

BLUEHAT

IL 2022


Hyntrospect: a Fuzzer for Hyper-V Devices



Diane Dubois

didu@google.com -  @0xdidu

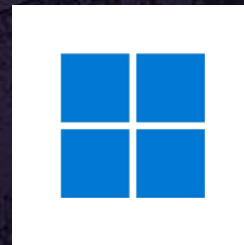
Whoami

- Security Engineer at **Google**
- Research project done as a 20% with Project Zero
- Passionate about vulnerability research on systems
- Active member of Women in Security networks
-  @0xdidu

Why Hyper-V?



- Project Zero mission
- Running on



- ... and because virtualization is a fun topic
 - Multiple layers from hardware to high level software
 - Complex implementations

Goals

- Instrumenting Hyper-V for vulnerability research
 - Fuzzer called Hyntrospect:
<https://github.com/googleprojectzero/Hyntrospect>
- Finding vulnerabilities and reporting them to Microsoft

Agenda

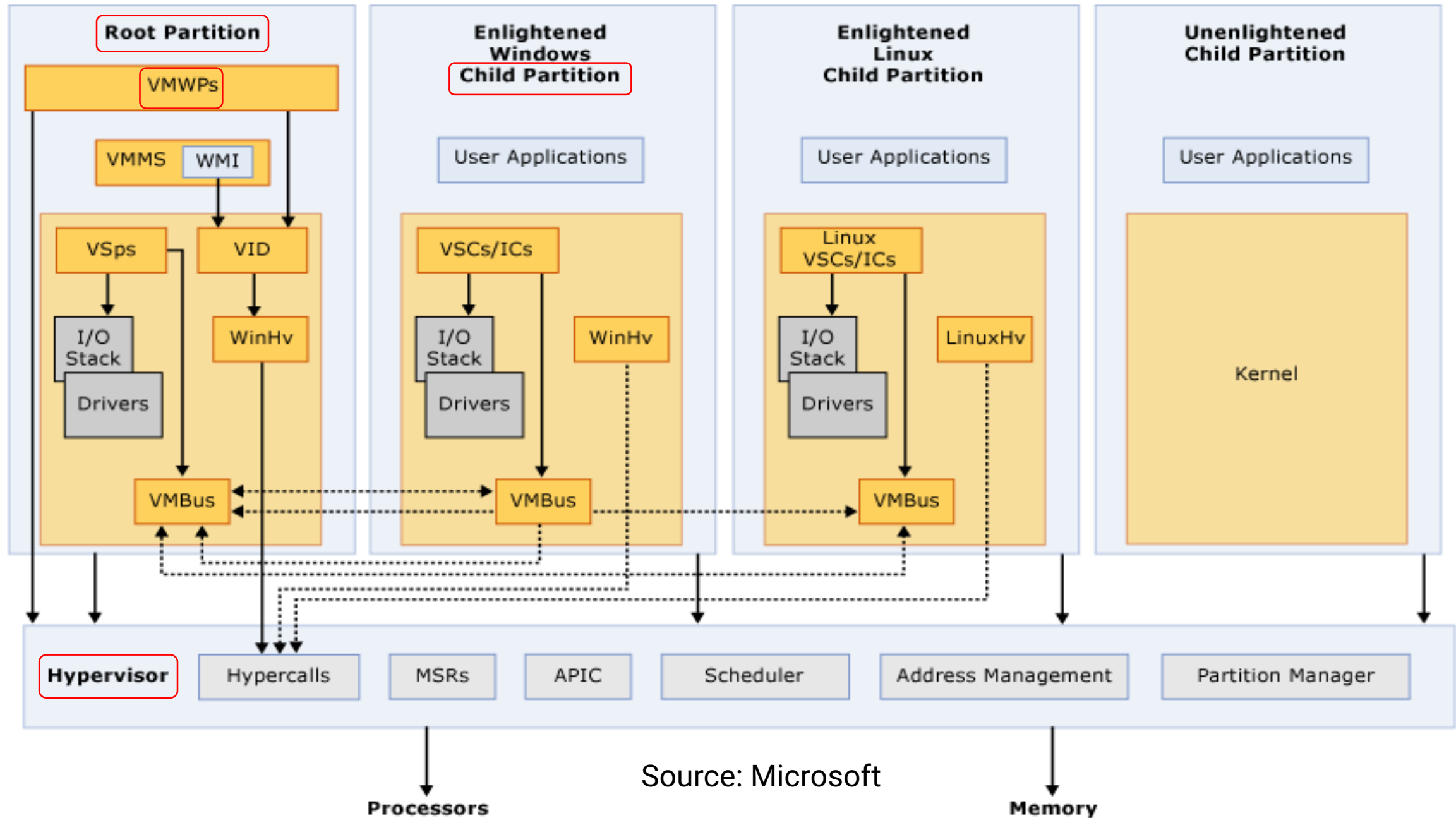
- Background on Hyper-V
- The Research Target
- Hyntrospect Fuzzer
- Current Results
- Future Endeavors



