Microsoft

Growing Hypervisor 0day with Hyperseed

Daniel King *@long123king*
Shawn Denbow *@sdenbow_*

MSRC Vulnerabilities & Mitigations Team
CDG Platform Security & Vulnerability Research

whoami

- Daniel King
  - MSRC V&M
  - Keen Security Lab
  - Member of “Master of Pwn” 2016
  - MSRC Nano Server Bounty Winner
  - Ruxcon 2016
  - CodeBlue 2016
  - ZeroNights 2015
- Shawn Denbow
  - CDG PSVR
  - USAF
  - Matasano
  - RPISEC

Overview

- Motivation
- Hyper-V Architecture
- Hypercalls
- Hyperseed
- Mutations
- Findings

Motivation – Why Hypercalls?

Importance

- Cornerstone of Hyper-V

Accessibility

- Accessible from kernel (Ring-0) mode

Simplicity

- Well-documented, good starting point

Complexity

- Diverse input & output format

Motivation – Bug Bounty

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)

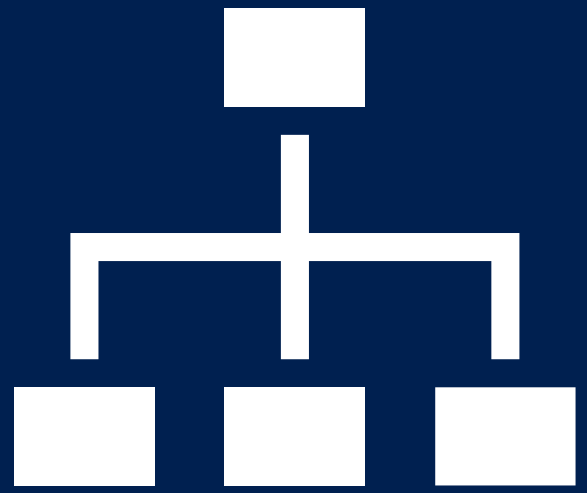
Information Disclosure

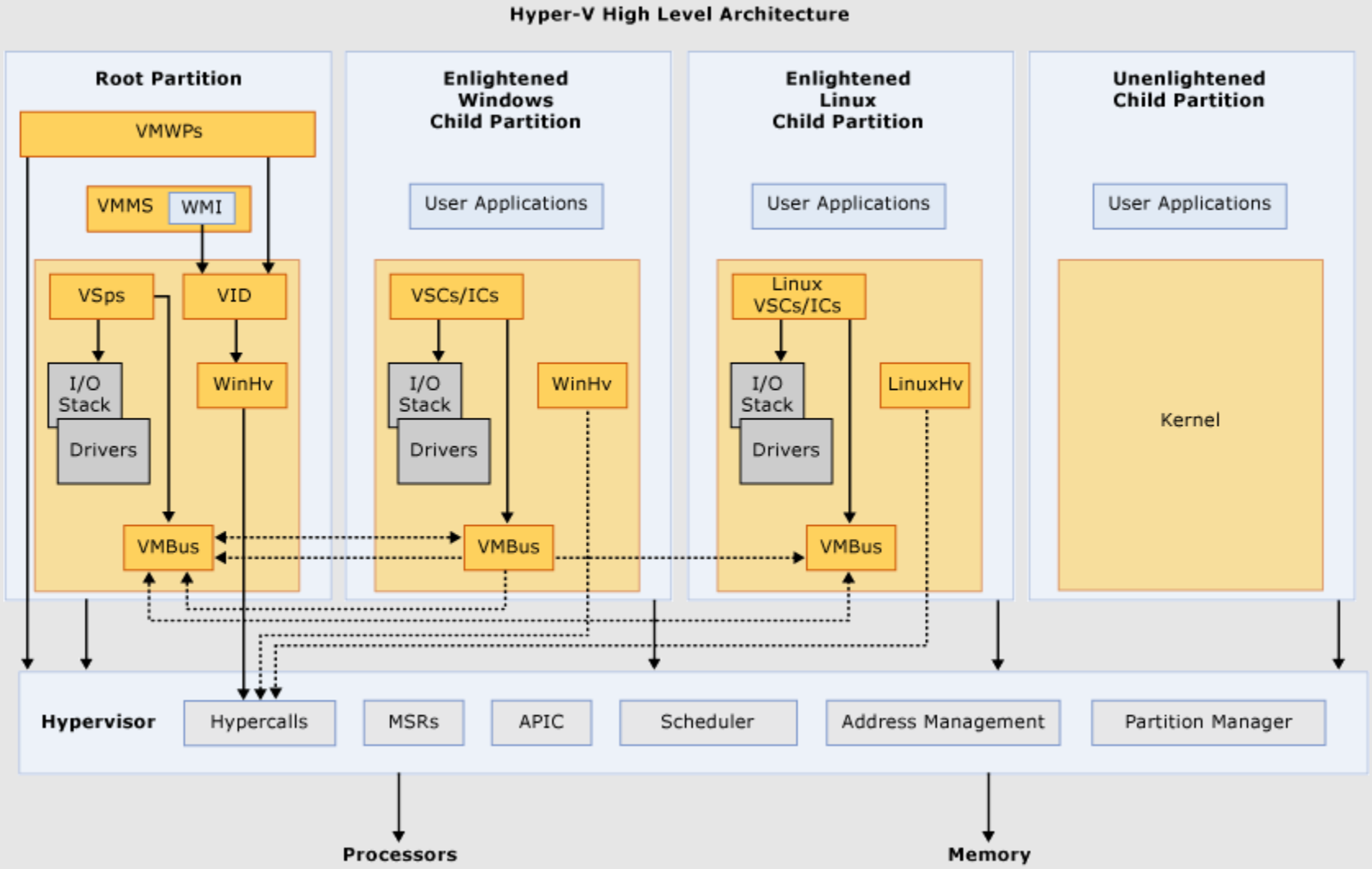
- $25,000 (Hypervisor/Kernel), $15,000 (User-mode)

Denial of Service

- $15,000 (Hypervisor/Kernel)

Hyper-V Architecture

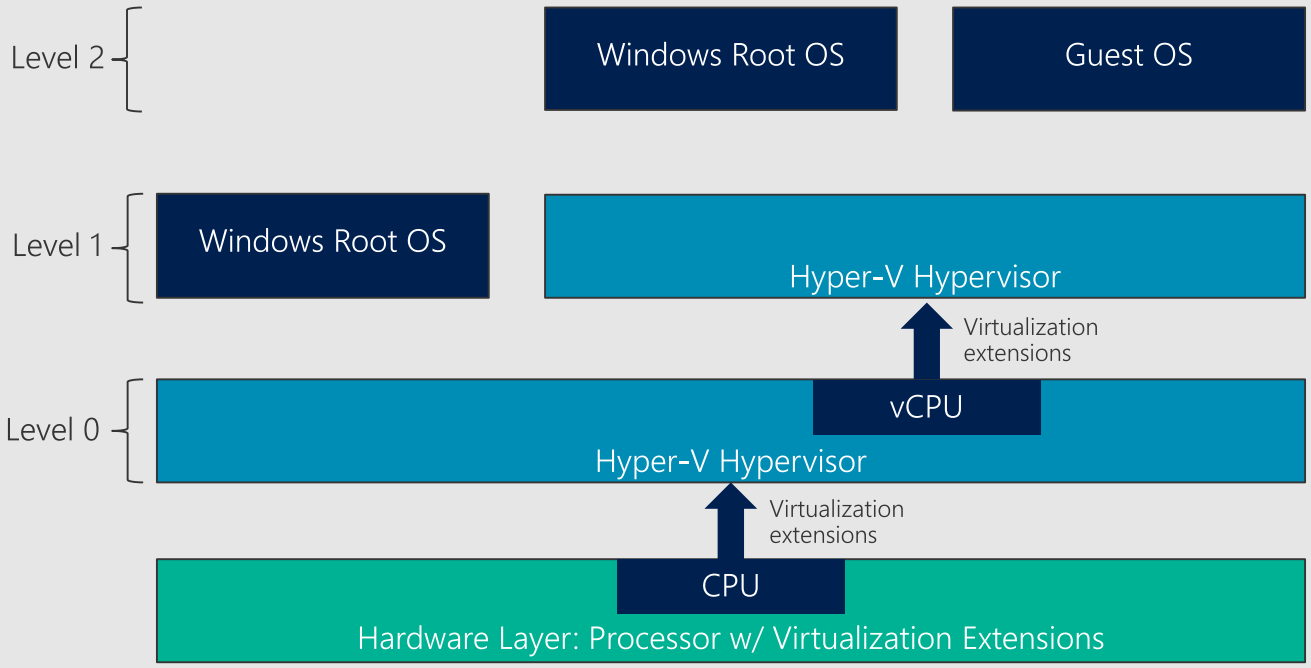




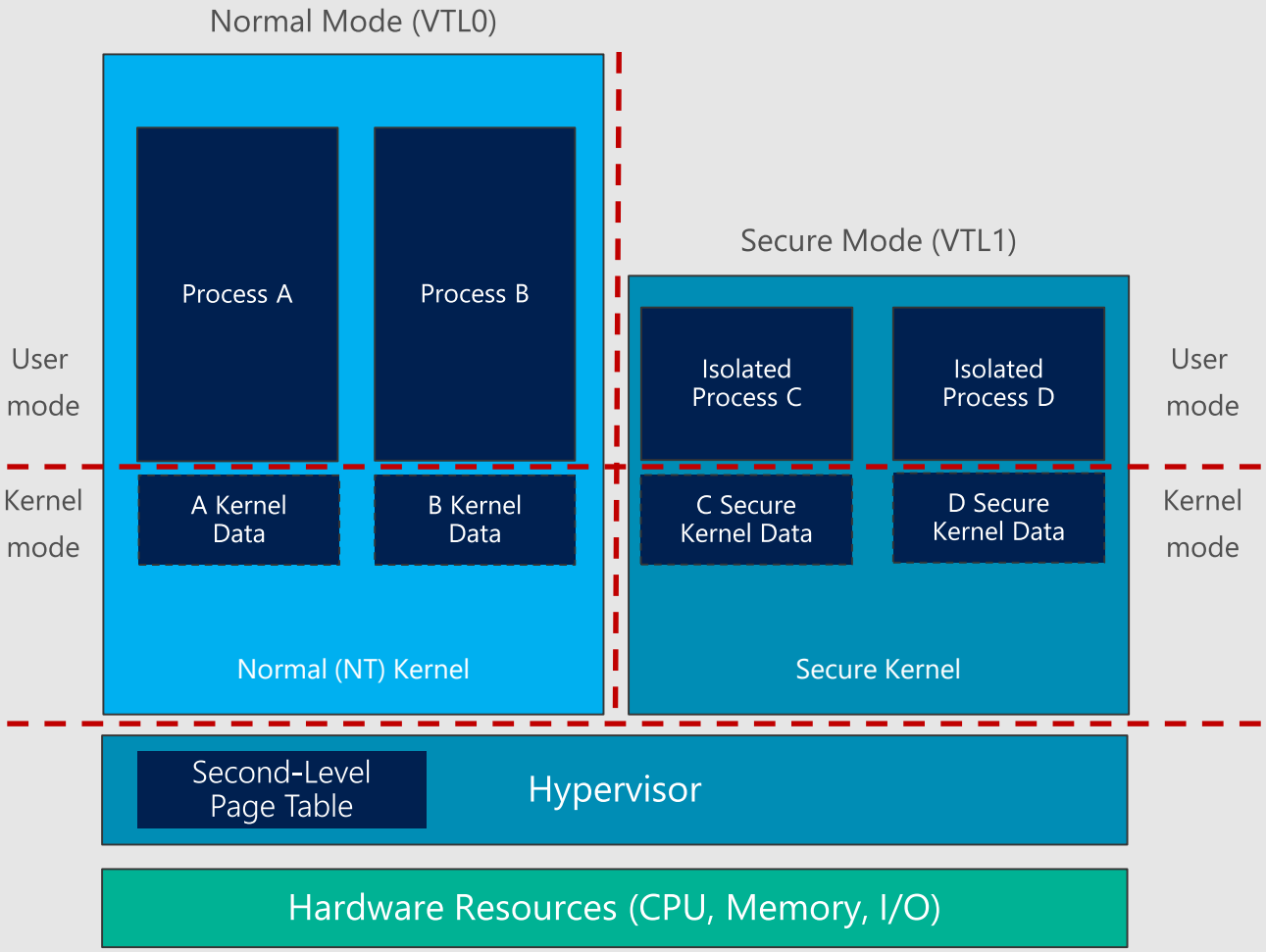
https://docs.microsoft.com/en-us/virtualization/hyper-v-on-windows/reference/hyper-v-architecture

Nested Virtualization

- L0 Hyper-V Hypervisor
- L1 Root
- L1 Guest
- L1 Hyper-V Hypervisor
- L2 Root
- L2 Guest



Virtualization Based Security



- VBS introduces concept of trust levels
  - VTL0 – normal mode
  - VTL1 – secure (privileged) mode
- VTL1 controls which resources VTL0 can access
  - Memory, privileged CPU registers
- Hypervisor leverages hardware virtualization to enforce access permissions
  - SLAT – memory access protection for CPU
  - IOMMU – memory access protection for devices
- VTL1 is NOT a separate VM
  - VTL1 and VTL0 share memory, CPU's
  - Secure kernel coordinates with normal kernel for memory management, scheduling, etc.

