and non-root operation

Virtualization is Imperfect

- Goals: compatibility, performance
  - Garfinkel [HotOs’07]: *«building a transparent VMM is fundamentally infeasible from a performance and engineering standpoint»*
- Many enhancements since first VT-x generation
  - fewer page faults, no TLB flush on VM entry
  - smaller latencies
- Transitions to VMM are inevitable though

THE CPUID CASE

- `cpuid` instruction causes a VM exit event
- Upon it VMM can control exposed properties of the virtual CPU

```
size_t ecx;
__asm__ volatile ( "cpuid" : "=c"(ecx) : "a"(1) : ...);
printf("%d\n", (int)(ecx >> 31));
```

31st bit of Extended Feature Information is the «hypervisor» bit

Timing VM Exit Events

```
movl $1, %eax
mfence
rdtsc
movl %eax, %esi
cpuid
rdtsc
subl %eax, %esi
negl %esi
```

An old detection: compare execution time of `cpuid` to bare-metal baseline

CPU: Intel i7-4980HQ

Native: ~300 cycles

VirtualBox 5.2: ~3000 cycles

Remediations for Sandboxes

- Track instructions causing VM exit (detection only)
- Optimize VMM code to reduce VM exit overhead
- **Fake values** returned by time sources
  - rewrite output of time APIs
  - make `rdtsc` cause a VM exit too, then alter its returned values (e.g. keep track of time spent in VMM)
  - simple faking schemes are easily broken

The JavaScript Lesson

- Microarchitectural attacks from browsers!
  - «The Spy in the Sandbox» (CCS'05) with `performance.now()`
  - Browser vendors reduced its resolution to 5 μs
- «Fantastic timers and where to find them» (FC'17)
  - recover resolution from coarse-grained clock
  - build alternative **covert time source**

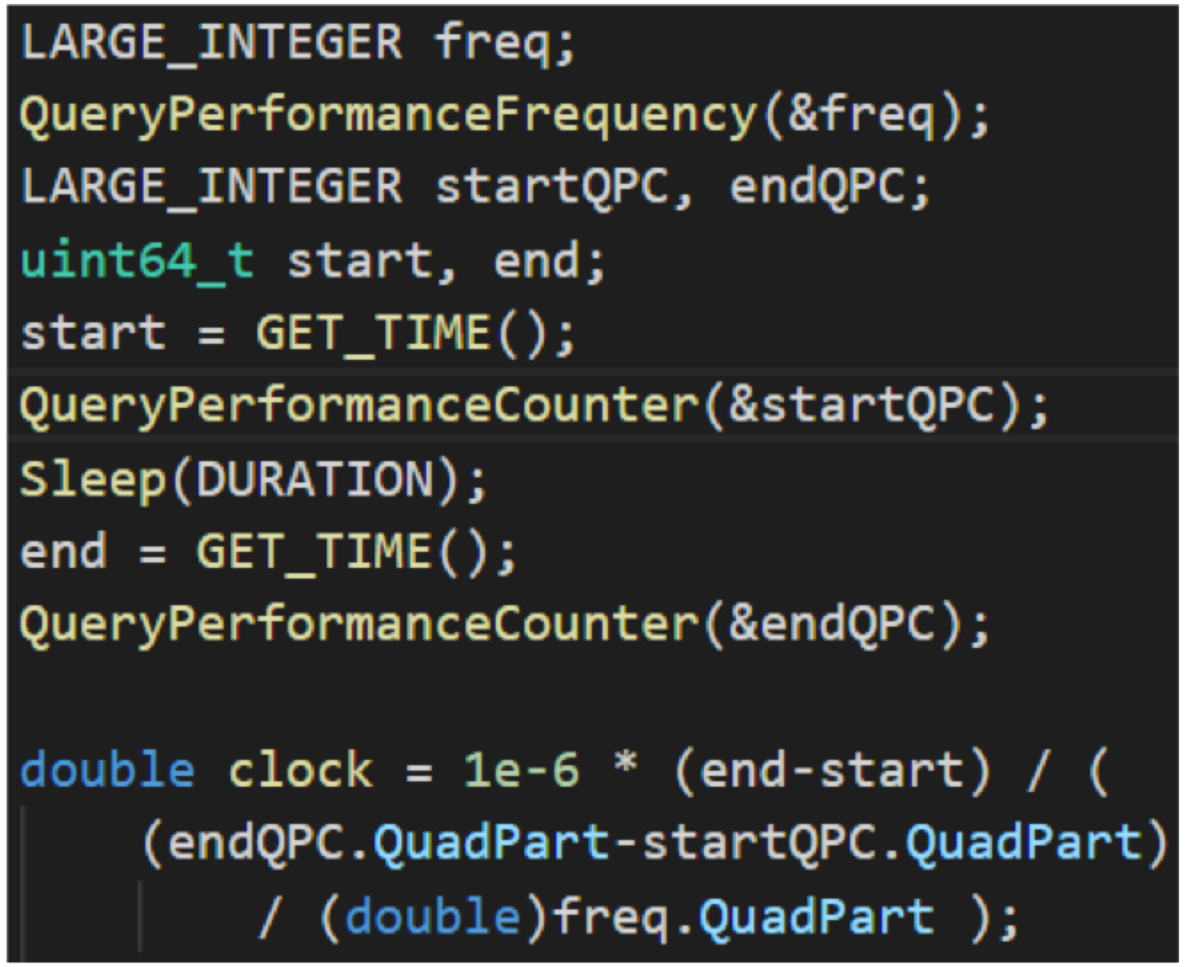
Building a Covert Time Source

```
volatile uint64_t counterClock;

// spawn a cthread
while (1) {
    counterClock++;
}

// main code
uint64_t start, end;
start = counterClock;
__asm__ volatile ("cpuid" ...);
end = counterClock;
```