Background on Hyper-V

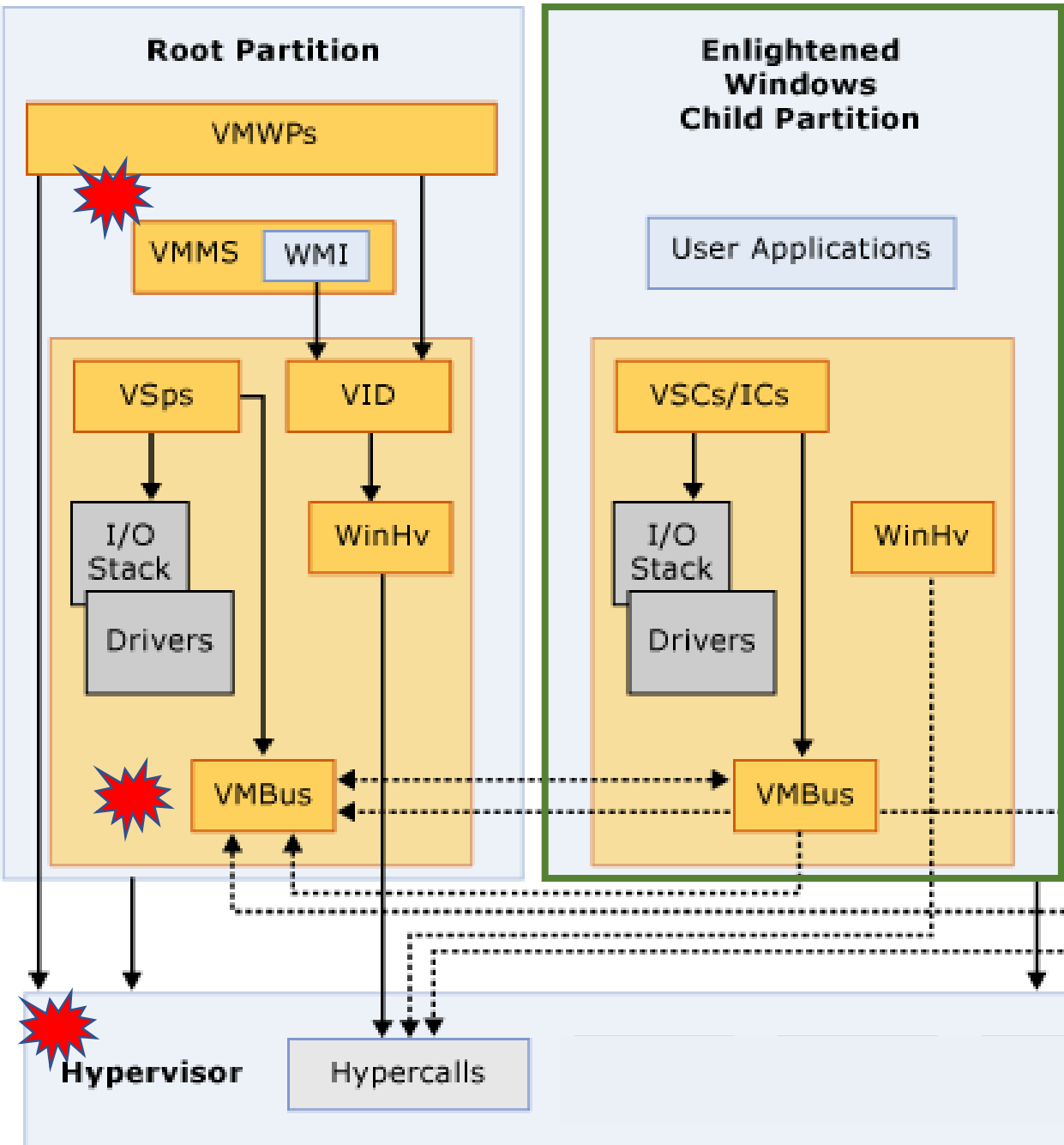


Hyper-V High Level Architecture

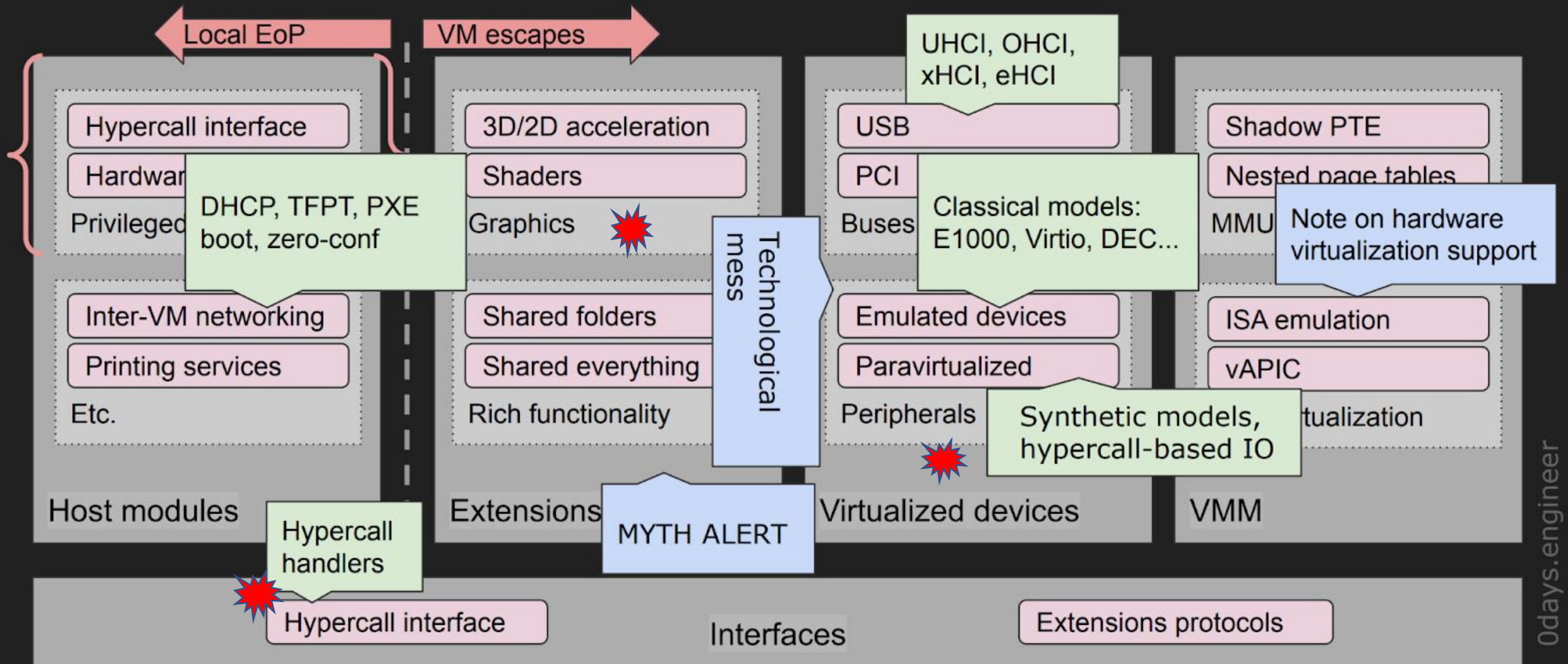


What is a Guest to Host Escape?

- Gaining **code execution** on one of the **hypervisor layers** from a **virtual machine**
- On Hyper-V: several layers
- Also Host denial of service, leaks ...

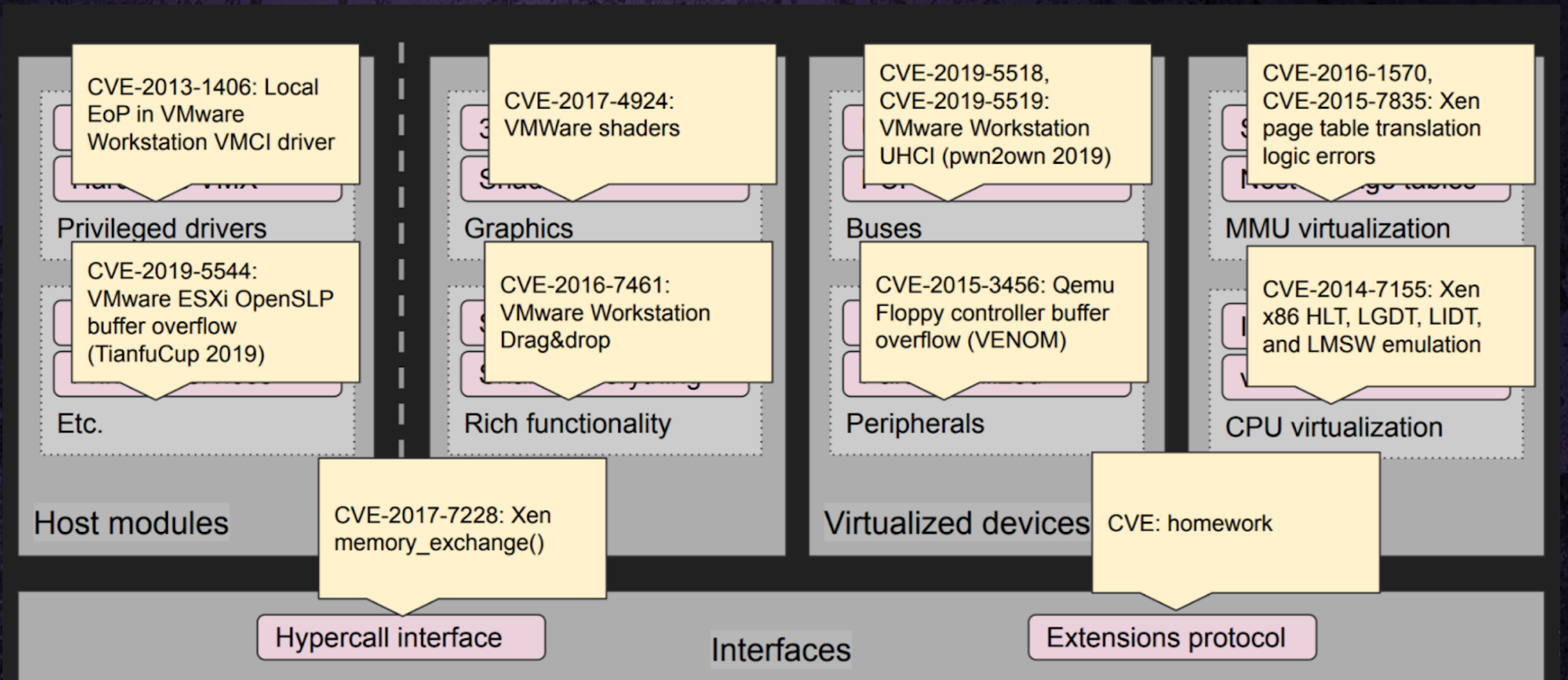


The Hypervisors' Attack Surface



As defined by Alisa Esage (Zer0Con 2020)

...and in practice



Hyper-V Attack Surface

Hypervisor

- Hypercall handlers
- Faults
- Instruction emulation
- Intercepts
- Register access (MSRs...)

Root partition

- Kernel
 - VMBus
 - Drivers (storage...)
- Userland
 - Emulated devices
 - Integration components

... and this list is not exhaustive

MSRC: [first-steps-in-hyper-v-research](#)

