VBS and VSM Internals

Saar Amar

whoami

- Saar Amar
- Security Researcher
    - vulnerabilities && exploitation
    - lowlevel internals
    - @AmarSaar
- Addicted to CTFs!
    - @pastenctf team member

Talk outline

- Windows before VBS
- VSM 101
  - VTLs, OS architecture, new components
- Attack surface
- Few bugs I found and reported ☺

Motivation

- Windows kernel exposes a HUGE attack surface
  - Networking, filesystems, fonts, scrollbars, etc.
  - However, the kernel is privileged => can basically do whatever it wants
- Can’t just change the entire OS architecture and design in one day…

Motivation

- *Before VBS* – kernel is the root of trust
  - Hyper-V fully trusts the root partition
- *After VBS* – we have a generic architecture to implement independent secure trusted components
  - Without a single binary for everything
  - If you own one component – you still have work to do!
- Outcome: we made ring0 more **restricted**, and **mitigated kernel exploits!** (at least some of them ;))

The world - after VBS

- Kernel code signing – ntos can’t have any +X unsigned pages
- Device Guard – strict execution control
  - PowerShell scripts
  - User/kernel mode binaries
  - Everything!
- Credential Guard – protects cryptographic secrets (LSA)
  - From anyone on the machine – even kernelspace code!
- Extendible architecture: many more, and more to come

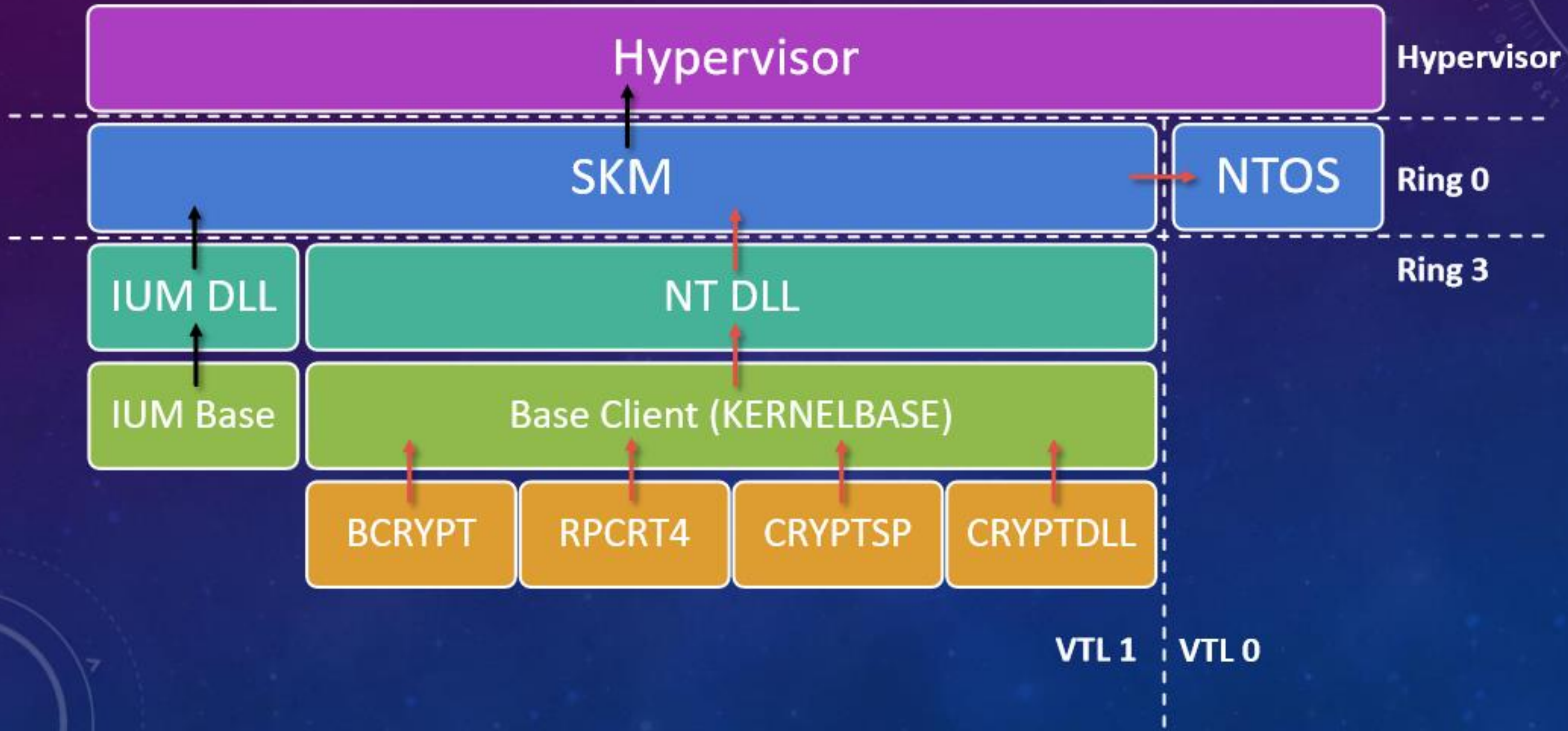
- Today we will focus on kernel exploit mitigations

VBS / VSM 101

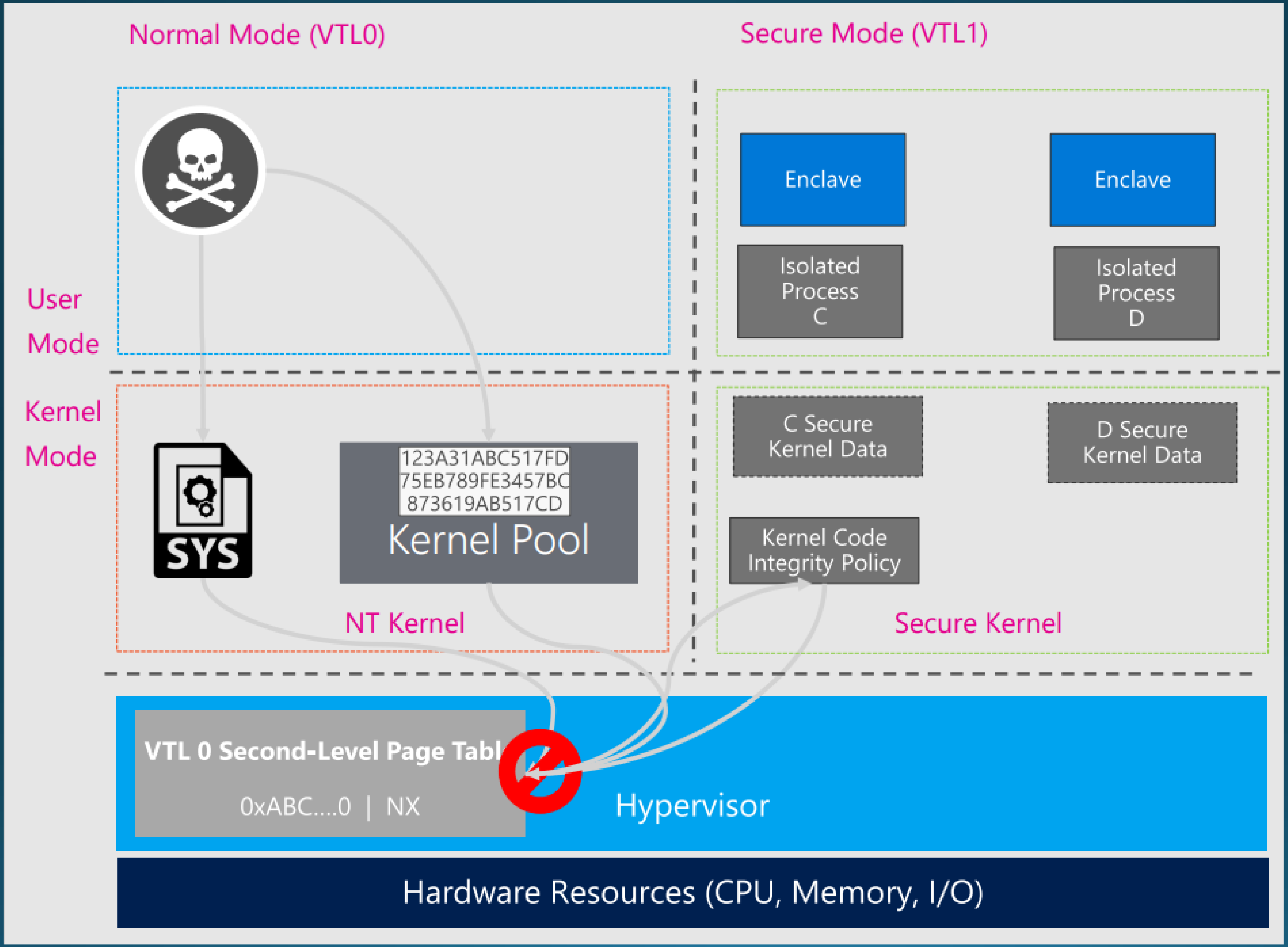
VSM 101

- Now, we have VTLs, which are orthogonal to rings
  - VTL1 is secure world, VTL0 is normal world => the higher the VTL is, the more privileged it gets
  - No more "ring -1/ring -2" confusion: more VTLs may be added in the future
  - Lets us write trusted code in **userspace** – runs in ring3vtl1
- All managed by Hyper-V
  - securekernel runs in ring0vtl1
  - ntos runs in ring0vtl0
- Hyper-V exposes 2 hypercalls for **normal call** and **secure call**
  - Normal call – services provided by NTOS to SK
  - Secure call – services provided by SK to NTOS
    - Securekernel!IumInvokeSecureService

