

I. Sennovskiy

i.sennovskiy@bi.zone

A new post-meltdown attack technique: how to use speculative instructions for virtualization detection

1. Intro

The attack shown in this paper is based on the cache side-channel attack used in Meltdown. The Meltdown attack uses speculative execution for accessing contents of memory which an unprivileged attacker would not be able to view. This attack differs from Meltdown as it doesn't use a cache timing threshold. This is possible because CPU executes certain instructions preemptively to speed up the computing process. In speculative execution, Meltdown reads buffers under attacker's control, that allows an attacker to use memory access timing measurement as a side channel.

However, speculative execution can also be used to disclose certain CPU settings, that an attacker should not know.

2. Virtualization

For certain instructions, for example for **rdtsc**, VT-x technology in Intel CPUs allows hypervisor to configure if a VMEXIT (that means a context switch) happens. By default, most virtualized environments are configured to create a VMEXIT on **rdtsc**, including Virtualbox, VMware, Hyper-V, Parallels on Apple or Parallels hypervisor. VMEXIT itself means a context switch, therefore instructions triggered a VMEXIT are executed longer, than in non-virtualized environments.

3. The attack

First, an attacker creates a memory buffer that spans several pages. Then, **rdtsc** is executed speculatively instead of speculative access to the privileged memory, and the result is used to access a certain part of the previously created buffer. The number of buffer's pages which can be accessed during speculative execution is limited, which allows later to discern actual speculative accesses from random mistakes. As soon as the function containing speculative execution is completed, the page number with the lowest access time is added to statistics. Then all cache in the memory region is

flushed. Below are the functions used to trigger speculative execution and memory accesses in 32-bit Windows OSs:

```

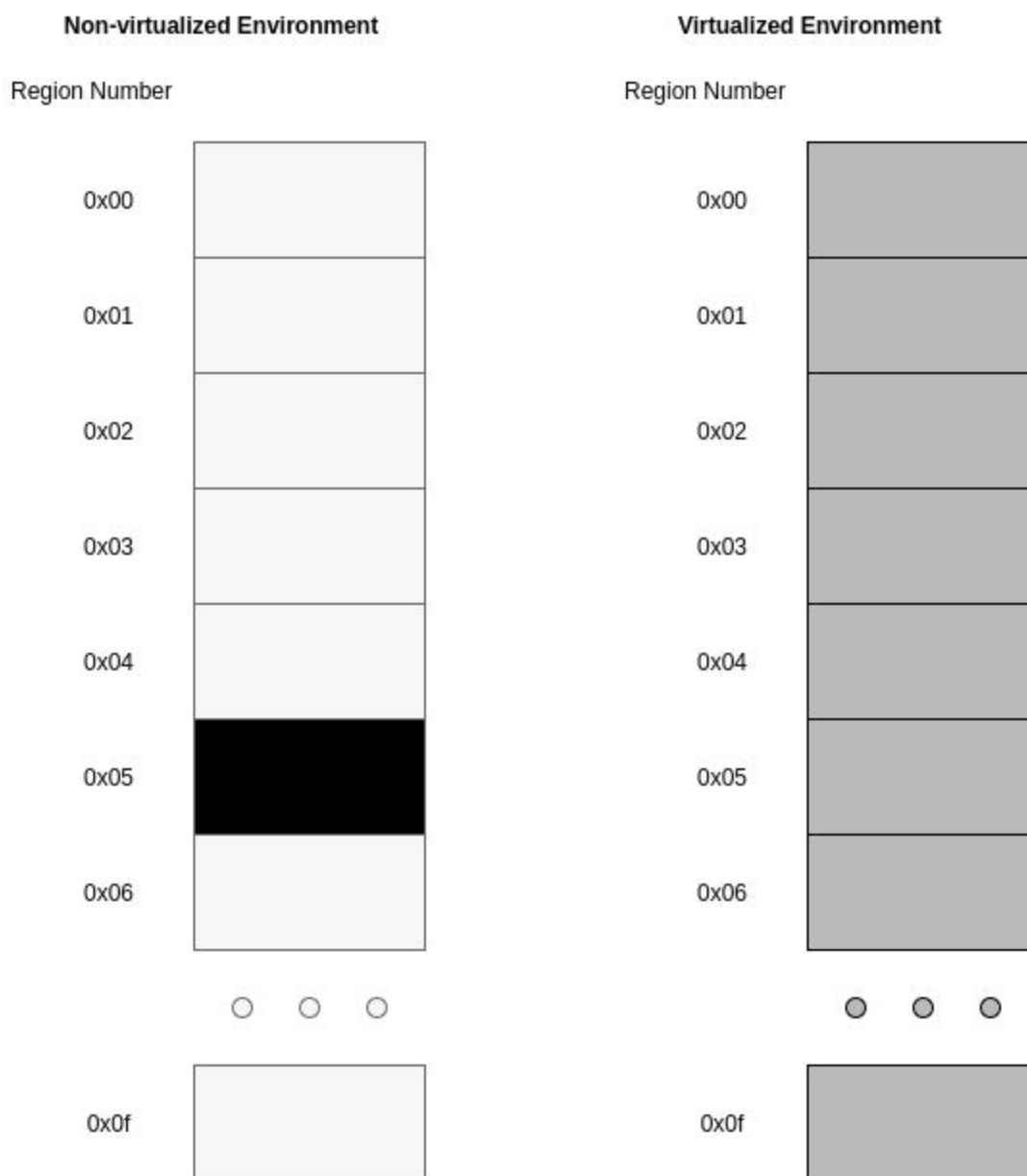
_declspec(naked) void herring() { //This function is used to trigger speculative
    __asm { //execution in the speculate function
        xorps xmm0, xmm0
        sqrtpd xmm0, xmm0
        sqrtpd xmm0, xmm0
        sqrtpd xmm0, xmm0
        sqrtpd xmm0, xmm0
        sqrtpd xmm0, xmm0
        sqrtpd xmm0, xmm0
        sqrtpd xmm0, xmm0
        sqrtpd xmm0, xmm0
        movd eax, xmm0
        lea esp, [esp+eax+4]
        ret
    }
}

_declspec(naked) void __fastcall speculate(const char* detector) {
    __asm { //This function speculatively executes rdtsc and accesses
        Mfence //memory based on its return value
        mov esi, ecx
        call herring
        rdtsc //These instructions are executed
        and eax, 7 //speculatively
        or eax, 32 //*
        shl eax, 12 //*
        movzx eax, byte ptr [esi+eax] //*
    }
}