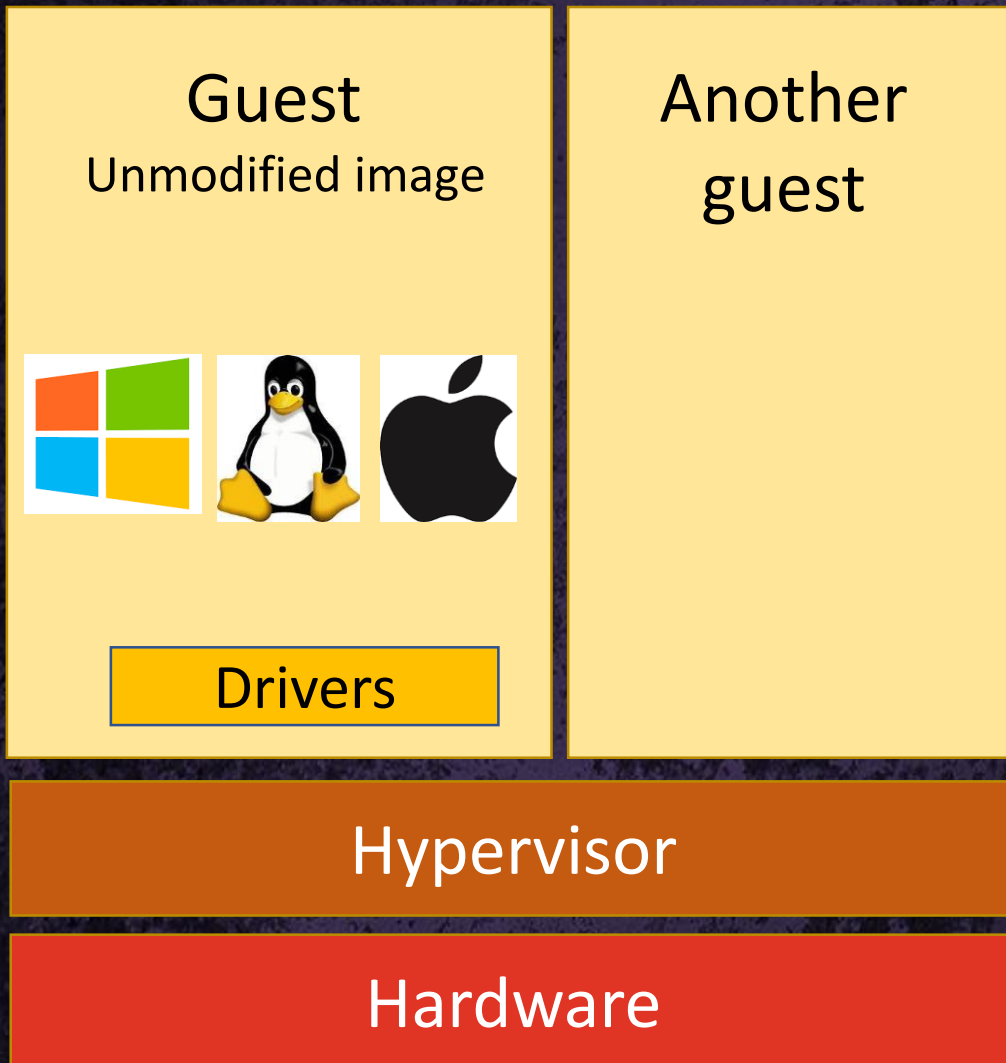
Research - State of the Art

- **MSRC and Microsoft** publish on Hyper-V
 - Security blog posts: e.g. First Steps in Hyper-V research
 - Posts on Hyper-V components
 - Talks on vulnerabilities found internally
e.g. Breaking VSM by Attacking SecureKernel at BlackHatUSA 2020
 - Symbols for some key components
- Active **external contributors**: @gerhart_x, @alisaesage, @OphirHarpaz, @peleghd...
And many more (GitHub/gerhart01/Hyper-V-Internals)

The Research Target

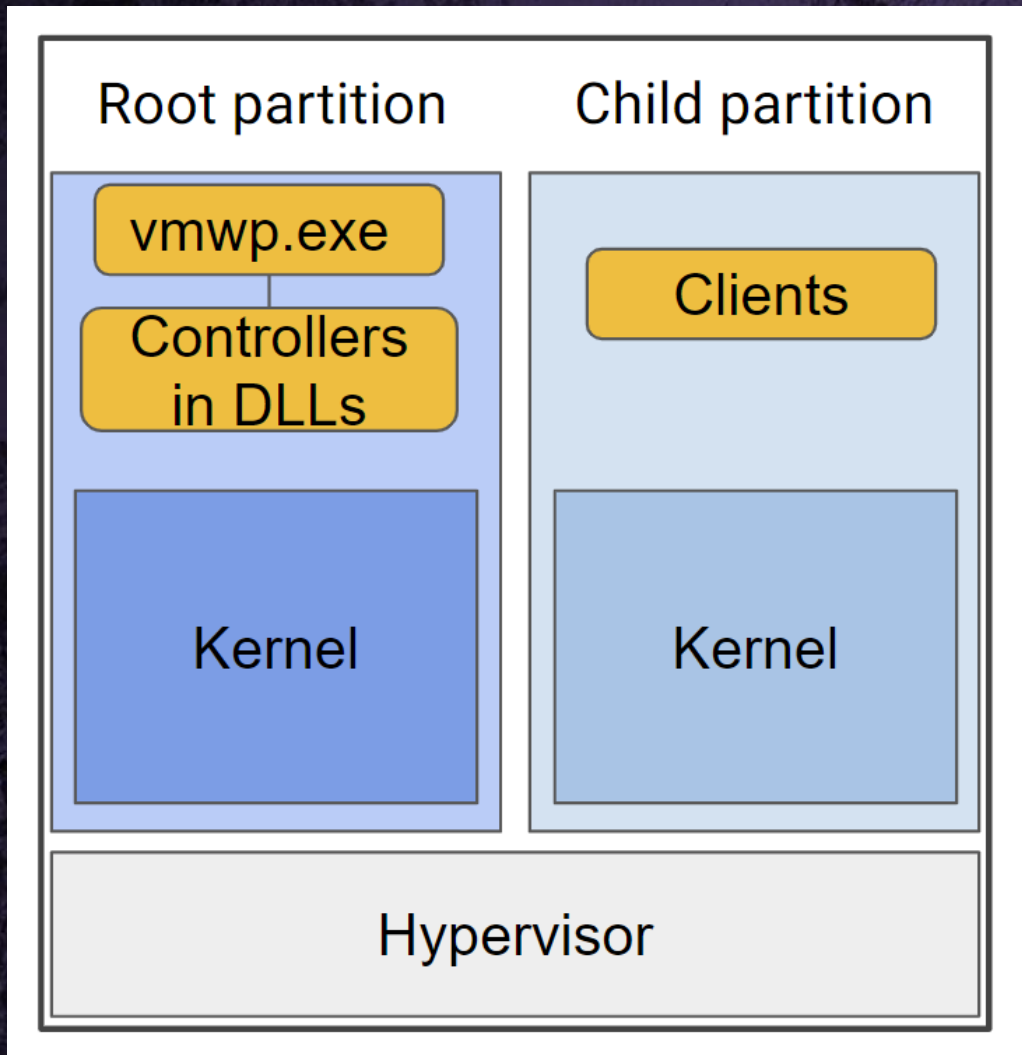


The Emulated Devices Controllers

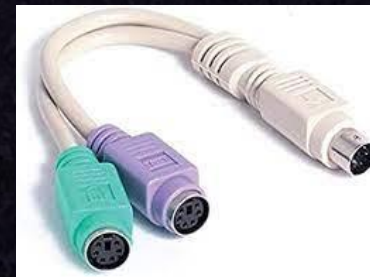


- Access by the VM to the hardware
- Emulation of hardware controller by the hypervisor
 - Hypervisor as a proxy
 - Guest operating systems unmodified
- Called “virtual devices” or “VDEVs” at Microsoft

...on Hyper-V



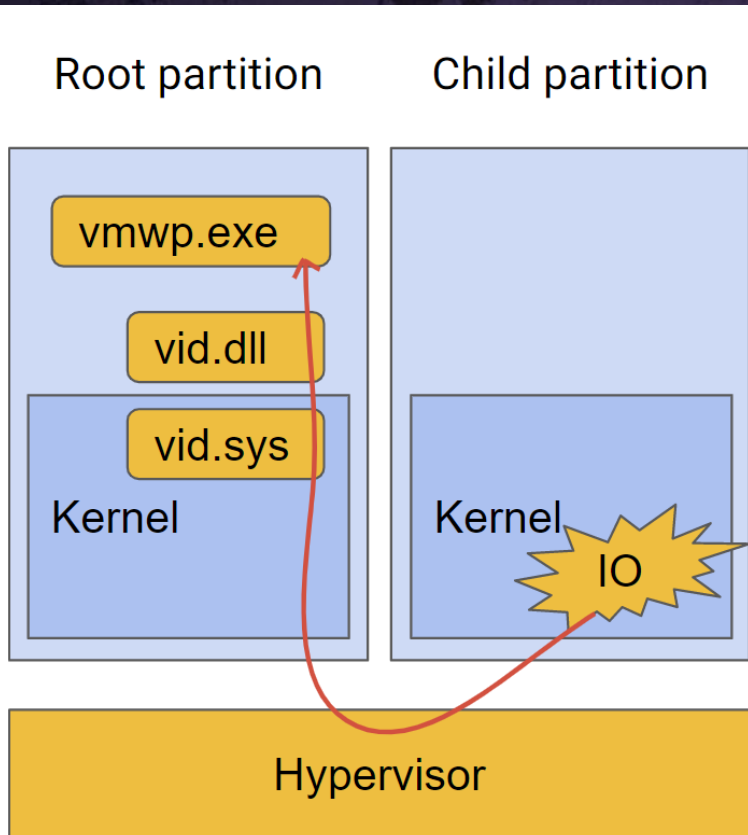
- Generation 1 VMs
 - Azure mostly uses this generation
- Userland of the root partition
- DLLs loaded by the worker process



Why Choose the Emulated Devices

- Complex (state machines)
 - For example: enable / disable ports, update a status register, wait for a command
- Several bugs on several hypervisors
- Azure mostly uses Generation 1 VMs
- Hyper-V is developed in C++
- Potential “guest to root partition” escapes

Life of a Request

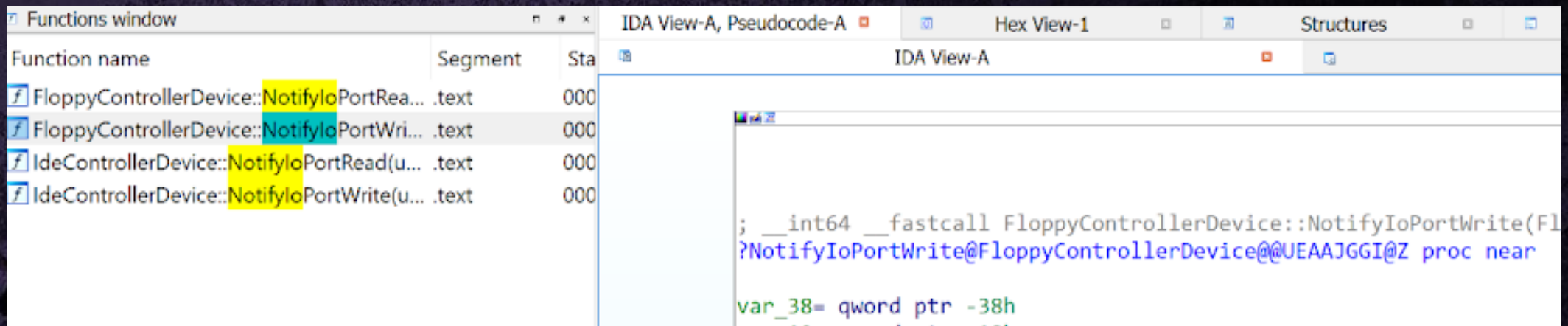


- Communication through IO ports
 - CPU instructions: IN / OUT
 - “IN EAX, DX”
 - “OUT DX, EAX”
- Communication through the hypervisor, the VID, and callbacks
- Some VDEVs are more complex
 - MMIO handling
 - Use of the VMBus
- More on MSRC blogpost “Attacking the VM worker process”

Some Reverse Engineering

- DLL implementing the controllers
- Typical IO handlers
 - \$Device::NotifyIoPortRead
 - \$Device::NotifyIoPortWrite

Example: VmEmulatedStorage.dll



+

Symbols

No particular difficulty (obfuscation...)