ARCHITECTURAL LAYER OVERVIEW

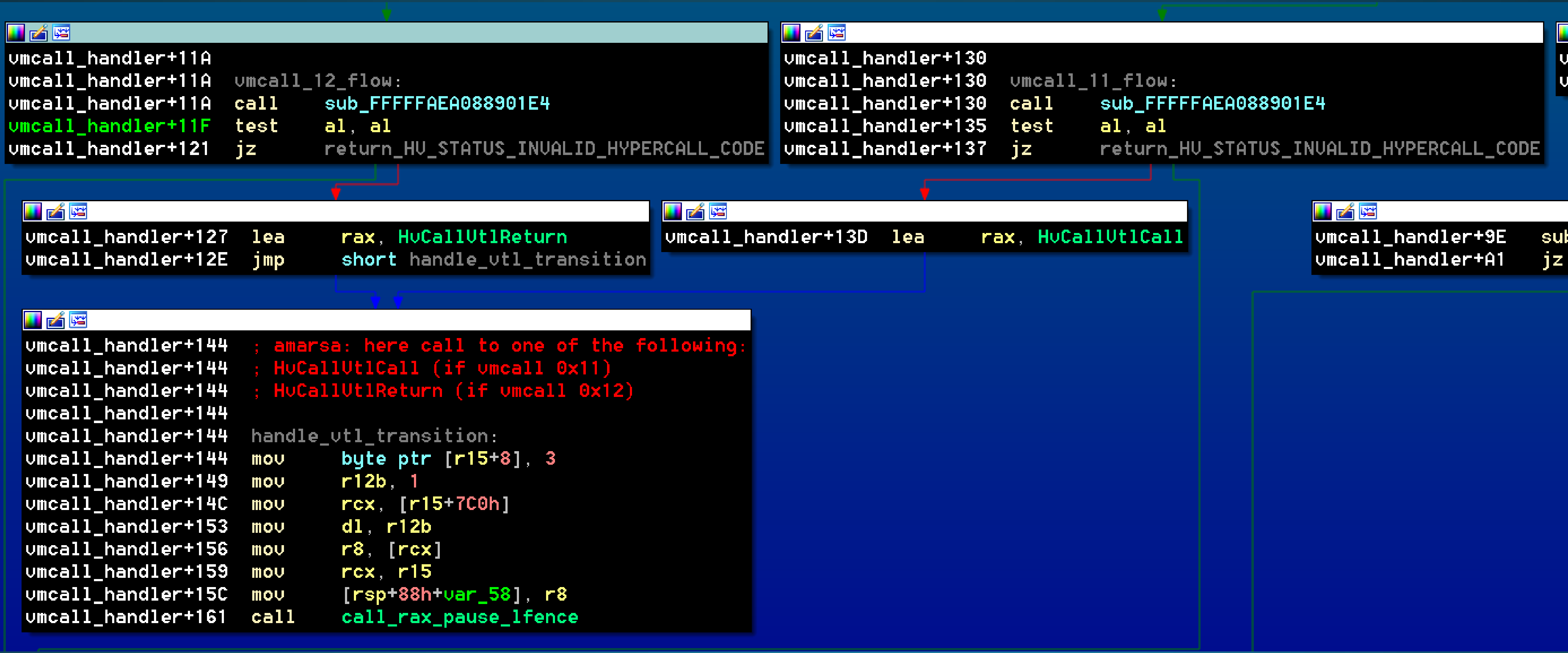


Source: Alex Ionescu, Blackhat 15



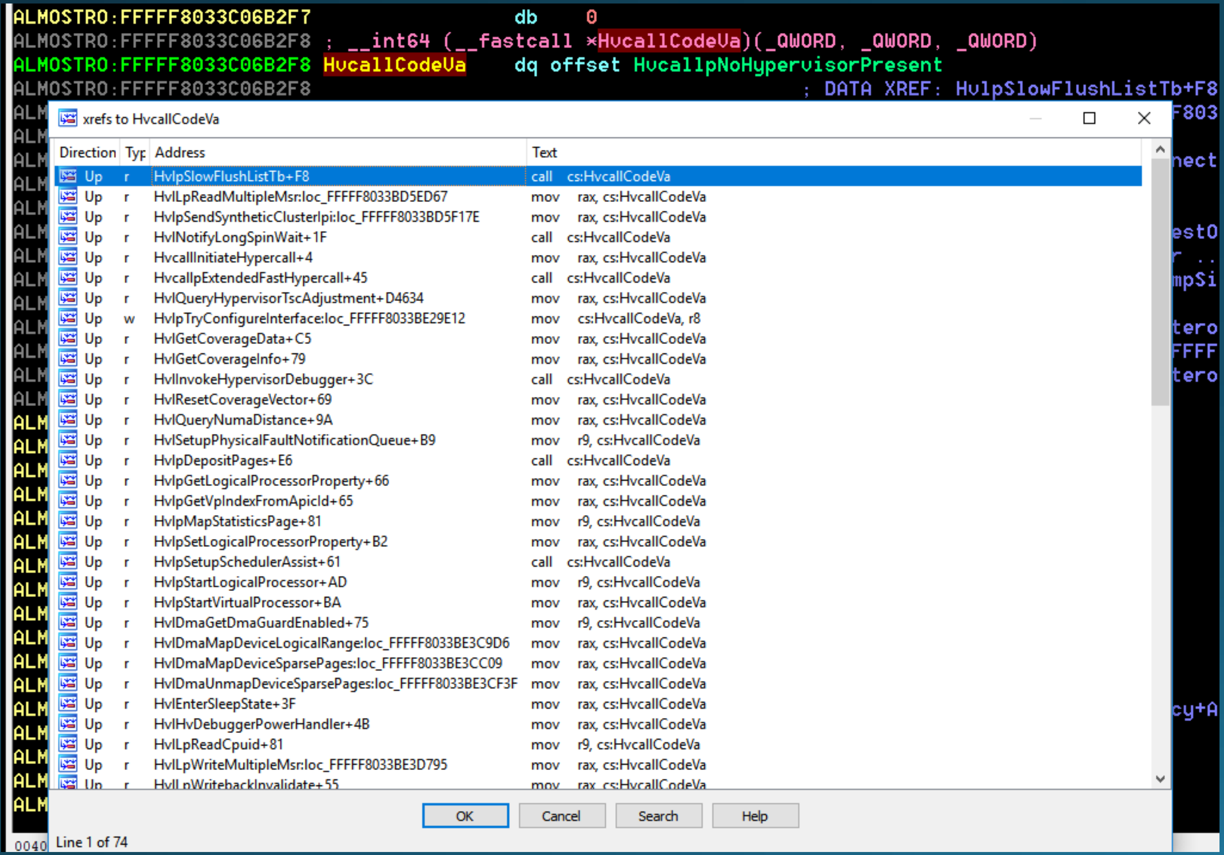
Source: Dave Weston && Alex Ionescu, OPCDE 2018

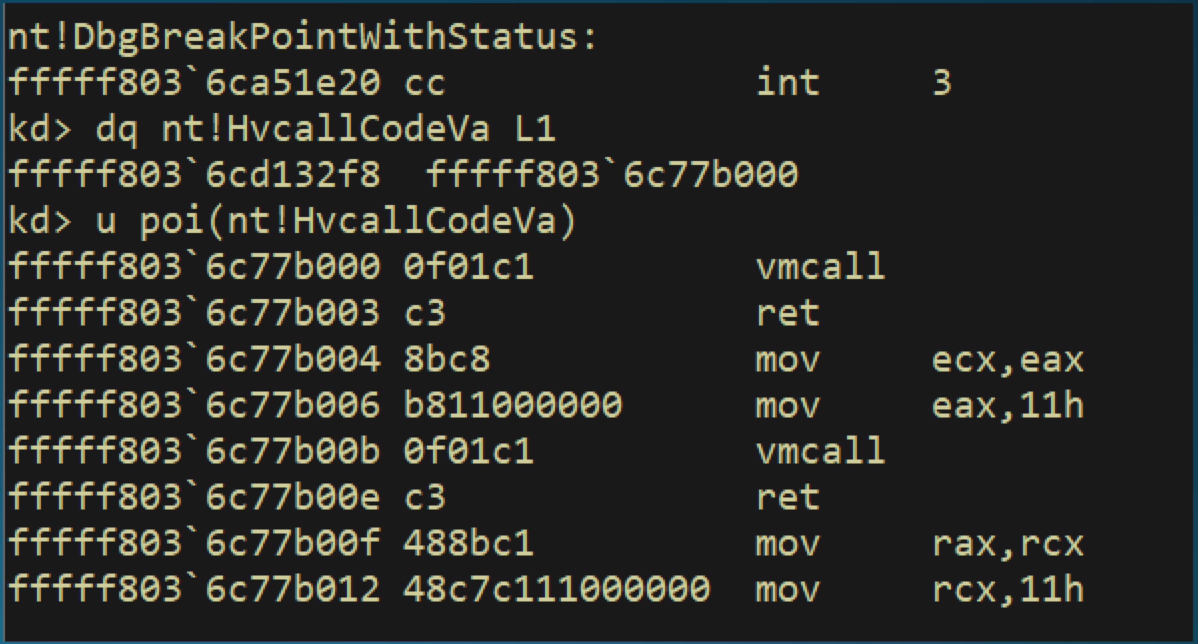
hv!vmcall_handler



In ntos

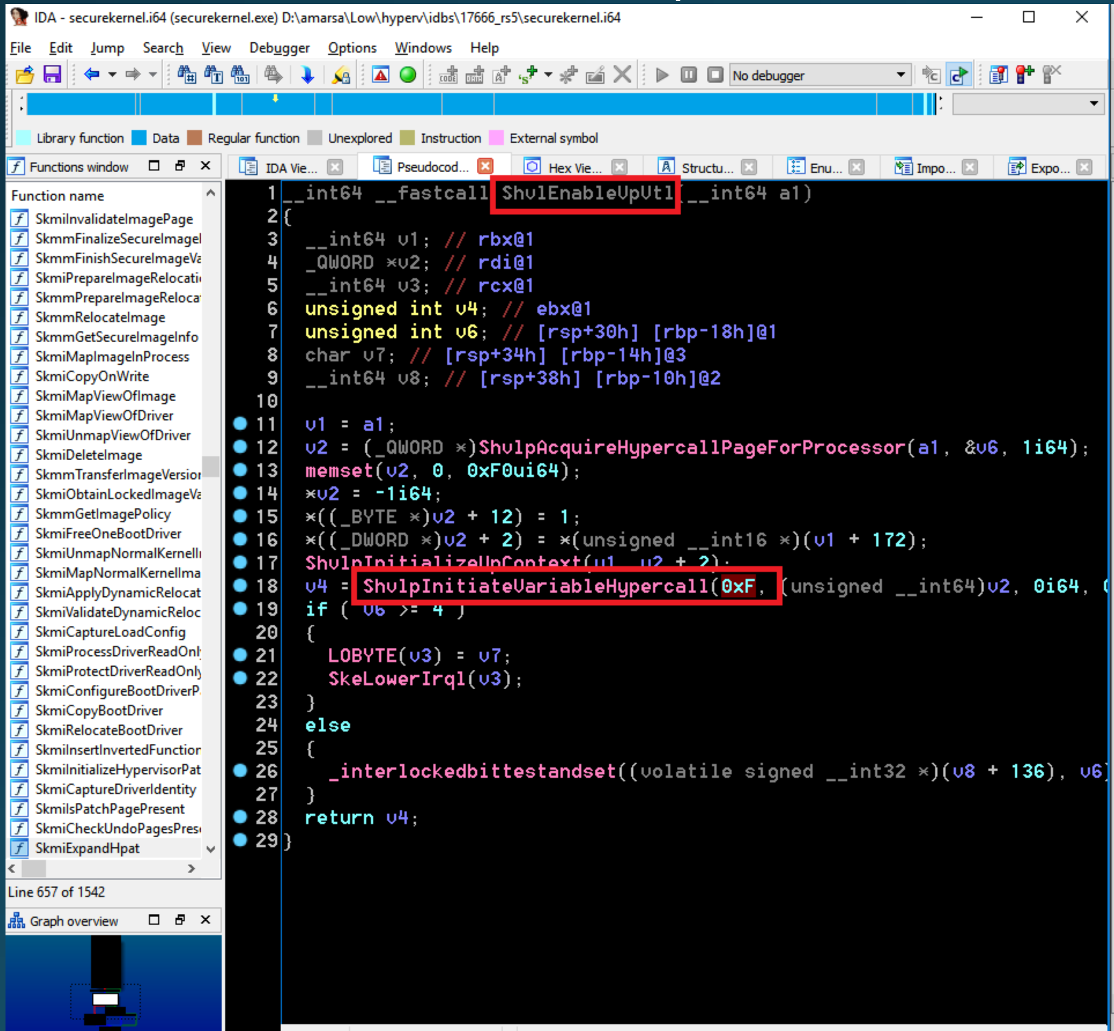
- Symbol – nt!HvcallCodeVa
- Calling to hypercalls wrapped by convention nt!HvI*

