
SPQR

New Directions in Hypervisor Detection

by cristianrichie

About Me

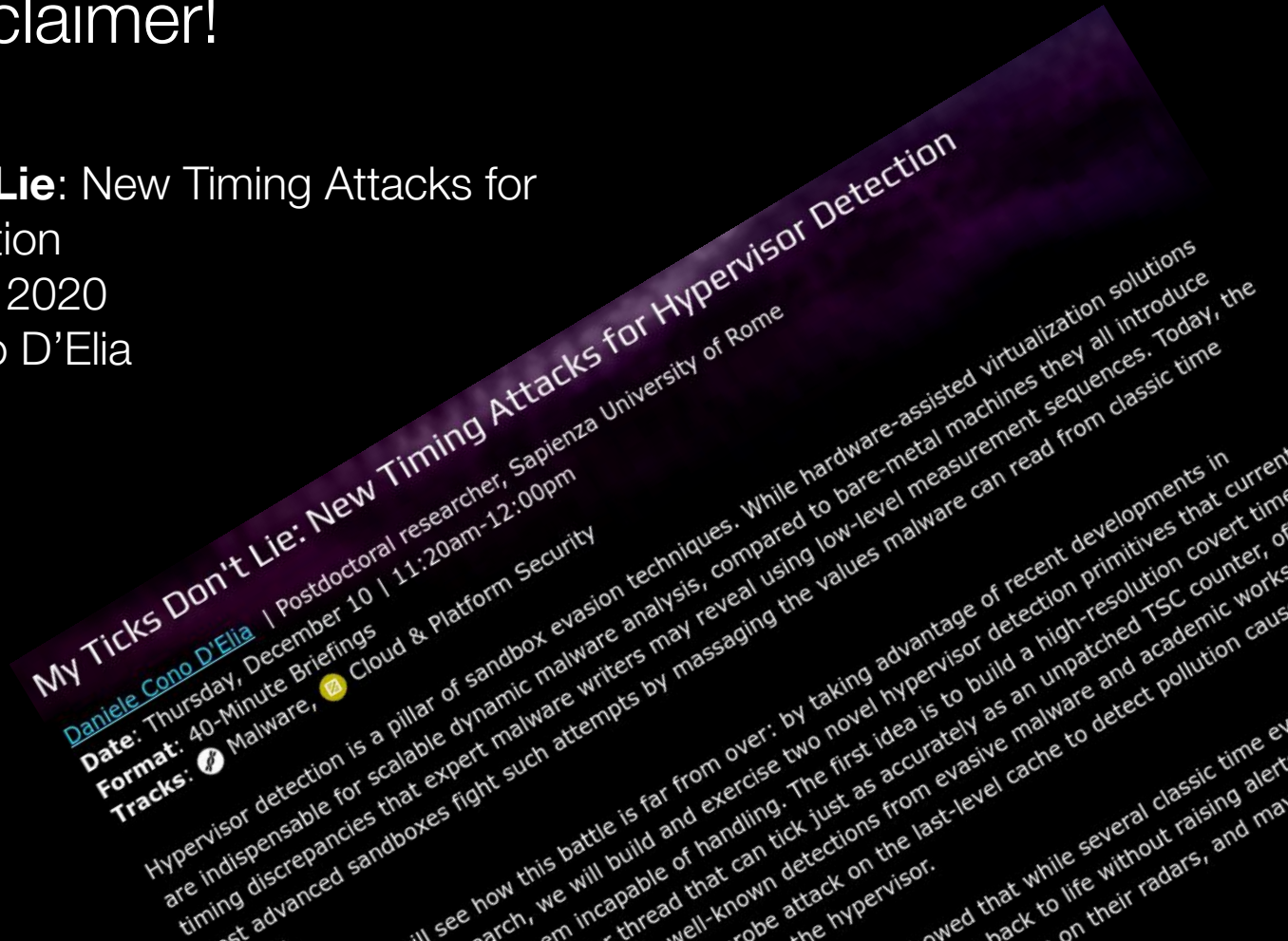
Cristian Assaiante [@cristianrichi3]

- 23 years old, student of Engineering in Computer Science.
- Reverse engineering, Binary Exploitation and Compilers.
- Capturing flags with TRX and mhackeroni (since 2018).

(stock image from DEFCON27)

!Disclaimer!