-

C++, indirect calls

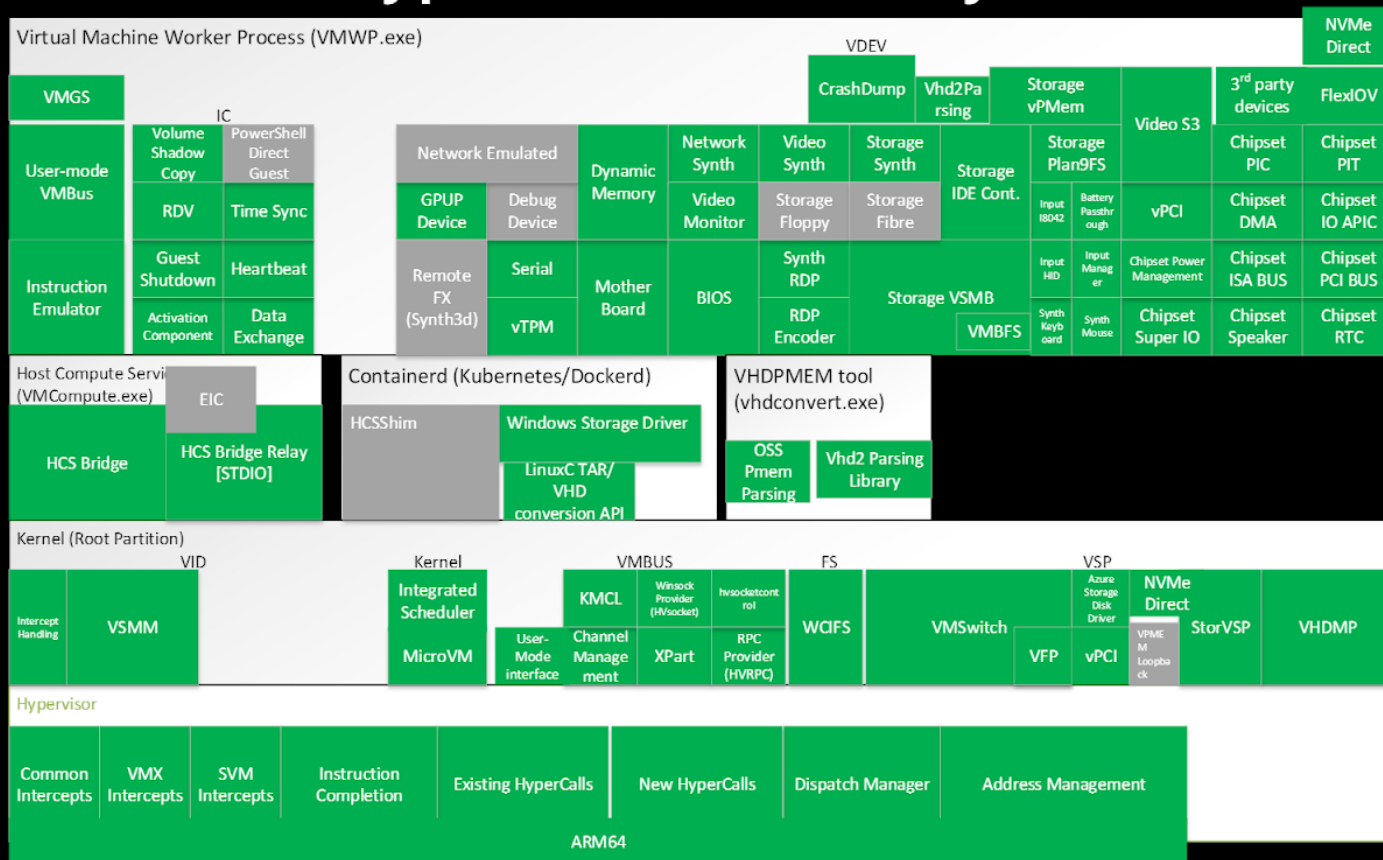
Hyntrospect: A Fuzzer for the Emulated Devices



<https://github.com/googleprojectzero/Hyntrospect>

Inspiration

Hyper-V: a case study



- libFuzzer: Coverage-guided approach
- Microsoft publication on their coverage (Keeping Windows Secure - Bluehat IL 2019)
- CVE-2018-0959 + Dedicated MSRC blogpost



How to do the same, closed source?

Existing Tools for Windows Binaries Fuzzing

Gathering Coverage	Fuzzers	Memory Corruption Detection
<ul style="list-style-type: none">• DynamoRIO• Intel Pin• Intel PT• Mesos• QDBI for Windows• TinyInst	<ul style="list-style-type: none">• WinAFL + DynamoRIO• Jackalope• whvp	PageHeap

So Why Another Tool?

- The target is a **DLL**
- **Emulating** only the relevant functions is hard
- vmwp binary and the DLL cannot be **restarted with instrumentation**
- The runtime operations are specific: **IOs injections**
- Some tools were developed during Hyntrospect development
- Managing all the blocks with a **minimal set of languages** is hard
- The fuzzer will be **ported** to similar use cases

Scope

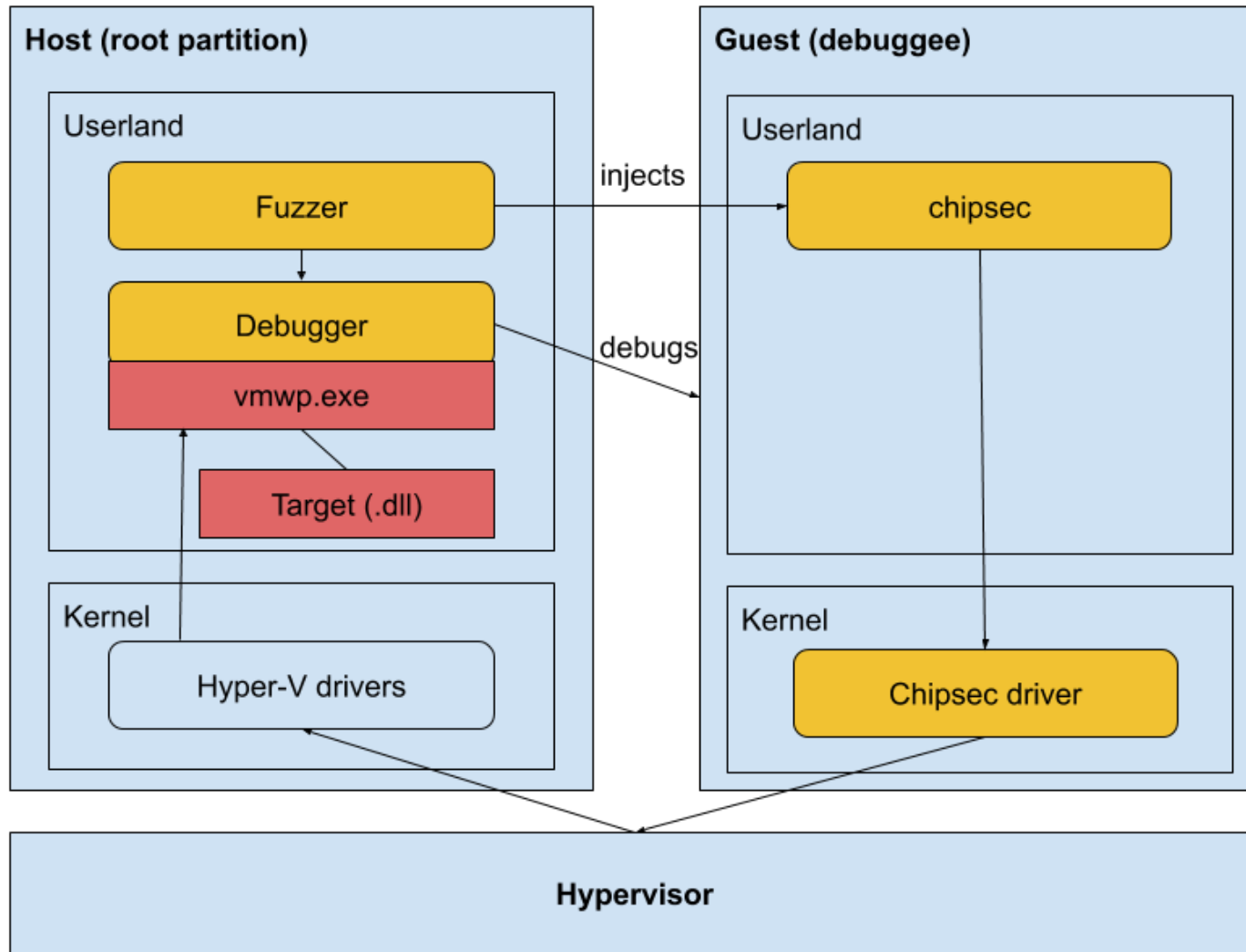
- Windows guest VM
- Intel CPU
- Generation 1 VMs
- Binaries (DLLs/EXEs) in the userland of the root partition