Issues:

- fast enough?
- reliable?
- serialization?

Approximate Resolution



```
LARGE_INTEGER freq;
QueryPerformanceFrequency(&freq);
LARGE_INTEGER startQPC, endQPC;
uint64_t start, end;
start = GET_TIME();
QueryPerformanceCounter(&startQPC);
Sleep(DURATION);
end = GET_TIME();
QueryPerformanceCounter(&endQPC);

double clock = 1e-6 * (end-start) / (
    (endQPC.QuadPart-startQPC.QuadPart)
        / (double)freq.QuadPart );
```

GET_TIME() is rdtsc or counterClock

In recent Intel CPUs TSC ticks at nominal frequency (check with `cpuid` for TSC bits `CONSTANT` and `NONSTOP`)

QPF's freq := counts per second

Approximate Resolution

Example: Intel(R) Core(TM) i7-4980HQ CPU @ 2.80GHz

QPF’s freq := 10 millions updates per second

Tried 100 repetitions with sleep of 1000 ms

rdtsc:	~2793.5 Mhz
cthread:	~540 Mhz (~400 without TurboBoost)

A Clever Implementation

```
volatile uint64_t counterClock;

// spawn a thread
__asm__ volatile(
    "xorq %%rax, %%rax ;"
    "movq %0, %%rcx ;"
    "1: incq %%rax ;"
    "   movq %%rax, (%%rcx) ;"
    "jmp 1b ; "
:
: "r"(&counterClock)
: "rax", "rcx" );
```

Trick: avoid reading counter value from memory to update it!

Why: cost of L1 access time impacts update frequency; **`inc`** and **`mov`** have good latency and throughput

From: «Malware Guard Extension: abusing Intel SGX to conceal cache attacks» by Schwarz, Weiser, Gruss, Maurice, Mangard. Springer Cybersecurity, 2020.

A Clever Implementation

```
volatile uint64_t counterClock;

// spawn a thread
__asm__ volatile(
    "xorq %%rax, %%rax ;"
    "movq %0, %%rcx ;"
    "1: incq %%rax ;"
    "   movq %%rax, (%%rcx) ;"
    "jmp 1b ; "
:
: "r"(&counterClock)
: "rax", "rcx" );
```

i7-4980HQ (Haswell)

rdtsc:	~2793.5 Mhz
cthread:	~540 Mhz (~400 no TB)
cthread$^+$:	~**3500** Mhz (~2230 no TB)

[Schwarz20] (Skylake)