https://myignite.techcommunity.microsoft.com/sessions/66666

Hypercalls

Hypercall Basics

- Calling mechanism for guests to interface with the hypervisor
  - Three calling conventions
    - Slow
    - Fast
    - XMM Fast
- Hypercall classes
  - Simple
  - Rep “repeat”
- Variable Sized Hypercalls
- Extended Hypercall Interface
  - Call codes above 0x8000

```
typedef union _HV_HYPERCALL_INPUT
{
    struct
    {
        UINT32 CallCode           : 16;
        UINT32 IsFast             : 1;
        UINT32 VariableHeaderSize : 9;
        UINT32 Reserved1          : 6;
        UINT32 CountOfElements    : 12;
        UINT32 Reserved2          : 4;
        UINT32 RepStartIndex      : 12;
        UINT32 Reserved3          : 4;
    };
    UINT64 AsUINT64;
} HV_HYPERCALL_INPUT, *PHV_HYPERCALL_INPUT;
```

```
typedef union _HV_HYPERCALL_OUTPUT
{
    struct
    {
        UINT16 CallStatus;
        UINT16 Reserved1;
        UINT32 ElementsProcessed : 12;
        UINT32 Reserved2         : 20;
    };
    UINT64 AsUINT64;
} HV_HYPERCALL_OUTPUT, *PHV_HYPERCALL_OUTPUT;
```

How To Issue a Hypercall

- Hypercalls must be invoked from the most privileged guest processor mode, CPL0 / Kernel mode
- All hypercalls should be invoked through the architecturally-defined hypercall interface
  - vmcall instruction for Intel VMX
  - vmmcall instruction for AMD SVM
- Hypercall Page: abstract the difference between different virtualization implementations
  - #define HV_X64_MSR_HYPERCALL 0x40000001

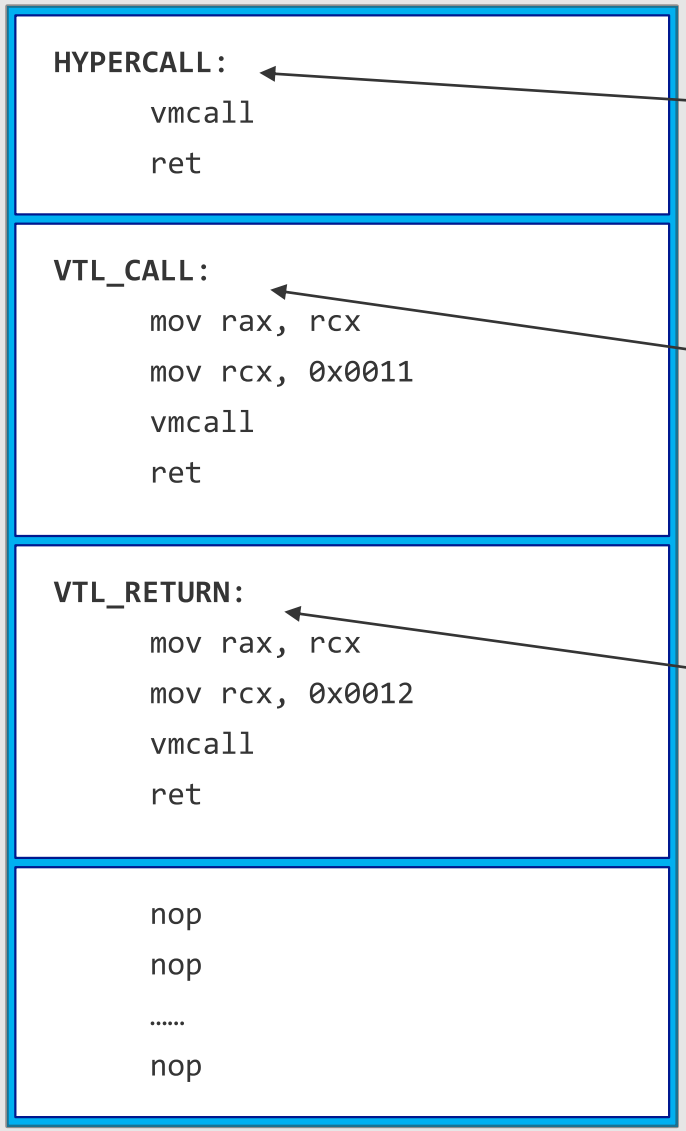
Establishing the Hypercall Interface

- Write guest id to HV_X64_MSR_GUEST_OS_ID
- Allocate a page of memory for the hypercall page
- Write to the hypercall MSR HV_X64_MSR_HYPERCALL
  - Set the Enable Hypercall Page bit
  - Write the GPA of the hypercall page

```
/*
 * Setup the hypercall page and enable hypercalls.
 * 1. Register the guest ID
 * 2. Enable the hypercall and register the hypercall page
 */
guest_id = generate_guest_id(0, LINUX_VERSION_CODE, 0);
wrmsrl(HV_X64_MSR_GUEST_OS_ID, guest_id);

hypercall_pg  = __vmalloc(PAGE_SIZE, GFP_KERNEL, PAGE_KERNEL_RX);

if (hypercall_pg == NULL) {
    wrmsrl(HV_X64_MSR_GUEST_OS_ID, 0);
    return;
}

rdmsrl(HV_X64_MSR_HYPERCALL, hypercall_msr.as_uint64);
hypercall_msr.enable = 1;
hypercall_msr.guest_physical_address = vmalloc_to_pfn(hypercall_pg);
wrmsrl(HV_X64_MSR_HYPERCALL, hypercall_msr.as_uint64);
```

FROM: https://github.com/LIS/lis-next

Hypercall Page



Calling here will issue a hypercall

Calling here will issue a secure call from VTL0 to VTL1

Calling here will issue a normal call from VTL1 to VTL0

HvCallVtlCall & HvCallVtlReturn

VTL 0 -> VTL 1

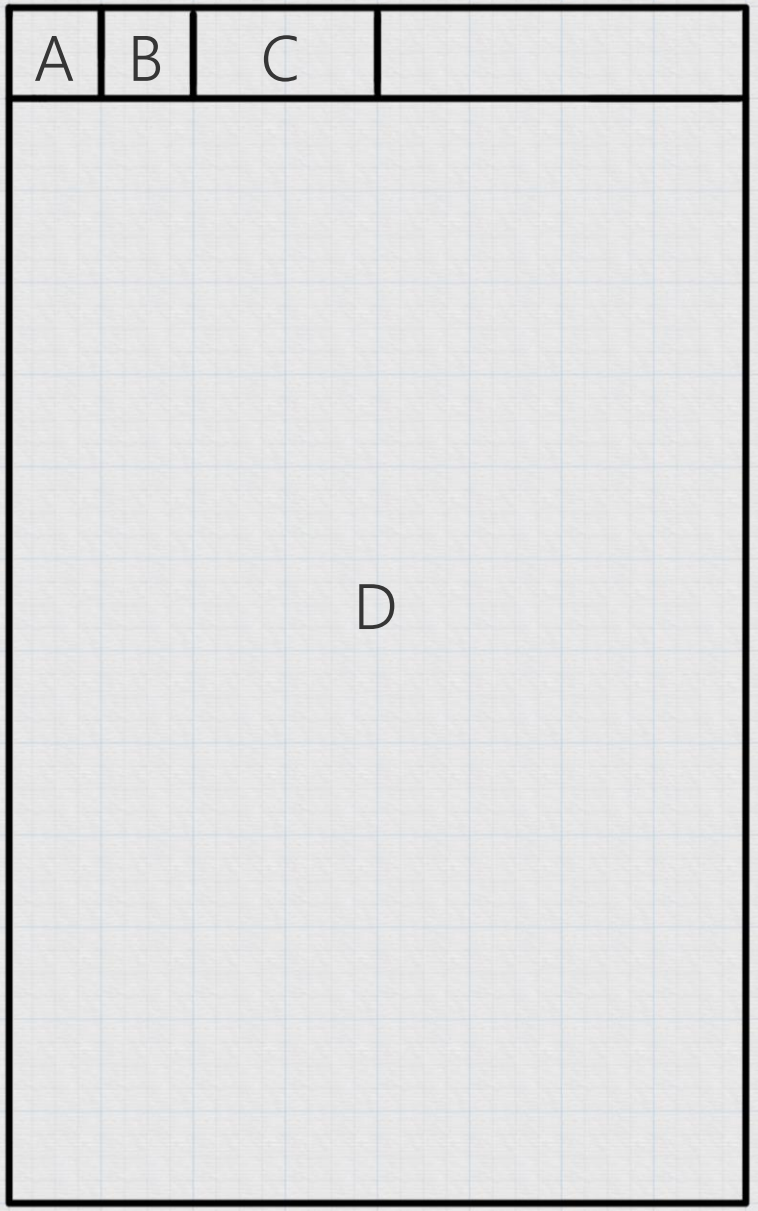
- nt!VslpEnterIumSecureMode
- vmcall(0x11)

VTL 1 -> VTL 0

- sk!SkCallNormalMode
- vmcall(0x12)

VTL Call & Return Buffer

- A [uint8_t]: SecureOperation
  - 2: Invoke Secure Service
- B [uint8_t]: NormalOperation
  - 0: Invoke Normal Service
- C [uint16_t]: ServiceCode
- D [uint64_t]: Parameters[12]



Restrictions Enforced by Hypercall Handlers

Restrictions Enforced by Hypercall Handlers

- The calling partition must possess a particular privilege

Partition Properties

- Partitions are identified by using a partition ID
  - #define HV_PARTITION_ID_INVALID ((HV_PARTITION_ID) 0x0)
  - #define HV_PARTITION_ID_SELF ((HV_PARTITION_ID) -1)
- Each partition has a set of properties
  - Read a property of a partition by invoking HvGetPartitionProperty
  - Modify partitions property by invoking HvSetPartitionProperty
  - HvPartitionPropertyPrivilegeFlags defines what the partition is allowed to access
- Partition privilege flags represented by a bit-field

Partition Privilege Flags

```
// Access to virtual MSRs
AccessVpRunTimeReg:1;
AccessPartitionReferenceCounter:1;
AccessSynicRegs:1;
AccessSyntheticTimerRegs:1;
AccessIntrCtrlRegs:1;
AccessHypercallMsrs:1;
AccessVpIndex:1;
AccessResetReg:1;
AccessStatsReg:1;
AccessPartitionReferenceTsc:1;
AccessGuestIdleReg:1;
AccessFrequencyRegs:1;
AccessDebugRegs:1;
AccessReenlightenmentControls:1;
Reserved1:18;
```

```
// Access to hypercalls
CreatePartitions:1;
AccessPartitionId:1;
AccessMemoryPool:1;
AdjustMessageBuffers:1;
PostMessages:1;
SignalEvents:1;
CreatePort:1;
ConnectPort:1;
AccessStats:1;
Reserved2:2;
Debugging:1;
CpuManagement:1;
Reserved:1;
Reserved:1;
Reserved:1;
AccessVsm:1;
AccessVpRegisters:1;
Reserved:1;
Reserved:1;
EnableExtendedHypercalls:1;
StartVirtualProcessor:1;
Reserved3:10;
```

Partition Privilege Flags - Child

```
// Access to virtual MSRs
AccessVpRunTimeReg:1;
AccessPartitionReferenceCounter:1;
AccessSynicRegs:1;
AccessSyntheticTimerRegs:1;
AccessIntrCtrlRegs:1;
AccessHypercallMsrs:1;
AccessVpIndex:1;
AccessResetReg:1;
AccessStatsReg:1;
AccessPartitionReferenceTsc:1;
AccessGuestIdleReg:1;
AccessFrequencyRegs:1;
AccessDebugRegs:1;
AccessReenlightenmentControls:1;
Reserved1:18;
```

```
// Access to hypercalls
CreatePartitions:1;
AccessPartitionId:1;
AccessMemoryPool:1;
AdjustMessageBuffers:1;
PostMessages:1;
SignalEvents:1;
CreatePort:1;
ConnectPort:1;
AccessStats:1;
Reserved2:2;
Debugging:1;
CpuManagement:1;
Reserved:1;
Reserved:1;
Reserved:1;
AccessVsm:1;
AccessVpRegisters:1;
Reserved:1;
Reserved:1;
EnableExtendedHypercalls:1;
StartVirtualProcessor:1;
Reserved3:10;
```

Partition Privilege Flags - Child

```
// Access to virtual MSRs
AccessVpRunTimeReg:1;
AccessPartitionReferenceCounter:1;
AccessSynicRegs:1;
AccessSyntheticTimerRegs:1;
AccessIntrCtrlRegs:1;
AccessHypercallMsrs:1;
AccessVpIndex:1;
AccessResetReg:1;
AccessStatsReg:1;
AccessPartitionReferenceTsc:1;
AccessGuestIdleReg:1;
AccessFrequencyRegs:1;
AccessDebugRegs:1;
AccessReenlightenmentControls:1;
Reserved1:18;
```

```
// Access to hypercalls
CreatePartitions:1;
AccessPartitionId:1;
AccessMemoryPool:1;
AdjustMessageBuffers:1;
PostMessages:1;
SignalEvents:1;
CreatePort:1;
ConnectPort:1;
AccessStats:1;
Reserved2:2;
Debugging:1;
CpuManagement:1;
Reserved:1;
Reserved:1;
Reserved:1;
AccessVsm:1;
AccessVpRegisters:1;
Reserved:1;
Reserved:1;
EnableExtendedHypercalls:1;
StartVirtualProcessor:1;
Reserved3:10;
```

Partition Privilege Flags - Root

```
// Access to virtual MSRs
AccessVpRunTimeReg:1;
AccessPartitionReferenceCounter:1;
AccessSynicRegs:1;
AccessSyntheticTimerRegs:1;
AccessIntrCtrlRegs:1;
AccessHypercallMsrs:1;
AccessVpIndex:1;
AccessResetReg:1;
AccessStatsReg:1;
AccessPartitionReferenceTsc:1;
AccessGuestIdleReg:1;
AccessFrequencyRegs:1;
AccessDebugRegs:1;
AccessReenlightenmentControls:1;
Reserved1:18;
```

```
// Access to hypercalls
CreatePartitions:1;
AccessPartitionId:1;
AccessMemoryPool:1;
AdjustMessageBuffers:1;
PostMessages:1;
SignalEvents:1;
CreatePort:1;
ConnectPort:1;
AccessStats:1;
Reserved2:2;
Debugging:1;
CpuManagement:1;
Reserved:1;
Reserved:1;
Reserved:1;
AccessVsm:1;
AccessVpRegisters:1;
Reserved:1;
Reserved:1;
EnableExtendedHypercalls:1;
StartVirtualProcessor:1;
Reserved3:10;
```