- **My Ticks Don’t Lie**: New Timing Attacks for Hypervisor Detection
  - BlackHat EU 2020
  - Daniele Cono D’Elia
  - [original talk]



Hypervisor Detection 101

Covert Time Source

Retrofitting Red Pills

LLC Prime+Probe Attack

Malware Analysis and Virtualization

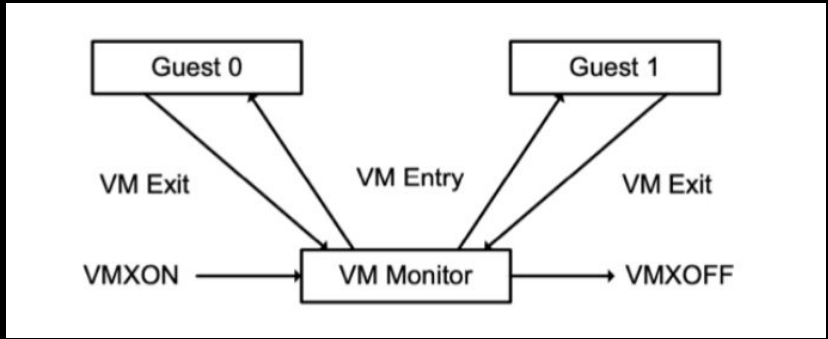
- Hypervisors cannot be avoided
  - analysts use VMs to safely perform their tasks
  - sandboxes run VMs on servers

- Hypervisor detection is a pillar of malware evasion techniques

Virtualization 101

VMX (Virtual Machine eXtension) enables CPU support for virtualization

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

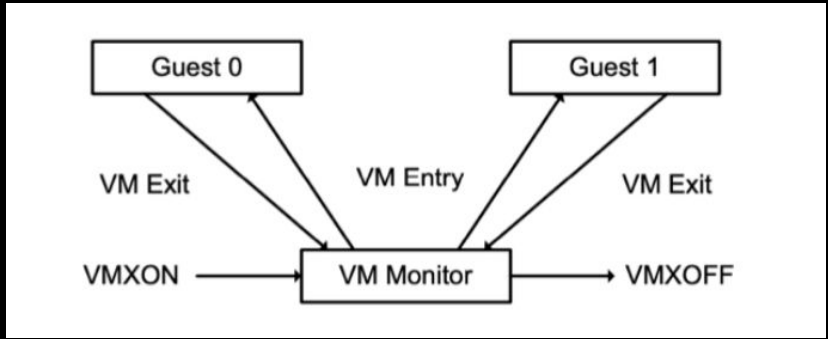


Interaction of a VMM and Guests
(from: Intel 64 and IA-32 Architectures SDM)

Virtualization 101

During the execution in guest environment, upon performing certain tasks, a transition to VMM occurs

- **VM Exit**: guest state is saved and VMM does its stuff
- **VM Entry**: guest state is restored and guest can continue its execution



Interaction of a VMM and Guests
(from: Intel 64 and IA-32 Architectures SDM)

The CPUID Case

- `cpuid` (CPU IDentification) instruction allows software to discover information about the running processor
- Input: the cpuid “leaf”, a 32 bit integer read from eax
  - for some leaves, also ecx can be set and it will be treated as input value
- Output: CPU information available within registers eax, ebx, ecx, edx

```
int eax = <leaf>, ebx, ecx, edx;
__asm__ volatile (“cpuid” :
                  “a=”(eax), “b=”(ebx), “c=”(ecx), “d=”(edx) :
                  “a”(eax));
```

The CPUID Case

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

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

31st bit of Extended Feature Information is the hypervisor bit

Timing VM Exit Events

Hypervisor detection via time comparison of `cpuid` execution.

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

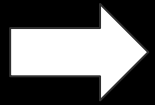
Native: ~300 cycles
VirtualBox 5.2: ~3000 cycles

Sandboxes Anti-Evasion Tricks

- Track instructions causing VM Exit
- Optimize VMM code to reduce VM Exit time overhead
- Fake time sources return values:
  - rewrite output of some APIs
  - make `rdtsc` cause a VM Exit too, then alter its returned values keeping track of time spent in VMM

How can we measure time without any available time source?