0.87 updates/cycle thanks to ILP

Cores Available

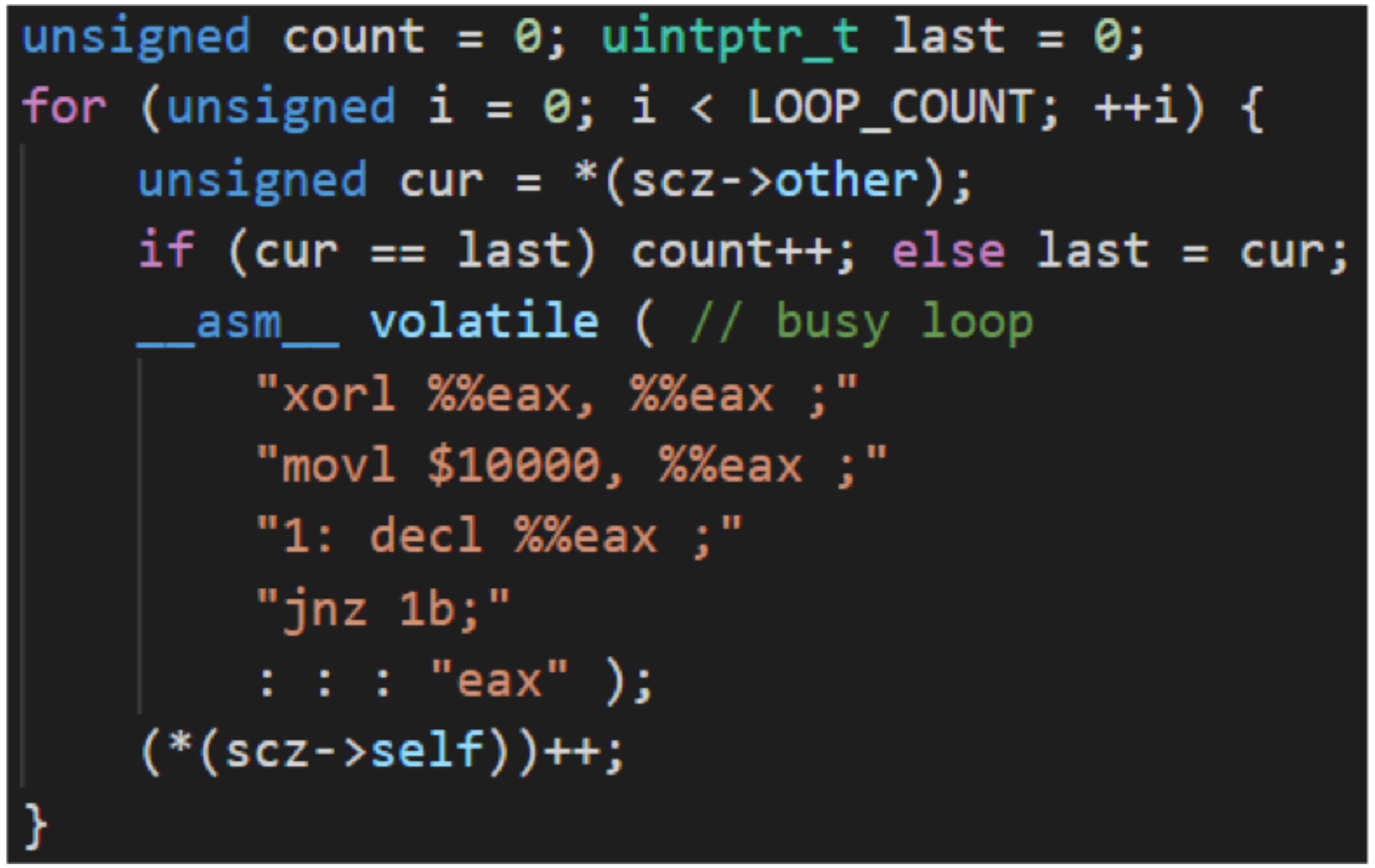
Idea: schedule two threads on mutually exclusive CPU sets, each runs a loop that checks if the other is running. Detects single-core machine

```
SYSTEM_PROCESS_INFORMATION spi;
SYSTEM_THREAD_INFORMATION sti*;
NtQuerySystemInformation(SystemProcessInformation, &spi, ...)
sti = spi[current_pid][other_tid]
if (sti->ThreadState == Ready) ... // not running
```

Adapted from «Detecting hardware-assisted virtualization» [DIMVA’16]

Cores Available

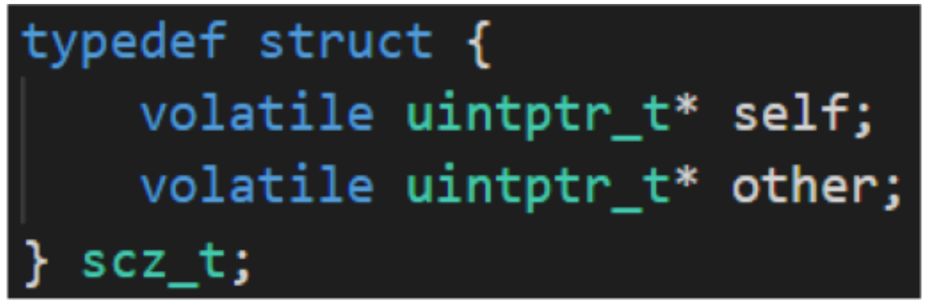
The sandbox may fake query results? We can avoid OS APIs



```
unsigned count = 0; uintptr_t last = 0;
for (unsigned i = 0; i < LOOP_COUNT; ++i) {
    unsigned cur = *(scz->other);
    if (cur == last) count++; else last = cur;
    __asm__ volatile ( // busy loop
        "xorl %%eax, %%eax ;"
        "movl $10000, %%eax ;"
        "1: decl %%eax ;"
        "jnz 1b;"
        : : : "eax" );
    (*(scz->self))++;
}
```

Race two threads, check sum of two count variables < LOOP_COUNT/2

(why: count increases when counter from other thread was not updated)



```
typedef struct {
    volatile uintptr_t* self;
    volatile uintptr_t* other;
} scz_t;
```

Counter Thread Scheduling

Counter threads may be descheduled, especially with few cores

Monotonicity preserved, but stale values are a problem

- set a high priority for the thread... 🙄
- what if we poke the counter?!? 👉🏻

Counter Thread Scheduling

```
uinptr_t last, start;
last = counterClock;
do {
    start = counterClock;
} while (start == last);
```

Check for a **heartbeat**

We read the current counter value, then read again until it changes: that's our start time...