Design Choices at a Glance

Emulation vs execution	Execution of a VM through a debugger (DbgShell) at runtime
Coverage	Tracked with the int3 technique described by @5aelo for TrapFuzz / @gamozolabs mesos
Memory corruption detection	Pageheap (gflags)
Type of bugs	Memory corruption State machine logic errors [Use after free] Race conditions

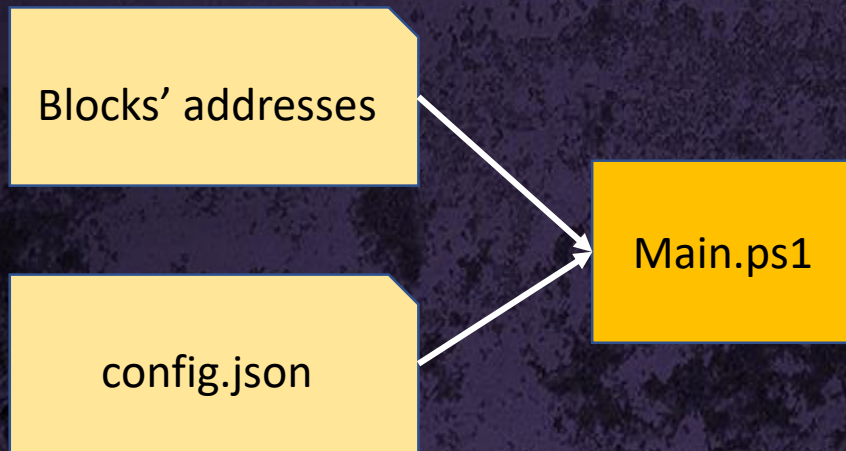
Design Choices at a Glance

Environment reset	Hyper-V checkpoints = snapshots
Mutation strategy	Custom
Language	PowerShell (except for the IDA scripts)
External dependencies	DbgShell , CHIPSEC , [pageheap], [LightHouse], [IDA]

