Attacking Hyper-V

Jaanus Kääp

Clarified Security

Who, from, what, why?

- Who: Jaanus Kääp
- From: Clarified Security, Estonia
- What: Vuln researcher & developer
- Why: Really like to talk @ POC
  Like to talk about security
  MSRC top list 4 last years

Why Hyper-V

- Difficult & interesting
- Currently my MSc thesis
- Not enough tools & info yet
  - MS has released more lately
    - Still no tools from them....
- Nice bounties also

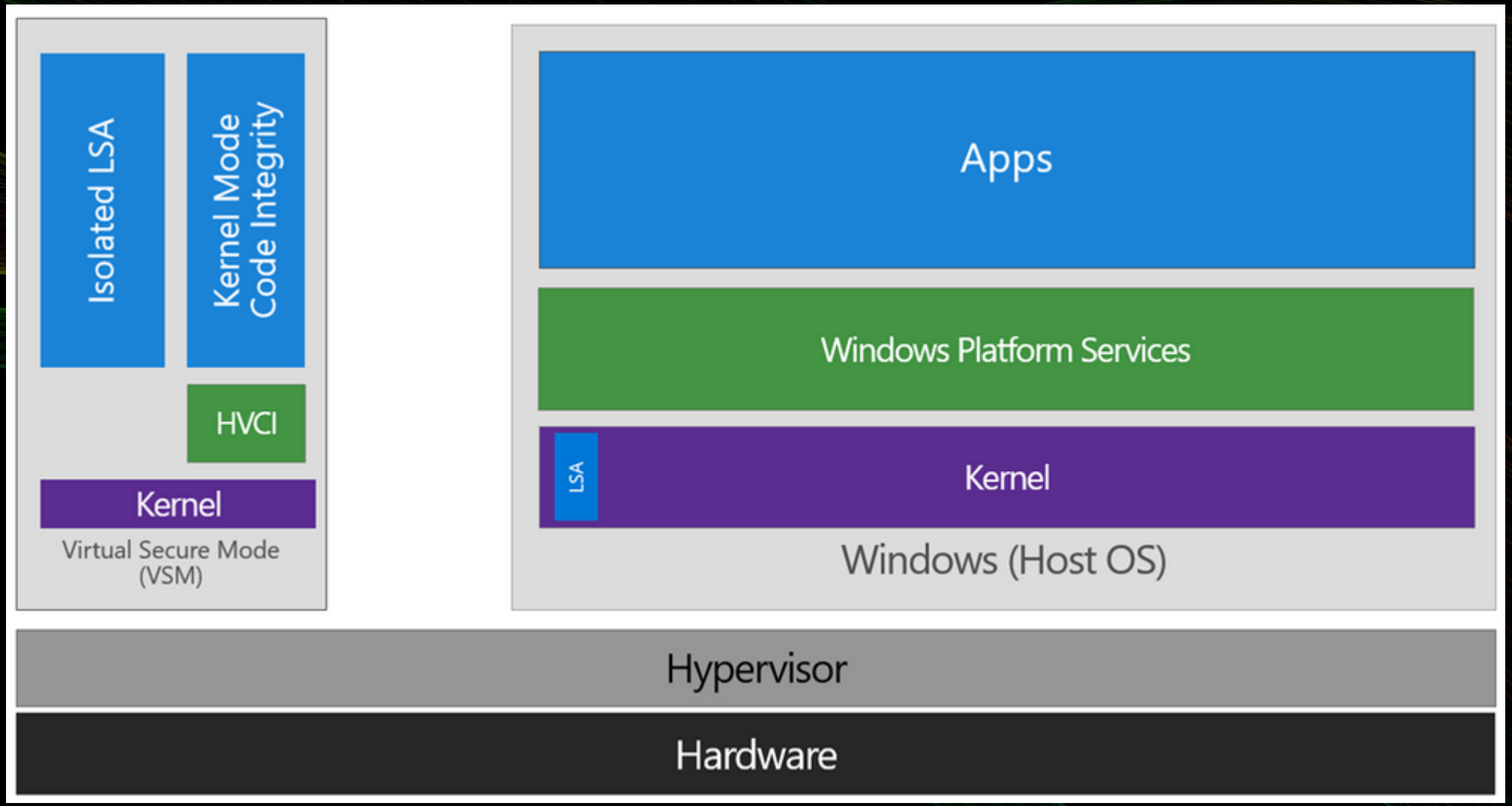
Back to basics & terms

- Hyper-V is type 1 hypervisor
- Partitions
- Host OS runs as VM (Root partition) also
  - Additional security via secured kernel (VBS)
- SLAT
- IOMMU

VBS & VTL

- Virtualization Based Security
- Virtual Trust Levels
- Created via SLAT and Hypervisor
- Used for kernel exploits mitigation
- Partition still same
- Rings still same
- VTL1 > VTL0