Use fences to serialize start and end measurements

Retrofitting Red Pills

We studied several time-based VM detections

- wrote rdtsc-based, serialized red pills
  - ignored EDX
  - did not rely on RDTSCP availability
- plugged CT primitives in the code
- compared results of two schemes

Detection 1: cpuid latency

Time to execute `cpuid` > threshold

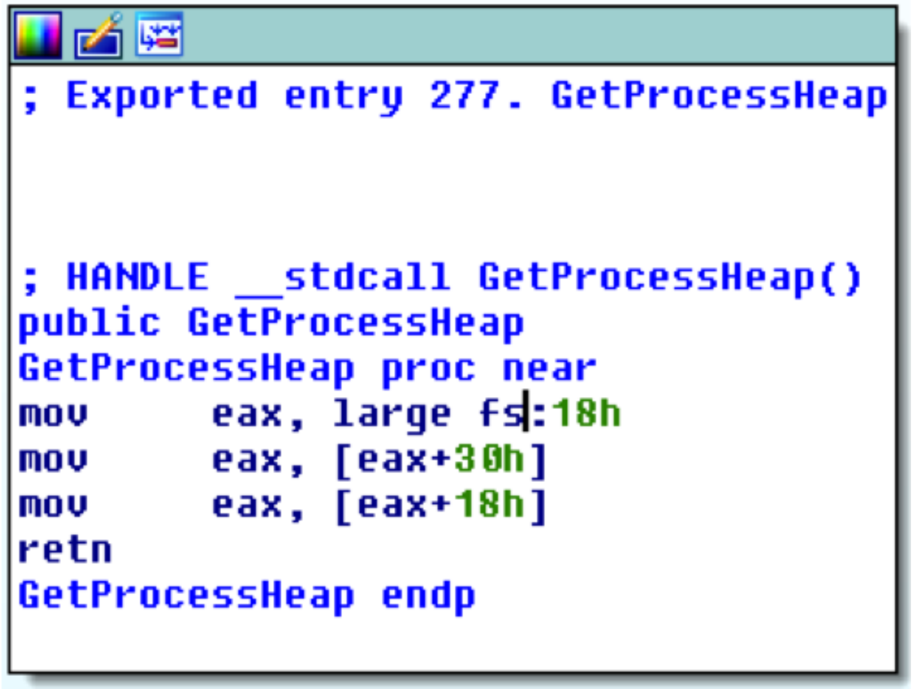
- A bad sign for a sandbox... ☺
- Initialize EAX=1
- Common settings
  - compute avg time from N=10 observations
  - 1000 threshold

Detection 2: Locky

Locky trick with GetProcessHeap/CloseHandle ratio

- GetProcessHeap() very fast on bare-metal
- ...but so is on hw-assisted virtualization!
- Compare execution time to slower CloseHandle()

Detects emulators or traps on PEB/TEB accesses



```
; Exported entry 277. GetProcessHeap



; HANDLE __stdcall GetProcessHeap()
public GetProcessHeap
GetProcessHeap proc near
mov     eax, large fs:18h
mov     eax, [eax+30h]
mov     eax, [eax+18h]
retn
GetProcessHeap endp
```

DETECTION 3: CPUID/NOP RATIO

Execution time ratio between `cpuid` and another instruction

- Absolute values: CPU-dependent, forgeable
- Idea: different instructions execute similarly slower or faster under different CPUs and conditions
- Use a low-latency instruction as reference (e.g. `nop`)

Reproduction of «Detecting hardware-assisted virtualization» [DIMVA'16]

Detection 4: TLB Eviction

Look for TLB entries evicted by VMM execution

- Fill TLB and cause a VM exit. Then analyze memory access latencies
- Pros: not fooled by subtracting VMM time
- Cons: universal TLB filling strategy?

Detection 4: TLB Eviction

Linear filling [DIMVA'16]

- Guess TLB_{size}
    - Try common values by increasing order
    - Correct value is usually the one right before self-eviction starts
- Allocate TLB_{size} pages. Use `clflush` for L* effects
- Measure max access time among all pages

Reproduction of «Detecting hardware-assisted virtualization» [DIMVA'16]

Poking Sandboxes

- We submitted a harmless poking program to many sandboxes
  - Public services, subscriptions
  - Many people downloaded it from feeds!
- Preliminary study
  - 272 execution instances
  - 81 not immediately interrupted
  - **76** completed at least 3/4 of our tests

COUNTERS IN SANDBOXES?

Timing primitives

➢ CT vs TSC frequency

>= TSC	44% of machines
>= 0.5 TSC	62.5%
>= 0.4 TSC	84.4%

(and in many cases TSC frequency was significantly different than what you'd expect for the CPU model exposed by the system...)

Results for CPUID

- Detection 1 fast but not very reliable. When lower, we considered also the the cpuid latencies recorded for Detection 3. Overall:

23/76	< 1000 TSC delta
33/76	[1000, 10000]
10/76	> 10000

- CT values also high when TSC>1000. As for the 23 below it:
  - 14 tested as single-core 🤭
  - 6 fake TSC values (CT high)
  - 1 QEMU case, 1 timed out too fast, 1 borderline

RESULTS FOR LOCKY AND DIMVA16

- **Locky** test gave surprising results
  - 23/76 were running in an emulator or trapping
  - 13 of them also tested as single-core
- **cpuid/nop ratio** detected 55/76 machines. Then:
  - 2 failed Locky, 7 had fake TSC values
  - 12 with high noise for nop, same with CT
- **TLB eviction** completed by 52 machines
  - high/unstable latencies for 23
  - 24/29 detected, no discrepancies