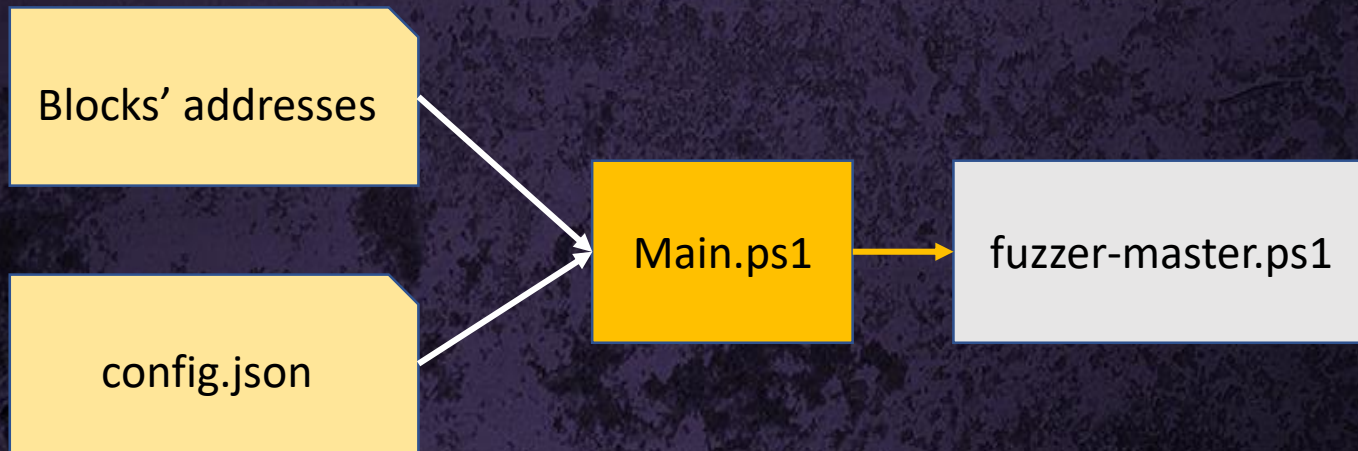


Overview of Hyntrospect

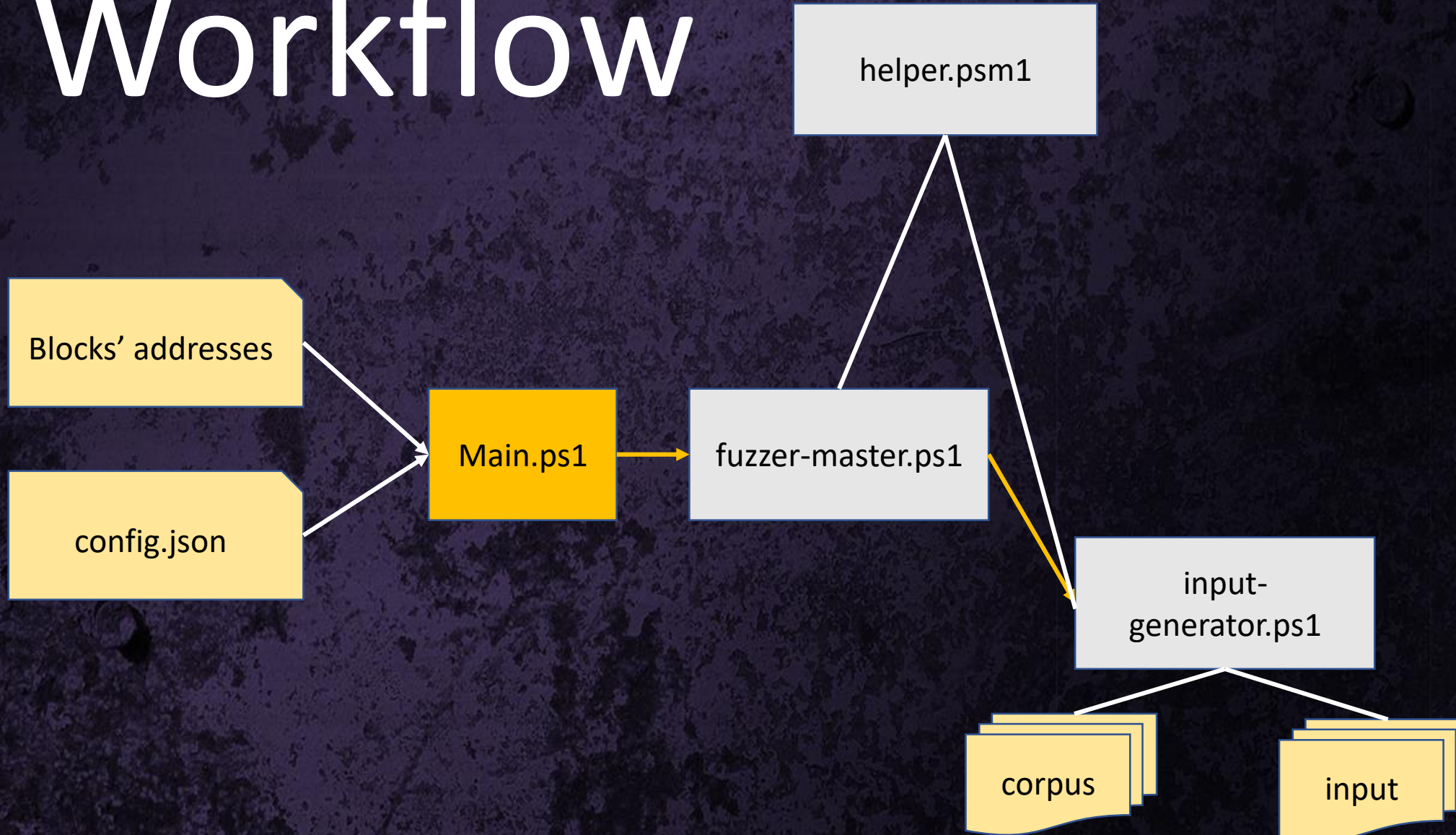
Workflow



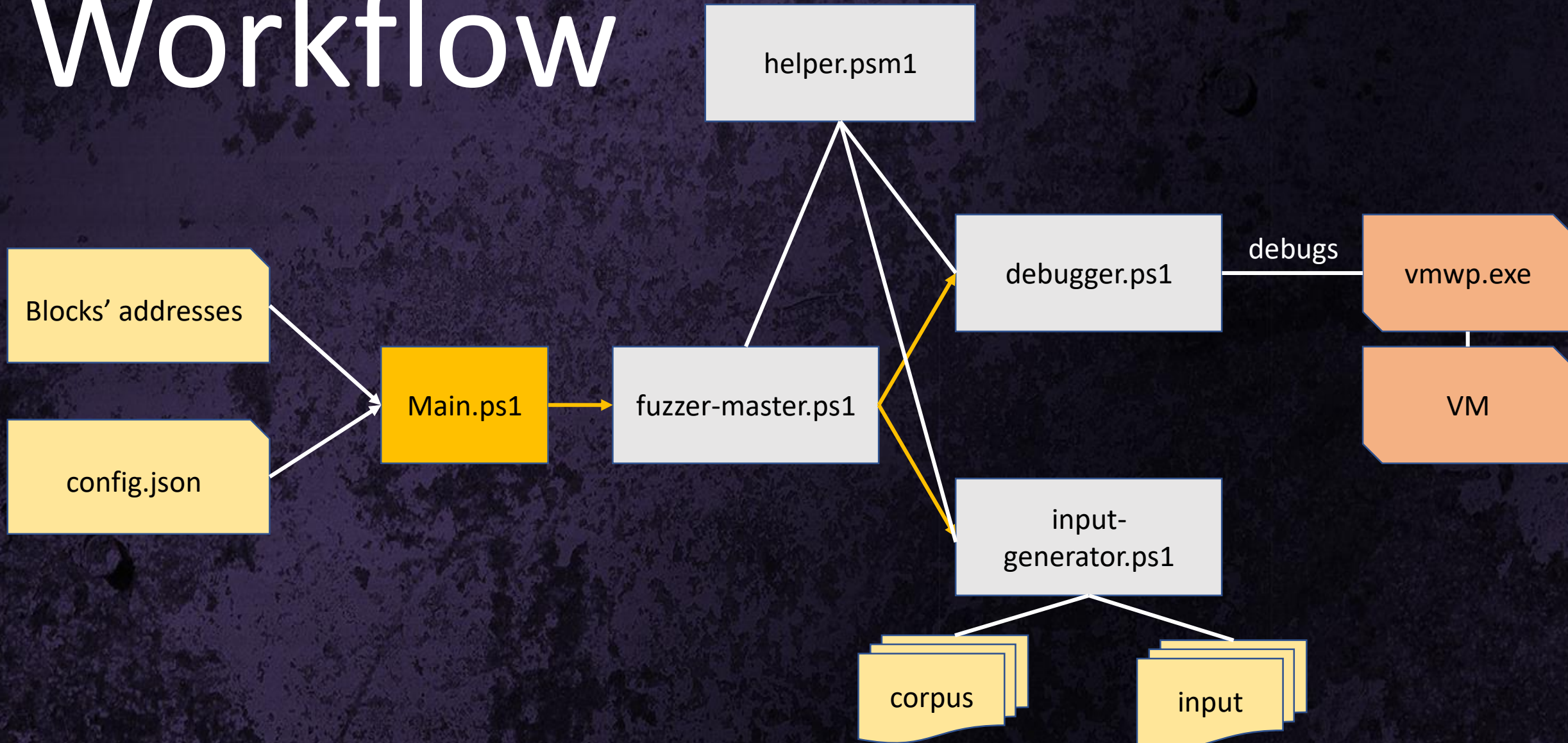
Workflow



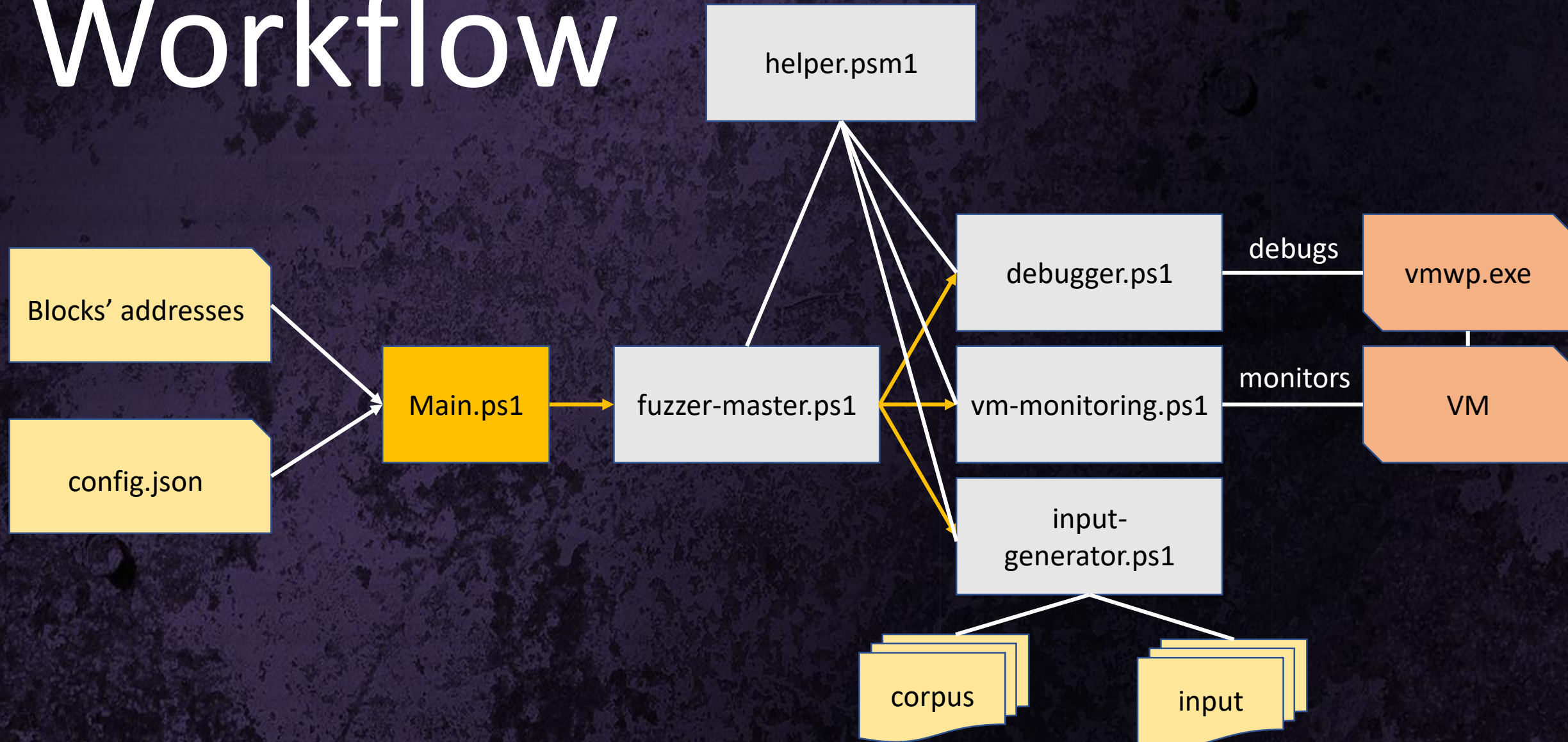
Workflow



Workflow

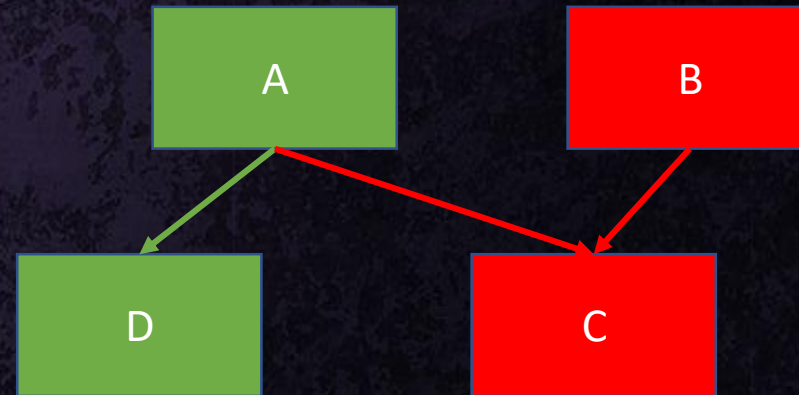


Workflow



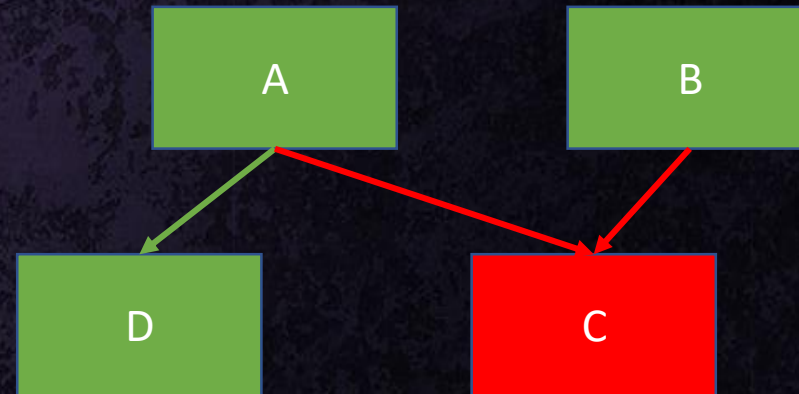
Coverage collection and guidance

- Block coverage



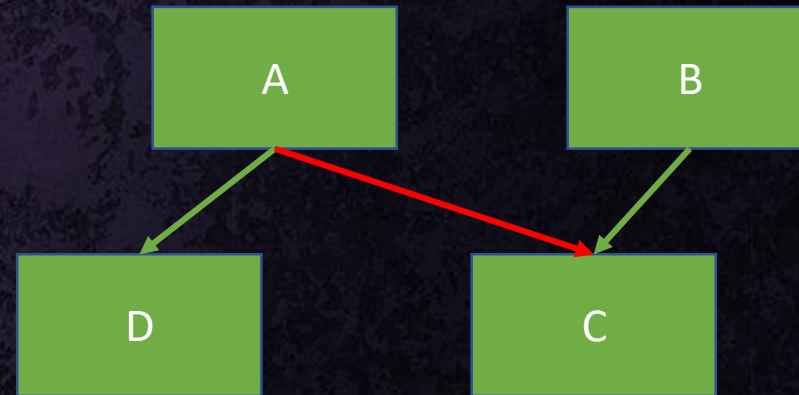
Coverage collection and guidance

- Block coverage



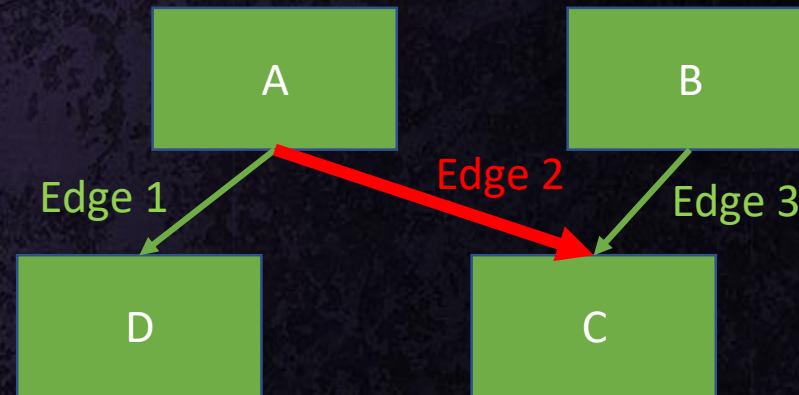
Coverage collection and guidance

- Block coverage



Coverage collection and guidance

- Block coverage
 - Versus edge coverage: easier to implement but does not promote rare paths
 - No counter