Hypervisor Detection 101



Covert Time Source

Retrofitting Red Pills

LLC Prime+Probe Attack

Building a Covert Time Source

```
volatile uint64_t ctr_clock;

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

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

Issues:

- fast enough?
- reliable?
- serialization?

Approximate Resolution:

- rdtsc: ~2793.5 Mhz
- cthread: ~540 MHz
  - w/ TurboBoost

A Clever Implementation

```
volatile uint64_t ctr_clock;

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

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

Approximate Resolution:

- rdtsc: ~2793.5 Mhz
- cthread: ~540 MHz
  - w/ TurboBoost
- cthread+: ~**3500** MHz
  - w/ TurboBoost

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

Cores Availability

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

At least two cores are needed to be able to use counter threads.
How can we check how many cores are available?

Cores Availability

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

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

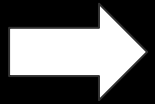
Cores Availability

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

Race two thread, check sum of two count variable < LOOP_COUNT/2 (why: count increases when counter from other thread was not updated)

Hypervisor Detection 101

Covert Time Source



Retrofitting Red Pills

LLC Prime+Probe Attack

Detection 1: cpuid latency

Time to execute `cpuid` > threshold

- Set eax = 1
- Compute average time
  - N = 10 observations
  - threshold = 1000

Detection 2: cpuid/nop ratio

Execution time ratio between `cpuid` and another instruction

- Use a low-latency instruction as reference (e.g. `nop`)
- Idea: different instructions execute similarly slower or faster under different CPUs and conditions
  - Native environment vs Guest environment

Reproduction of “Detecting hardware-assisted virtualization” [DIMVA ‘16]

Detection 3: TLB Eviction

Look for TLB entries evicted by VMM execution

- **TLB** (Translation Lookaside Buffer): keeps track of translations between Virtual and Physical addresses to avoid repeating translations
  - a kind of cache memory for MMU
- Fill TLB and cause a VM Exit. Then analyze memory access latencies

Reproduction of “Detecting hardware-assisted virtualization” [DIMVA ‘16]

Results

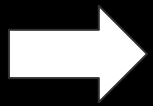
76 completed executions on online available sandboxes

- Detection 1: 43/76 detected
  - fake TSC values for 6
  - single core detected for 14
- Detection 2: 55/76 detected
  - fake TSC values for 7
  - high noise for `nop` for 12 (also with CT)
- Detection 3: 24/29 detected
  - completed only by 52 machines
  - unstable latencies for 23

Hypervisor Detection 101

Covert Time Source

Retrofitting Red Pills



LLC Prime+Probe Attack

New Detection

Look for effects on caches quite reliable to measure

Idea:

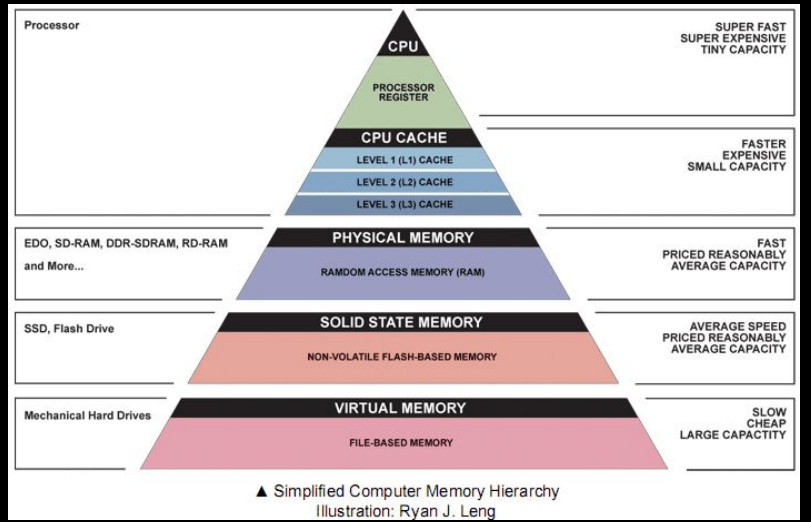
- VMM execution may evict LLC lines
- Search for those lines to detect VMM

Why LLC?