Partition Privilege Check

```
//
// Check for CpuManagement permission.
//
status = HV_STATUS_SUCCESS;
privileges.AsUINT64 = 0;
privileges.CpuManagement = TRUE;
HT_CHK(status = VmCheckCurrentPartitionPrivileges(privileges));
```

Restrictions Enforced by Hypercall Handlers

- The calling partition must possess a particular privilege
- The partition being acted upon must be in a particular state
- The partition must be either a parent or child

Partition State Check

```
//
// This function locates a partition with the specified partition Id and
// references it only if its state matches one of the specified values.
//
HT_CHK(status = ObReferencePartitionById(InputHeader->PartitionId,
                                         ObPartitionReferenceIfActive,
                                         ObPartitionRightParent,
                                         ObPartitionSecurityNormal,
                                         &partition,
                                         &partitionReference));
```

Restrictions Enforced by Hypercall Handlers

- The calling partition must possess a particular privilege
- The partition being acted upon must be in a particular state
- The partition must be either a parent or child
- The partition must be the root

Root Partition Check

```
//
// This hypercall can only be made by the highest enabled VTL in the root partition.
//
partition = VmGetCurrentPartition();
if (!partition->Settings.IsRootPartition)
{
    HT_ERR(status = HV_STATUS_ACCESS_DENIED);
}
HT_CHK(status = VmCheckCurrentVtlAuthority(VmRootPartition, VmGetPtHighestEnabledVtl()));
```

Restrictions Enforced by Hypercall Handlers

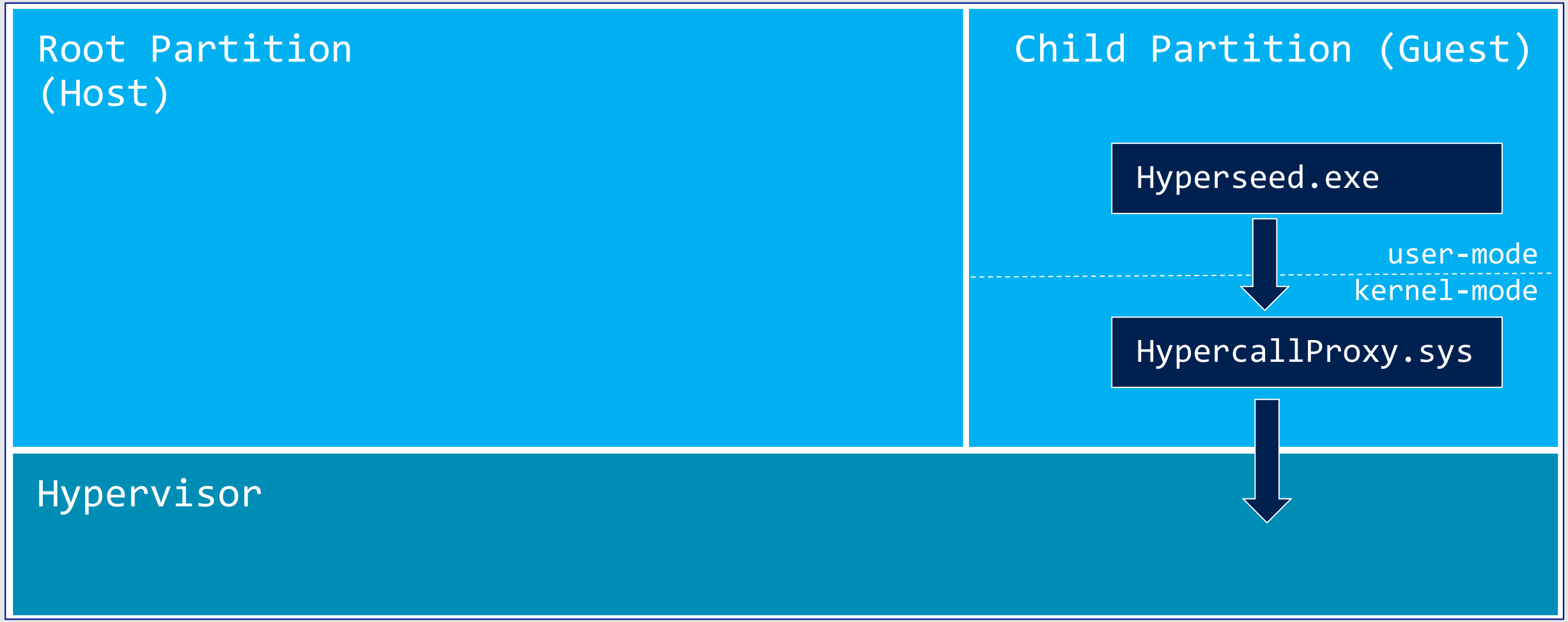
- The calling partition must possess a particular privilege
- The partition being acted upon must be in a particular state
- The partition must be the root
- The partition must be either a parent or child
- The virtual processor must be in a particular state

Virtual Processor State Check

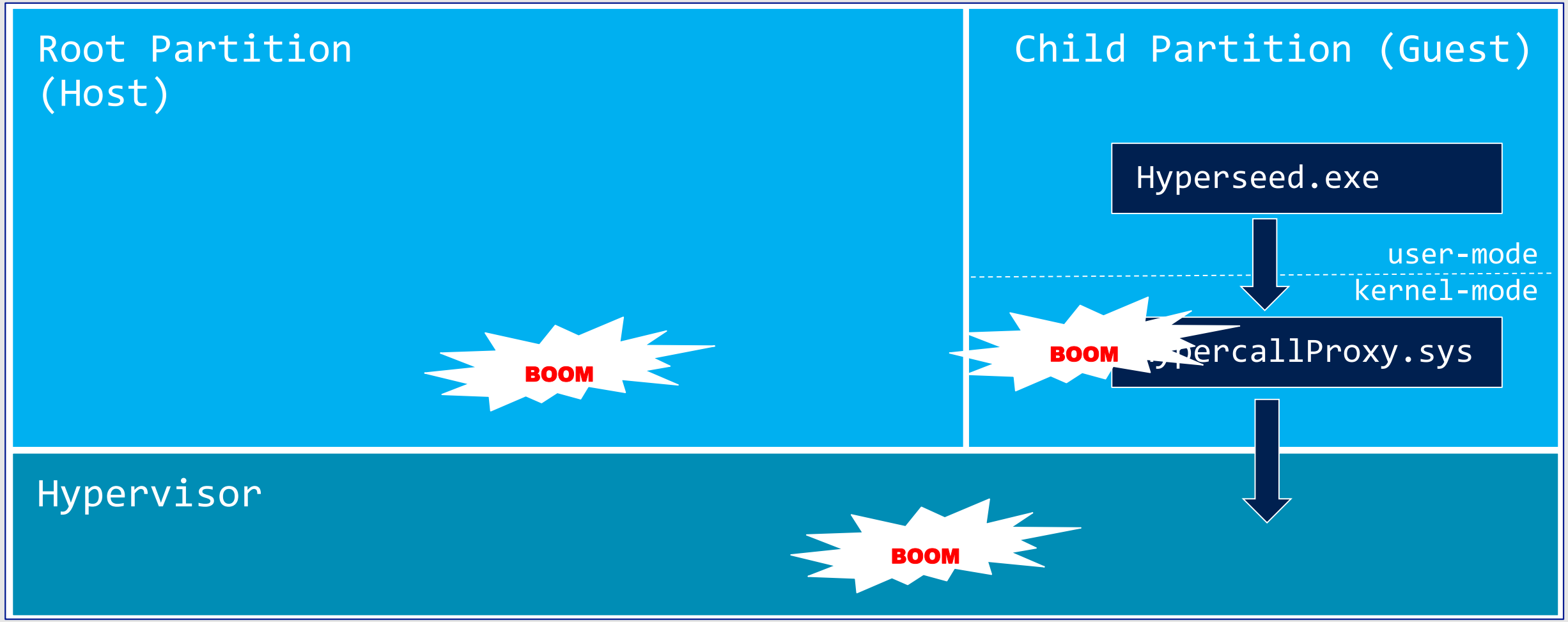
```
//
// VP must be explicitly suspended before restore.
//
if (!Info->Vp->Activity.ExplicitSuspend)
{
    HT_ERR(status = HV_STATUS_INVALID_VP_STATE);
}
```