VBS & VTL

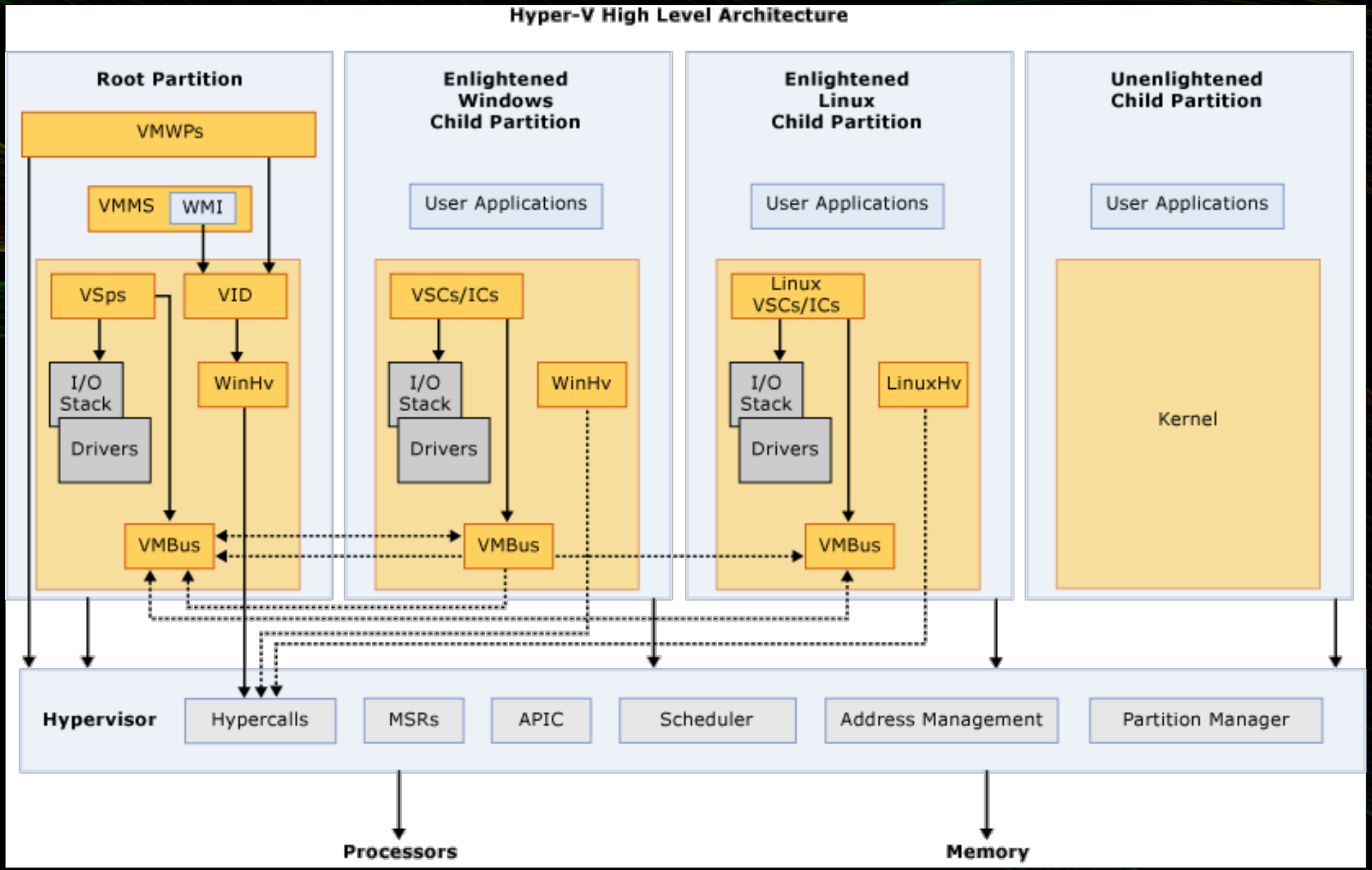


https://blogs.technet.microsoft.com/ash/2016/03/02/windows-10-device-guard-and-credential-guard-demystified/

VMs and terms

- Partition = VM
- VMWP = Host useland, per VM, emulation