- High resolution
- Shared between cores

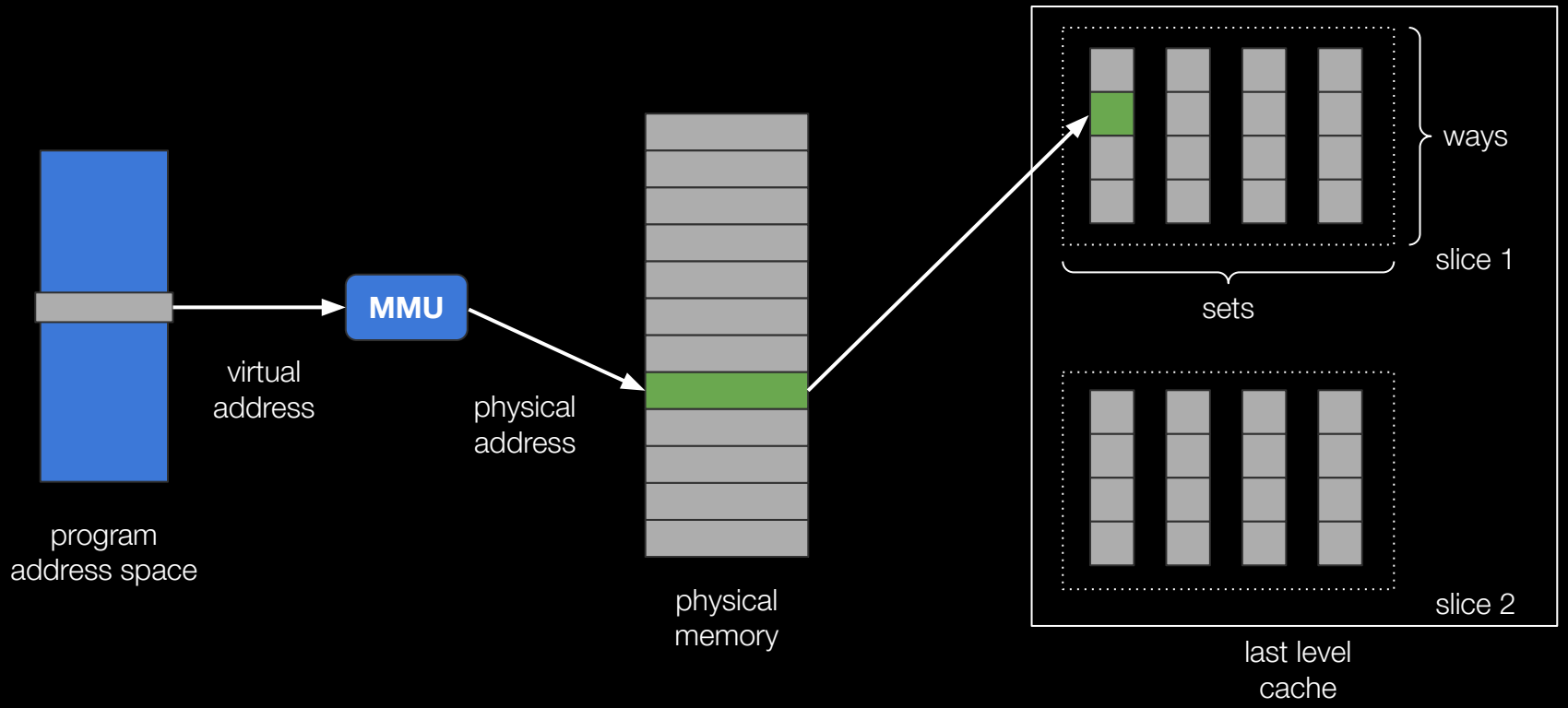
Memory Hierarchy

- Huge difference between CPU and Physical Memory
- To speed up the memory accesses, cache memories are in between
  - L1d/L1i per-core
  - L2 per-core
  - L3 shared
    - a.k.a. **LLC** (last level cache)



▲ Simplified Computer Memory Hierarchy
Illustration: Ryan J. Leng

LLC Functioning



Prime+Probe Attack

i-th cache set
(16-way associative)

fill each cache set entry (how?)