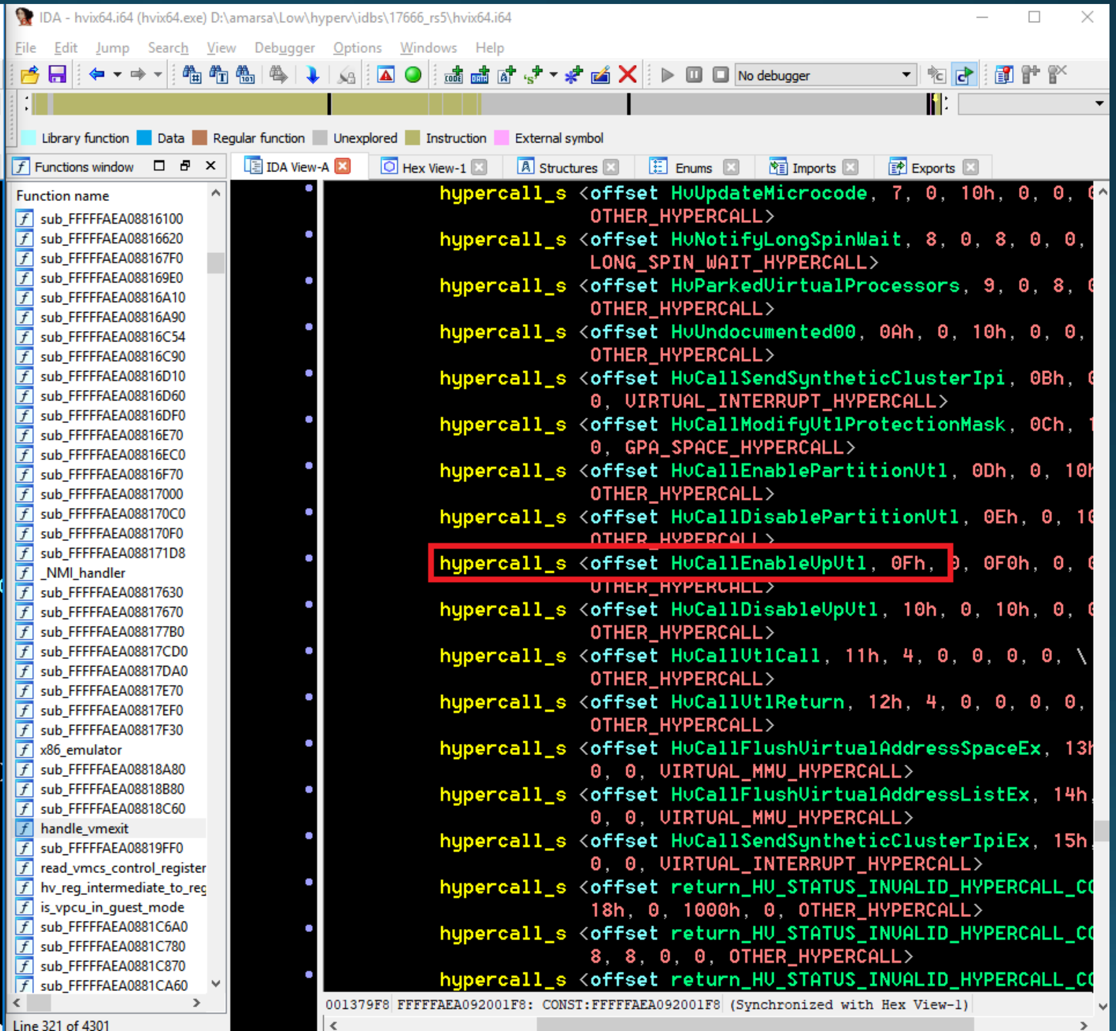




In securekernel

- Symbol – securekernel!HvcallCodeVa
- Prefixes: shvl*, skp*, …





MMU Security

- MMU configuration is restricted to VTL1 only
- The processor uses the SLAT (EPT) instead of PTEs
- ntos manages PTEs, Hyper-V and sk are the only ones with write access to EPT
- Result: with VBS enabled, virtual to physical translation is not managed by ntos!
  - MMU uses SLAT instead of PTEs

Implementing Mitigations

- So we split the kernel into separate spaces
- And we have IPC between VTLs
- Now it's trivial to implement the mitigations we talked about!

- VTL1, Hyper-V and CPU can have these "contracts":
  - Nobody can read this physical page - *Credential Guard*, LsaIso.exe
  - Block +RX/+RWX – *Device Guard*
  - Don't execute code from this physical page (even when PTE has –NX, *kills kernel exploits that generate code*)

Execution Security

- New: Mode Based Execution Control (MBEC)
  - Even if the processor doesn't support it, Hyper-V emulates it
  - Can't disable SMEP and jump directly to userspace anymore!
- The new design lets us build really strong mitigations like that

- Implementation:
  - ntos can't allocate executable physical pages (only sk can)
    - ExAllocatePoolWithTag(poolType=0) actually causes a BSOD
  - Instead, it calls into sk for this allocation
  - sk uses skci to verify that all code in ring0 is signed

Other pieces of the puzzle

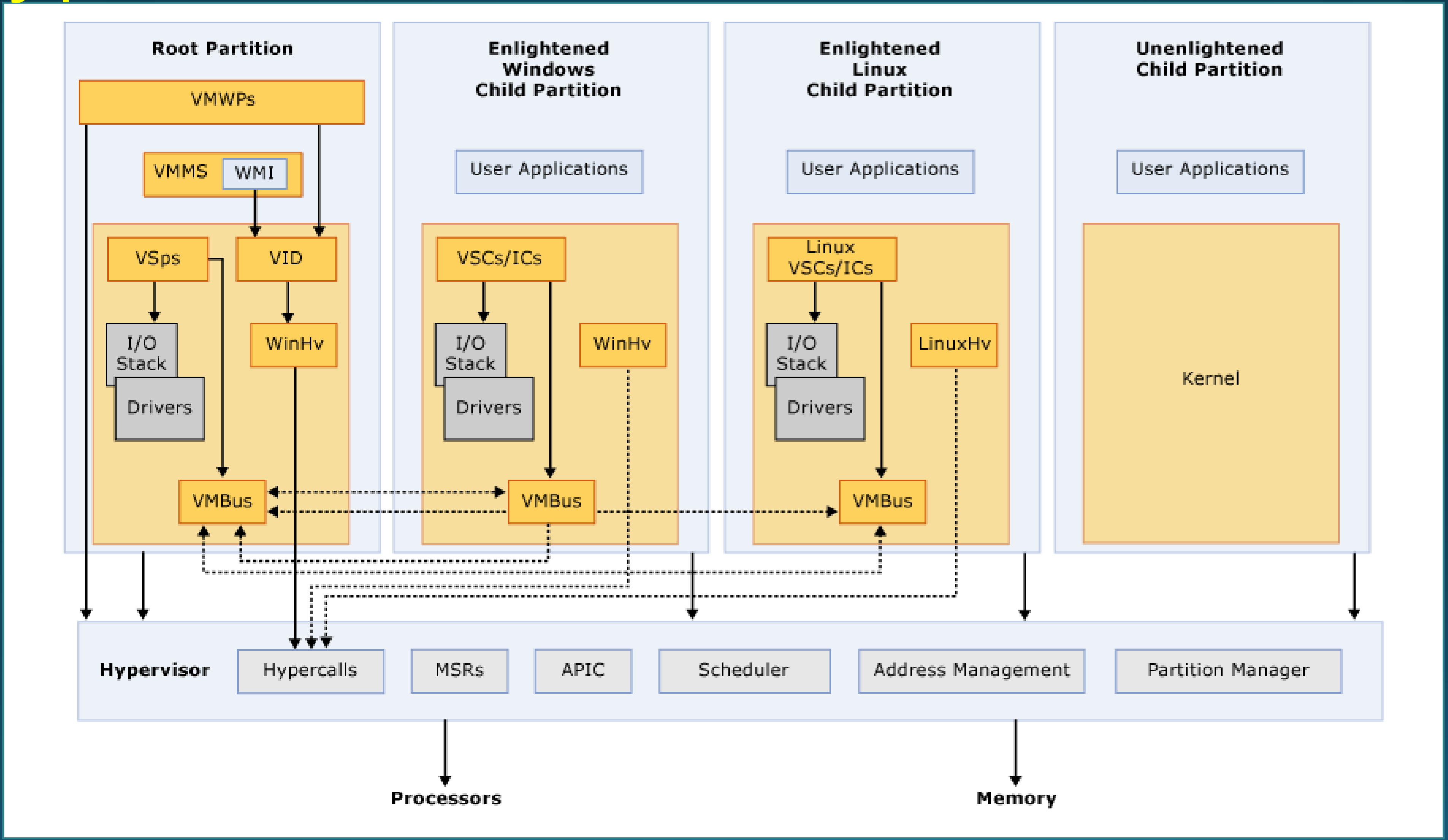
- Secureboot
  - Otherwise, can write into bootloader from the root partition, reboot and profit
- VTd
  - Otherwise, possible to overwrite HV via DMA attacks
- TPM
  - Only way to securely hold encryption keys for hiberfile across sleep states
  - Otherwise, hibernate, modify Hyper-V text section, boot and profit
  - Critical to secure S4 sleep state

VBS Attack Surface

Attack surface

- Heart of all that architecture is what we covered
  - Hypervisor platform/services, VTL1 securekernel and trustlets
- So, immediate attack surface:
  - Services exposed by VTL1: securekernel!IumInvokeSecureService
  - VTL1 calls into VLT0 for normal services – and it has to sanitize responses correctly
  - RPC with trustlets
  - Hyper-V: hypercalls, MSRs, IO ports, VMBus, etc…

Hypervisor architecture



Source: https://docs.microsoft.com/en-us/virtualization/hyper-v-on-windows/reference/hyper-v-architecture

Hypervisor attack surface

- Hypercall handlers
- MSRs
  - Reads and writes MSR handlers called from vmexit loop handler
- x86_emulator
  - Called from the vmexit loop handler, emulates many interesting things:
    - Instructions
    - Triple-fault
    - etc.
- Virtual address emulation

```
  case EXIT_REASON_APIC_WRITE:
    read_EXIT_QUALIFICATION(&v_exit_qualification);
    emulate_apic_write(
      v93,
      v_exit_qualification & 0xFF0,
      *(_DWORD *)(*(_QWORD *)(*(_QWORD *)(v16 + 3344) + 280i64) + (v_exit_qualification & 0xFF0)));
    goto LABEL_219;
  case EXIT_REASON_APIC_ACCESS|EXIT_REASON_EXTERNAL_INTERRUPT:
    read_EXIT_QUALIFICATION(&v106);
    emulate_apic_access_or_external_interrupt_handling(v17, v106);
    goto LABEL_219;
}
if ( (_DWORD)v_exit_reason != EXIT_REASON_APIC_ACCESS )
{
  if ( (_DWORD)v_exit_reason == EXIT_REASON_EPT_VIOLATION )
  {
    handle_ept_violation(v16 + 0xD00, a1);
  }
  else if ( (_DWORD)v_exit_reason == EXIT_REASON_EPT_MISCONFIG )
  {
    if ( (_DWORD)_RCX == 0x10000031 )
      handle_ept_misconfig((__int64)v17);
  }
  else
  {
    x86_emulator(
      (unsigned __int64 **)a1,
      v_exit_reason,
      a5,
      (__int64)a1,
      v19,
      a6,
      a7,
      a8,
```
0001A288 handle_vmexit:681

```
if ( (_DWORD)exit_reason == EXIT_REASON_TRIPLE_FAULT )
{
  if ( is_vpcu_in_guest_mode((__int64)*(a1 - 200)) )
    sub_FFFFFAEA088F6088((__int64)v16, 2, 0i64, 0);
  else
    check_if_keRaiseSystemError((__int64)v16, v76);
  return;
}
if ( (_DWORD)v_exit_reason == EXIT_REASON_INIT )
  call_do_reboot_((__int64)a1, exit_reason);
switch ( (_DWORD)v_exit_reason )
{
  case EXIT_REASON_PENDING_VIRT_NMI:
    if ( is_vpcu_in_guest_mode((__int64)*(a1 - 200)) && (unsigned __int8)does_cpu_have(v69, 0x400000i64) )
    {
      *((_DWORD *)v16[33] + 92) = 26;
      read_GUEST_INTERRUPTIBILITY_INFO(&v86);
      if ( v86 & 1 )
      {
        v86 = v86 & 0xFFFFFFFE | 2;
        write_GUEST_INTERRUPTIBILITY_INFO(&v86);
      }
```