- VSP = Host driver, some emulation
- VMBUS = One bus to bind them all

VMs



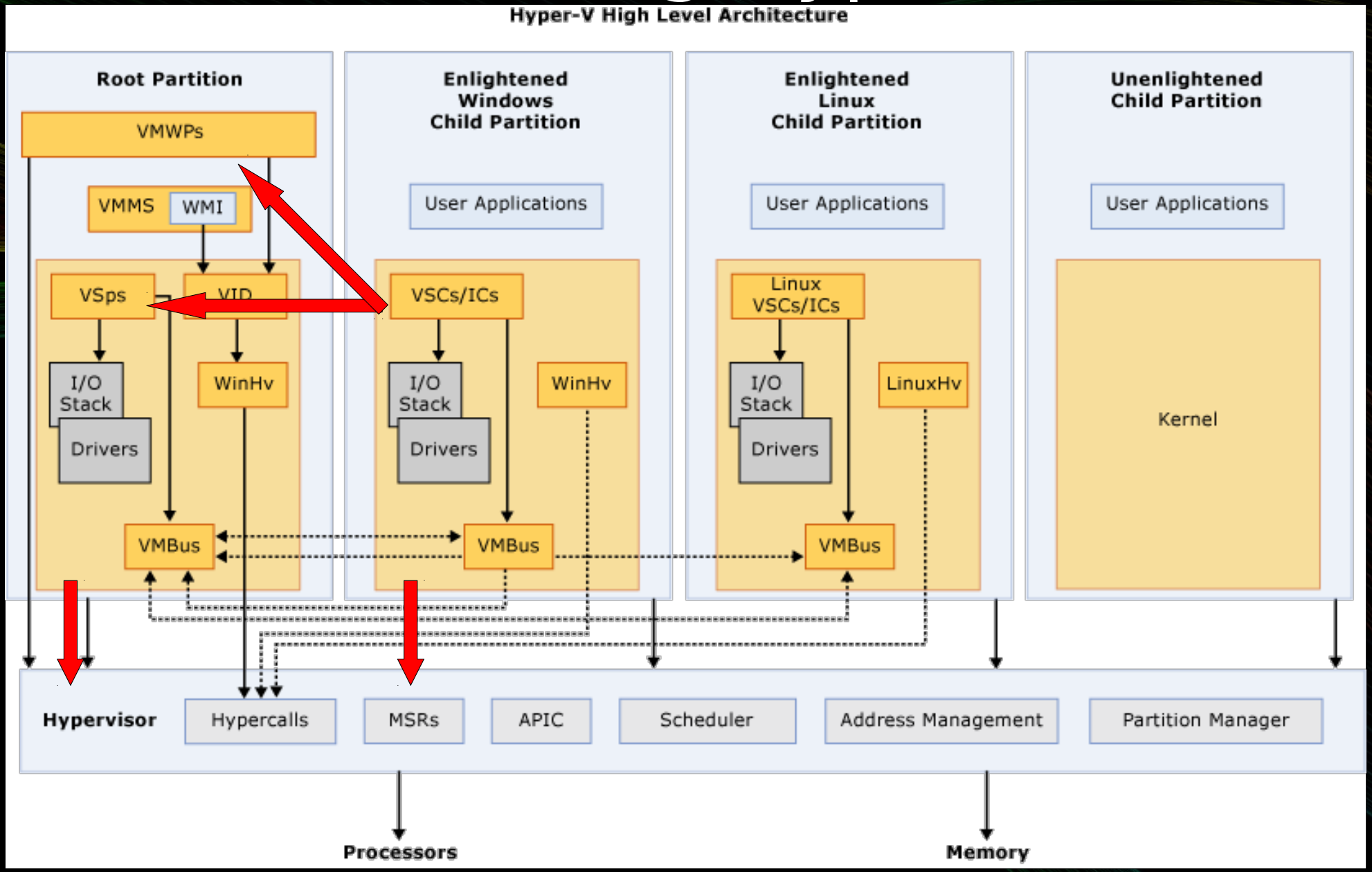
https://docs.microsoft.com/en-us/virtualization/hyper-v-on-windows/reference/hyper-v-architecture