Hyperseed

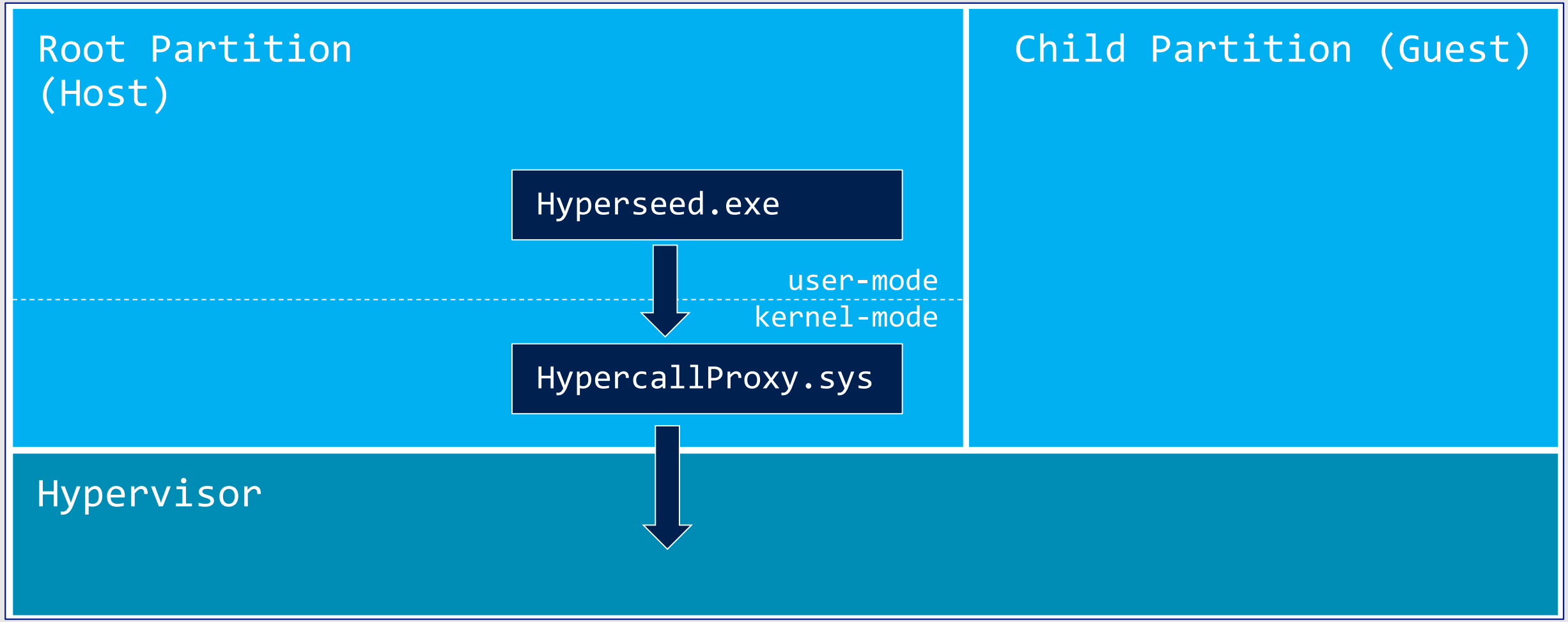
Fuzzing from Guest



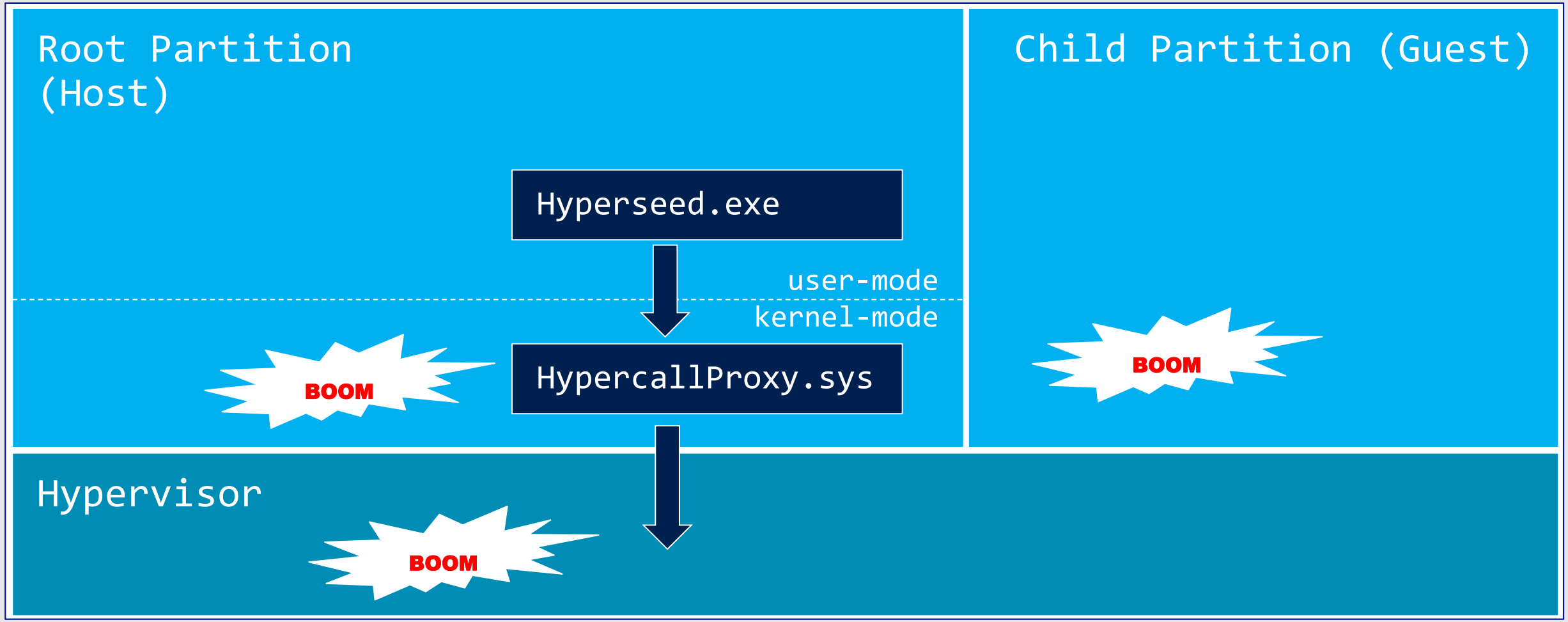
Fuzzing from Guest



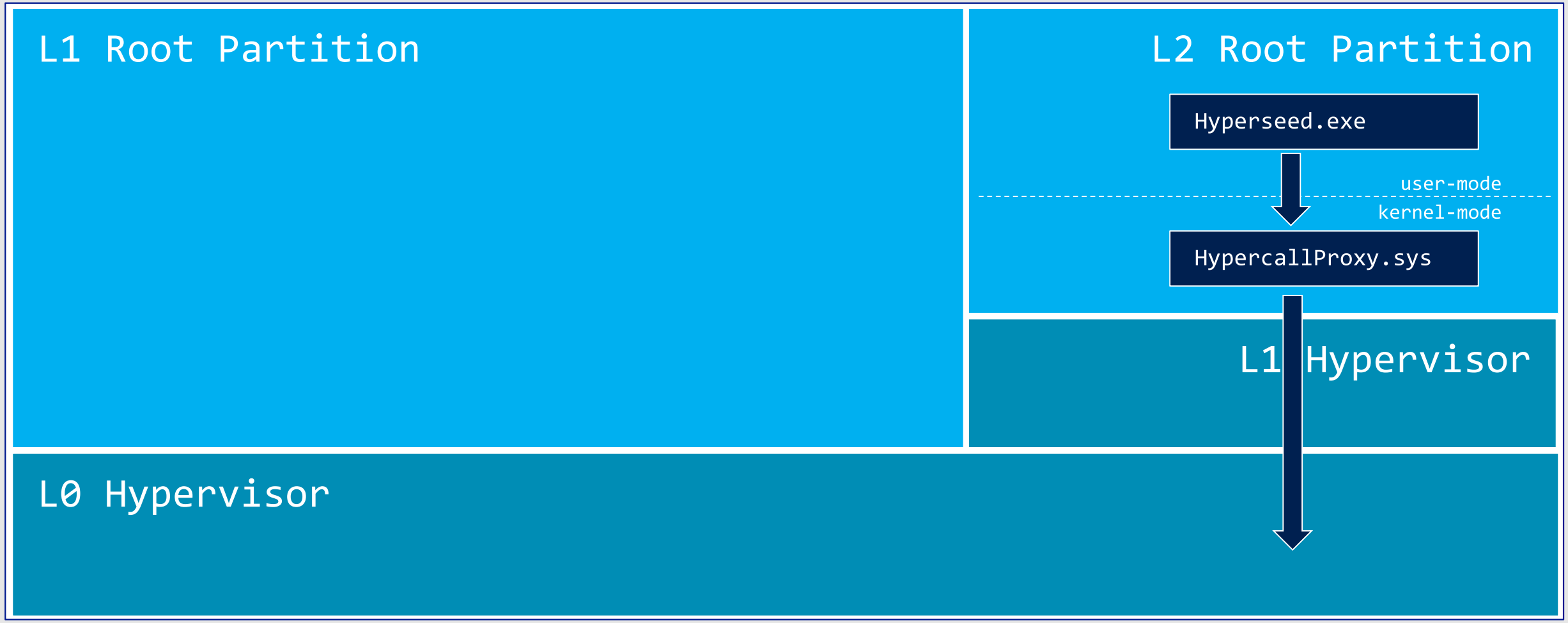
Fuzzing from Root



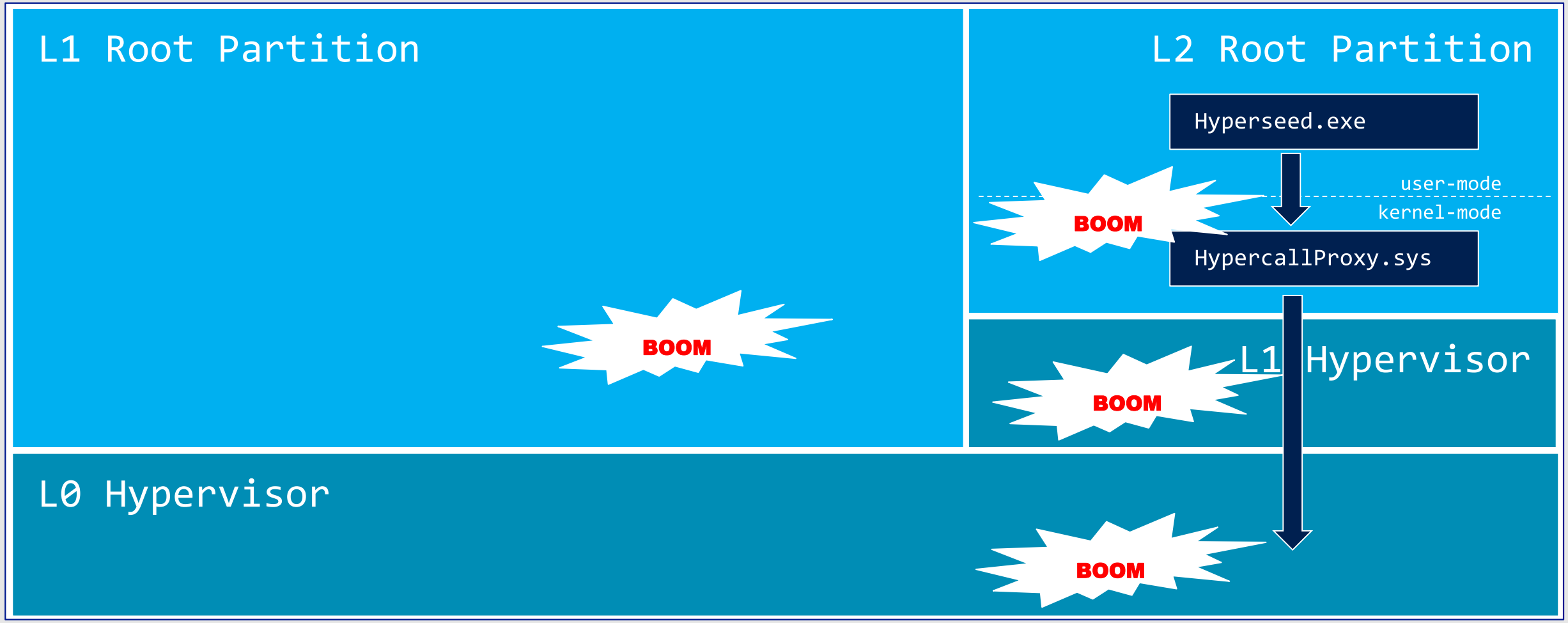
Fuzzing from Root



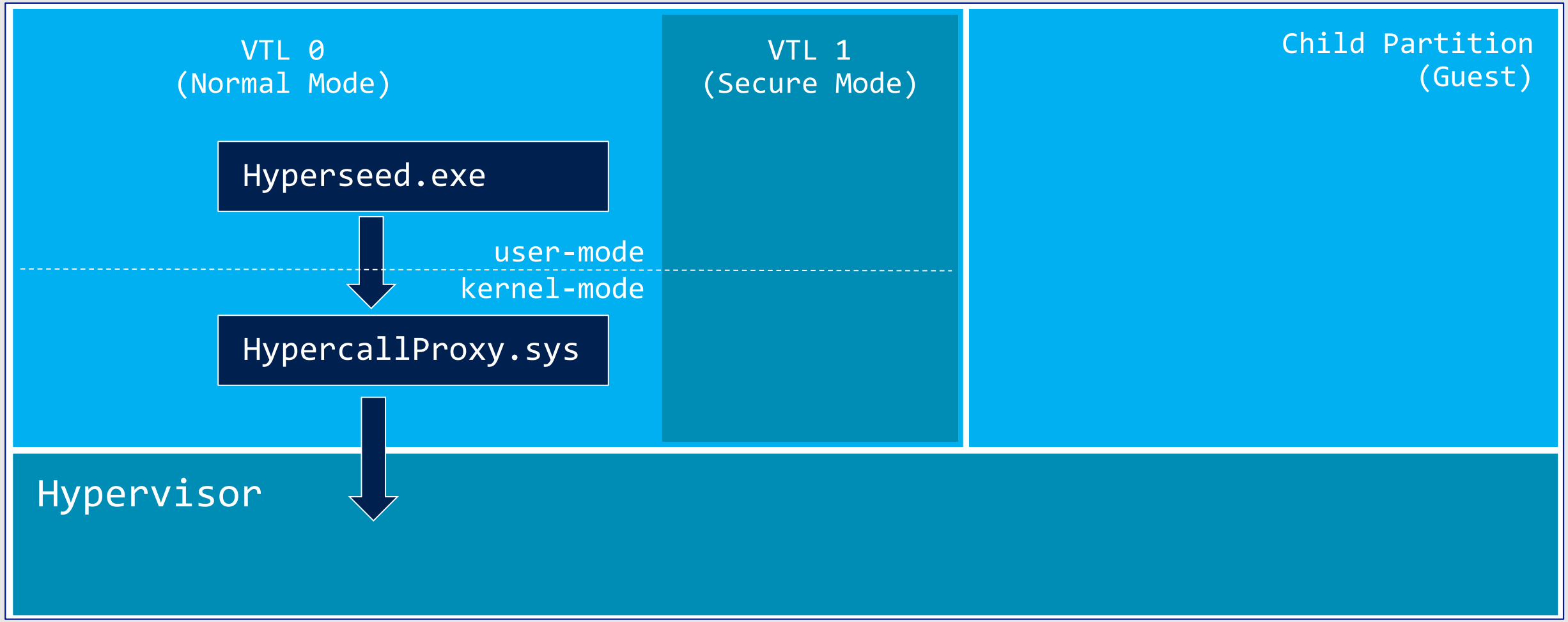
Fuzzing from L2 Root Partition



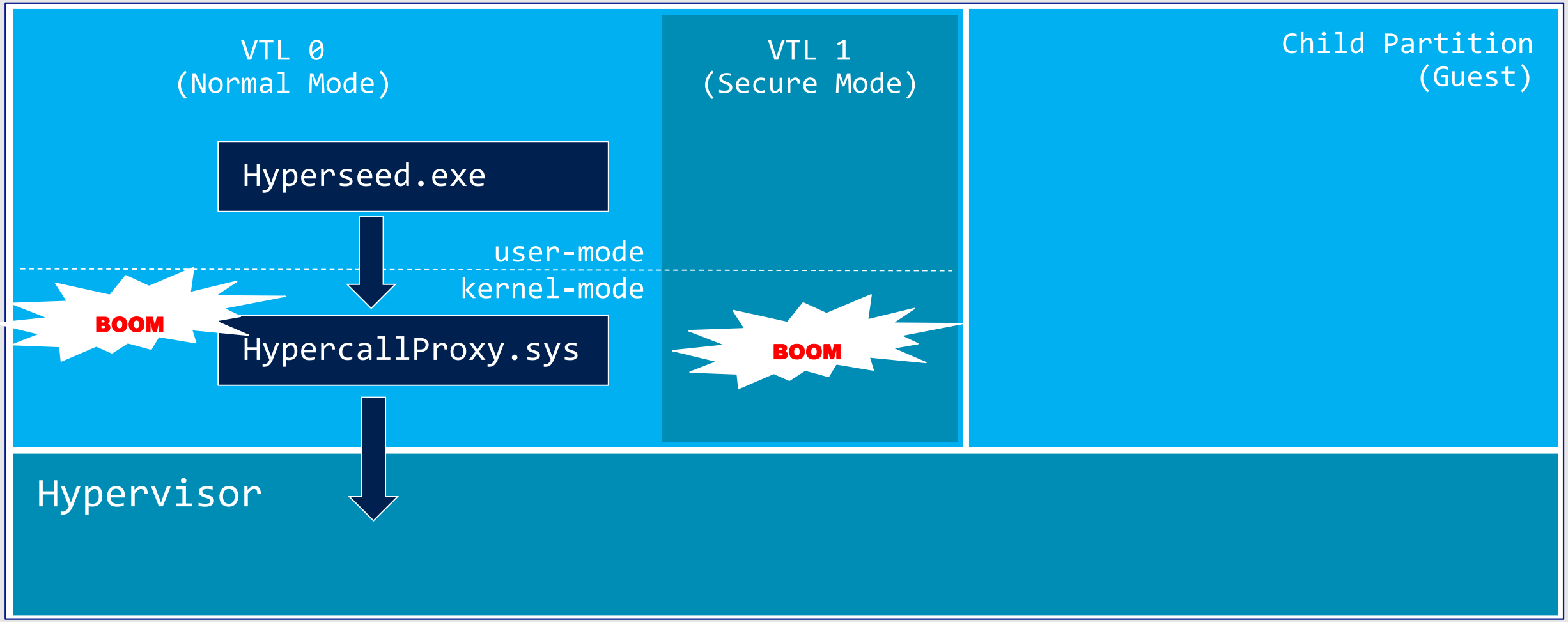
Fuzzing from L2 Root Partition



Fuzzing from Host VTL 0



Fuzzing from Host VTL 0



Format-Aware Fuzzing

Past Fuzzing Experience

- CLFS, Deathnote of Microsoft Windows Kernel
- Intel PT & kAFL

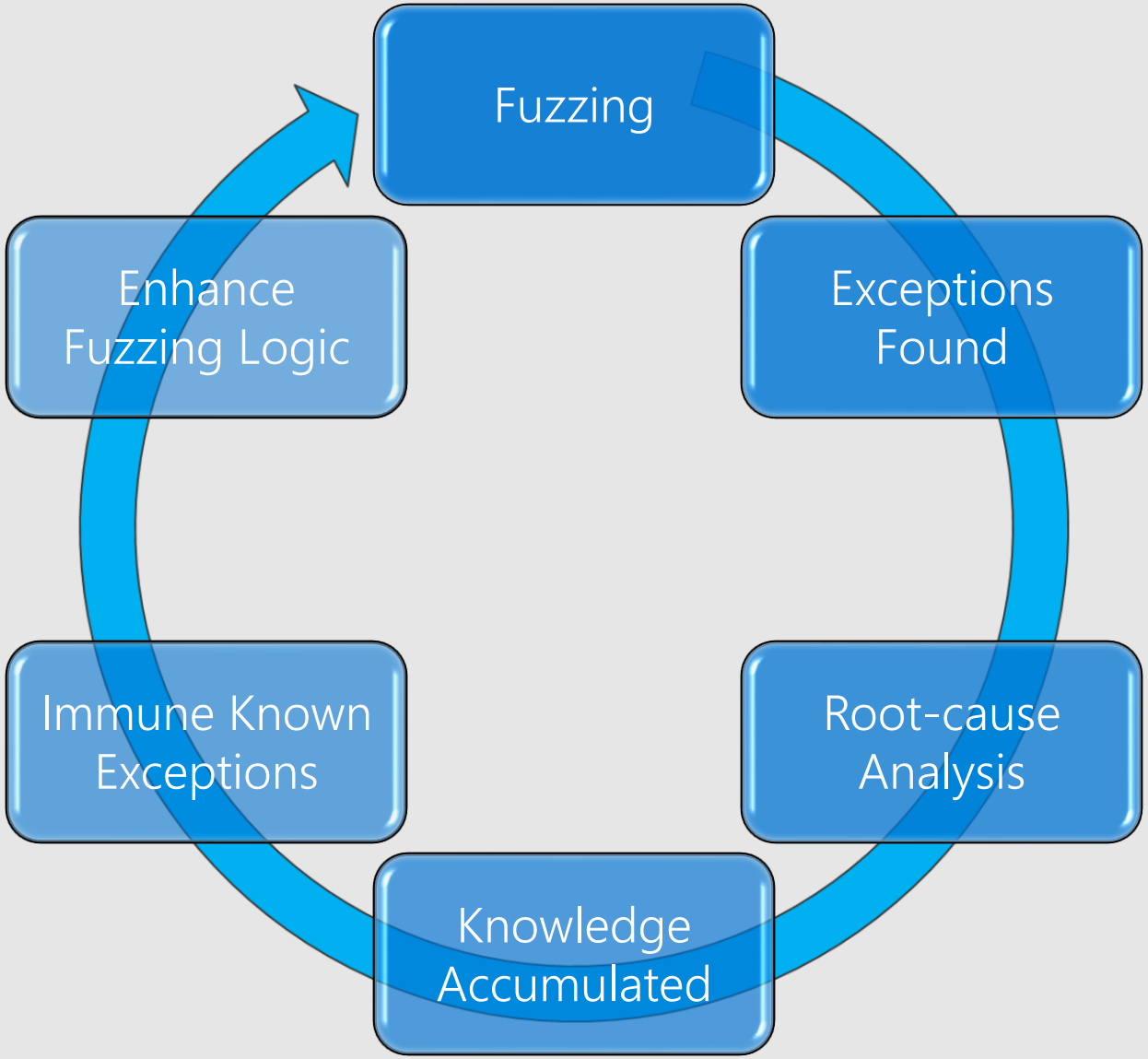
Pros

- Effective to bypass fields constraints
- Easy to scale, from coarse-grained to fine-grained, from less hypercalls to more hypercalls.

Cons

- Assumptions may overlook some special circumstance