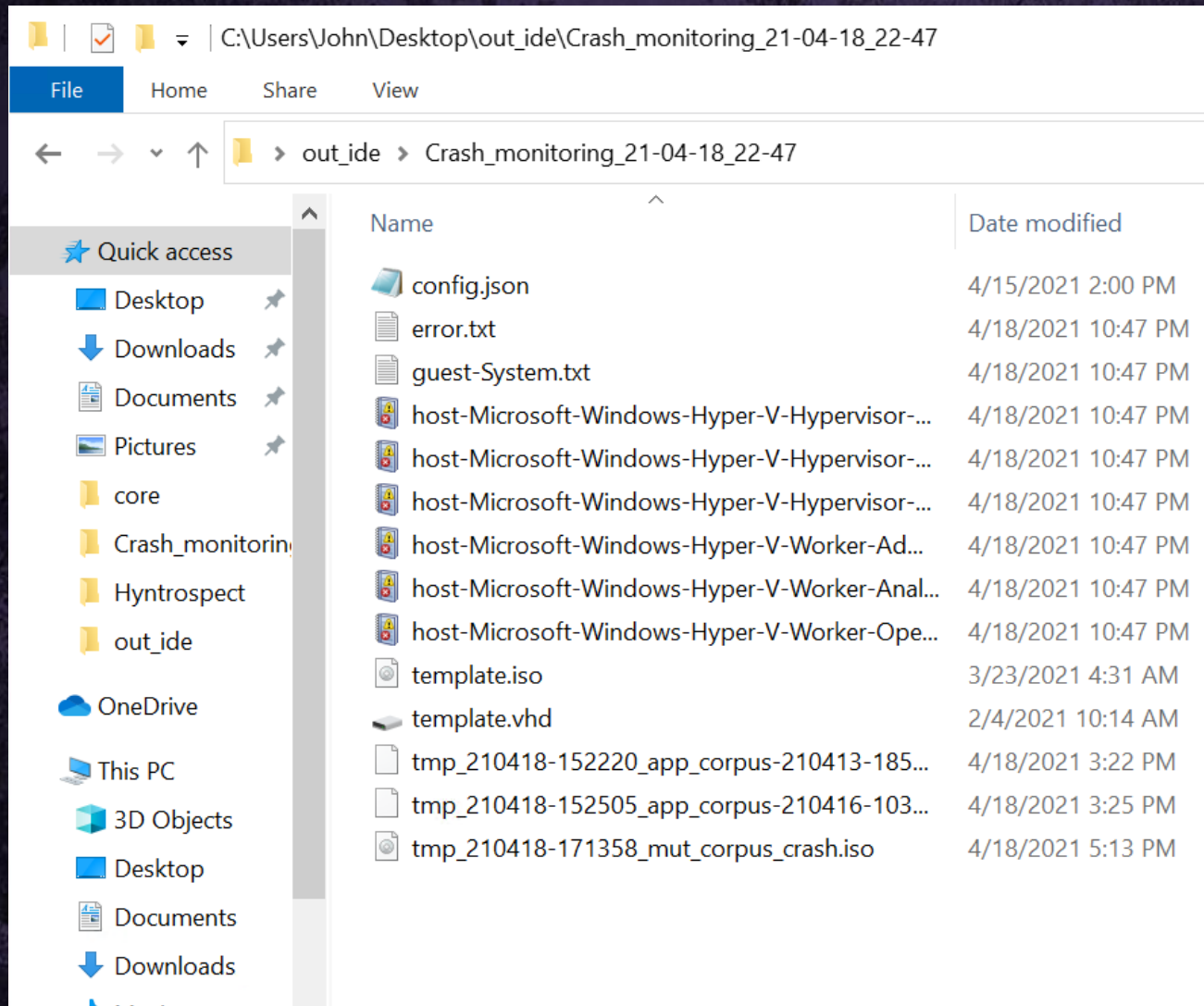
int3 technique

- Pre-compute the list of targeted blocks' addresses
- Set int3 at the beginning of each block
- Each int3 reached = coverage increase
- The int3 is removed, input file handled, execution resumes
- Faster over time

Generation of the Input File

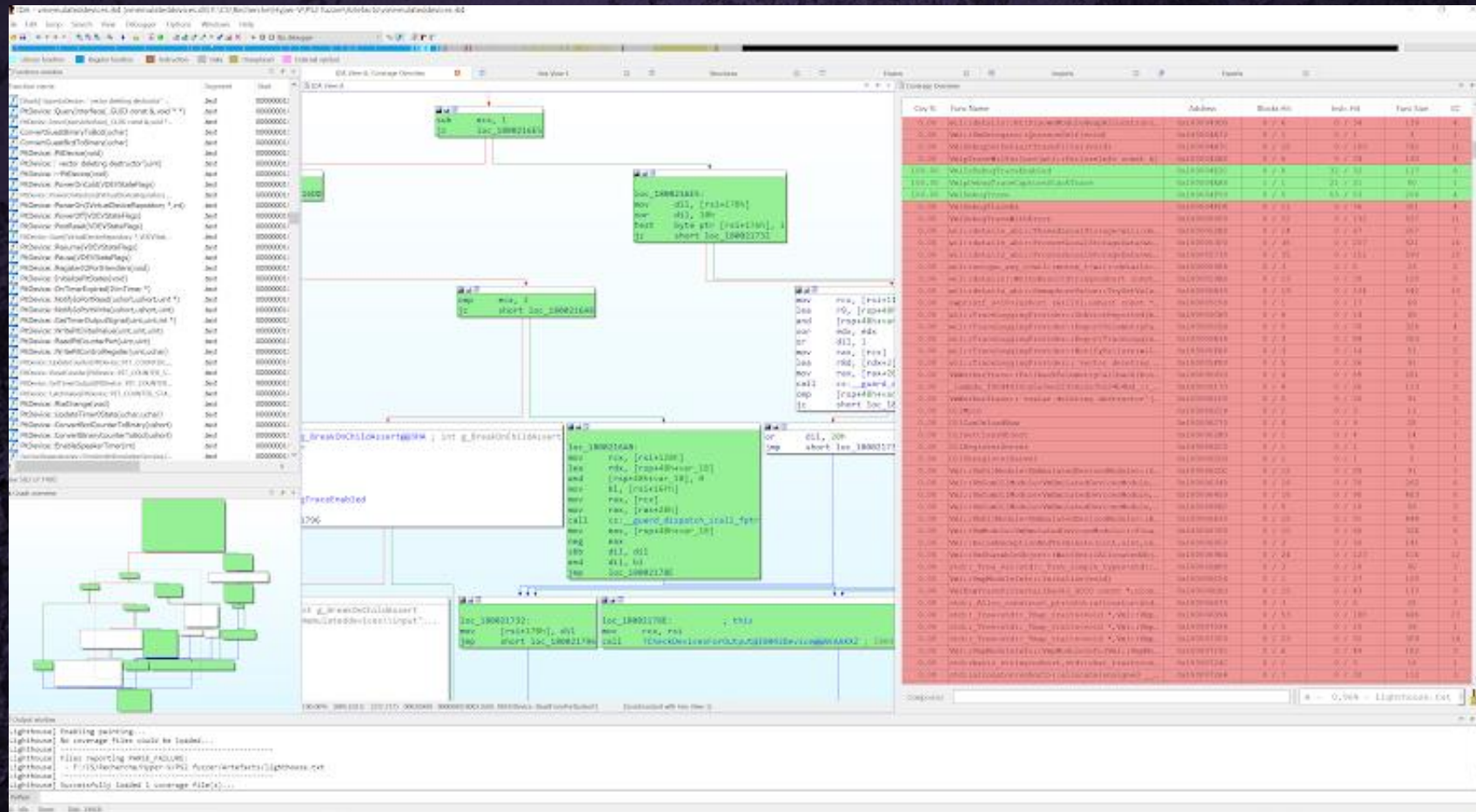
- Record of **seeds** at the beginning [optional]
- **Corpus** of “interesting files”
- **Coverage increase** -> truncated input file added to the corpus
- 3 **strategies**: mutate, append, generate randomly
- **Format** of input files
 - Byte 0 % 2 -> IN / OUT operation
 - Byte 1 % (number of ports) -> selected IO port
 - Byte 2 % 3 -> length
 - If OUT and based on length -> value