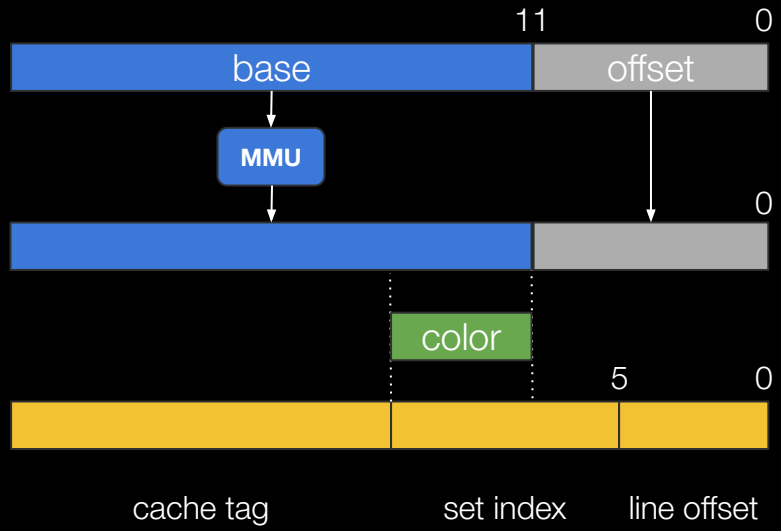
VMM may evict one on more lines

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

Eviction Sets

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

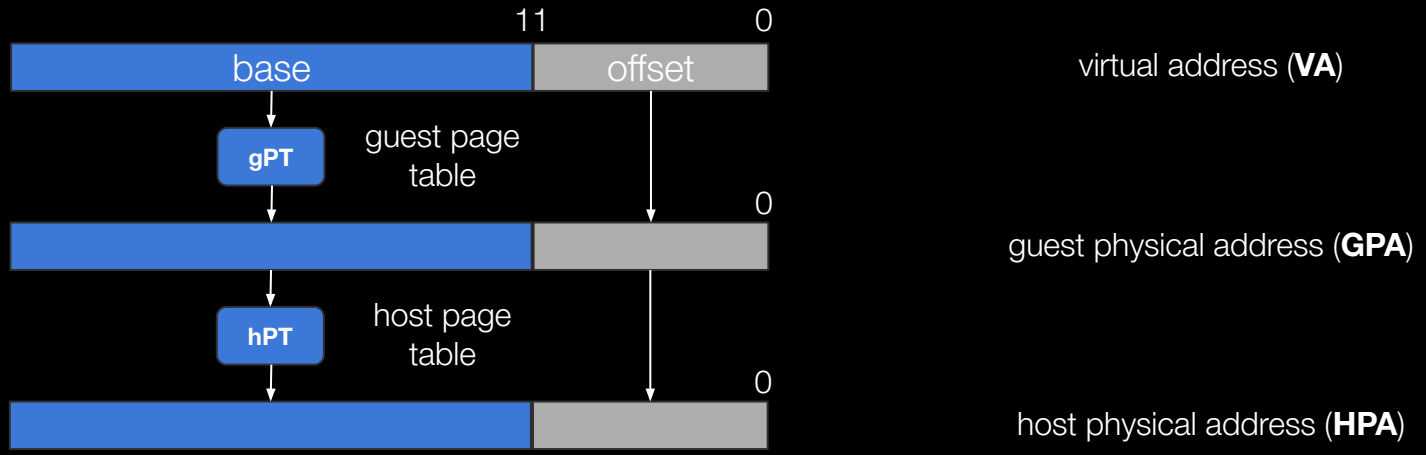
LLC Addressing



virtual address

physical address

Addressing inside the VM

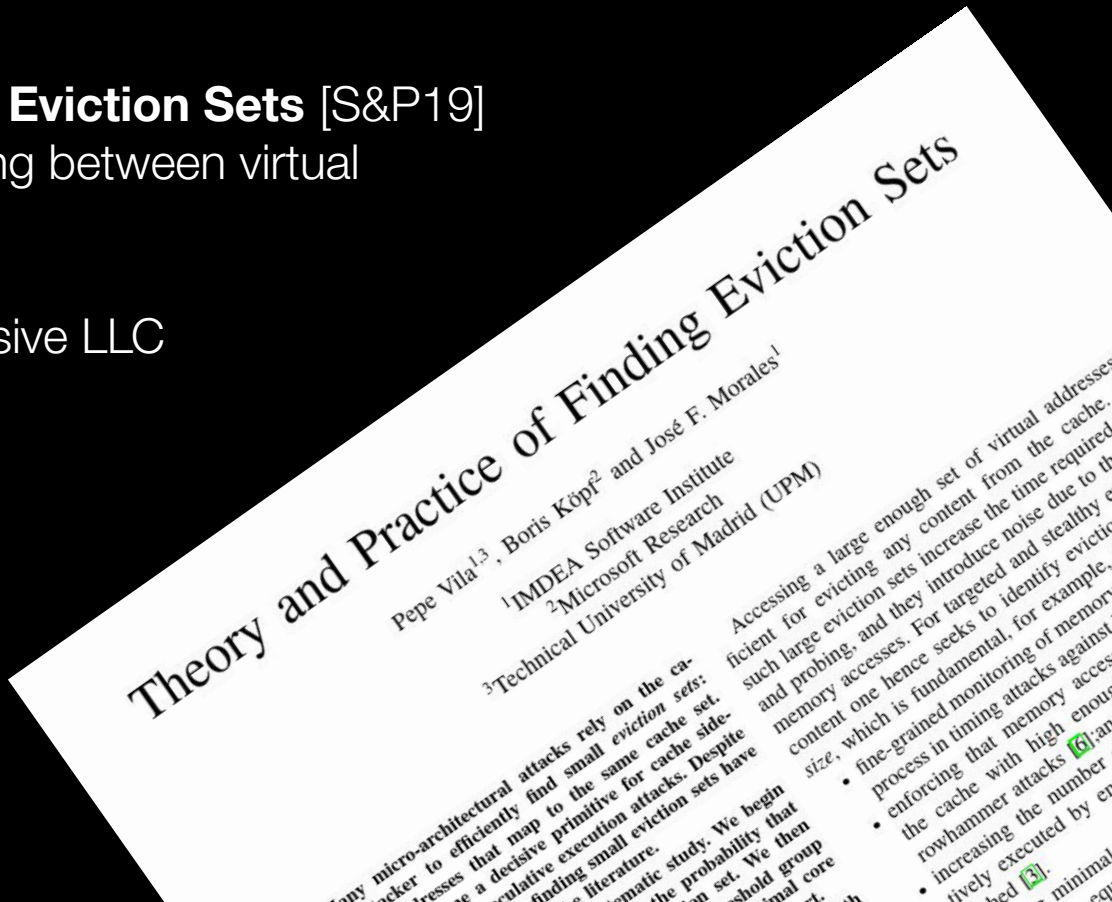


In a sandbox we have no knowledge of the mapping VA-HPA

Finding Minimal Eviction Sets