Attacking Hyper-V

- Attack directions
  - Host to hypervisor
  - Guest to hypervisor
  - Guest to host kernel
  - Guest to host userland

  - Host to host secure kernel*

Attacking Hyper-v



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

GUEST/HOST → HYPERVISOR

Communication with Hyper-V

- Hypercalls
- MSRs

Hypercalls

- Could think as syscalls++
- Give over 3 values (unless extended)
- Input 1 (RCX)
  - Call code : 16 bit
  - Is fast: 1 bit
  - Count of elements: 12 bit
  - Start Index: 12 bit

Hypercalls

- Could think as syscalls++
- Give over 3 values (unless extended)
- Input 1 (RCX)

```
struct
{
    UINT32 CallCode : 16;
    UINT32 IsFast : 1;
    UINT32 dontCare1 : 15;
    UINT32 CountOfElements : 12;
    UINT32 dontCare2 : 4;
    UINT32 RepStartIndex : 12;
    UINT32 dontCare3 : 4;
};
```

Hypercalls

- Returned (RAX)

```
struct
{
    HV_STATUS CallStatus;
    UINT16 dontCare1;
    UINT32 ElementsProcessed : 12;
    UINT32 dontCare2 : 20;
};
```

Hypercalls – fast

- Input 2 (RDX) – integer input simply
- Input 3 (R8) – integer input simply

Hypercalls – slow

- Input 2 (RDX) – Physical mem addr for input buffer
- Input 3 (R8) – Physical mem addr for output buffer

Hypercalls – fast extended*

- Input 2 (RDX)– integer input simply
- Input 3 (R8)– integer input simply
- Some more registers
  - XMM0-XMM5

Hypercalls in kernel

- Op VMCALL/VMMCALL
- Nobody calls directly
- nt!HvcallCodeVa → executable memory

```
1: kd> u poi(nt!HvcallCodeVa)
fffff803`60840000 0f01c1          vmcall
fffff803`60840003 c3              ret
```

- NB:
  - ~~*bp poi(nt!HvcallCodeVa)*~~
  - *ba e 1 poi(nt!HvcallCodeVa)*

Making Hypercalls

- Kernel exports HvlInvokeHypercall
- Can use it to make hypercalls easily
- Input/Output buffers >1 page
  - Buffers physical pages contiguous
  - MmAllocatePartitionNodePagesForMdlEx