- Knowledge of input format needed

Knowledge Accumulation Cycle



How To Mutate?

Primitive Data Types Mutator (uint64_t)

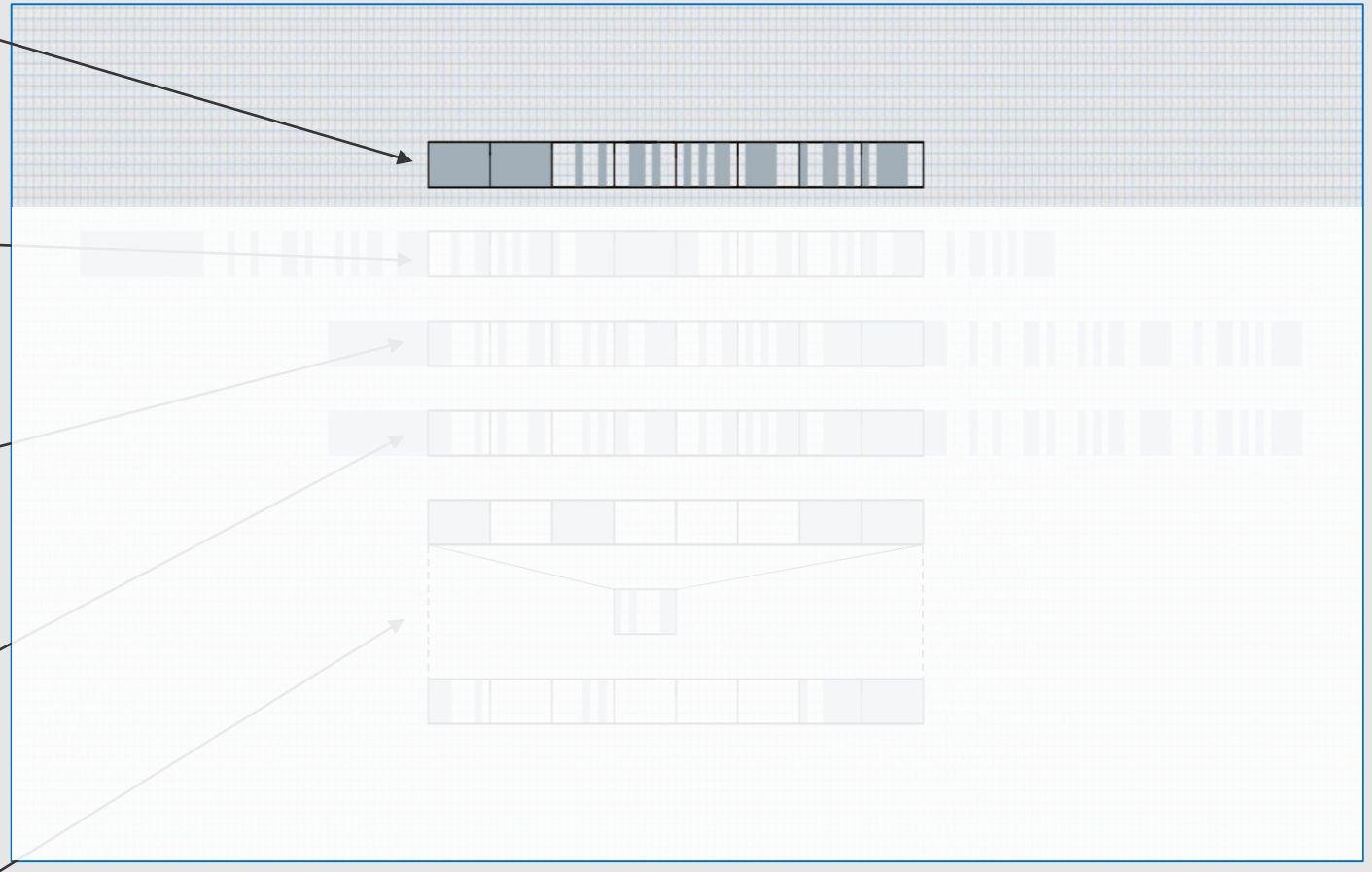
CRandomMutator:
`rand_64() -> 0xFFFF123456789ABC`

CSlidingMutator:
`sliding(45) -> 0x13579FFFE2468ACF`

CSwapHalfMutator:
`swap_half() -> 0xE2468ACF13579FFF`

CBitFlipMutator:
`bitflip(47) -> 0xE2460ACF13579FFF`

CMaskMutator:
`mask(0xA3) -> 0xE2000A0000009FFF`



Primitive Data Types Mutator (uint64_t)

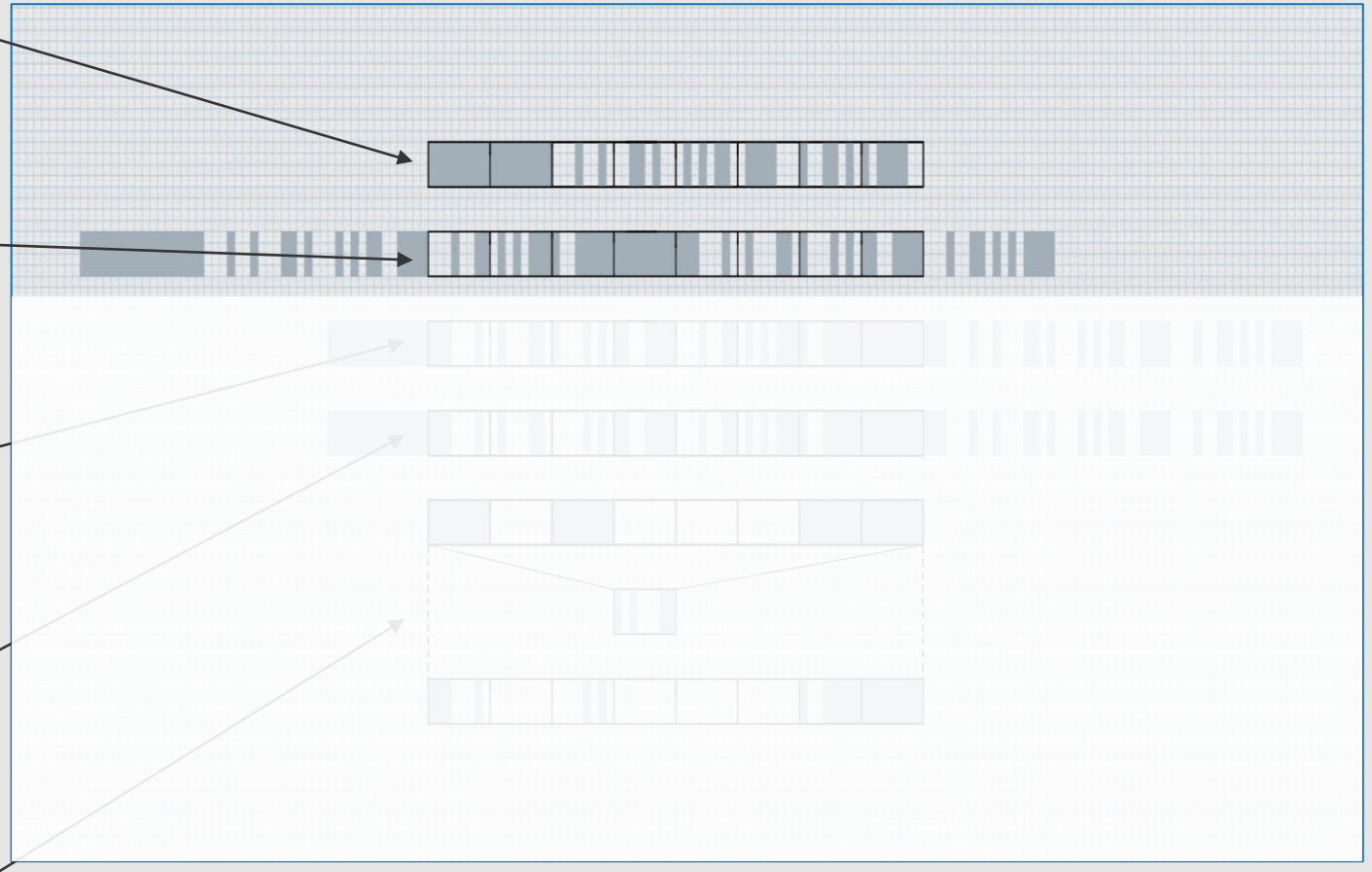
CRandomMutator:
`rand_64() -> 0xFFFF123456789ABC`

CSlidingMutator:
`sliding(45) -> 0x13579FFFE2468ACF`

CSwapHalfMutator:
`swap_half() -> 0xE2468ACF13579FFF`

CBitFlipMutator:
`bitflip(47) -> 0xE2460ACF13579FFF`

CMaskMutator:
`mask(0xA3) -> 0xE2000A0000009FFF`



Primitive Data Types Mutator (uint64_t)