Theory and Practice of Finding Eviction Sets [S&P19]

- no assumption on the mapping between virtual addresses and cache sets
- $O(n*w^2)$ makes it practical
- does not work with non-inclusive LLC



Theory and Practice of Finding Eviction Sets

Pepe Vila[1,3], Boris Köpf[2] and José F. Morales[1]

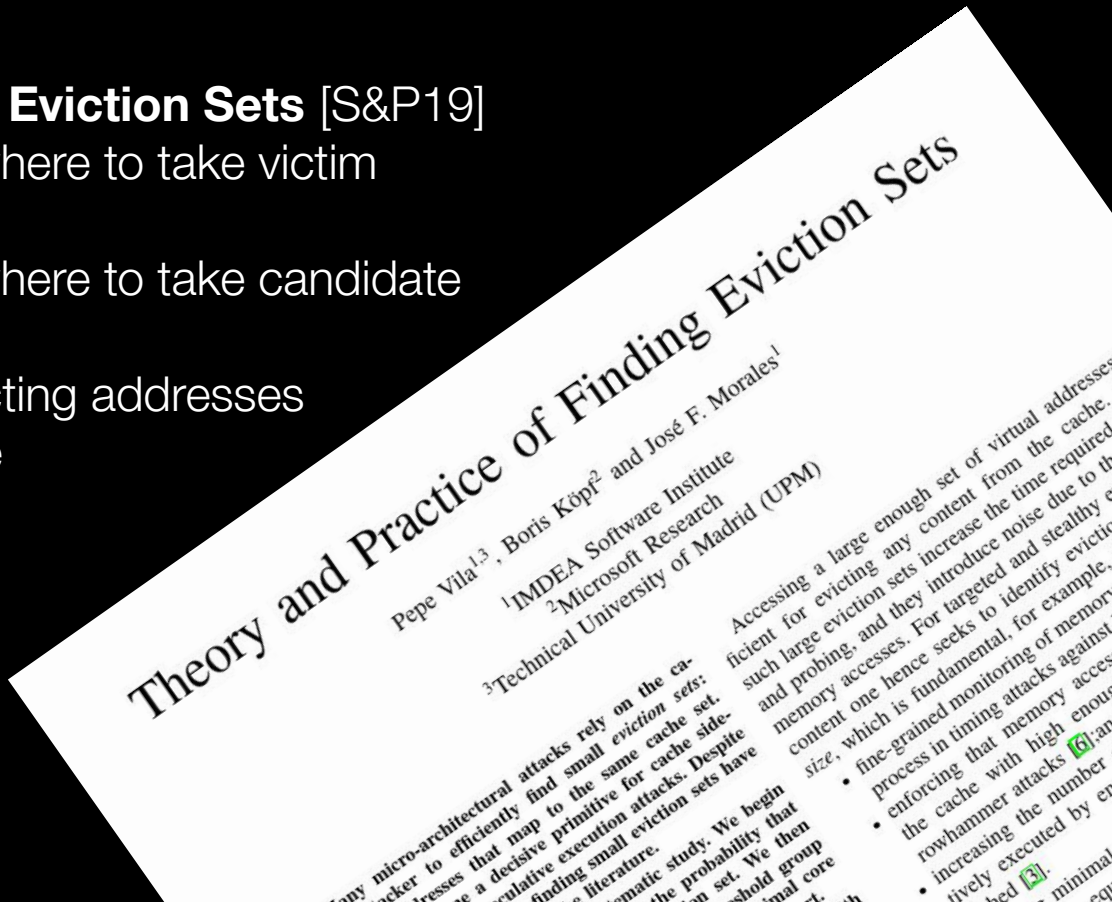
[1]IMDEA Software Institute
[2]Microsoft Research
[3]Technical University of Madrid (UPM)