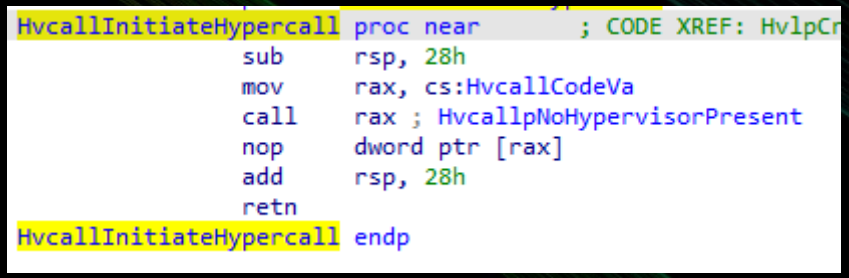
- Alex Ionescu blog is recommended

Recording hypercalls

- Simple with debugger
  - DAMN slow – lot of hypercalls
- My solution – execution redirection
  - My driver
  - Overwrite nt!HvcallCodeVa
  - Do stuff
  - Jmp to original destination

Recording hypercalls

- But where is nt!HvcallCodeVa
  - Not exported by the kernel
  - Referenced by MANY functions
  - HvlInvokeHypercall – exported and useful



Recording hypercalls

- Where to record
  - I picked filesystem
  - IRQL sometimes not PASSIVE_LEVEL
    - Can’t write right away!
  - Work queue works well
    - Some timeloss from copying buffers

Recording hypercalls

- Additional issues
  - Extra logic from fast, slow, fast extended
    - That’s life
  - With slow – no idea of input size
    - I just record 1 page
      - OVERKILL!
  - Some hypercalls happen constantly
    - Skip them early in the code

My recording logic

- Driver has hook function in asm
  - Main logic in separate function (C++)
- Locates nt!HvcallCodeVa via HvlInvokeHypercall
- Overwrites nt!HvcallCodeVa

Hypercalls fuzzing

- Fuzzing random/manual inputs
- Mutating recorded hypercalls
- Mutating on the fly
  - Will break things
  - Found CVE-2019-0695 like this

Hypercalls in Hypervisor

- Handler pointer of each hypercall
  - hv+0xC000000+code*0x18

MSRs

- Intercepted by Hypervisor
- Some results are faked/emulated & others relayed
- Handlers easy to find
  - huge switch statement
  - Values like 0x40000000, 0x40000001, 0x40000002, 0x40000003, 0x40000004, 0x4000000D

MSR read/write

- Easy to intercept with debugger
- Lot of traffic
- Tricky to inject code more legit way
- Easy to inject via debuger
- Text segment has lot of space at the end

My MSR recording

- Windbg extension
- Short asm to hv+0x330000 for filtering MSRs
- Insert jump to read/write handler
- Data relayed via debugger

Fuzzing MSRs

- Trying through reads/writes
- Going through CPU manuals
- Possible special cases

GUEST → HOST KERNEL

VMBUS

- Communication between partitions
- Used and managed via Hyper-V
- Based on ringbuffer in shared memory
- Organized to channels
- Special case: pipes

VMBUS channels

- Data between guest-host via VMBUS
- GUID based identification
- Callback based
- Data via VMBUS and GPADL
- GPADL is MDL between partitions
- GPADL → MDL (seen by most)

VMBUS channels

- vmbkmclr.sys/vmbkmcl.sys
- Can send simple buffer
- Can send MDL as external data
  - Not copied over, so guest can modify
- Handler is executed

VMBUS channels

- VMBCHANNEL – channel object
- Structure not public
- First channel: vmbkmclr!KmclChannelList
- Some important offset:
  - 0x64C VM ID
  - 0x700 Process packet callback
  - 0x760 Next channel
  - 0x960 GUID

VMBUS channels request

- vmbuskernelmodeclientlibapi.h
- Existing connection found via linked list
- Sending request straightforward
  - VmbPacketAllocate
  - VmbPacketSend

VMBUS channels request

- VmbPacketSend is asynchronous
- VMBPACKET returned by VmbPacketAllocate
- VmbPacketSetCompletionRoutine to add completion routine
- Or simply pointer @ offset 0x64

VMBUS channel recording

- vmbkmclr!KmclChannelList not exported
- Referenced by vmbkmclr!DllInitialize

```
INIT:00000001C001A12F                lea     rax, KmclChannelList
```