New Detection: LLC Prime+Probe

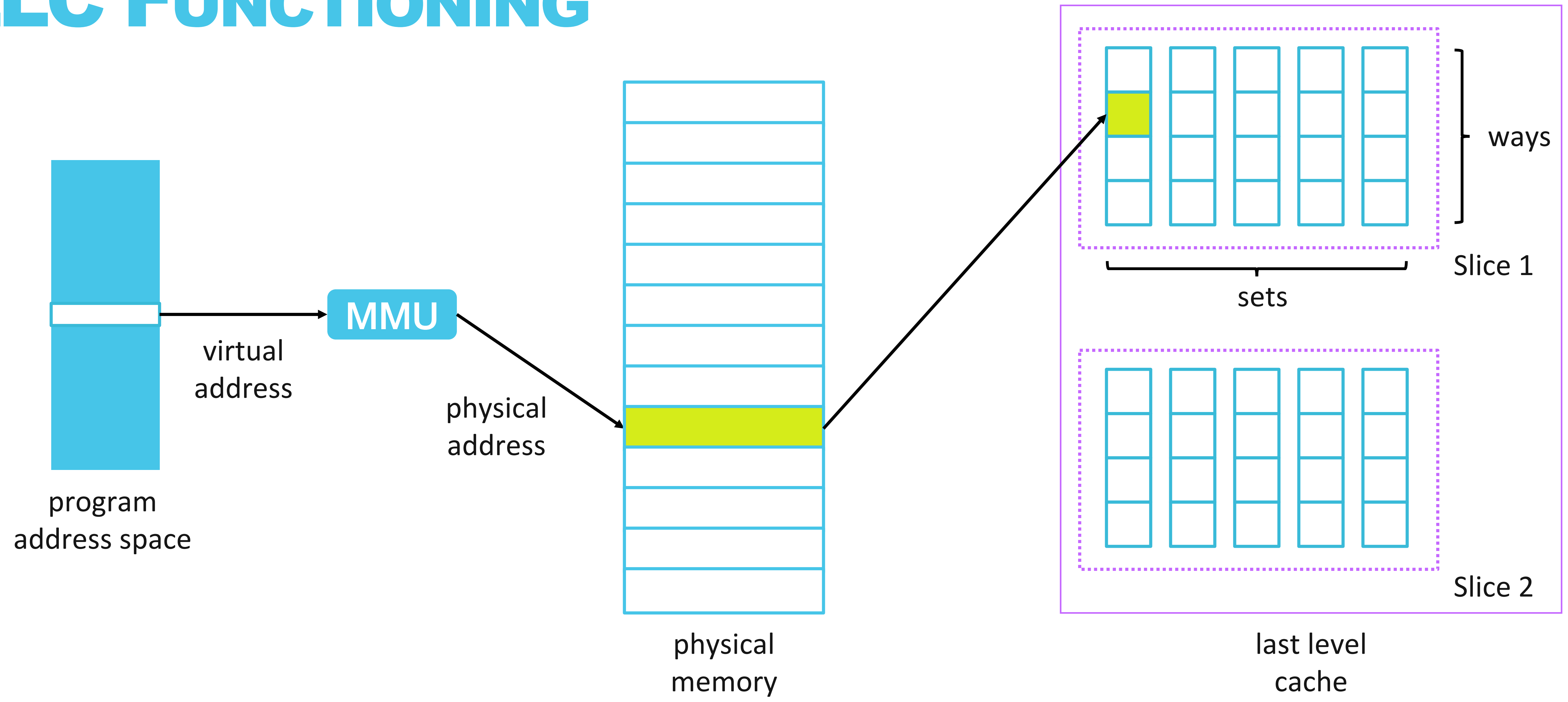
Look for effects on caches quite reliable to measure