Crash Qualification



- 2 levels of monitoring:
 - Debugger level
 - Monitoring process
- Tip: track the VM uptime
- Crash folder created with logs and artefacts to re-run the case

Coverage Visualization in IDA



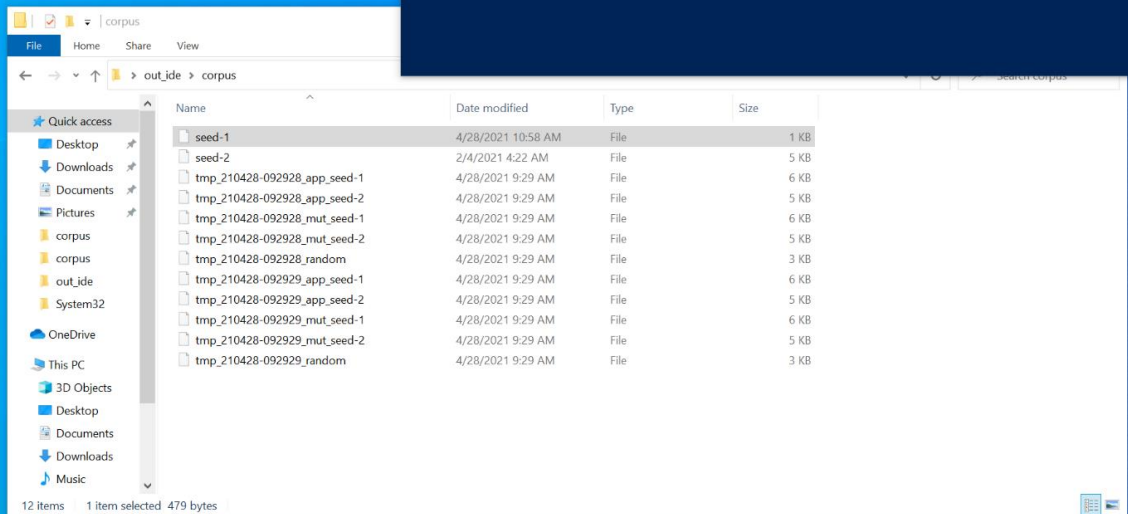
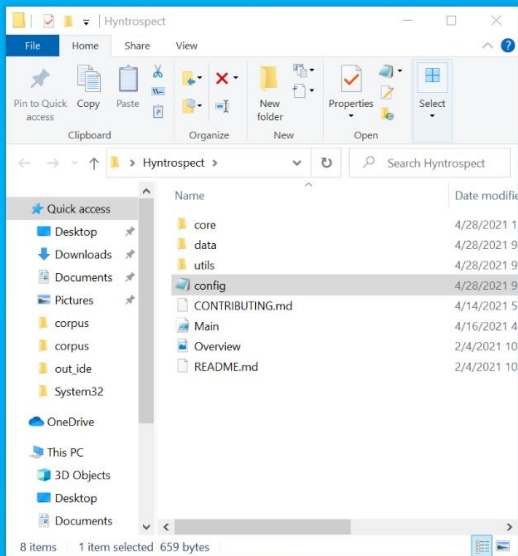
Optionally, using a helper and IDA + LightHouse


```
Administrator: C:\Users\John\Desktop\Hyntrospect\core\DbgShell\x64\DbgShell.exe
ModLoad: 00007FFE`35600000 00007FFE`3567F000 fwpucint
ModLoad: 00007FFE`131B0000 00007FFE`131E5000 vmcsynthstor
ModLoad: 00007FFE`397E0000 00007FFE`39855000 vmrdrvcore
ModLoad: 00007FFE`41730000 00007FFE`417d0000 shcore
ModLoad: 00007FFE`401C0000 00007FFE`401cc000 netutil
ModLoad: 00007FFE`3a490000 00007FFE`3a4a9000 samcl
ModLoad: 00007FFE`397b0000 00007FFE`397d0000 srvc
ModLoad: 00007FFE`3e120000 00007FFE`3e134000 WTSAPI32
ModLoad: 00007FFE`39790000 00007FFE`397a8000 NETAPI32
ModLoad: 00007FFE`40520000 00007FFE`4052c000 CRYPTBASE
ModLoad: 00007FFE`401e0000 00007FFE`40222000 LOGONCL
ModLoad: 00007FFE`34b30000 00007FFE`34b3a000 vmisifproxystub
ModLoad: 00007FFE`13100000 00007FFE`1317c000 w32time
(5a4.1760): Break instruction exception - code 80000003 (first chance)
ntdll!IDbgBreakPoint

rax=000000723a74a000 rbx=0000000000000000 rcx=0000000000000000
rdx=00007FFE4365ba10 rsi=0000000000000000 rdi=0000000000000000
rip=00007FFE4362e8a0 rsp=000000723b57fb18 rbp=0000000000000000
r8=0000000000000000 r9=00007FFE4365ba10 r10=0000000000000000
r11=0000000000000000 r12=0000000000000000 r13=0000000000000000
r14=0000000000000000 r15=0000000000000000
iopl=0   TBD: flags
cs=0033  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000246
ntdll!IDbgBreakPoint:
00007FFE`4362e8a0 cc             int     3

Setting the breakpoints. That operation can take a long time depending on the size of the list (~sec to hour).
Starting the execution.
```

```
Administrator: Windows PowerShell
PS C:\Users\John\Desktop\Hyntrospect> .\Main.ps1
Starting DbgShell with fuzzer-master parametered through config.json.
c:\Users\John\Desktop\out_ide\corpus\seed-1
```



Current results



Local runs

- **First targets:** i8042 (PS/2), videoS3, floppy, IDE
 - Example: I8042 device with IO ports 0x60, 0x61, 0x62, 0x64
- **Local setup:**
 - Dedicated workstation
 - 32 GB RAM and Intel Core i9 CPU
 - 8 GB per VM, 1 or 2 vCPUs
 - Windows 10
- **Speed limitation**
 - Main factor: number of breakpoints
 - Time to set them / update them in DbgShell
 - Not linear
- **Next goal:** port the fuzzer to GCP/Azure

Number of breakpoints	Time to set up the breakpoints in DbgShell at each iteration
150	immediate
500	6 seconds
1000	20 seconds
2000	1 minute 15 seconds

Coverage (3 days run)

vmemulateddevices.dll	Current coverage
VideoS3Device	42.7%
i8042Device	40%

VmEmulatedStorage.dll	Current coverage
FloppyControllerDevice	43.3%
IdeControllerDevice	28.8%

- Start / init / stop functions not called
 - Attaching to a running VM
- Debug strings blocks skipped

Guest VM Crash found

- On i8042 device
- Reproducible
- BSOD of the VM with different error messages at each run
 - SYSTEM_SERVICE_EXCEPTION (0x3b)
 - PFN_LIST_CORRUPT (4e)
 - ATTEMPTED_WRITE_TO_READONLY_MEMORY (0xbe)
 - KERNEL_SECURITY_CHECK_FAILURE (0x139)
 - ...
- Memory corruption error

Some more investigation

```
; __int64 __fastcall PciBusDevice::HandleA20GateChange(PciBusDevice * __hidden this)
?HandleA20GateChange@PciBusDevice@@AEAAJXZ proc near

var_38= qword ptr -38h
var_30= qword ptr -30h
var_18= qword ptr -18h
var_10= qword ptr -10h
arg_8= qword ptr 10h

mov     [rsp+arg_8], rbx
push    rdi
sub     rsp, 50h
xor     edi, edi
mov     rbx, rcx
cmp     [rcx+14Fh], dil
jz      short loc_180018AC5
```

- Narrowed down the case
 - Sequence of 2 OUT operations
 - State machine, path accessible in 2 steps
- PciBusDevice::HandleA20GateChange
 - Legacy A20 device
 - Updates the host memory mapping
 - ... but the guest keeps the same mapping
- Question: possible compromise of VBS?

Follow-up

- In practice, impossible to exploit
- Not a security bug
- Shared with MSRC
- Validates the behavior of the fuzzer, crash handling and reproduction scripts
- Highlights that this surface has probably been well covered

Future Endeavours



Design Limitations

- **Space**: Restricted to the **userland** of the root partition
- **Time**: Not optimized for **speed**

Future Work

- Development of the fuzzer internals
 - Mutation strategy
 - Redevelop some parts in C++ or C# for performance
 - Speed-related updates: towards a minimal debugger?
 - [not prioritized] Userland vs kernel (or hAFL1?)
- Porting to GCP / Azure
 - Port to new devices
 - Run faster and longer
- Adapting to other root partition targets
 - Keeping the frame and “basic blocks”
 - Changing the commands and input consumption
 - Other interesting dlls loaded in vmwp.exe (generation 2).

Conclusion



<https://github.com/googleprojectzero/Hyntrospect>

Windows PowerShell

```
PS C:\> Write-Output "Do you have any questions?"
```