Accessing a large enough set of virtual addresses
ficient for evicting any content from the cache,
such large eviction sets increase the time required
and probing, and they introduce noise due to th
memory accesses. For targeted and stealthy e
content one hence seeks to identify evictio
which is fundamental, for example,
fine-grained monitoring of memor
process in timing attacks against
enforcing that memory acces
the cache with high enou
rowhammer attacks [6];an
increasing the number
tively executed by e
hed [3].
minimal

Finding Minimal Eviction Sets

Theory and Practice of Finding Eviction Sets [S&P19]

- map a memory region from where to take victim addresses
- map a memory region from where to take candidate addresses for the eviction set
- get a large random set of evicting addresses
- reduce the set to minimal size



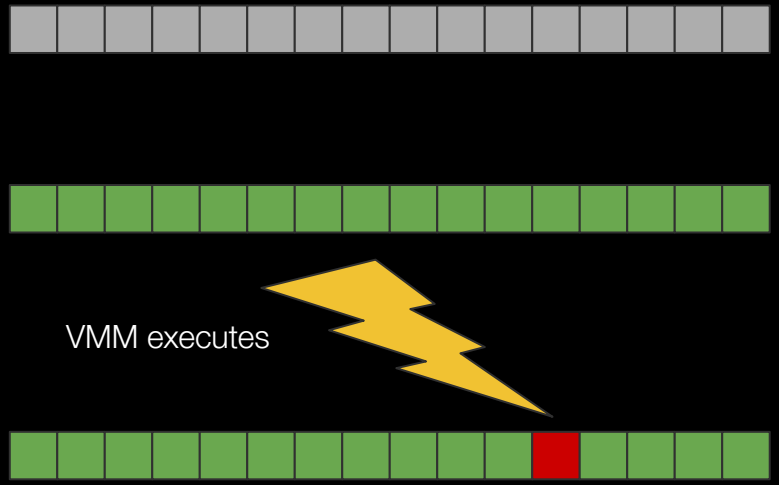
Theory and Practice of Finding Eviction Sets

Pepe Vila[1,3], Boris Köpf[2] and José F. Morales[1]

[1]IMDEA Software Institute
[2]Microsoft Research
[3]Technical University of Madrid (UPM)

Accessing a large enough set of virtual addresses
ficient for evicting any content from the cache,
such large eviction sets increase the time required
and probing, and they introduce noise due to th
memory accesses. For targeted and stealthy e
content one hence seeks to identify evictio
size, which is fundamental, for example,
• fine-grained monitoring of memor
process in timing attacks against
• enforcing that memory acces
the cache with high enou
rowhammer attacks [6];an
• increasing the number
tively executed by e
hed [3].

LLC Prime+Probe for VM Detection



pick eviction set(s)

preload stage + prime

`cpuid`

compute max access time among lines in the set

Results

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

- VirtualBox 6.1, Windows host: 20/128
- VirtualBox 6.1, Linux host: 18/128
- VMware Workstation Pro, Windows host: 10/90
  - Eviction sets not found for every color
- QEMU-KVM 4.2.50: 13/128

Limitations

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

Questions?