Idea: search for LLC lines evicted by VMM execution

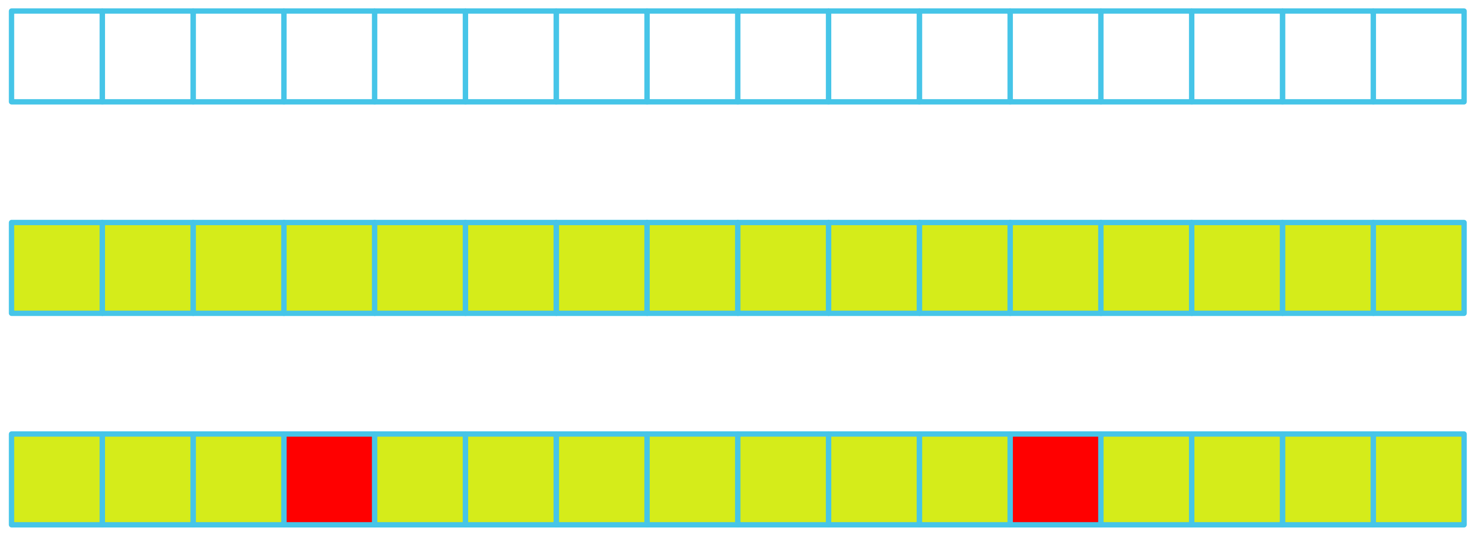
Why LLC?

- high resolution
- shared between cores
- (usually) inclusive

LLC Functioning



Prime+Probe Attack



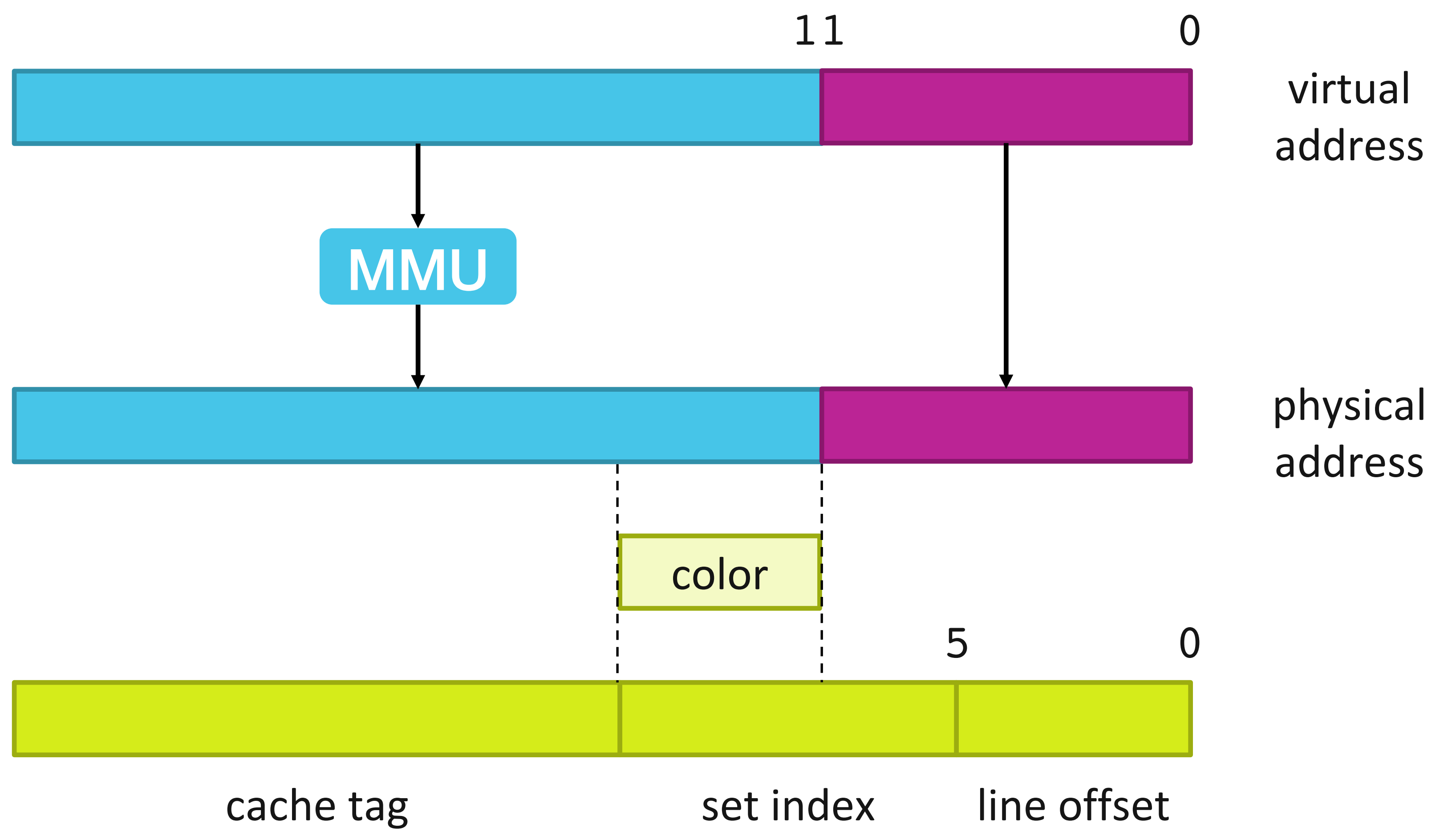
Some attacker-controlled lines will see higher latency from LLC miss!