00018350 x86_emulator:76

```
    case EXIT_REASON_TASK_SWITCH:
      handle_reason_task_switch(v15 + 3328, (__int64)v16);
      return;
    case EXIT_REASON_CPUID:
      if ( !is_vpcu_in_guest_mode((__int64)*(a1 - 200)) )
      {
        handle_reason_cpuid((__int64)v16);
        return;
      }
      *((_DWORD *)v16[33] + 92) = 18;
do_exit_inst_flow_return:
      do_exit_inst_flow();
      return;
    case EXIT_REASON_INVD:
      if ( !is_vpcu_in_guest_mode((__int64)*(a1 - 200)) )
      {
        handle_reason_invd((__int64)v16);
        return;
      }
      goto do_exit_inst_flow_return;
    case EXIT_REASON_INVLPG:
      read_EXIT_QUALIFICATION(v16 + 2);
      if ( is_vpcu_in_guest_mode(v67) && (unsigned __int8)does_cpu_have(v68, 0x200i64) )
      {
```

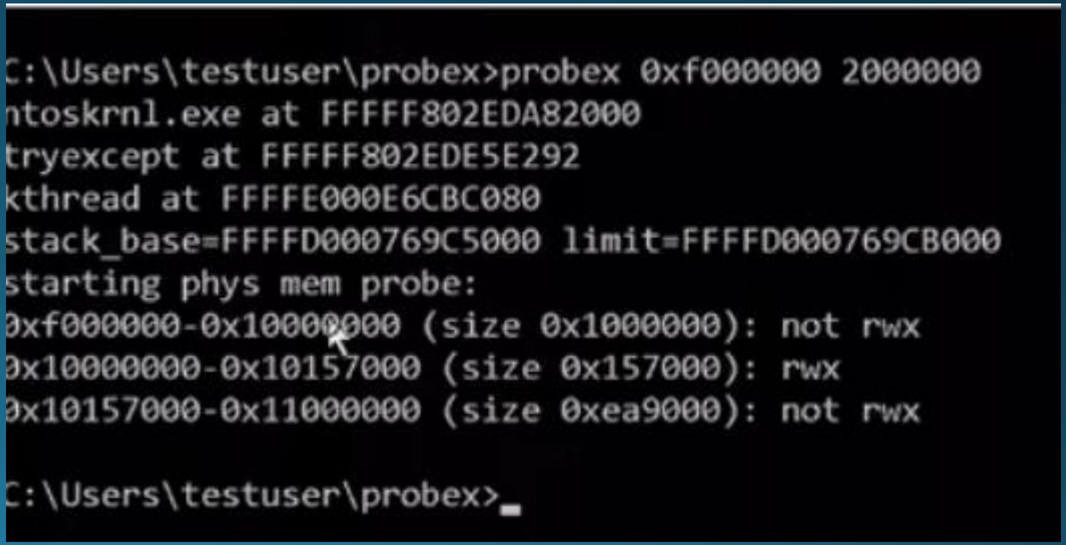
0018CF1 x86_emulator:133

Hypervisor attack surface

- The root partition can access almost the entire physical address space, including:
  - MMIO
  - exceptions being LAPIC
  - VTd bars
- Some pages are shared with VTL1
  - Libraries, like iumdll.dll
  - Marked as RO in the root partition's EPT
- With HVCI enabled:
  - UEFI runtime pages are marked as RO
  - UEFI runtime executes in VTL1 context
- Without HVCI
  - UEFI runtime pages are marked as writable!
  - UEFI runtime executes in root partition kernel context

Example: HVCI bypass

- CVE-2016-0181 by Rafal Wojtczuk
- There were some pages with RWX permission in ring0 EPT
  - Likely artifacts of early boot phase / EFI code
- Trivial HVCI bypass – execute unsigned code in ring0vtl0
  - With VBS, it **is** a security boundary ☺



Hypervisor attack surface

- PCIe config space (accessible via MMCFG)
  - Device-specific registers
  - Memory bars locations
  - REMAP_LIMIT/REMAP_BASE are locked (trivially)
  - What can we overlap?
- IO port
  - All are directly accessible, with some exceptions
  - Easy to see them, configured in the root partition's VMCS IO bitmap
- Finally, in RS4: protect vtl0 from DMA attacks
  - not only vtl1 :)
  - Using IOMMU remapping (hal!dmr*)
  - https://twitter.com/AmarSaar/status/985618204184768513

VTL1 attack surface

- Services exposed by VTL1
  - LsaIso RPC services
    - Lots of RPC demarshalling code (vtl1ring3)
    - >50 services implemented in securekernel!IumInvokeSecureService (vtl1ring0)
- VTL1 extensively calls to the root partition for some services
  - We want to maintain lots of the current mechanisms in VTL0
    - To keep VTL1 attack surface as small as possible
  - So, lots of sanitizing/parsing of responses from VTL0 occurs in VTL1

VMBus && vmswitch

- VMBus has a serious part in this design
  - IPC between partitions
  - Really important in Azure
- VMBus has to do lots of parsing, marshalling, etc…
  - Can be complex and sensitive code
- Same goes for vmswitch
  - Google Project Zero found a potential g2h there
  - https://bugs.chromium.org/p/project-zero/issues/detail?id=688

Hyper-V vmswitch.sys VmsMpCommonPvtHandleMulticastOids Guest to Host Kernel-Pool Overflow

Reported by kost...@google.com, Jan 4 2016

This bug is subject to a 90 day disclosure deadline. If 90 days elapse without a broadly available patch, then the bug report will automatically become visible to the public.

This function is reachable by sending a RNDIS Set request with OID 0x01010209 (OID_802_3_MULTICAST_LIST) from the Guest to the Host.

This function potentially allocates a buffer based on the addresses sent.
The number of entries is determined by dividing the length of the data by 6:

```
.text:000000000001D717 mov eax, 0AAAAAAABh
.text:000000000001D71C mov r13b, 1
.text:000000000001D71F mul r14d
.text:000000000001D722 mov ebp, edx
.text:000000000001D724 shr ebp, 2
.text:000000000001D727 test ebp, ebp ; ebp=r14d//6
.text:000000000001D729 jz loc_31B04
.text:000000000001D72F
.text:000000000001D72F loc_1D72F: ; CODE XREF: VmsMpCommonPvtHandleMulticastOids+144CEj
.text:000000000001D72F cmp ebp, [rbx+0EE8h]
.text:000000000001D735 jz loc_31B2B
.text:000000000001D73B mov r8d, 'mcMV' ; Tag
.text:000000000001D741 mov rdx, r14 ; NumberOfBytes
.text:000000000001D744 mov ecx, 200h ; PoolType
.text:000000000001D749 mov r12, r14
.text:000000000001D74C call cs:__imp_ExAllocatePoolWithTag .text:000000000001D752 mov r14, rax
.text:000000000001D755 test rax, rax
.text:000000000001D758 jz loc_1D7E8
.text:000000000001D75E mov r8, r12 ; Size
.text:000000000001D761 mov rdx, r15 ; Src
.text:000000000001D764 mov rcx, rax ; Dst
.text:000000000001D767 call memmove
```

An interesting test is located at 0x1D72F.
If the number of entries is identical to the currently stored one, then we jump to this piece of code:

So… what’s new?

Last year…

- MS keeps adding code, features and patches to these components
- It’s REALLY important to keep track of that…

Microcode update && Hotpatch

- Microcode update and hotpatch mechanisms were added to the hypervisor in RS4
- Microcode update:
  - look out for a new hypercall: 0x7
  - Called from ntos: nt!HvlUpdateMicrocode
- Easy flow to reverse, see known MSRs:

```
__writemsr(MSR_IA32_UCODE_REV, 0i64);
_RAX = 1i64;
__asm { cpuid }                             // ; amarsa: Intel supplies the microcode
                                            // ; revision value in MSR 0x8b,
                                            // ; after CPUID(1) has been executed.
                                            // ; Execute it each time before reading that MSR.
                                            //
v_code_revision = __readmsr(MSR_IA32_UCODE_REV);
v_code_revision_ = (((unsigned __int64)HIDWORD(v_code_revision) << 32) | (unsigned int)v_code_revision) >> 32;
*pOutCodeRevision = v_code_revision_;
if ( (_DWORD)v_code_revision_ != a1[1] )
  __writemsr(MSR_IA32_UCODE_WRITE, microcode_bits);// write microcode->bits array
```

IDA View-A | Hex View-1 | Structures | Enums | Imports | Exports | Matched Functions | Primary Unmatched

EA	Name	Basicblock	Instructions	Edges
0000000140019E10	ShvlGetInterceptData	1	20	0
0000000140019E68	ShvlCompleteIntercept	9	50	11
000000014001A2E8	ShvlPrepareHypervisorPatch	18	78	24
000000014001A414	ShvlLoadHypervisorPatch	22	174	32
000000014001F080	SkmiConvertImagePfn	4	19	4
000000014001F0D4	SkmiReplicateGapFrame	5	59	6
0000000140025A10	SkmiReserveHpatAddresses	12	61	18
0000000140034F44	SkmiInitializeHypervisorPatching	17	70	24
000000014003505C	SkmiCaptureDriverIdentity	8	38	10
00000001400350E8	SkmiIsPatchPagePresent	8	29	10
0000000140035148	SkmiCheckUndoPagesPresent	19	61	27
000000014003520C	SkmiExpandHpat	24	197	34
0000000140035534	SkmiFreeDummyPatchMapping	9	56	13
0000000140035618	SkmiCreateDummyPatchMapping	67	301	103
0000000140035A50	SkmiPrepareNormalDummyPatchContext	18	165	26
0000000140035CE8	SkmiPrepareGuardHotPatch	10	65	15
0000000140035DDC	SkmiPreApplyHotPatch	16	112	22
0000000140035FA4	SkmiApplyPatchesToPatchImage	4	39	4
000000014003604C	SkmiApplyHotPatch	88	419	142
0000000140036640	SkmiRevertHotPatch	14	110	19
0000000140036800	SkmiPrepareHotPatchBaseImage	11	55	15
00000001400368C4	SkmiMatchPatchImage	10	32	14
0000000140036934	SkmiFreeSecureImagePages	10	50	14
0000000140036A04	SkmiCopyPatchImage	25	174	38
0000000140036CB4	SkmiCopySecurePatchImage	8	71	11
0000000140036DCC	SkmiUnloadPatchImage	12	60	16
0000000140036EB8	SkmiPrepareSecurePatchContext	18	95	25
000000014003703C	SkmiCleanupSecurePatchContext	6	31	8
00000001400370BC	SkmiPatchSecureImage	20	138	29
00000001400372D4	SkmiApplyHypervisorHotPatch	28	146	42
00000001400374EC	SkmiApplySecureHotPatch	12	54	17
00000001400376F8	SkmmDetermineHotPatchUndoTableSize	16	62	24
00000001400377FC	SkmmObtainHotPatchUndoTable	34	158	53
0000000140037A5C	SkmiPrepareDriverPatchContext	6	67	8
0000000140037B60	SkmmApplyHotPatch	74	325	118
00000001400380E8	SkmmRevertHotPatch	19	77	29
00000001400404A0	SkpgEnumerateSections	22	86	33
0000000140042AF0	SkpgVerifyKernelVaProtectionExtent	19	87	27
000000014004380C	SkpgCompareExtensionTableEntries	3	12	2
0000000140043834	SkpgCompareVas	3	10	2
0000000140043854	SkpgLocateImageExtents	23	133	35
0000000140043A3C	SkpgHotpatchApply	8	32	11

Trusted Boot

- SMX support was added to the hypervisor in RS4
  - For example, new MSR, 0x40000116
- According to the documentation:
  - SMX registers are memory mapped to 0xFED20000 - FED3FFFF
  - TPM registers are mapped to 0xFED40000 - 0xFED4FFFF
  - These regions are listed as allocated resources in the Device Manager application, so the system is finding them
  - https://software.intel.com/en-us/forums/virtualization-software-development/topic/297709

```
FFFFFAEA088A5910 mov     rcx, [rbx+2380h]
FFFFFAEA088A5917 lea     r9, HalpTxtProcStartup
FFFFFAEA088A591E mov     r8d, 6
FFFFFAEA088A5924 mov     edx, esi
FFFFFAEA088A5926 call    HalMapIoSpaceWithStatus
FFFFFAEA088A592B movzx   edi, ax
FFFFFAEA088A592E test    ax, ax
FFFFFAEA088A5931 jnz     return_error
```

```
FFFFFAEA088A5937 lea     r9, HalpTxtPrivateSpace
FFFFFAEA088A593E xor     r8d, r8d
FFFFFAEA088A5941 mov     edx, esi
FFFFFAEA088A5943 mov     ecx, 0FED20000h
FFFFFAEA088A5948 call    HalMapIoSpaceWithStatus
FFFFFAEA088A594D movzx   edi, ax
FFFFFAEA088A5950 test    ax, ax
FFFFFAEA088A5953 jnz     return_error
```