- First ”lea rax, ???”
- Stepping though linked list to find all channels

VMBUS channel recording

- Overwrite channel Process packet callback
- Recording logic same as in hypercalls
  - Worker threads
  - Timeloss from copying buffers
- Redirect to correct handler

VMBUS channels fuzzing

- Sending random
- Sending modified recordings
  - Sooner or later crashes guest kernel
- Modifying traffic on the fly
  - Crashes guest kernel quite fast

GUEST → HOST VMWP

VMBUS pipes

- All pipes are channels
  - All channels are not pipes :)
- Pipes have no packet process handlers
- Handled by vmbusr.sys/vmbus.sys
- NtReadFile/ntWriteFile from userspace
- No MDLs – only usual buffer

VMBUS pipes reading path

- NtReadFile
  → ...
  → vmbusr!PipeRead
  → vmbusr!PipeTryRead
  → ...
  → vmbusr!PipeTryReadSingle
- Uses vmbusr!PkGetReceiveBuffer for shared buffer

VMBUS pipes recording

- Can use vmbusr!PipeTryRead
  - RCX == pipe object (ptr from channel)
  - RDX == IRP of the NtReadFile
- But don’t know the result yet
- Most reads return 0 bytes

VMBUS pipes recording

- Ending of the vmbusr!PipeTryRead

```
loc_1C0001CC2:          ; End of the PipeTryRead
mov     rbx, [rsp+38h+arg_0]
mov     rsi, [rsp+38h+arg_8]
add     rsp, 30h
pop     rdi
retn
PipeTryRead endp
```

- At the start of that block
  - RSI == irp
  - RBX == pipe object (ptr from channel)

VMBUS pipes recording

- Hook end of the vmbusr!PipeTryRead
- Irp→IoStatus.Information == read length
  - Often zero, so nothing to record
- Channel found via pipe object
  - Channel pointer @ pipe object + 0x100

- Problem: PipeTryRead not exported
  - Search based on signature
    - not good solution: updates might break

VMBUS pipes recording

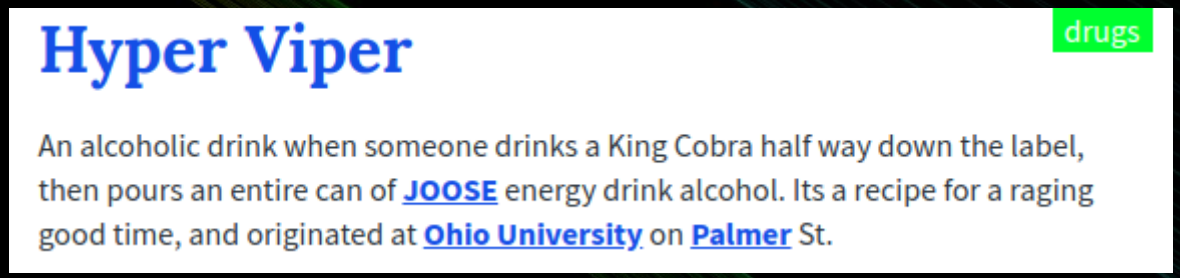
- Hook end of the vmbusr!PipeTryRead
- Filter 0 reads
- Recording logic same as in hypercalls
  - Worker threads
  - Timeloss from copying buffers
- Return as in original vmbusr!PipeTryRead

VMBUS pipes fuzzing

- Sending random
- Sending modified recordings
  - Sooner or later crashes guest/-service
- Modifying traffic on the fly
  - Crashes guest/-service quite fast

HyperV*iper*

- My new tool/toolset
- First named - googled later
- **Of course** exists in urban dictionary



HyperV*iper*

- My new tool/toolset
  - Driver
  - Userland tools for driver
  - DLL for other tools
  - WinDbg extension*
  - Python library*

