Semantic gap

by Honorary_BoT

The other day I was walking. Walking page tables of course. On Windows. On Intel x64 CPU.

Every time I try to exploit something, I avoid using known gadgets or techniques. Instead, I prefer getting to know the execution environment, like what is there in the address space, which properties it has and so on. I am also lazy, so I look for the easiest way possible.

I needed an RWX memory in the kernel. I was aware that Microsoft does a really good job with mitigations and was not expecting any. But anyway, I decided to scan Windows page tables.

I was not using any specifics of the Windows memory manager, I skipped Software PTEs. My idea was doing it in a hardware way: if the page has a P bit set in PTE, then the mapping is there, no matter what semantics Windows puts on that memory.

I used PulseDbg, a hypervisor-based debugger for that. This way ensures the OS to be frozen and not modifying the page tables on the fly. For the scanning process itself refer to the Offzone 2019 presentation "(Mis)configuring page tables"[1] or, even better, to the Intel Software Developers Manual (Vol 3, Chapter 4), it has all the details. In fact, SDM has everything, so always refer it. I also would suggest for you to read it before you go to bed.

Surprise! Windows kernel does have RWX regions of memory. In my case it was Windows 10 1809. And an Intel Haswell CPU on a Gigabyte Q87 chipset motherboard, let me explain why it matters.

The first thing I identified was an area with UEFI Runtime services being mapped as RWX. It is because the firmware typically doesn't set the protection on loaded modules. And Windows is not aware of the semantics of the firmware loader. The only option for the OS is to rely on the firmware for those services to work.

HAL keeps UEFI Runtime function pointer table at `hal!HalEfiRuntimeServicesBlock`. Those functions can be triggered from user mode, for instance by launching "System information", which would trigger reading a UEFI variable.

The good news is if you use Microsoft Surface devices, you're fine, since MSFT firmware assigns protection to UEFI modules. Good job, Microsoft!

Besides that, some drivers create custom allocations as RWX, which is inevitable, I guess. But not for `MmMapIoSpace` function, which has an interesting behavior. Check out the prototypes:

```
PVOID MmMapIoSpace(PHYSICAL_ADDRESS
PhysicalAddress, SIZE_T NumberOfBytes,
MEMORY_CACHING_TYPE CacheType);

PVOID MmMapIoSpaceEx(PHYSICAL_ADDRESS
PhysicalAddress, SIZE_T NumberOfBytes,
ULONG Protect);
```

The first one is a legacy one, the "Ex" one is only available on Windows 10. Third party drivers would use the old one. There is an implicit mapping between specified caching type and protection:

- MmNonCached converted to RWX
- MmCached converted to RWX
- MmWriteCombined converted to RW

So, the driver must decide if it wants backward compatibility, or a fine-grained protection of the mapping.

The good news again is there is a Virtualization-Based Security. Virtual secure mode uses hardware virtualization features for security and protection of Windows 10. It has a W^X enforcement in EPT – extended page tables, controlled by the hypervisor. It does not allow guest kernel memory to be both writable and executable at the same time. If you're concerned about your Windows security, you should definitely turn VBS on.

There are more RWX regions present in the kernel. If you're interested, then take a walk. On page tables, of course.

1. https://offzone.moscow/report/mis-configuring-page-tables/