Hyper-V
ASLR

ASLR

- Usually we need to gain relative/arbitrary read primitive
- Fixed addresses are very helpful…
  - PTE base (and then… Anniversary update!)
  - HAL's heap (and then… Creators update!)
  - sharedpage
    - Now we even have a hypervisor sharedpage: ntdll!RtlphypervisorSharedUserVa
      - Not fixed, but can be predicted…
  - etc…
- Some pages in hv are not fully randomized…
  - PTE base (0xffffff7f80000000)
  - hv_lowstub (0x1000 / 0x2000)
  - self-referenced PTE entry 0x1fe (0xffffff7fbfdfe000)
  - And I keep finding stuff…

Saar Amar
@AmarSaar

In continuation to my "fixed addresses in hv" tweet: time to talk about new structure, which I saw at address 0x40000 in few boots in a row. Details will come soon!

```
*
*                    THIS IS NOT A BUG OR A SYSTEM CRASH
*
* If you did not intend to break into the debugger, press the "g"
* press the "Enter" key now.  This message might immediately reapp
* does, press "g" and "Enter" again.
*
*******************************************************************
hv+0x27bfe0:
fffffb06`aa47bfe0 cc              int     3
0: kd> dd 40000
00000000`00040000  00380001 00000000 00000009 00000000
00000000`00040010  00048000 00000000 00040000 00000000
00000000`00040020  00048000 00000000 000411e8 00000000
00000000`00040030  000411e0 00000000 01080007 00000000
00000000`00040040  ffe11290 ffe1129c ffe1120c ffe19168
00000000`00040050  ffe19174 ffe19180 ffe1918c ffe21058
00000000`00040060  ffe2c154 00000000 00000000 00000000
00000000`00040070  00000000 00000000 00000000 00000000
0: kd> g
```

Saar Amar
@AmarSaar

beside to my new 0x40000 structure (twitter.com/AmarSaar/statu ...), you can count in 0xfffffe0000000000 as well (hardcoded in binary). Looks the same between boots (more details about it will be coming soon :))

Kernel 'com:port=\\.\pipe\Vm0,pipe,resets=0,reconnect' - WinDbg:10.0.14321.1024 AMD64

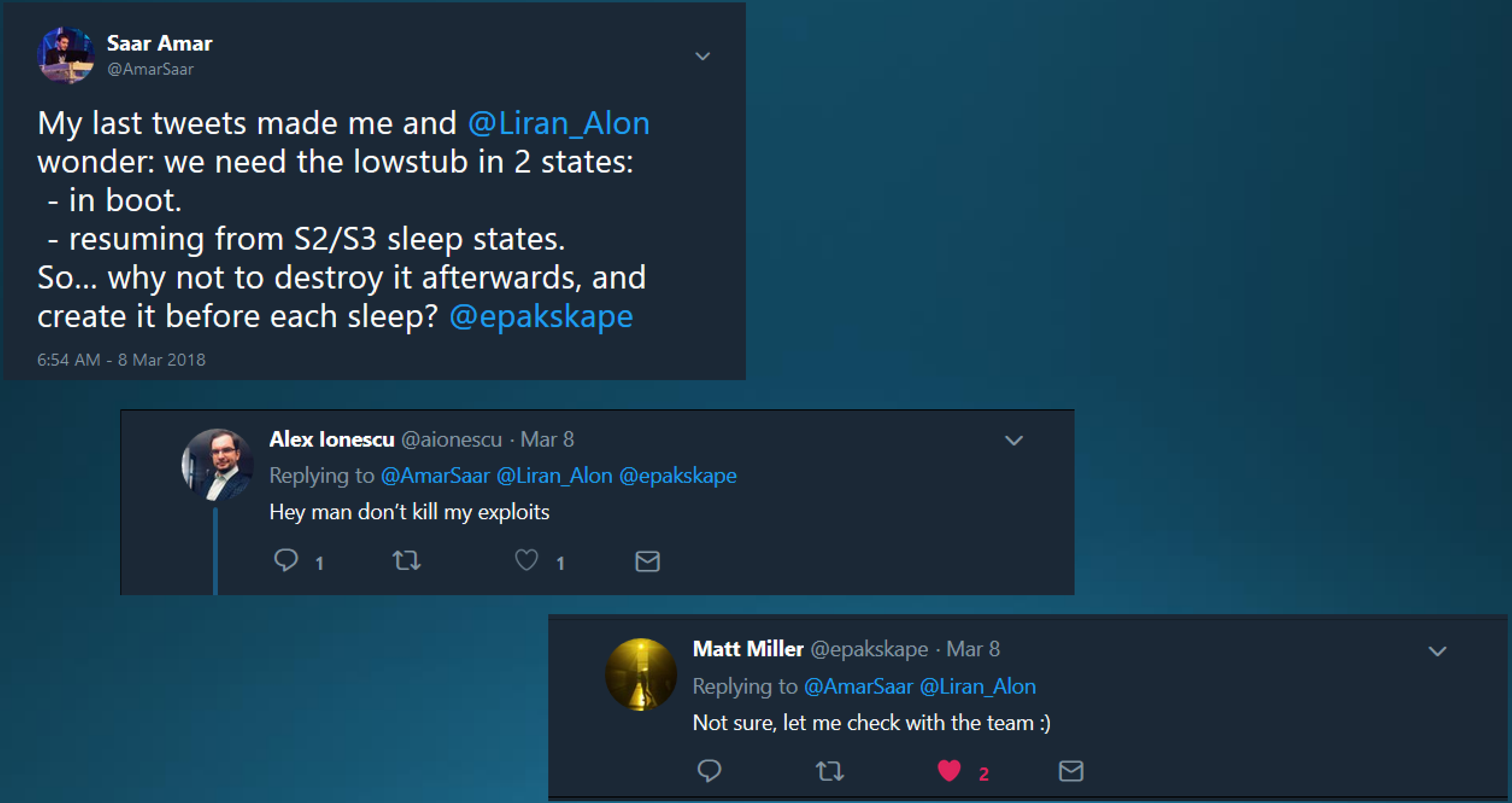
File Edit View Debug Window Help

Command - Kernel 'com:port=\\.\pipe\Vm0,pipe,resets=0,reconnect' - WinDbg:10.0.14321.1024 AMD64

```
kd> g
Break instruction exception - code 80000003 (first chance)
*******************************************************************************
*                                                                             *
*   You are seeing this message because you pressed either                    *
*       CTRL+C (if you run console kernel debugger) or,                       *
*       CTRL+BREAK (if you run GUI kernel debugger),                          *
*   on your debugger machine's keyboard.                                      *
*                                                                             *
*                   THIS IS NOT A BUG OR A SYSTEM CRASH                       *
*                                                                             *
* If you did not intend to break into the debugger, press the "g" key, then   *
* press the "Enter" key now.  This message might immediately reappear.  If it *
* does, press "g" and "Enter" again.                                          *
*                                                                             *
*******************************************************************************
hv+0x278470:
fffff86c`9fe78470 cc              int     3
kd> dq fffffe0000000000
fffffe00`00000000  00000000`00000000 00000000`00000031
fffffe00`00000010  00000000`0000000d 00000000`00000031
```

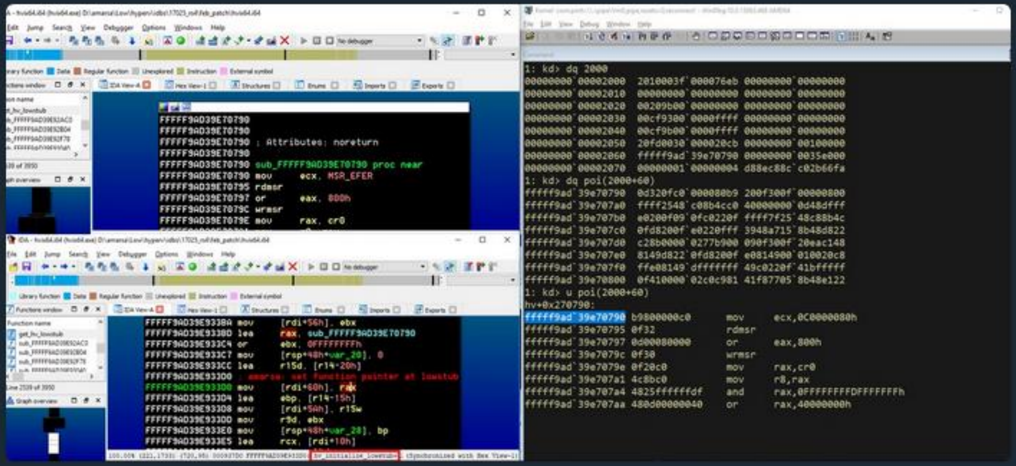
lowstub

- The reason I wake up every morning
- Extremely interesting structure, contains lots of interesting stuff
  - Located in a predictable physical address
  - 16bit code stub
  - Self referenced in virtual memory
  - KPROCESSOR_STATE (cr3, rip, rsp, rcx)
- Has 2 responsibilities:
  - On boot time– bring up from real mode to protected mode, and then to long mode
  - When processor resumes from S2/S3 sleep states
- Its (physical) address can be predicted – and we have +X memory there!
- Question: What bothers me about this?


Saar Amar
@AmarSaar
My last tweets made me and @Liran_Alon wonder: we need the lowstub in 2 states:
- in boot.
- resuming from S2/S3 sleep states.
So... why not to destroy it afterwards, and create it before each sleep? @epakskape
6:54 AM - 8 Mar 2018
Alex Ionescu @aionescu · Mar 8
Replying to @AmarSaar @Liran_Alon @epakskape
Hey man don't kill my exploits
1
1
Matt Miller @epakskape · Mar 8
Replying to @AmarSaar @Liran_Alon
Not sure, let me check with the team :)
2

Saar Amar
@AmarSaar

Little bonus to last tweet - the lowstub has a pointer to code, so if you know its' addr (and it's pretty much fixed), you can break randomization over hv base (credit goes to @aionescu and his talk about type-c attacks and lowstub internals)