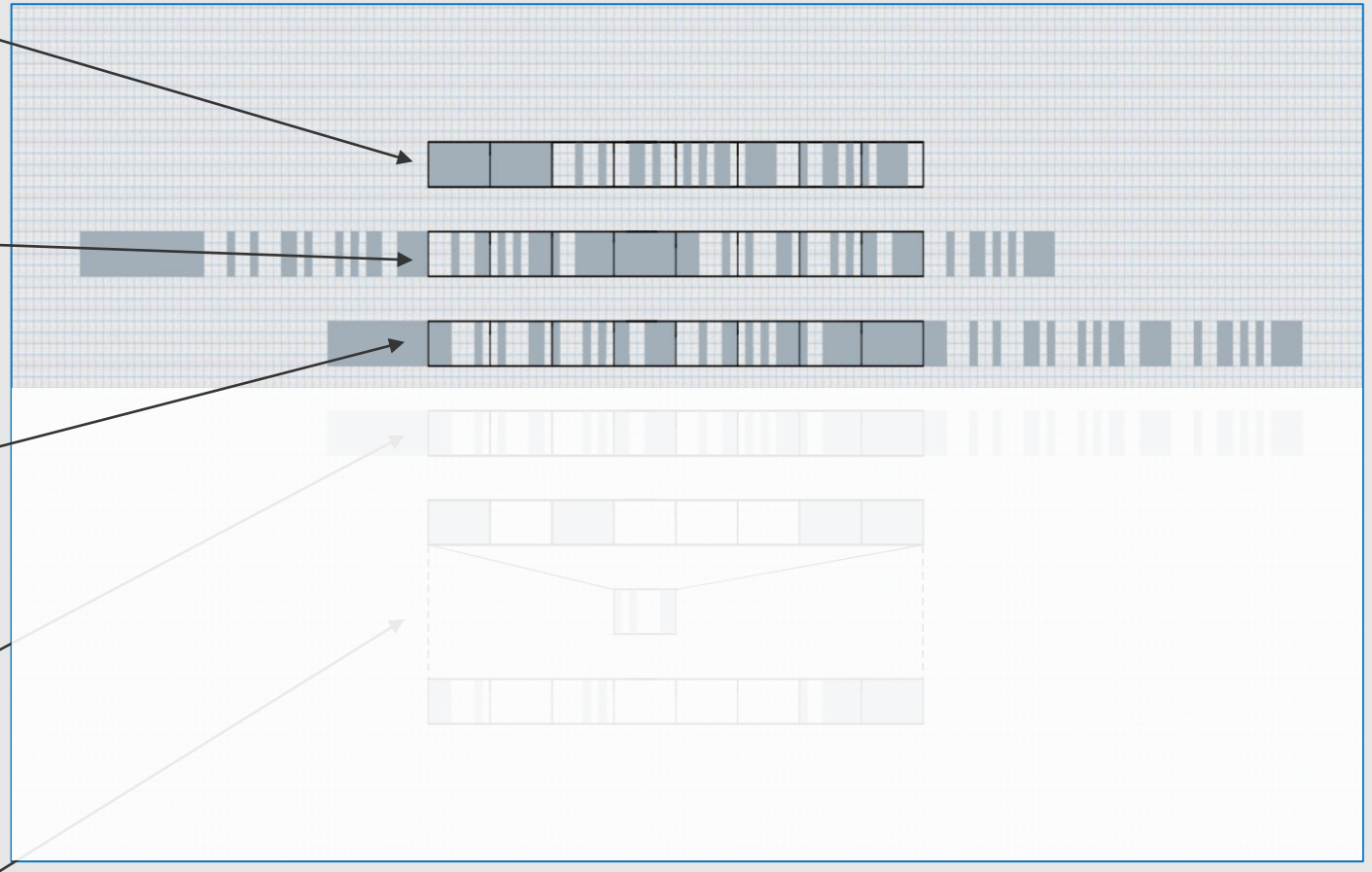
CRandomMutator:
`rand_64() -> 0xFFFF123456789ABC`

CSlidingMutator:
`sliding(45) -> 0x13579FFFE2468ACF`

CSwapHalfMutator:
`swap_half() -> 0xE2468ACF13579FFF`

CBitFlipMutator:
`bitflip(47) -> 0xE2460ACF13579FFF`

CMaskMutator:
`mask(0xA3) -> 0xE2000A0000009FFF`



Primitive Data Types Mutator (uint64_t)

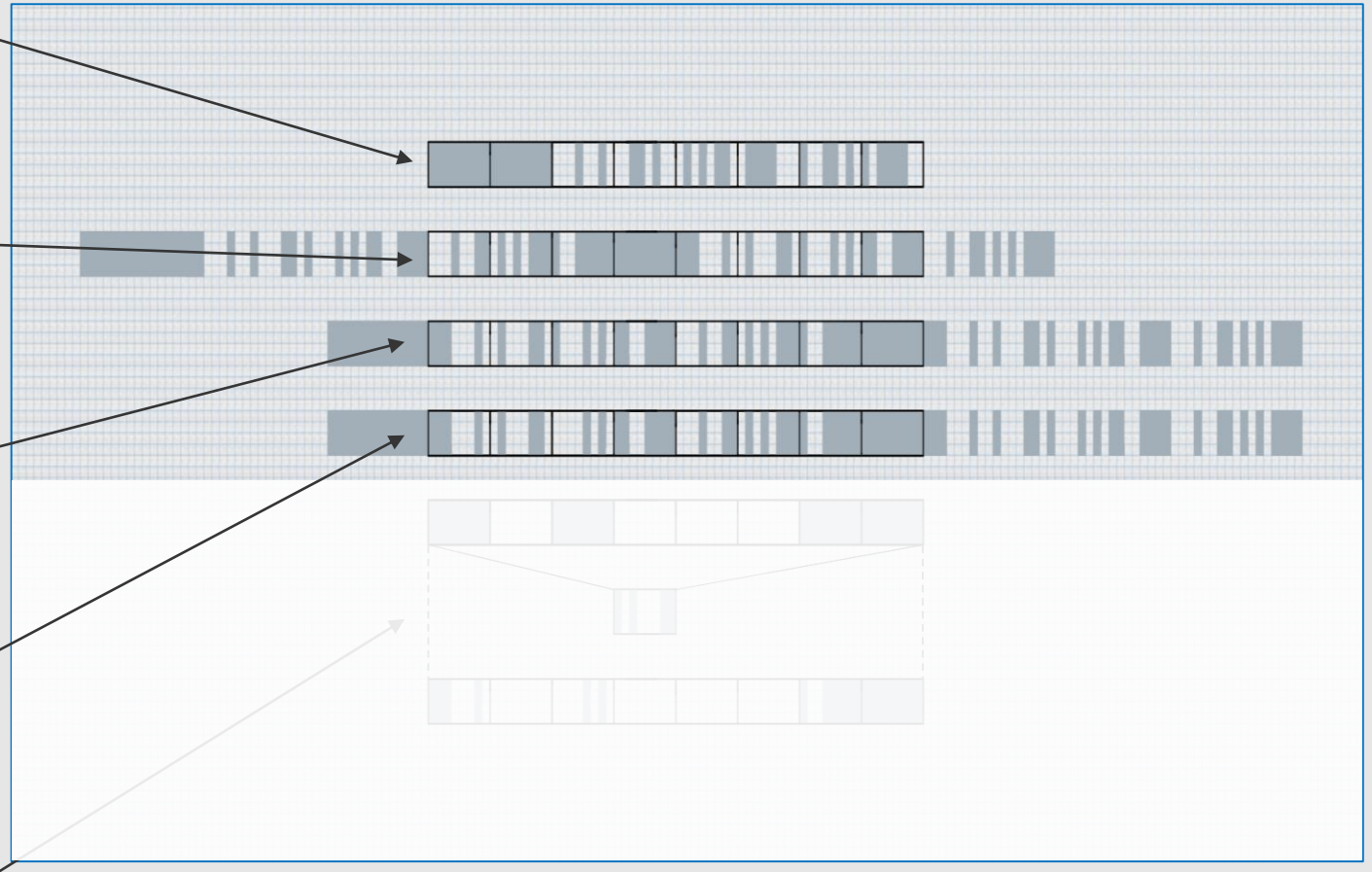
CRandomMutator:
rand_64() -> 0xFFFF123456789ABC

CSlidingMutator:
sliding(45) -> 0x13579FFFE2468ACF

CSwapHalfMutator:
swap_half() -> 0xE2468ACF13579FFF

CBitFlipMutator:
bitflip(47) -> 0xE2460ACF13579FFF

CMaskMutator:
mask(0xA3) -> 0xE2000A0000009FFF



Primitive Data Types Mutator (uint64_t)

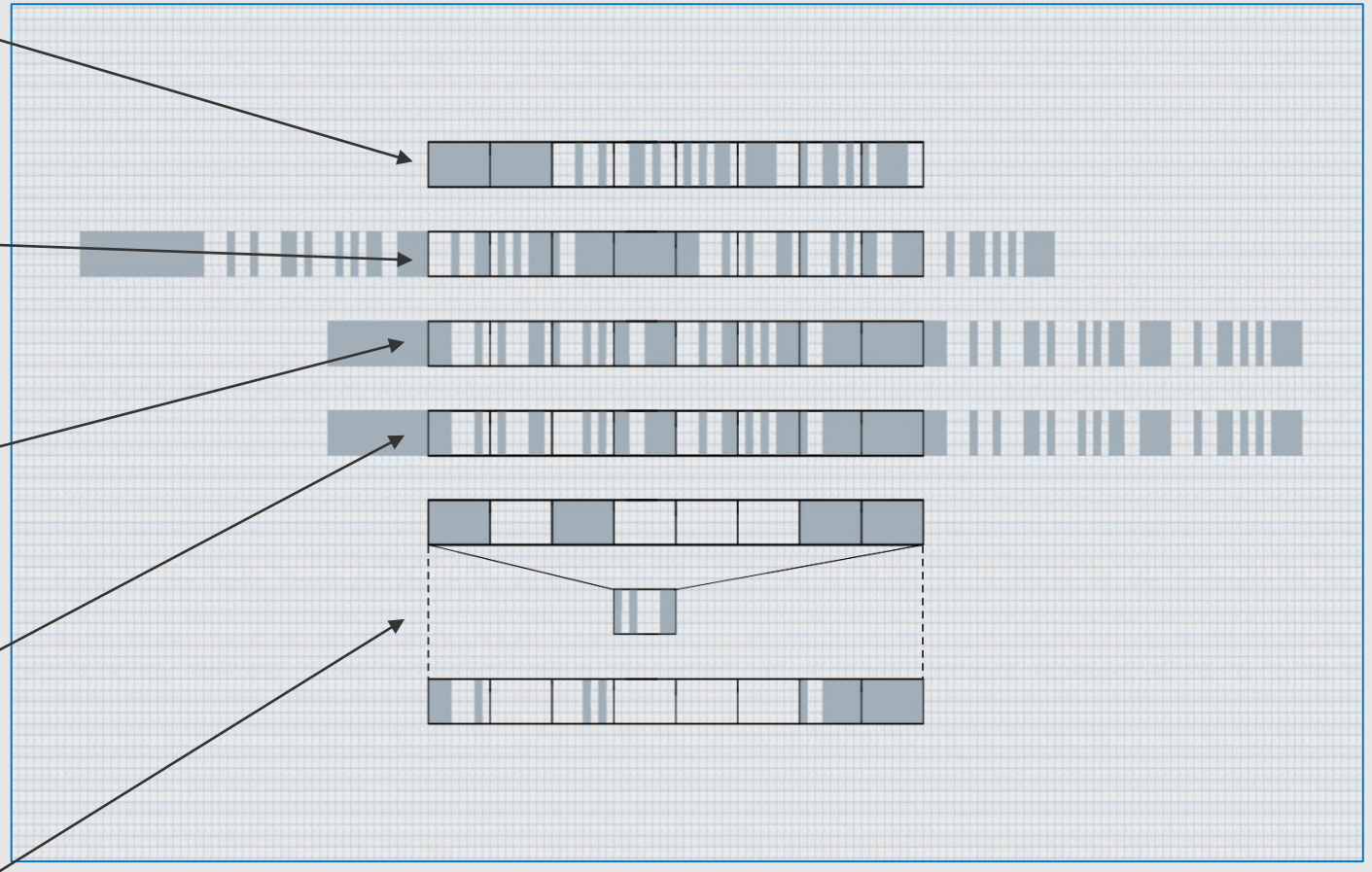
CRandomMutator:
`rand_64() -> 0xFFFF123456789ABC`

CSlidingMutator:
`sliding(45) -> 0x13579FFFE2468ACF`

CSwapHalfMutator:
`swap_half() -> 0xE2468ACF13579FFF`

CBitFlipMutator:
`bitflip(47) -> 0xE2460ACF13579FFF`

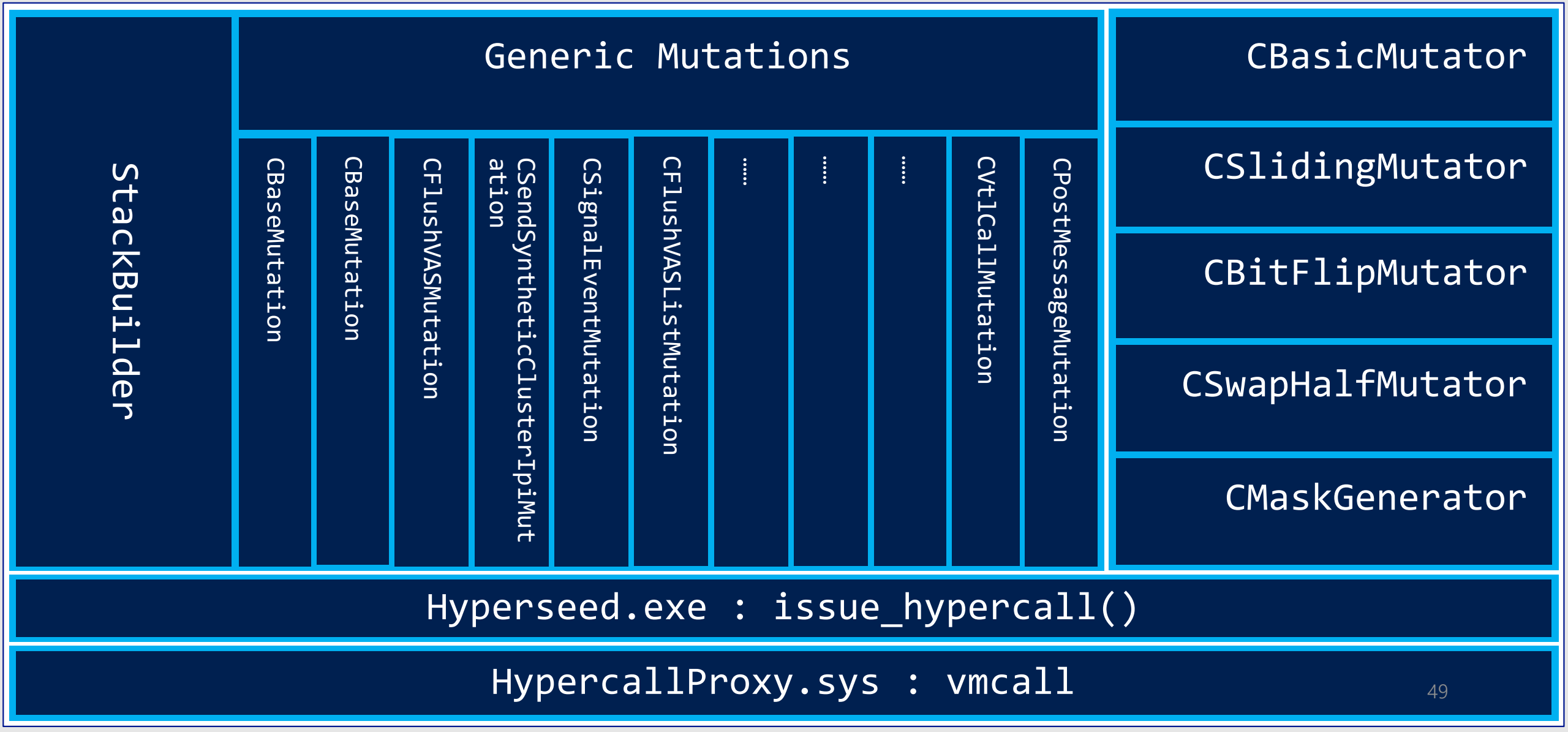
CMaskMutator:
`mask(0xA3) -> 0xE2000A0000009FFF`



