I. Sennovskiy

isennovskiy@gmail.com

A new post-meltdown technique. Using speculative instructions for virtualization detection.

1. Intro

The attack shown in this paper is based on cache side channel used in Meltdown. Meltdown attack uses speculative execution for accessing contents of memory which an unprivileged attacker would not be able to view. This is possible because CPU executes certain instructions preemptively to speed up execution. Meltdown leverages accesses to attacker-controlled buffers in speculative execution, changing the state of CPU cache in such a way, that the attacker can use memory access timing as a side channel.

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

2. Virtualization

VT-x technology in Intel CPUs allows hypervisor to choose for certain instructions, whether a VMEXIT (essentially a context switch) will happen, for example ***rdtsc***. Most virtualized environments in standard configuration are set to create a VMEXIT on ***rdtsc***, including but not limited to Virtualbox, VMware, Parallels running on Apple hypervisor and Parallels hypervisor. VMEXIT in itself is a change of context of execution, which means that instructions that trigger a VMEXIT execute longer, than if they were executed in a non-virtualized environments.

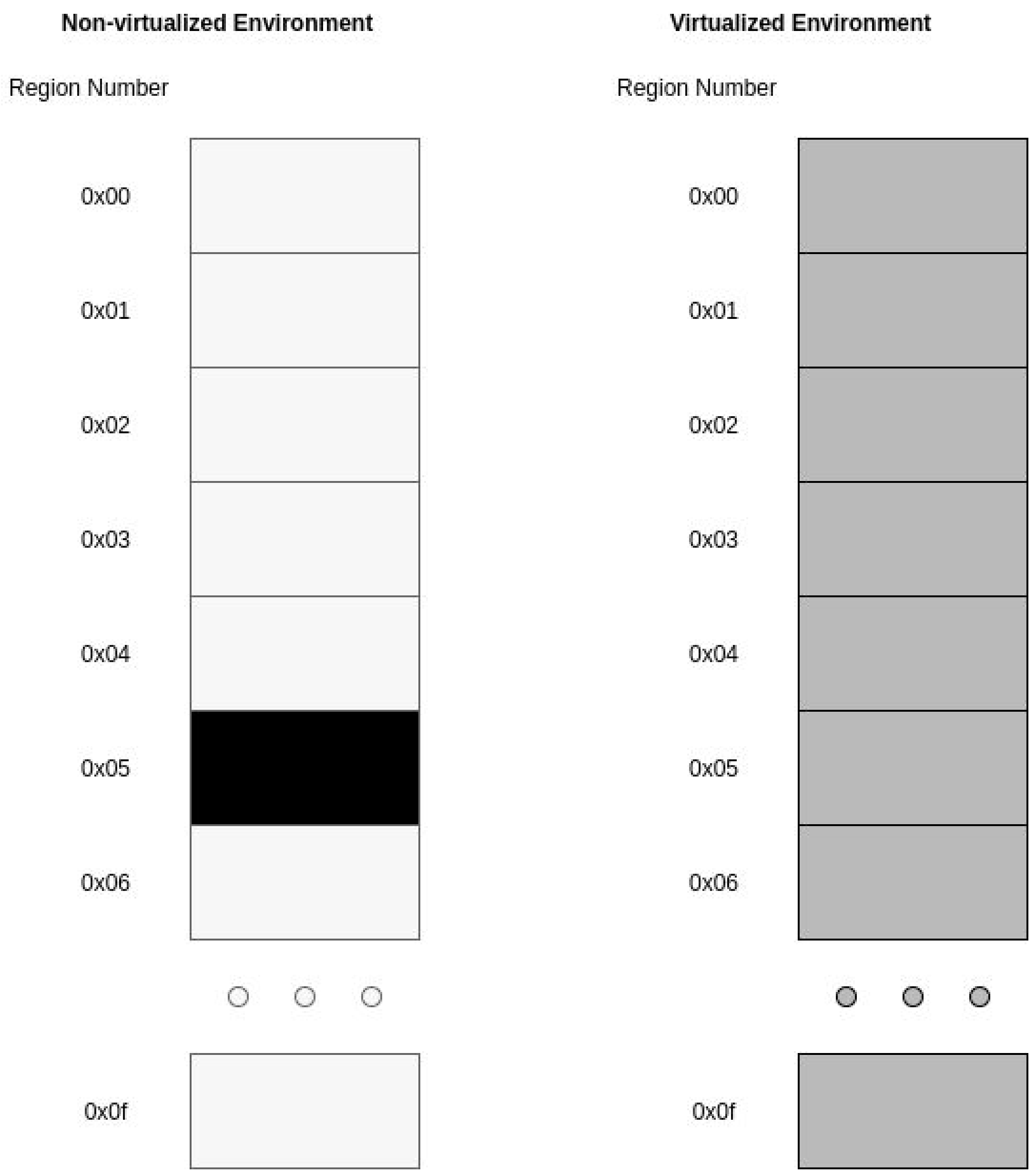
3. The attack

A buffer that spans several pages is created. The attack differs from Meltdown, it doesn’t use a cache timing threshold. Instead of speculatively accessing regions of memory to get data, ***rdtsc*** is executed speculatively, then the result of ***rdtsc*** execution, which is used for accessing a certain part of the buffer. The number of pages from the buffer which can be accessed during speculative execution is limited, which allows later to discern actual speculative accesses from random mistakes. After the function containing speculative execution finishes the number of the page with the lowest access time is added to statistics. Then all cache in the memory region is flushed. The functions used to trigger speculative execution and memory in 32-bit windows system:

```
_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]       //*
    }
}
```

For a successful attack this is done many times (10000 cycles were used in this test case).The number of instances of fastest page accesses to wrong pages is computed. In virtualized environments where VMEXIT on ***rdtsc*** was 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. This is represented on picture 1 (the darkness of the region represents how many times it’s been hit). Mac OS, Ubuntu, Debian and Windows non-virtualized hosts were used for testing. Ubuntu, Debian and Windows were used as guest environments.



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 the way ***rdtsc*** is executed. In a non-virtualized environment ***rdtsc*** would be executed on the CPU itself. CPU would just return the counter to the user. In a virtualized environment, where the hypervisor set the “RDTSC exiting” bit in the IA32_VMX_PINBASED_CTLS MSR, executing ***rdtsc*** is essentially a context switch, which would take too long. CPU decides that ***rdtsc*** would take too long to execute in a VM and doesn’t execute it unless the flow reaches it directly. In a virtualized environment ***rdtsc*** and instructions directly following it are not executed speculatively, and in non-virtualized they are.

5. **Conclusions and future work**

This attack uses the new Meltdown caching technique to create a side channel, but instead of accessing privileged memory regions it discloses information about CPU operation. There already existed several methods for detecting virtualization, however they mostly relied heavily on using ***rdtsc*** for timing, leaving them vulnerable to a smart hypervisor, that could possibly fake the values. While this attack in its simplest form is vulnerable to that too, it can be changed to operate on limited memory regions during one cycle. If the memory regions are deterministically changed between cycles in a certain manner, it becomes infeasible for the hypervisor to try faking the memory accesses as if they were cached, because it doesn’t know which ones should have been cached. A POC will be created that uses this property.

This creates an interesting observation: if the ***rdtsc*** creates a VMEXIT, the attack introduced in this article can show the presence of virtualization, while many previous attacks 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 standard 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.