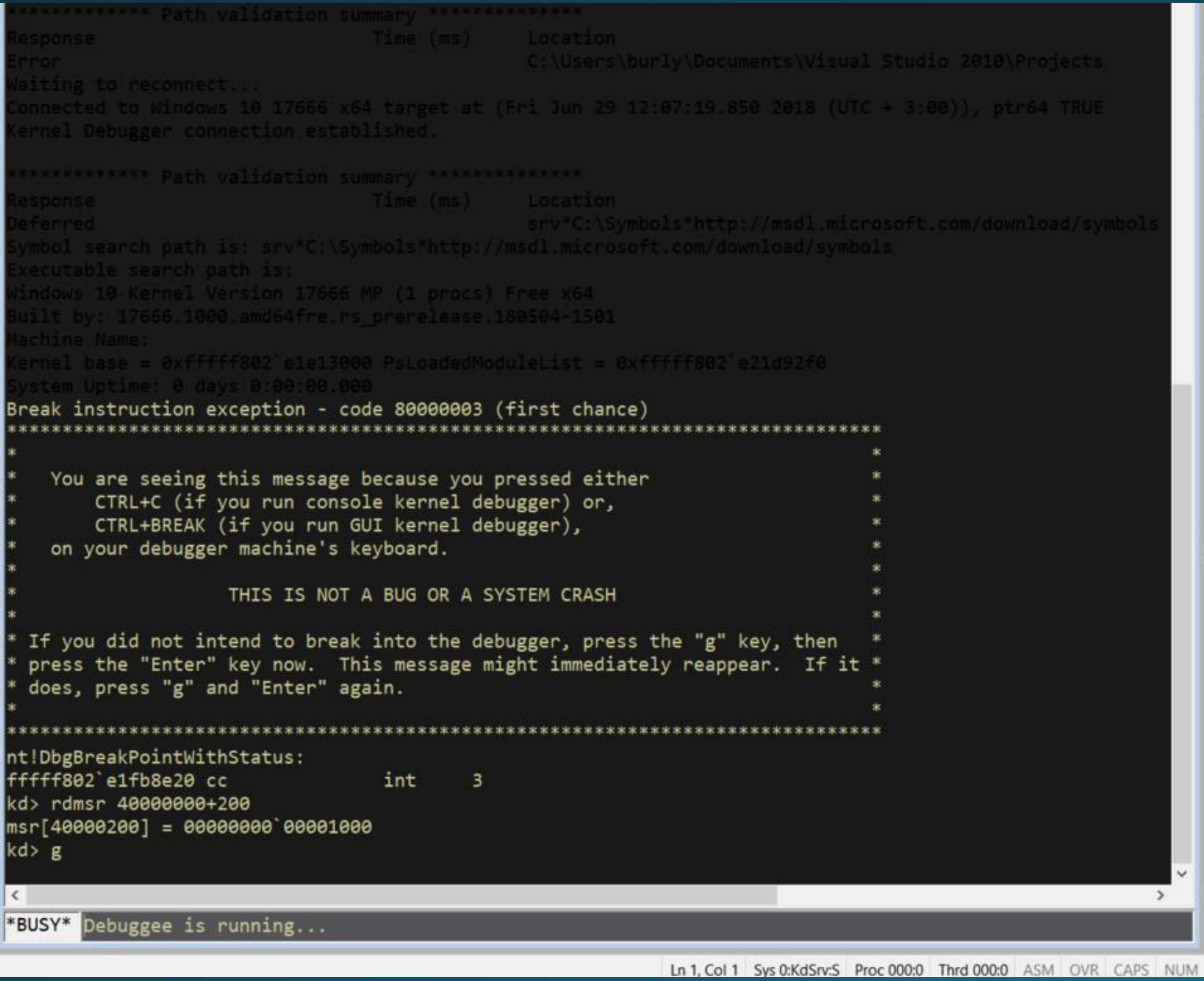
Command

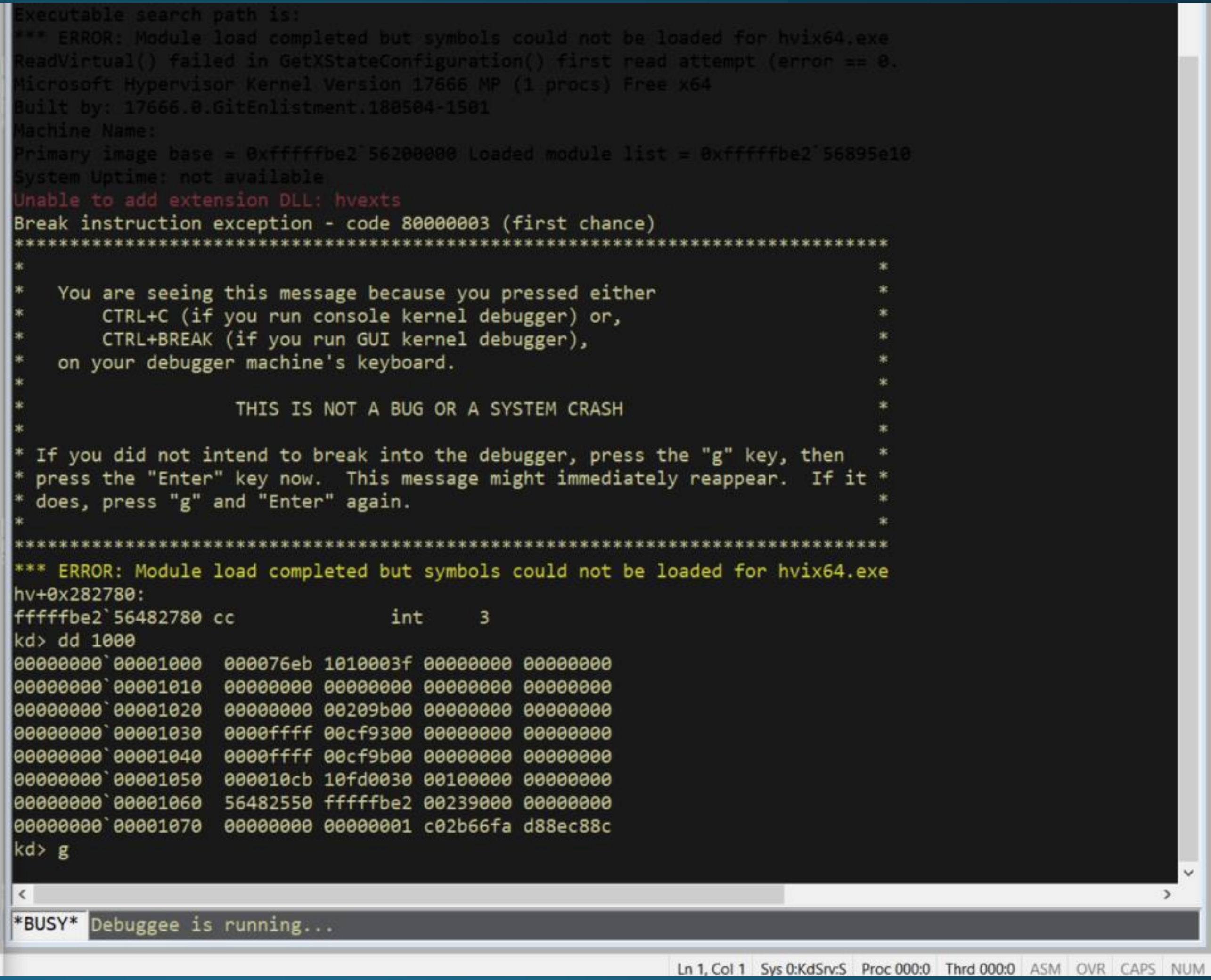
```
*** ERROR: Module load completed but symbols could not be loaded for hvix64.exe
hv+0x282780:
fffff8f3`d6282780 cc              int     3
kd> lm
start             end                 module name
fffff8f3`d5f78000 fffff8f3`d5f84000   kdstub     (deferred)
fffff8f3`d6000000 fffff8f3`d7600000   hv         (no symbols)
kd> dd 1000
00000000`00001000  000076eb 1010003f 00000000 00000000
00000000`00001010  00000000 00000000 00000000 00000000
00000000`00001020  00000000 00209b00 00000000 00000000
00000000`00001030  0000ffff 00cf9300 00000000 00000000
00000000`00001040  0000ffff 00cf9b00 00000000 00000000
00000000`00001050  000010cb 10fd0030 00100000 00000000
00000000`00001060  d6282550 fffff8f3 00239000 00000000
00000000`00001070  00000000 00000001 c02b66fa d88ec88c
kd> u 1000
00000000`00001000 eb76            jmp     00000000`00001078
00000000`00001002 0000            add     byte ptr [rax],al
00000000`00001004 3f              ???
00000000`00001005 0010            add     byte ptr [rax],dl
00000000`00001007 1000            adc     byte ptr [rax],al
00000000`00001009 0000            add     byte ptr [rax],al
00000000`0000100b 0000            add     byte ptr [rax],al
00000000`0000100d 0000            add     byte ptr [rax],al
kd> u 1078
00000000`00001078 fa              cli
00000000`00001079 662bc0          sub     ax,ax
00000000`0000107c 8cc8            mov     eax,cs
00000000`0000107e 8ed8            mov     ds,ax
00000000`00001080 66be7000        mov     si,70h
00000000`00001084 0000            add     byte ptr [rax],al
00000000`00001086 6766c7060100    mov     word ptr [esi],1
00000000`0000108c 0000            add     byte ptr [rax],al
```

The lowstub MSR

- Apparently, in RS4 we got a new MSR: 0x40000200
- This MSR simply reads (not writes) the address of hv_lowstub
  - Accessible from the root partition

```
case 0x40000200:
  v_partitionPermissionOK = *(__readgsqword(CurrentPartition) + offsetof(partition_s, partitionPrivilegeFlags)) &
  v_retval = HV_STATUS_ACCESS_DENIED;
  if ( v_partitionPermissionOK )
    v_retval = HV_STATUS_SUCCESS;
  if ( v_partitionPermissionOK )
    *pOut = hv_lowstub;
```


************* Path validation summary **************
Response   Time (ms)   Location
Error   C:\Users\burly\Documents\Visual Studio 2010\Projects
Waiting to reconnect...
Connected to Windows 10 17666 x64 target at (Fri Jun 29 12:07:19.850 2018 (UTC + 3:00)), ptr64 TRUE
Kernel Debugger connection established.
************* Path validation summary **************
Response   Time (ms)   Location
Deferred   srv*C:\Symbols*http://msdl.microsoft.com/download/symbols
Symbol search path is: srv*C:\Symbols*http://msdl.microsoft.com/download/symbols
Executable search path is:
Windows 10 Kernel Version 17666 MP (1 procs) Free x64
Built by: 17666.1000.amd64fre.rs_prerelease.180504-1501
Machine Name:
Kernel base = 0xfffff802`e1e13000 PsLoadedModuleList = 0xfffff802`e21d92f0
System Uptime: 0 days 0:00:00.000
Break instruction exception - code 80000003 (first chance)
****
*   You are seeing this message because you pressed either
*   CTRL+C (if you run console kernel debugger) or,
*   CTRL+BREAK (if you run GUI kernel debugger),
*   on your debugger machine's keyboard.
*   THIS IS NOT A BUG OR A SYSTEM CRASH
* If you did not intend to break into the debugger, press the "g" key, then
* press the "Enter" key now. This message might immediately reappear. If it
* does, press "g" and "Enter" again.
****
nt!DbgBreakPointWithStatus:
fffff802`e1fb8e20 cc   int   3
kd> rdmsr 40000000+200
msr[40000200] = 00000000`00001000
kd> g
BUSY Debuggee is running...
Ln 1, Col 1 Sys 0:KdSrv:S Proc 000:0 Thrd 000:0 ASM OVR CAPS NUM


Executable search path is:
*** ERROR: Module load completed but symbols could not be loaded for hvix64.exe
ReadVirtual() failed in GetXStateConfiguration() first read attempt (error == 0.
Microsoft Hypervisor Kernel Version 17666 MP (1 procs) Free x64
Built by: 17666.0.GitEnlistment.180504-1501
Machine Name:
Primary image base = 0xfffffbe2`56200000 Loaded module list = 0xfffffbe2`56895e10
System Uptime: not available
Unable to add extension DLL: hvexts
Break instruction exception - code 80000003 (first chance)
****
*   You are seeing this message because you pressed either
*   CTRL+C (if you run console kernel debugger) or,
*   CTRL+BREAK (if you run GUI kernel debugger),
*   on your debugger machine's keyboard.
*   THIS IS NOT A BUG OR A SYSTEM CRASH
* If you did not intend to break into the debugger, press the "g" key, then
* press the "Enter" key now. This message might immediately reappear. If it
* does, press "g" and "Enter" again.
****
*** ERROR: Module load completed but symbols could not be loaded for hvix64.exe
hv+0x282780:
fffffbe2`56482780 cc   int   3
kd> dd 1000
00000000`00001000 000076eb 1010003f 00000000 00000000
00000000`00001010 00000000 00000000 00000000 00000000
00000000`00001020 00000000 00209b00 00000000 00000000
00000000`00001030 0000ffff 00cf9300 00000000 00000000
00000000`00001040 0000ffff 00cf9b00 00000000 00000000
00000000`00001050 000010cb 10fd0030 00100000 00000000
00000000`00001060 56482550 fffffbe2 00239000 00000000
00000000`00001070 00000000 00000001 c02b66fa d88ec88c
kd> g
BUSY Debuggee is running...
Ln 1, Col 1 Sys 0:KdSrv:S Proc 000:0 Thrd 000:0 ASM OVR CAPS NUM