Mutator Dispatcher

```
class CMutatorDispatcher
{
public:
    template<class T>
    static bool mutateBasicType(T& data, size_t hint = 0)
    {
        size_t bits = CPatternGenerator::Instance()->next_rand();
        size_t index = 0;

        if (is_bit_set(bits, index++)) CRandomMutator::mutateBasicType<T>(data, hint);
        if (is_bit_set(bits, index++)) CBitFlipMutator::mutateBasicType<T>(data, 1);
        if (is_bit_set(bits, index++)) CSlidingMutator::mutateBasicType<T>(data, hint);
        if (is_bit_set(bits, index++)) CBitFlipMutator::mutateBasicType<T>(data, 2);
        if (is_bit_set(bits, index++)) CSwapHalfMutator::mutateBasicType<T>(data, hint);
        if (is_bit_set(bits, index++)) CBitFlipMutator::mutateBasicType<T>(data, 4);
        if (is_bit_set(bits, index++)) CBitFlipMutator::mutateBasicType<T>(data, 8);
        if (is_bit_set(bits, index++)) CMaskMutator::mutateBasicType<T>(data, hint);

        return true;
    }
}
```

Hyperseed Components



CFlushVASMutation

Wrapper Interface

```
HV_STATUS
HvFlushVirtualAddressSpace(
      __in HV_ADDRESS_SPACE_ID      AddressSpace,
      __in HV_FLUSH_FLAGS     Flags,
      __in UINT64 ProcessorMask
      );
```

Native Interface

HvFlushVirtualAddressSpace	
	Call Code = 0x0002
➡ Input Parameters	
0	AddressSpace (8 bytes)
8	Flags (8 bytes)
16	ProcessorMask (8 bytes)

```
#define SELECT              CPatternGenerator::Instance()->select
#define FROM_RANGE          CPatternGenerator::Instance()->from_range
#define RAND_8()            CPatternGenerator::Instance()->next_rand()
#define RAND_16()           CPatternGenerator::Instance()->next_rand_word()
#define RAND_32()           CPatternGenerator::Instance()->next_rand_dword()
#define RAND_64()           CPatternGenerator::Instance()->next_rand_qword()
#define BM(x)               CMutatorDispatcher::mutateBasicType(x)
#define BMN0(x)             CMutatorDispatcher::mutateBasicType_Nonzero(x)
#define DM(x, y)            CMutatorDispatcher::mutateData(x, y)
```

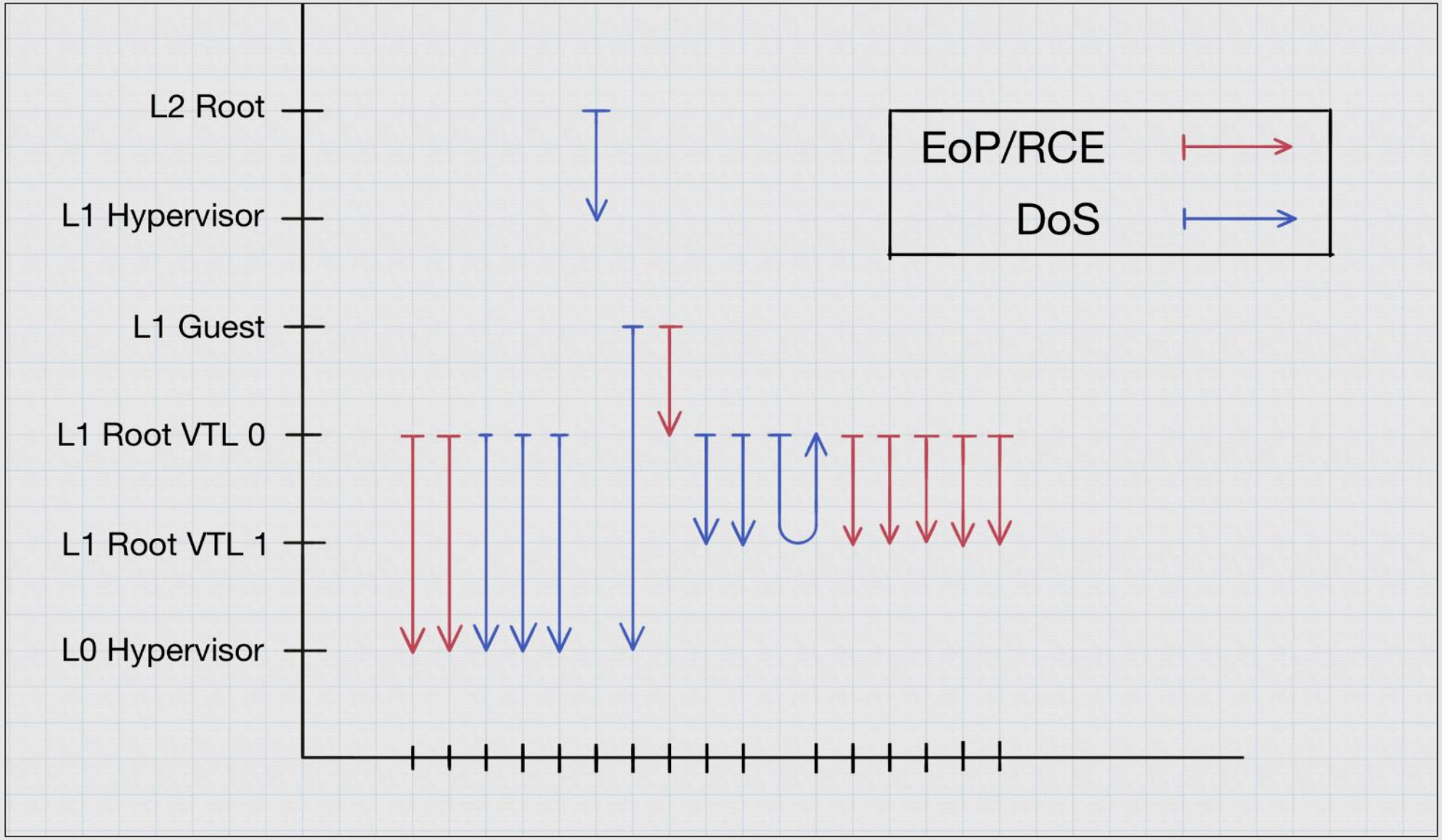
```
template<>
bool mutateCustom<HvCallFlushVirtualAddressSpaceTrait>(CBaseMutation* Mutation,
                  HvCallFlushVirtualAddressSpaceBuffer& m_HypercallBuffer,
                  shared_ptr<CBaseFeed> accum_feed)
{
    PHV_INPUT_FLUSH_VIRTUAL_ADDRESS_SPACE input = m_HypercallBuffer.InputBuffer();

    BM(input->Header.AddressSpace);
    // #define HV_FLUSH_ALL_PROCESSORS              (0x00000001)
    // #define HV_FLUSH_ALL_VIRTUAL_ADDRESS_SPACES  (0x00000002)
    // #define HV_FLUSH_NON_GLOBAL_MAPPINGS_ONLY    (0x00000004)
    // #define HV_FLUSH_USE_EXTENDED_RANGE_FORMAT   (0x00000008)
    input->Header.Flags = RAND_8() & 0x07;
    BMN0(input->Header.ProcessorMask);

    accum_feed->set_input(string((char*)input, m_HypercallBuffer.input_len));
    return true;
}
```

Findings

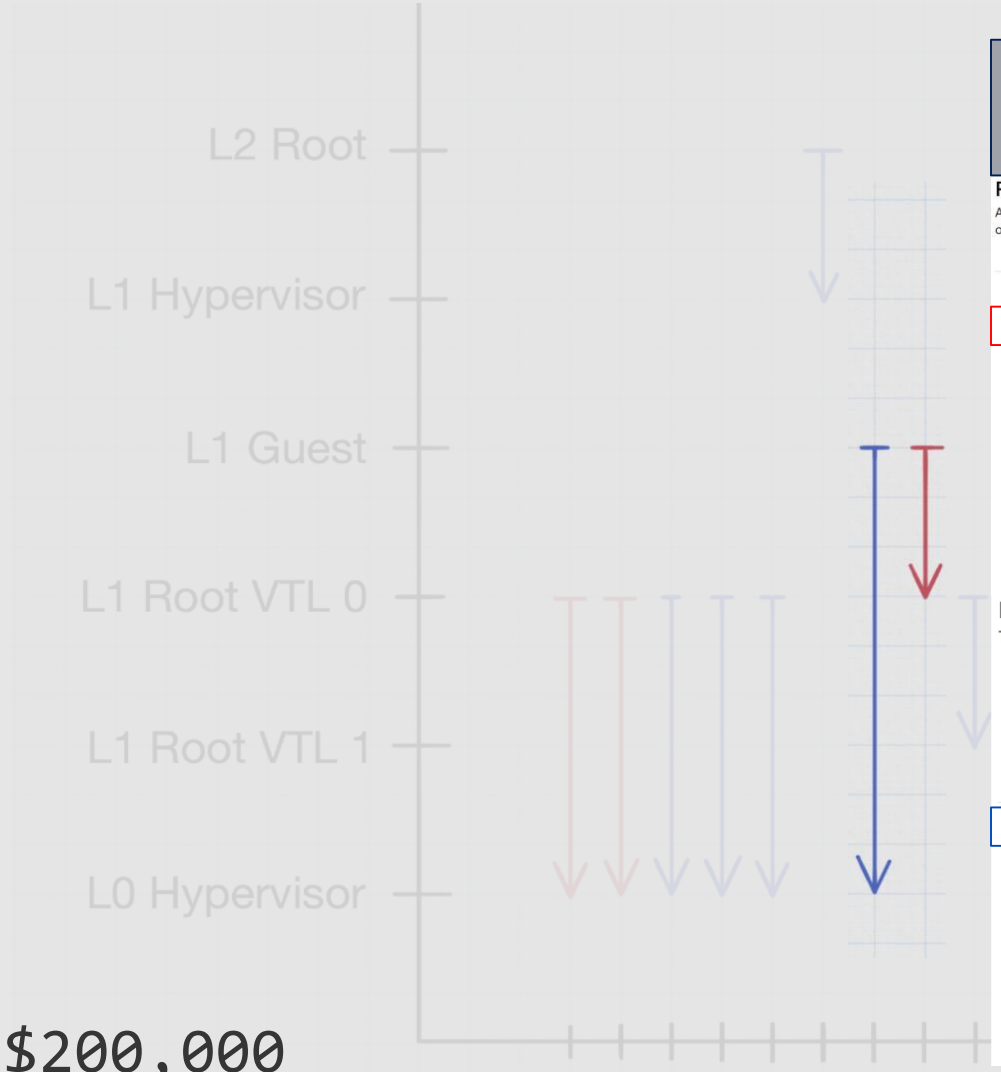
Fuzzing Stats

- Initiated 1 Year Ago
- 2 Contributors
- 350+ Source Files, 16K+ Lines of Code
- 169 Hypercalls Mutated
- 16 Findings So Far
  - 8 EoP/RCE
  - 8 DoS
  - 2 Qualified for Hyper-V Bounty
  - 7 Qualified for WIP Bounty

Findings


L2 Root
L1 Hypervisor
L1 Guest
L1 Root VTL 0
L1 Root VTL 1
L0 Hypervisor
EoP/RCE
DoS

How Much Are They Worth? (Hyper-V Bounty)



FROM: https://www.microsoft.com/en-us/msrc/bounty-hyper-v

Remote Code Execution

An eligible submission includes a RCE vulnerability in Microsoft Hyper-V that enables a guest virtual machine to compromise the hypervisor, escape from a guest virtual machine to the host, or escape from one guest virtual machine to another guest virtual machine.