```

To make the attack successful, an attacker should repeat the above described activities until he (or she) finds the distribution of cached pages, and as many times as necessary to collect reliable statistics (10 000 cycles were used in this test case). Then the number of misses of the expected region are computed. In virtualized environments with VMEXIT on **rdtsc** enabled, the percentage of time non-designated areas are hit spans from 50% to 99%. On non-virtualized systems it is less than one percent, as

shown on the picture below (the darker the region's color the more often it's been hit). In this test case, I use Mac OS, Ubuntu, Debian and Windows non-virtualized hosts and Ubuntu, Debian and Windows guest environments. The same attack is possible with *rdmsr* instruction.



Picture 1 - Cached pages' distribution in different environments

4. Explanation of the attack

The attack uses speculative execution to trick the CPU into disclosing information about how **rdtsc** is executed. In a non-virtualized environment **rdtsc** would be executed on the CPU itself, and CPU would just return the counter. In a virtualized environment, where the hypervisor set the “RDTSC exiting” bit in the IA32_VMX_PINBASED_CTLX MSR, in fact, executing **rdtsc** is a context switch, which would take too much time.

As I detected this vulnerability, I tried to find its reasons, but there is no sufficient information in available Intel CPUs documentation to be completely sure. I assume that the CPU either decides that **rdtsc** would take too much time to execute in a VM and doesn't execute it unless the flow reaches it directly, or the CPU doesn't speculatively execute instructions which trigger a VMEXIT. In a virtualized environment **rdtsc** and directly following instructions are not executed speculatively, unlike non-virtualized ones.

5. Conclusions and further research

This attack uses the new Meltdown caching technique to create a side channel, but instead of accessing privileged memory regions it discloses current CPU operation mode. There are several well-known methods for detecting virtualization, however these methods mostly rely on using **rdtsc** for timing, and are not applicable for a smart hypervisor that could possibly fake the values. This attack can be mitigated, but an exploit with slight changes works correctly. Possibly I will provide some PoCs to bypass smart hypervisors later.

This creates an interesting observation: if the **rdtsc** creates a VMEXIT, the described attack can show the presence of virtualization, while many previous attacks, for example TLB cache profiling, can be mitigated, and if there is no VMEXIT on **rdtsc**, previous attacks are useful.

This attack allows quick and simple detection of virtualized environments configured with default settings, or environments that use **rdtsc** exiting knowingly to defend themselves from over virtualization checks. It was tested on an example of a virtualized sandbox and detected the sandbox easily without raising any suspicions.

The PoC code is located at <https://github.com/bi-zone/rdtsc-checkvirt>.