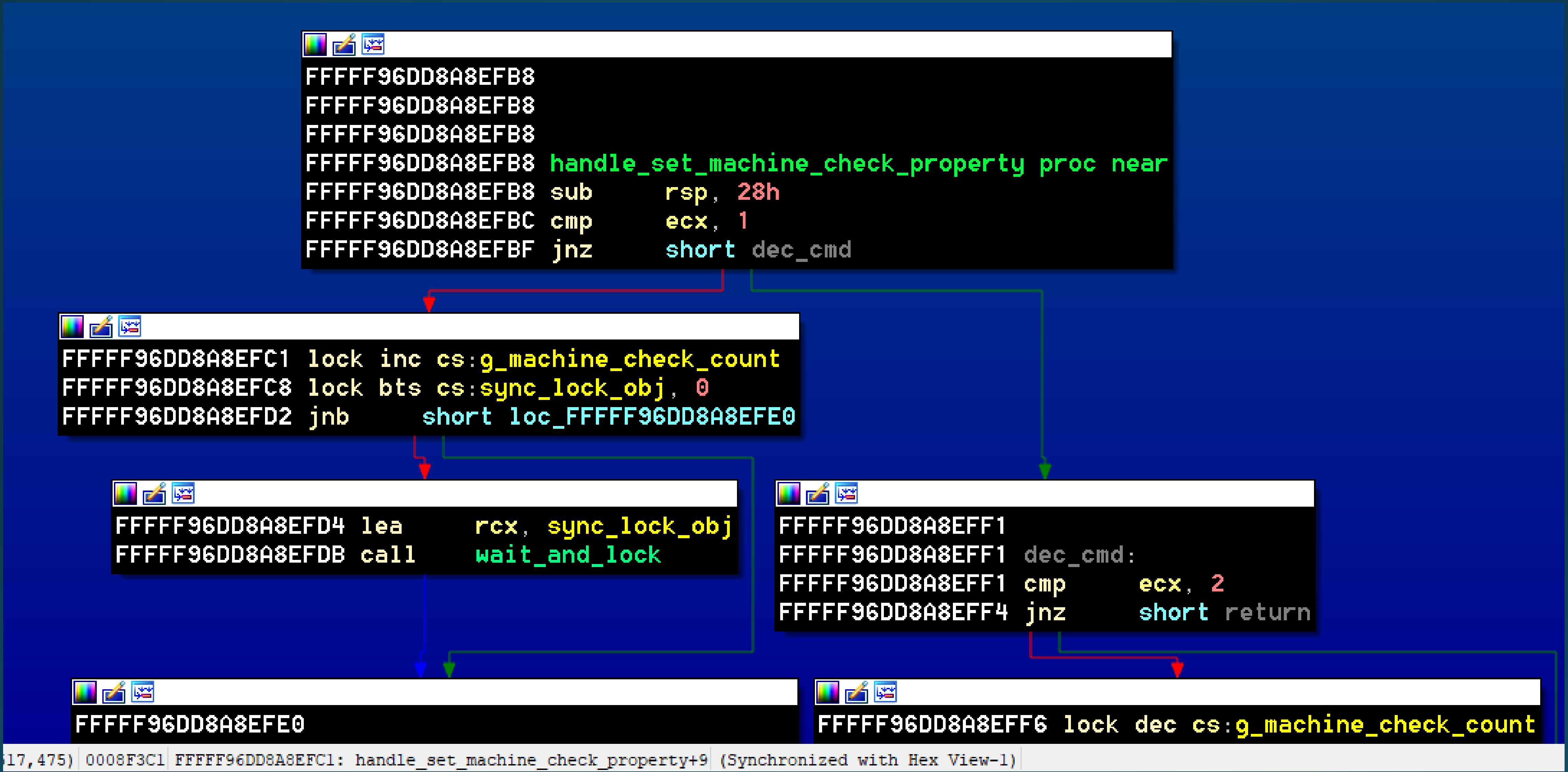
So… let’s talk about NEW bugs

Bootlib

- Bootlib.dll, statically linked to winload, ci, skci, etc…
- Contains many interesting pieces of code
  - Including DER parsing
- I found 2 bugs in *MinAsn1StringToOid*
  - One is wrong return value
  - Second is off-by-one on a static buffer passed from the caller
- None of them is a security issue, but MS fixed them after all ☺
  - https://twitter.com/AmarSaar/status/976797760308633602

hv!handle_machine_check_property

- Really cute bug, accessible from a trivial hypercall flow
  - *HvCallSetSystemProperty*
- Integer overflow/underflow of global counts vars
- Can't crash, so it isn't security issue
- Should be fixed in the next version ☺


FFFFF96DD8A8EFB8
FFFFF96DD8A8EFB8
FFFFF96DD8A8EFB8
FFFFF96DD8A8EFB8 handle_set_machine_check_property proc near
FFFFF96DD8A8EFB8 sub rsp, 28h
FFFFF96DD8A8EFBC cmp ecx, 1
FFFFF96DD8A8EFBF jnz short dec_cmd
FFFFF96DD8A8EFC1 lock inc cs:g_machine_check_count
FFFFF96DD8A8EFC8 lock bts cs:sync_lock_obj, 0
FFFFF96DD8A8EFD2 jnb short loc_FFFFF96DD8A8EFE0
FFFFF96DD8A8EFD4 lea rcx, sync_lock_obj
FFFFF96DD8A8EFDB call wait_and_lock
FFFFF96DD8A8EFF1
FFFFF96DD8A8EFF1 dec_cmd:
FFFFF96DD8A8EFF1 cmp ecx, 2
FFFFF96DD8A8EFF4 jnz short return
FFFFF96DD8A8EFE0
FFFFF96DD8A8EFF6 lock dec cs:g_machine_check_count
17,475) 0008F3C1 FFFFF96DD8A8EFC1: handle_set_machine_check_property+9 (Synchronized with Hex View-1)

Kernel 'com:port=\\.\com2,pipe,resets=0,reconnect' - WinDbg:10.0.17134.1 AMD64

File Edit View Debug Window Help

Command - Kernel 'com:port=\\.\com2,pipe,resets=0,reconnect' - WinDbg:10.0.17134.1 AMD64

```
kd> bp HvcallInitiateHypercall
kd> g
Breakpoint 0 hit
nt!HvcallInitiateHypercall:
fffff802`25fffca0 4883ec28        sub     rsp,28h
kd> rrcx=1006f
kd> rrdx=4
kd> rr8=200000002
kd> rr9=0
kd> g
```

Kernel 'com:port=\\.\com1,pipe,resets=0,reconnect' - WinDbg:10.0.17134.1 AMD64

File Edit View Debug Window Help

Command

```
kd> lm
start             end                 module name
fffff96d`d869a000 fffff96d`d86a6000   kdstub     (deferred)
fffff96d`d8800000 fffff96d`d9e00000   hv         (no symbols)
kd> dd fffff96d`d8ee1310 L2
fffff96d`d8ee1310  00000000 00000000   .data:FFFFF96DD8EE1310 g_machine_check_count dd ?
kd> g
Break instruction exception - code 80000003 (first chance)
*******************************************************************************
*                                                                             *
*   You are seeing this message because you pressed either                    *
*       CTRL+C (if you run console kernel debugger) or,                       *
*       CTRL+BREAK (if you run GUI kernel debugger),                          *
*   on your debugger machine's keyboard.                                      *
*                                                                             *
*                   THIS IS NOT A BUG OR A SYSTEM CRASH                       *
*                                                                             *
* If you did not intend to break into the debugger, press the "g" key, then   *
* press the "Enter" key now.  This message might immediately reappear.  If it *
* does, press "g" and "Enter" again.                                          *
*                                                                             *
*******************************************************************************
hv+0x282780:
fffff96d`d8a82780 cc              int     3
kd> dd fffff96d`d8ee1310 L2
fffff96d`d8ee1310  ffffffff 00000000
kd> g
```

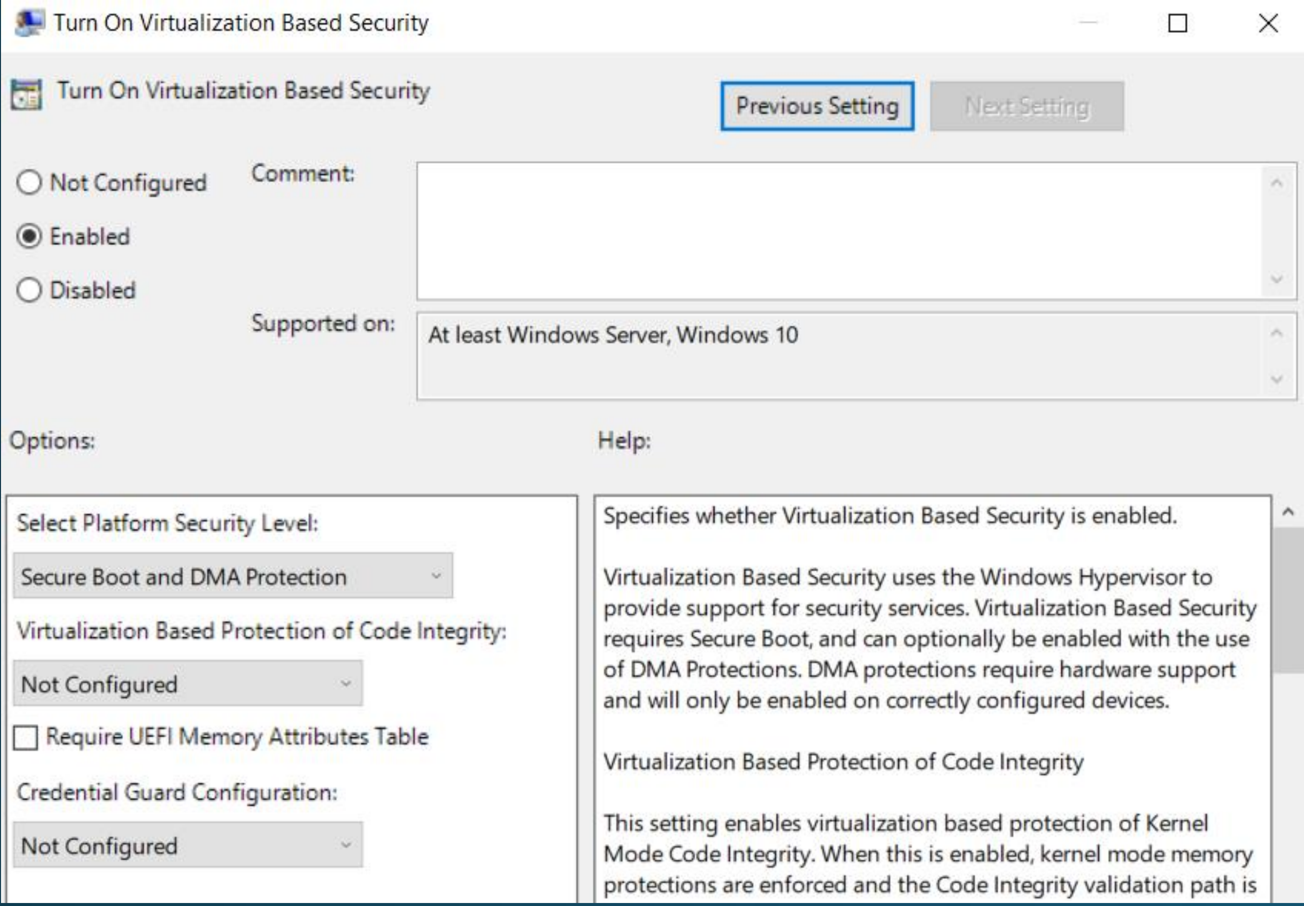
Segfault in HV!