HyperV*iper*

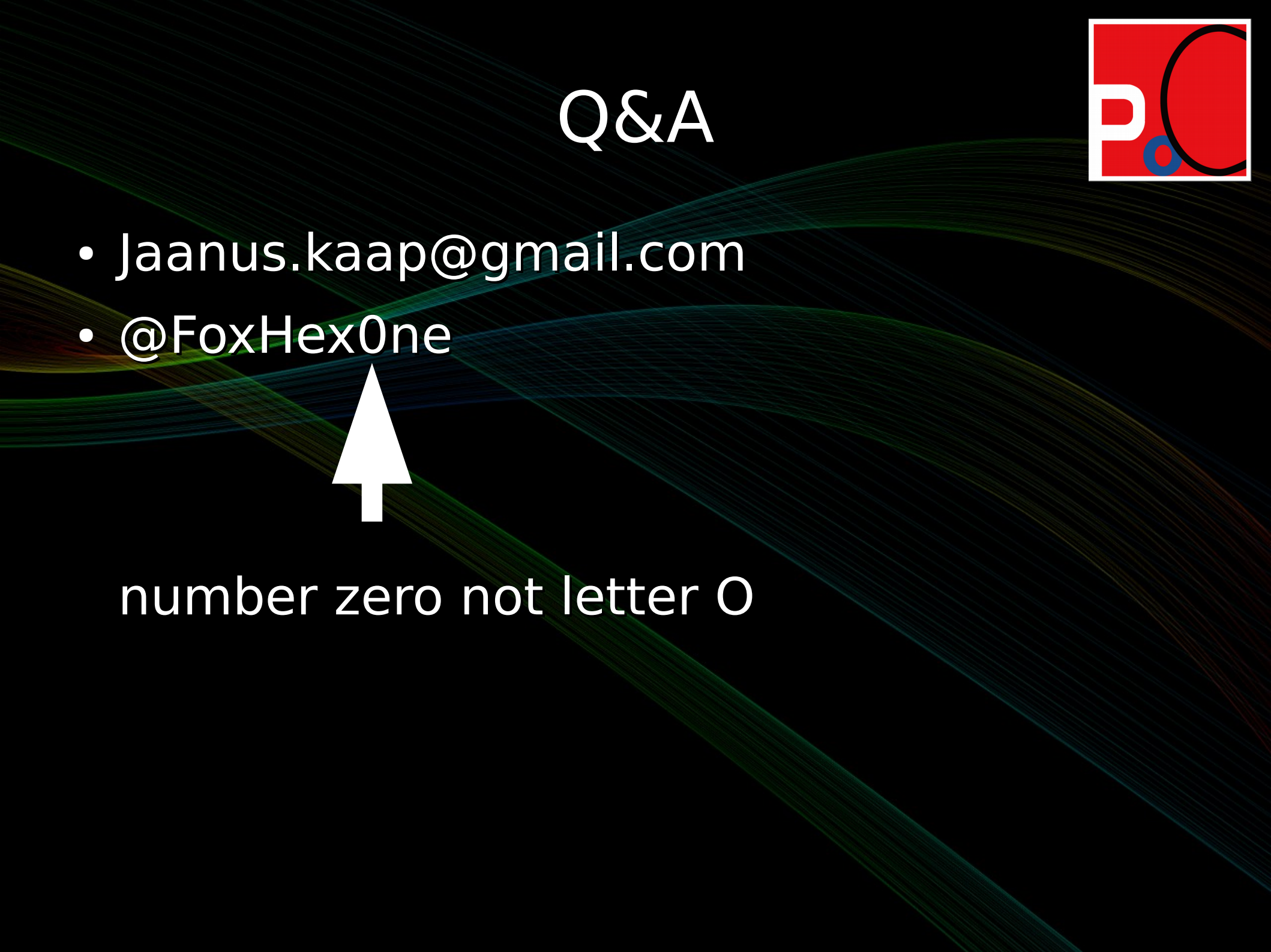
- Make request
- Record request
- Fuzzing from kernel
- Listing channels
- Debugger extension for reversing help
- Developed to be built on ;)
- ....

HyperV*iper*

- https://github.com/FoxHex0ne/HyperViper

number zero not letter O

Q&A



- Jaanus.kaap@gmail.com
- @FoxHex0ne

number zero not letter O