Vulnerability Type	Tier	Proof of concept	Functioning Exploit	Report Quality	Payout range (USD)*
RCE	Tier 1	Required	Yes	High	$250,000
			No	High	$200,000
			No	Low	$50,000
RCE	Tier 2	Required	Yes	High	$150,000
			No	High	$100,000
			No	Low	$25,000
RCE	Tier 3	Required	Yes	High	$20,000
			No	High	$15,000
			No	Low	$5,000

Denial of Service and Information Disclosure

The vulnerability should result in one of the following:

- Crash the host machine, resulting in a denial of service condition
- Cause a failure to start and stop VMs
- Gain sensitive information from the host machine or another guest

Vulnerability Type	Tier	Proof of concept	Report Quality	Payout range (USD)
DOS	Tier 1	Required	High	$15,000
			Low	$5,000
Info Disclosure	Tier 1	Required	High	$25,000
			Low	$5,000
	Tier 2	Required	High	$15,000
			Low	$5,000

✓ 1 x $200,000

✓ 1 x $15,000

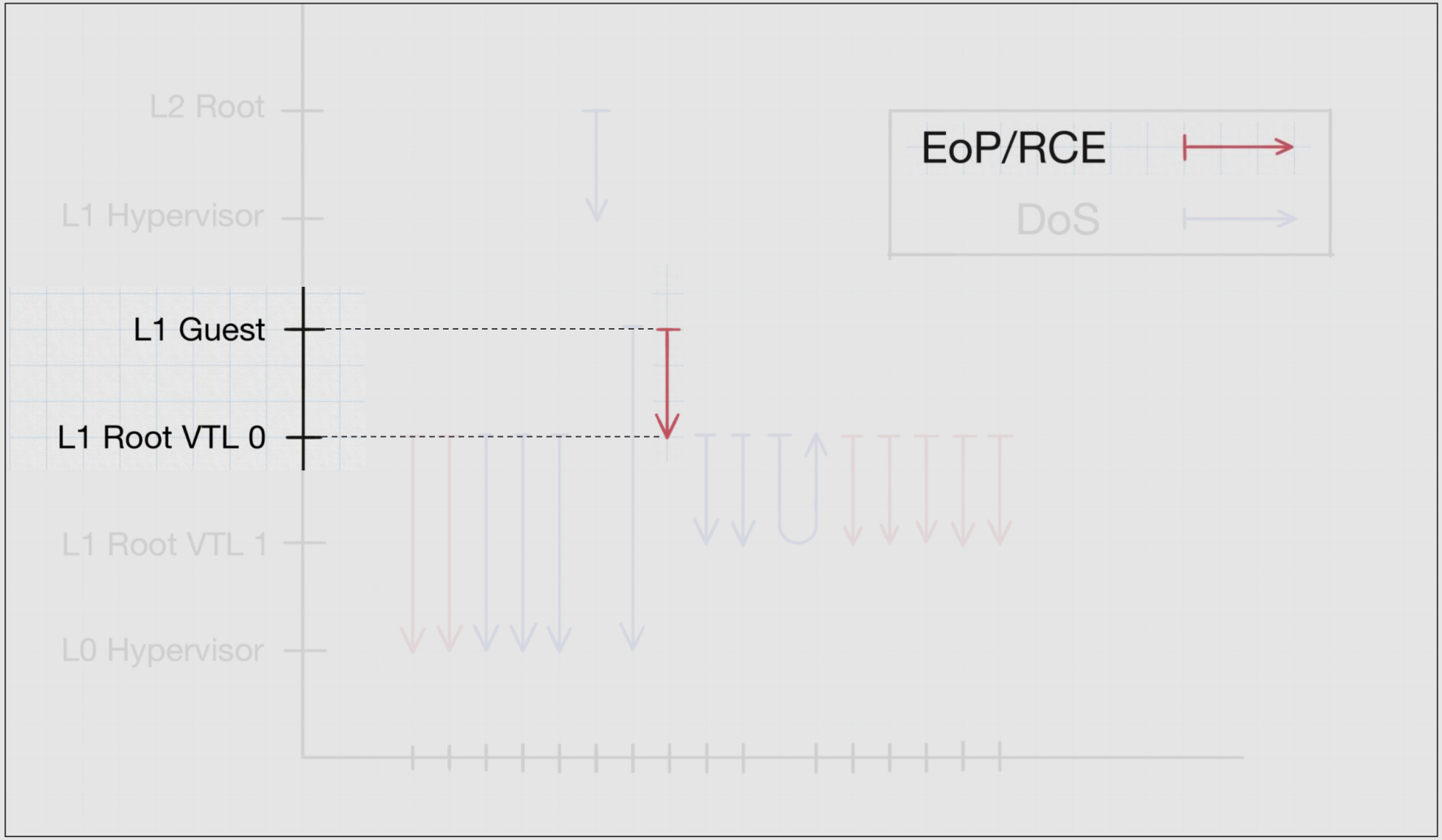
How Much Are They Worth? (WIP Bounty)

FROM: https://www.microsoft.com/en-us/msrc/bounty-windows-insider-preview

Impact	Report Quality	Severity			
		Critical	Important	Moderate	Low
Remote Code Execution	High Medium Low	$30,000 $12,000 $3,000	$25,000 $8,000 $2,000	N/A	N/A
Elevation of Privilege	High Medium Low	N/A	$20,000 $8,000 $2,000	$0	N/A
Security Feature Bypass	High Medium Low	N/A	$20,000 $8,000 $2,000	$0	$0
Information Disclosure	High Medium Low	N/A	$10,000 $3,000 $1,000	$0	$0

- ✓ 2 x $20,000
- ✓ 5 x $20,000

CVE-2018-8439



CVE-2018-8439

FROM: https://portal.msrc.microsoft.com/en-US/security-guidance/advisory/CVE-2018-8439

CVE-2018-8439 | Windows Hyper-V Remote Code Execution Vulnerability

Security Vulnerability

Published: 09/11/2018
MITRE CVE-2018-8439

A remote code execution vulnerability exists when Windows Hyper-V on a host server fails to properly validate input from an authenticated user on a guest operating system. To exploit the vulnerability, an attacker could run a specially crafted application on a guest operating system that could cause the Hyper-V host operating system to execute arbitrary code.

An attacker who successfully exploited the vulnerability could execute arbitrary code on the host operating system.

The security update addresses the vulnerability by correcting how Hyper-V validates guest operating system user input.

On this page

Exploitability Assessment

The following table provides an exploitability assessment for this vulnerability at the time of original publication.

Publicly Disclosed	Exploited	Latest Software Release	Older Software Release	Denial of Service
No	No	2 - Exploitation Less Likely	2 - Exploitation Less Likely	Not Applicable

Acknowledgements

Daniel King (@long123king), MSRC Microsoft

CVE-2018-8439

HvPostMessage

- The hypervisor provides a simple inter-partition communication facility that allows one partition to send a parameterized message to another partition. (Because the message is sent asynchronously, it is said to be posted.) The destination partition may be notified of the arrival of this message through an SynIC(Synthetic Interrupt Controller) interrupt.

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

Wrapper Interface

```
HV_STATUS
HvPostMessage(
      __in HV_CONNECTION_ID   ConnectionId,
      __in HV_MESSAGE_TYPE    MessageType,
      __in UINT32 PayloadSize,
      __in_ecount(PayloadSize)
           PCVOID Message
      );
```

Native Interface

HvPostMessage		
	Call Code = 0x005C	
➡ Input Parameters		
0	ConnectionId (4 bytes)	Padding (4 bytes)
8	MessageType (4 bytes)	PayloadSize (4 bytes)
16	Message[0] (8 bytes)	
⋮	⋮	
248	Message[29] (8 bytes)	

CVE-2018-8439

- Reproduce Steps
  I. Run Hyperseed inside guest VM to keep fuzzing HvPostMessage with ChannelMessageOpenChannel(0x05)/ChannelMessageCloseChannel(0x07) continuously.
  II. Manually reset guest VM from host.

- Result
  I. Host BSOD

CVE-2018-8439

```
0: kd> !analyze -v
*******************************************************************************
*                                                                             *
*                        Bugcheck Analysis                                    *
*                                                                             *
*******************************************************************************

DRIVER_IRQL_NOT_LESS_OR_EQUAL (d1)
An attempt was made to access a pageable (or completely invalid) address at an
interrupt request level (IRQL) that is too high.  This is usually
caused by drivers using improper addresses.
If kernel debugger is available get stack backtrace.
Arguments:
Arg1: ffffba81d8001008, memory referenced
Arg2: 0000000000000002, IRQL
Arg3: 0000000000000001, value 0 = read operation, 1 = write operation
Arg4: fffff80fb4b51122, address which referenced memory
0: kd> .trap 0xfffff58dbc6ca7a0
NOTE: The trap frame does not contain all registers.
Some register values may be zeroed or incorrect.
rax=ffff8780eb36adc0 rbx=0000000000000000 rcx=0000000000000000
rdx=ffffba81d8001008 rsi=0000000000000000 rdi=0000000000000000
rip=fffff80fb4b51122 rsp=fffff58dbc6ca930 rbp=0000000080000005
 r8=0000000000000001  r9=0000000000000000 r10=000000000000a000
r11=ffffba81d7a6a010 r12=0000000000000000 r13=0000000000000000
r14=0000000000000000 r15=0000000000000000
iopl=0         nv up ei ng nz na pe nc
vmbusr!BusChSendInterrupt+0x42:
fffff80f`b4b51122 f00fab0a        lock bts dword ptr [rdx],ecx ds:ffffba81`d8001008=????????
0: kd> kf
  *** Stack trace for last set context - .thread/.cxr resets it
 #   Memory  Child-SP          RetAddr           Call Site
00           fffff58d`bc6ca930 fffff80f`b5542e5c vmbusr!BusChSendInterrupt+0x42
01        30 fffff58d`bc6ca960 fffff80f`b55429a5 vmbkmclr!InCompletePacket+0x4ac
02        c0 fffff58d`bc6caa20 fffff80f`b72f112b vmbkmclr!VmbChannelPacketComplete+0x15
03        30 fffff58d`bc6caa50 fffff80f`b55413a2 storvsp+0x112b
04        60 fffff58d`bc6caab0 fffff80f`b4b55680 vmbkmclr!InpProcessingWorkerRoutine+0x212
05        80 fffff58d`bc6cab30 fffff800`d28fdae7 vmbusr!AwWorkerThread+0xb0
06        60 fffff58d`bc6cab90 fffff800`d29bbb86 nt!PspSystemThreadStartup+0x47
07        50 fffff58d`bc6cabe0 00000000`00000000 nt!KiStartSystemThread+0x16
0: kd> !pte rdx
                                           VA ffffba81d8001008
PXE at FFFF98CC66331BA8    PPE at FFFF98CC66375038    PDE at FFFF98CC6EA07600    PTE at FFFF98DD40EC0008
contains 0A0000000515D863  contains 0A0000000515E863  contains 0A0000023ACB5863  contains 001D845D00000000
pfn 515d      ---DA--KWEV  pfn 515e      ---DA--KWEV  pfn 23acb5    ---DA--KWEV  not valid
                                                                                   Page has been freed
```

```
0: kd> dt vmbusr!_HV_MONITOR_PAGE -b
   +0x000 TriggerState     : _HV_MONITOR_TRIGGER_STATE
      +0x000 AsUINT32          : Uint4B
      +0x000 GroupEnable       : Pos 0, 4 Bits
      +0x000 RsvdZ             : Pos 4, 28 Bits
   +0x004 RsvdZ1           : Uint4B
   +0x008 TriggerGroup     : _HV_MONITOR_TRIGGER_GROUP
      +0x000 AsUINT64          : Uint8B
      +0x000 Pending           : Uint4B
      +0x004 Armed             : Uint4B
   +0x028 RsvdZ2           : Uint8B
   +0x040 NextCheckTime    : Int4B
   +0x240 Latency          : Uint2B
   +0x340 RsvdZ3           : Uint8B
   +0x440 Parameter        : _HV_MONITOR_PARAMETER
      +0x000 ConnectionId      : _HV_CONNECTION_ID
         +0x000 AsUINT32          : Uint4B
         +0x000 Id                : Pos 0, 24 Bits
         +0x000 Reserved          : Pos 24, 8 Bits
      +0x004 FlagNumber        : Uint2B
      +0x006 RsvdZ             : Uint2B
   +0x840 RsvdZ4           : UChar
```

CVE-2018-8439

- Monitored Notification Page
  - Allocated in vmbusr!ParentCreateMonitors
    - When initializing VMBus connection from the client
  - Freed in vmbusr!ParentDeleteMonitors
    - When resetting, destroying partition or failing VMBus connection
  - Cache a reference in vmbusr!PncEnableInterrupt
    - BusFdoOpenChannel, holds the reference even if the channel is not opened successfully
  - Clear the cached reference in vmbusr!PncDisableInterrupt
    - BusFdoCloseChannel, won't call this routine if it is not opened successfully
  - Accessing the cached reference after being freed leads to UAF

CVE-2018-8439 (Fix)

- Monitored Notification Page
  - Allocated in vmbusr!ParentCreateMonitors
    - When initializing VMBus connection
  - Freed in vmbusr!ParentDeleteMonitors
    - When resetting, destroying partition or failing VMBus connection
  - Cache a reference in vmbusr!PncEnableInterrupt
    - BusFdoOpenChannel, holds the reference even if the channel not opened successfully
  - Clear the cached reference in vmbusr!PncDisableInterrupt
    - BusFdoCloseChannel, won't be called if the channel is not opened successfully
    - BusFdoOpenResult, clear the cached reference if the channel is not opened properly.

Future

Future

- Keep Improving Format-Aware Fuzzing
- Adopt Coverage-Guided Fuzzing
  - Intel PT & AFL
- More Details on Findings After Being Patched
- Open-Sourcing
- More Targets

Acknowledgements

- Microsoft Security Response Center
- CDG Platform Security & Vulnerability Research
- Hyper-V Development Team

Appendix

Past Research

- Security Assessment of Microsoft Hyper-V (ERNW Newsletter 43 / May 2014, Felix Wilhelm, Mattias Luft)
- Ring 0 to Ring -1 (Syscan 2015 – Alex Ionescu)
- The Battle of SKM and IUM (Blackhat 2015 - Alex Ionescu)
- Analysis of the Attack Surface of Windows 10 Virtualization-Base Security (Blackhat 2016 - Rafal Wojtczuk)
- VBS and VSM Internals (Saar Amar)
- A Dive in to Hyper-V Architecture and Vulnerabilities (Blackhat 2018 - Joe Bialek & Nicolas Joly)
- Hardening Hyper-V through offensive security research (Blackhat 2018 – Jordan Rabet)
- First steps in Hyper-V research (Saar Amar)
- Fuzzing para-virtualized devices in Hyper-V (MSFT Virtualization Security Team)
- Writing a Hyper-V “Bridge” for Fuzzing (Alex Ionescu)

Useful Public Resources

- Hypervisor Top-Level Functional Specification
  - https://docs.microsoft.com/en-us/virtualization/hyper-v-on-windows/reference/tlfs
- Hyper-V Debugging Symbols
  - https://blogs.technet.microsoft.com/virtualization/2018/04/25/hyper-v-symbols-for-debugging/
- Linux Integration Service for Microsoft Hyper-V
  - https://github.com/LIS
- Unofficial hdk – Hyper-V Development Kit
  - https://ionescu007.github.io/hdk/
- Report quality definitions for Microsoft's Bug Bounty programs
  - https://www.microsoft.com/en-us/msrc/bounty-example-report-submission

Q&A