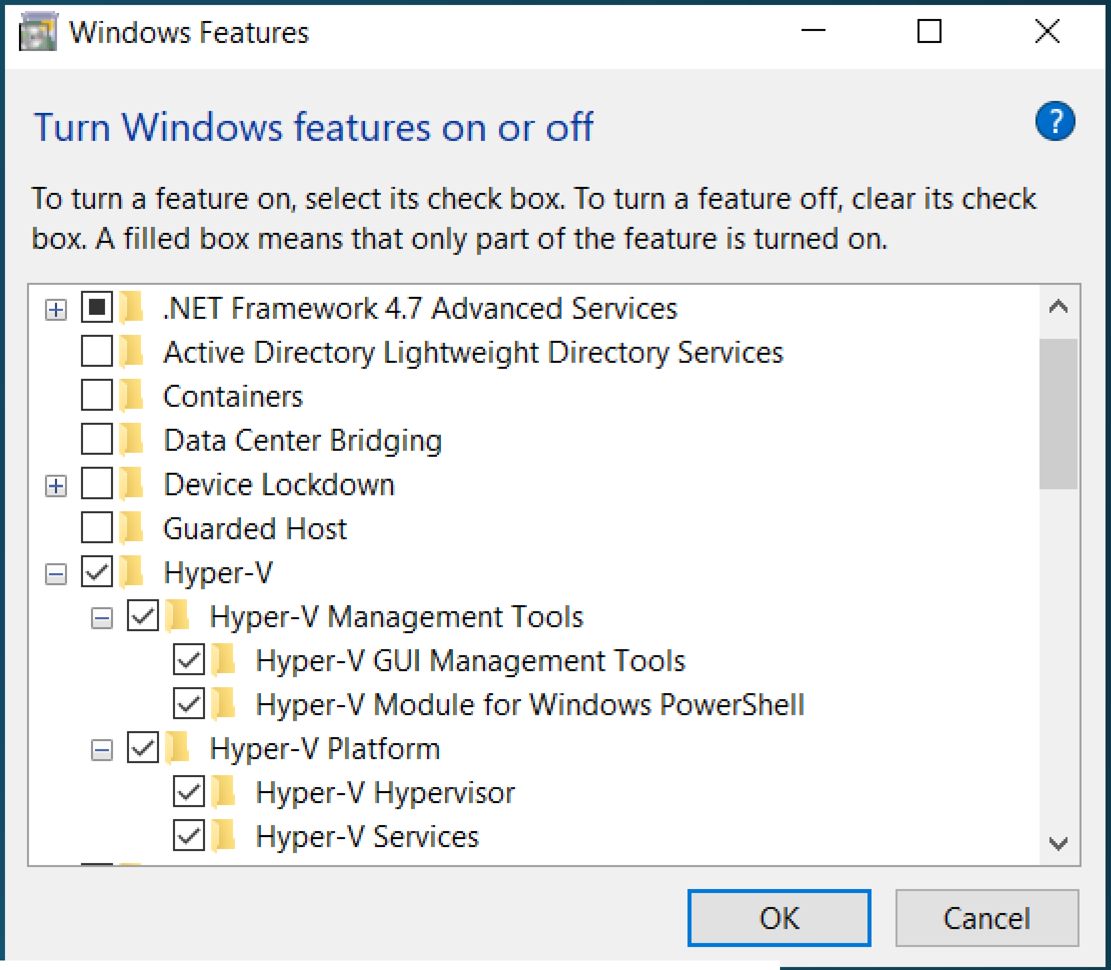
Got fixed in build 17711 (RS5)

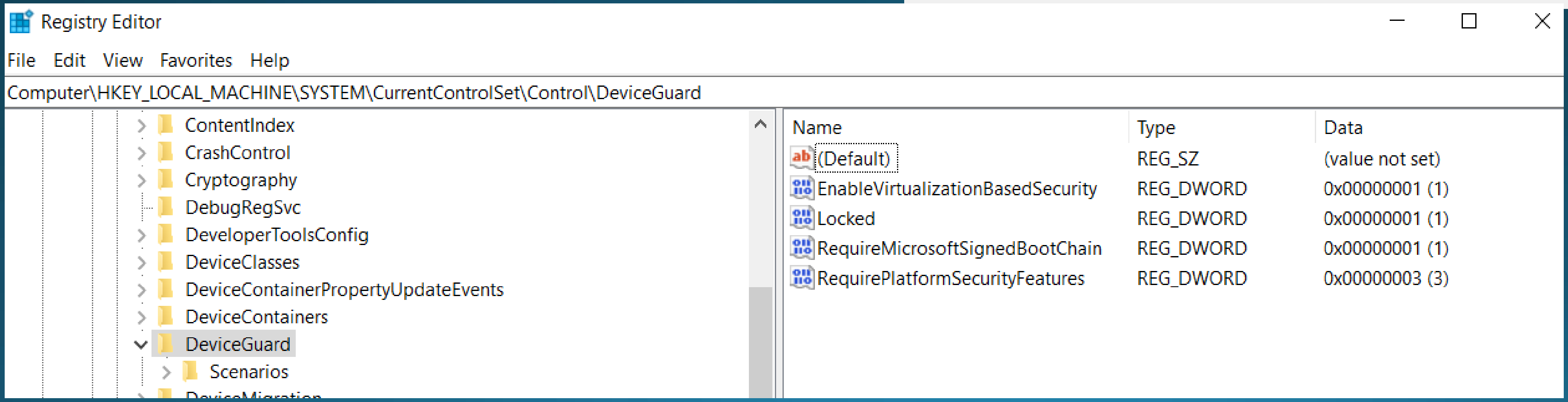
```
System Uptime: not available
Unable to add extension DLL: hvexts
Access violation - code c0000005 (!!! second chance !!!)
*** ERROR: Module load completed but symbols could not be loaded for hvix64.exe
hv+0x216837:
fffff96f`30616837 48396b10        cmp     qword ptr [rbx+10h],rbp
kd> dq @rdi
fffff96f`30ae2000  fffff96f`30ae2000 00000000`00000000
fffff96f`30ae2010  00000020`d3abc4b0 00000000`204f33cd
fffff96f`30ae2020  00000000`00000000 00fbf6f1`ece7e2dd
fffff96f`30ae2030  00000000`9b081c40 ffffe800`0042b000
fffff96f`30ae2040  ffffe800`0042b000 00000000`00000021
fffff96f`30ae2050  00000000`25fbd736 00000000`00000000
fffff96f`30ae2060  00000000`00020000 00000000`00000001
fffff96f`30ae2070  00000000`00000000 00000000`00000000
kd> u @rip
hv+0x216837:
fffff96f`30616837 48396b10        cmp     qword ptr [rbx+10h],rbp
fffff96f`3061683b 774b            ja      hv+0x216888 (fffff96f`30616888)
fffff96f`3061683d 0f1f00          nop     dword ptr [rax]
fffff96f`30616840 488b03          mov     rax,qword ptr [rbx]
fffff96f`30616843 48395808        cmp     qword ptr [rax+8],rbx
fffff96f`30616847 0f8515010000    jne     hv+0x216962 (fffff96f`30616962)
fffff96f`3061684d 488b4b08        mov     rcx,qword ptr [rbx+8]
fffff96f`30616851 483919          cmp     qword ptr [rcx],rbx
kd> u @rip - a
hv+0x21682d:
fffff96f`3061682d 488b9f90460100  mov     rbx,qword ptr [rdi+14690h]
fffff96f`30616834 4533ff          xor     r15d,r15d
fffff96f`30616837 48396b10        cmp     qword ptr [rbx+10h],rbp
fffff96f`3061683b 774b            ja      hv+0x216888 (fffff96f`30616888)
fffff96f`3061683d 0f1f00          nop     dword ptr [rax]
fffff96f`30616840 488b03          mov     rax,qword ptr [rbx]
fffff96f`30616843 48395808        cmp     qword ptr [rax+8],rbx
fffff96f`30616847 0f8515010000    jne     hv+0x216962 (fffff96f`30616962)
kd> dq @rdi+14690
fffff96f`30af6690  00000100`00003c40 fffff96f`30af6740
fffff96f`30af66a0  00000000`00000001 00000000`25fb5a58
fffff96f`30af66b0  00000000`25fb7ec2 00000000`25fb7ec2
fffff96f`30af66c0  fffff96f`30af6680 fffff96f`30af6780
fffff96f`30af66d0  ffffffff`ffffffff 00000000`00000000
fffff96f`30af66e0  00000000`00000001 fffff96f`306b5370
fffff96f`30af66f0  00000000`00000000 00000000`00000000
fffff96f`30af6700  fffff96f`30ab5b80 fffff96f`30af6680
kd> rrbx
rbx=0000010000003c40
```

Public service announcement

- Enable VSM, HVCI, Device Guard and Credential Guard
  - it really changes the rules of the game
  - And… you get kCFG for free ☺
- Supposed to be on by default in RS4+ clean installs


Turn On Virtualization Based Security
Turn On Virtualization Based Security
Previous Setting
Next Setting
Not Configured
Enabled
Disabled
Comment:
Supported on:
At least Windows Server, Windows 10
Options:
Help:
Select Platform Security Level:
Secure Boot and DMA Protection
Virtualization Based Protection of Code Integrity:
Not Configured
Require UEFI Memory Attributes Table
Credential Guard Configuration:
Not Configured
Specifies whether Virtualization Based Security is enabled.
Virtualization Based Security uses the Windows Hypervisor to provide support for security services. Virtualization Based Security requires Secure Boot, and can optionally be enabled with the use of DMA Protections. DMA protections require hardware support and will only be enabled on correctly configured devices.
Virtualization Based Protection of Code Integrity
This setting enables virtualization based protection of Kernel Mode Code Integrity. When this is enabled, kernel mode memory protections are enforced and the Code Integrity validation path is


Windows Features
Turn Windows features on or off
To turn a feature on, select its check box. To turn a feature off, clear its check box. A filled box means that only part of the feature is turned on.
.NET Framework 4.7 Advanced Services
Active Directory Lightweight Directory Services
Containers
Data Center Bridging
Device Lockdown
Guarded Host
Hyper-V
Hyper-V Management Tools
Hyper-V GUI Management Tools
Hyper-V Module for Windows PowerShell
Hyper-V Platform
Hyper-V Hypervisor
Hyper-V Services
OK
Cancel


Registry Editor
File Edit View Favorites Help
Computer\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\DeviceGuard
ContentIndex
CrashControl
Cryptography
DebugRegSvc
DeveloperToolsConfig
DeviceClasses
DeviceContainerPropertyUpdateEvents
DeviceContainers
DeviceGuard
Scenarios
Name Type Data
(Default) REG_SZ (value not set)
EnableVirtualizationBasedSecurity REG_DWORD 0x00000001 (1)
Locked REG_DWORD 0x00000001 (1)
RequireMicrosoftSignedBootChain REG_DWORD 0x00000001 (1)
RequirePlatformSecurityFeatures REG_DWORD 0x00000003 (3)

References

- http://www.alex-ionescu.com/syscan2015.pdf
- http://www.alex-ionescu.com/blackhat2015.pdf
- https://www.blackhat.com/docs/us-16/materials/us-16-Wojtczuk-Analysis-Of-The-Attack-Surface-Of-Windows-10-Virtualization-Based-Security-wp.pdf
- http://alex-ionescu.com/Publications/OPCDE/octagon.pdf
- https://bugs.chromium.org/p/project-zero/issues/detail?id=688

close()

- Thanks to the great folks @BluehatIL
- Follow me on twitter for lowlevel internals
  - @AmarSaar