Filling LLC Sets?

- An **eviction set** contains virtual addresses that map to one cache set
- Cache associativity determines optimal size
- We need to build a minimal eviction set for all available **colors**

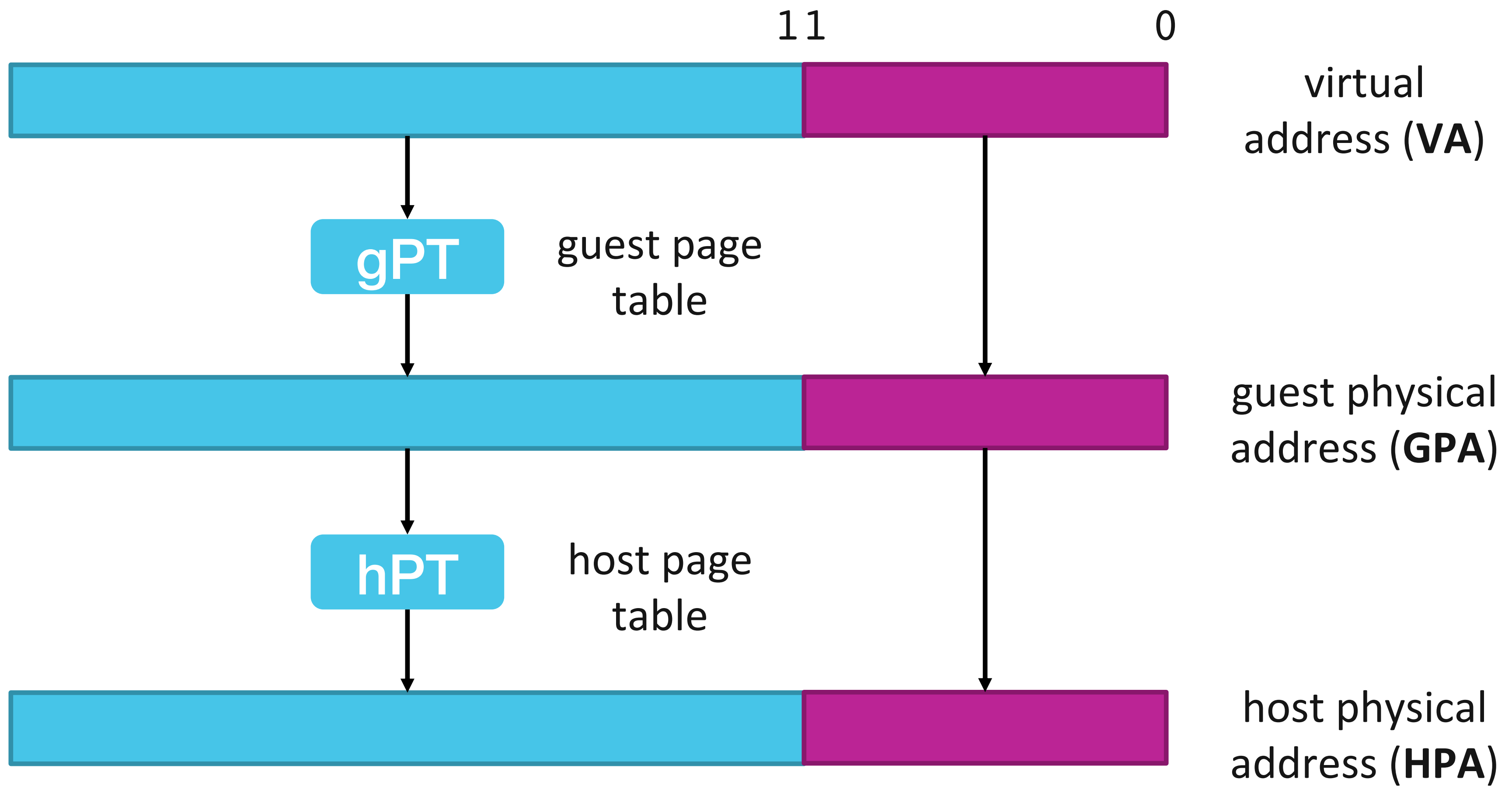
black hat EUROPE 2020

LLC Addressing



#BHEU @BLACKHATEVENTS

EVICTION SETS IN A VM?



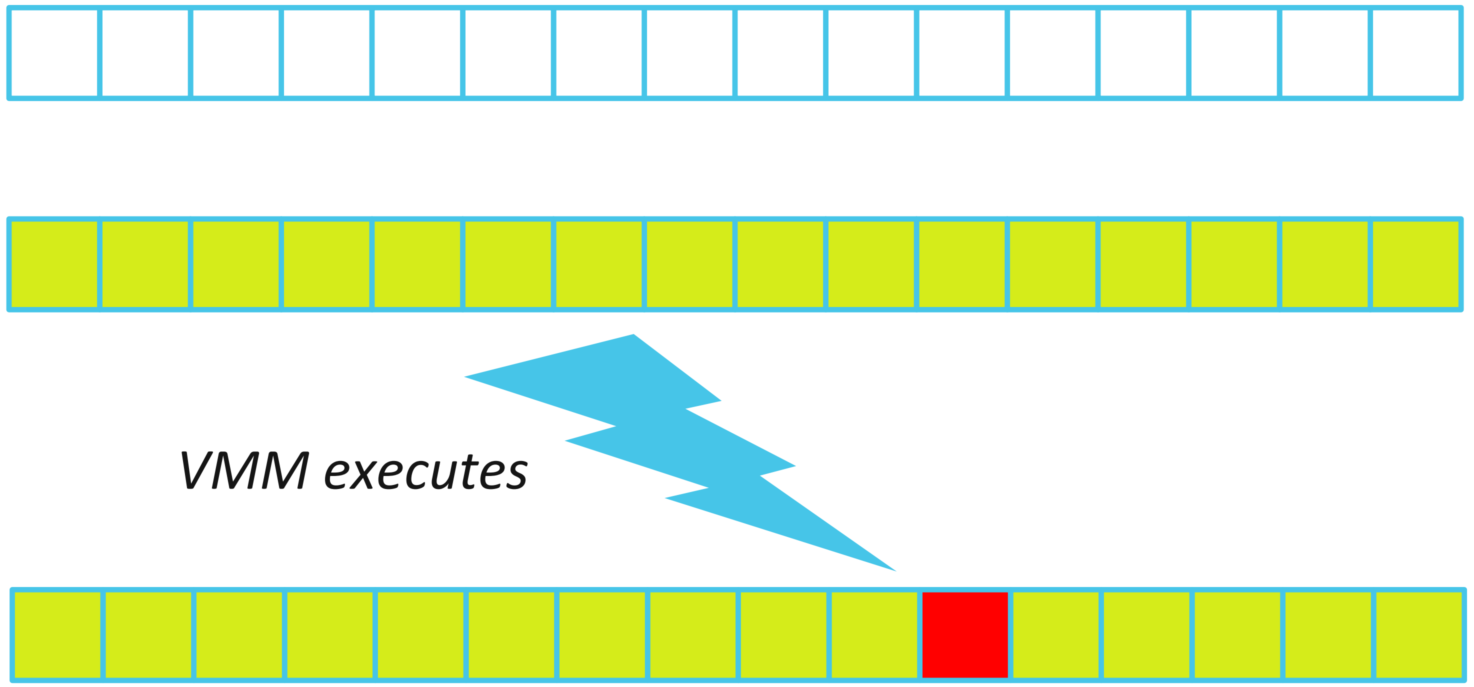
In a sandbox we have no knowledge of the mapping VA-HPA

FINDING EVICTION SETS

Theory and Practice of Finding Eviction Sets [S&P'19]

- no assumptions on the mapping between VAs and cache sets
- choose a buffer large enough (≈ cache size) to evict a target
- prune it to build an eviction set
- $O(n*w^2)$ makes it rather practical

LLC P+P FOR VM DETECTION



pick eviction set(s)

preload stage + prime

`cpuid`

compute max access time among lines in set

EXPERIMENTS

Implementation

- OS-agnostic, can use rdtsc or counter threads
- tested on Intel CPUs from Ivy Bridge to Whiskey Lake
- different combinations of VirtualBox, VMware, KVM, Xen

Selected Results

i7-8665U (8MB, 16w, 128 colors)

VirtualBox 6.1, Win host:	20/128
VirtualBox 6.1, Linux host:	18/128
VMware W. Pro 15, Win host:	10/90 found
QEMU-KVM 4.2.50:	13/128
Typical running time:	2-3’

More VirtualBox configurations:

5.2.44, Linux host, i7-3437U:	7/64
5.2.18, Mac host, i7-4980HQ:	9/128
6.1.16, Win host, i7-4770HQ:	17/128
5.2.38, Win host, i9-8950HK:	10/192

For other CPU/hypervisor/host configs we observed very similar trends

A custom VMM may pollute even more cache sets during analysis

Limitations

- Execution time may be long (> timeout) for big caches
- Eviction set construction may fail (e.g. non-inclusive LLC)

Next directions

- «Attack Directories, Not Caches» [S&P'19] non-inclusive
- «Dynamically Finding Minimal Eviction Sets Can Be Quicker Than You Think for Side-Channel Attacks against the LLC» [RAID'19] speed

Outlook

«µarch 🤝 malware» could be a promising research area

Defenses

- static & dynamic code analyses
- performance counters

Threats

- look for specific VMM features
- try other µarch «vectors»

Stay tuned :)

@dcdelia

CREDITS

For their help in different stages 🤗

- Cristian Assaiante
- Pietro Borrello
- Federico Palmaro

...and to FlatIcon.com for making this presentation just a bit more entertaining!

black hat EUROPE 2020

Thank You!