
长亭科技
CHAITIN
Box Escape: Discovering 10+ Vulnerabilities in VirtualBox
ChenNan
2020
CHAITIN

About Me

- Security Researcher of Chaitin Security Research Lab
- Virtualization, IOT and Kernel Bug Hunting & Exploit
- Microsoft Most Valuable Researcher of 2019
- Speaker of 44Con, CSS2018 and insomnihack.

About Chaitin

- Beijing Chaitin Tech Co., Ltd(@ChaitinTech)

  https://chaitin.cn/en

  https://realworldctf.com/

- Chaitin Security Research Lab
  - Pwn2Own 2017 3rd place
  - GeekPwn 2015/2016/2018/2019 awardees
    - PS4 Jailbreak, Android rooting, IoT Offensive Research, ESXi Escape
  - CTF players from team b1o0p, Tea Deliverers
    - 2nd place at DEFCON 2016
    - 3rd place at DEFCON 2019
    - 1st place at HITCON 2019
    - 4st place at DEFCON 2020

Agenda

01 VirtualBox Overview

02 Bug Hunting

03 Case Study

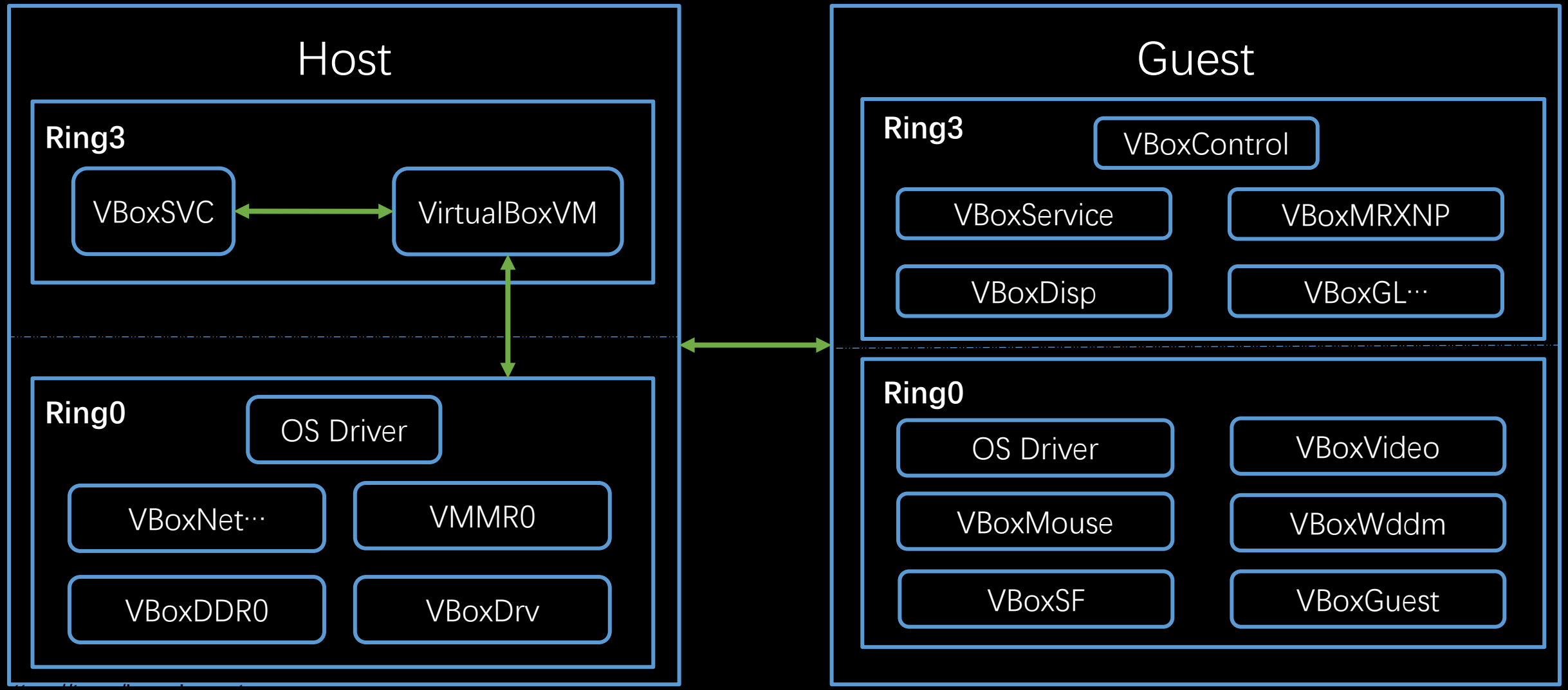
04 Demo Time

PART 1

VirtualBox Overview

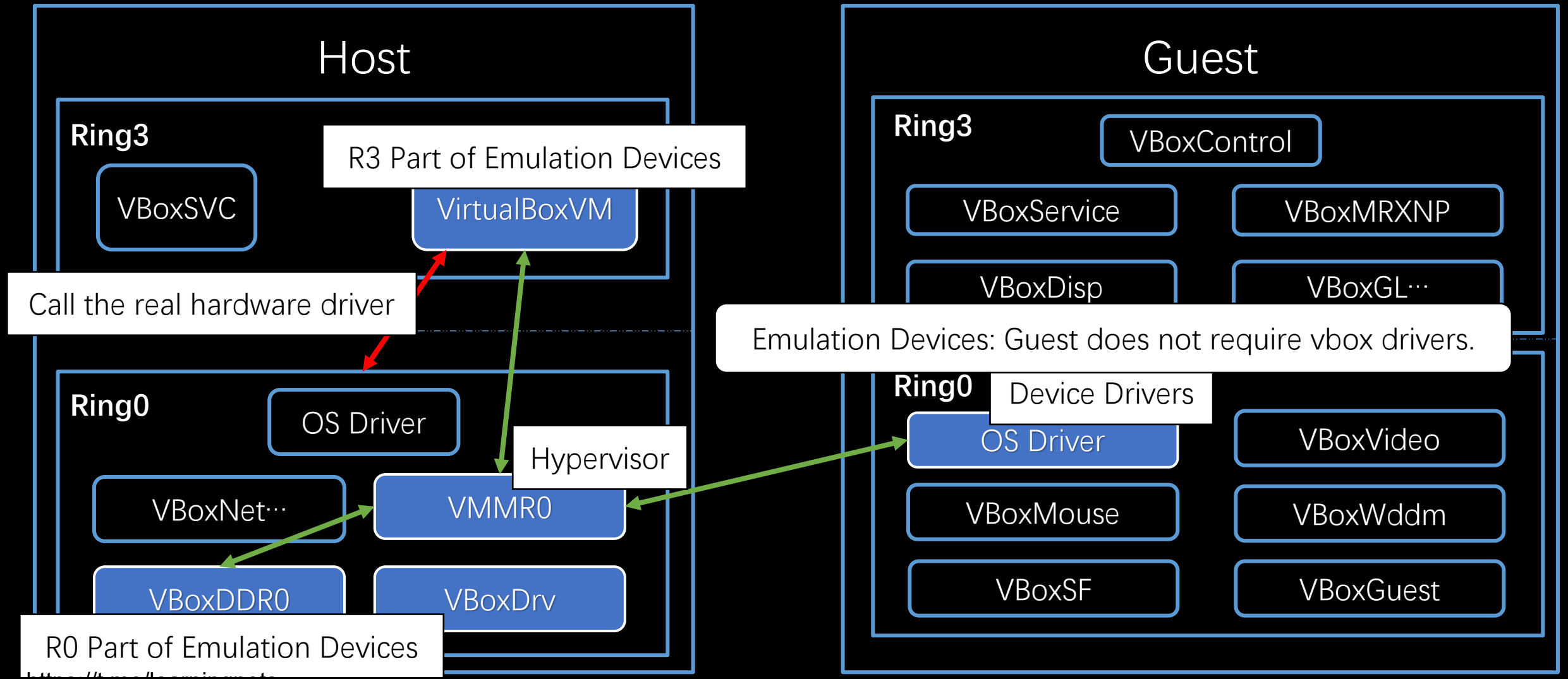
VirtualBox Architecture

VirtualBox Architecture

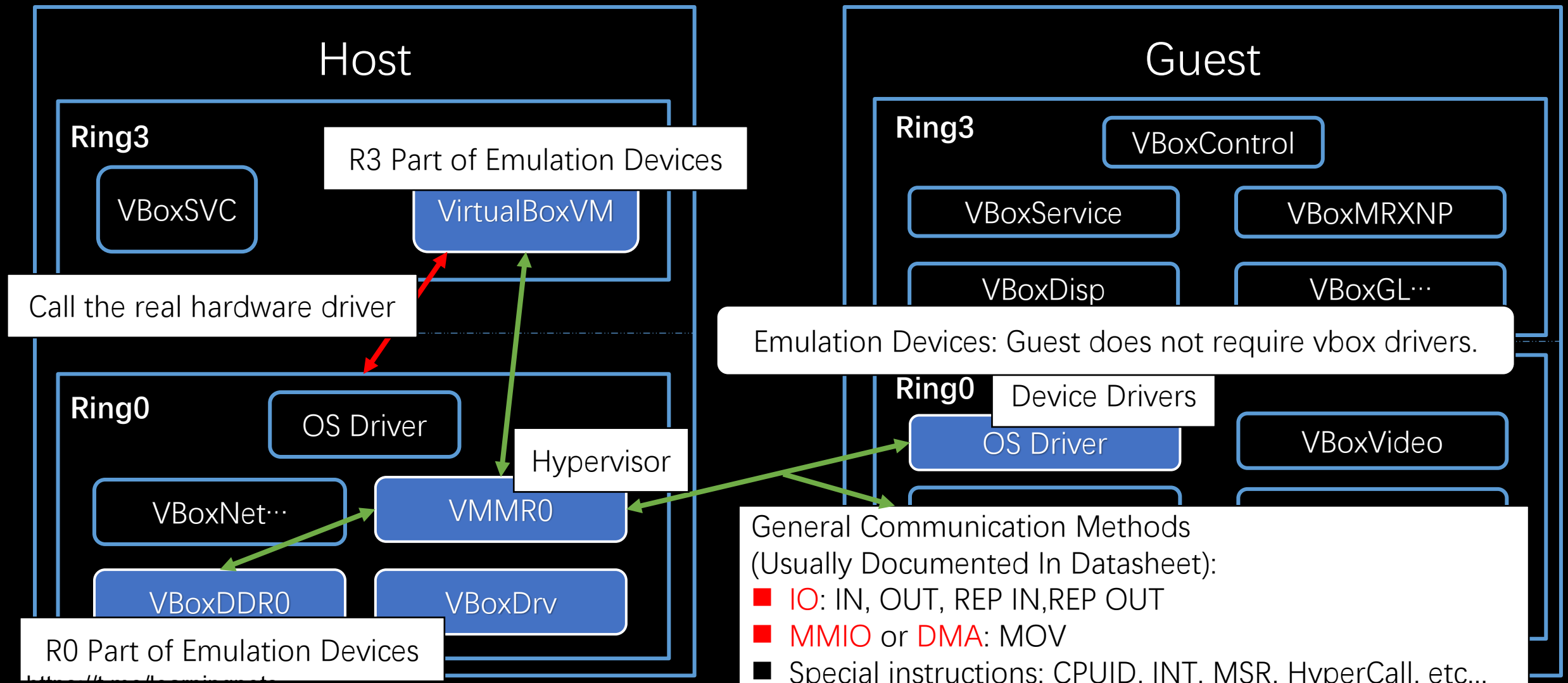


General Communication

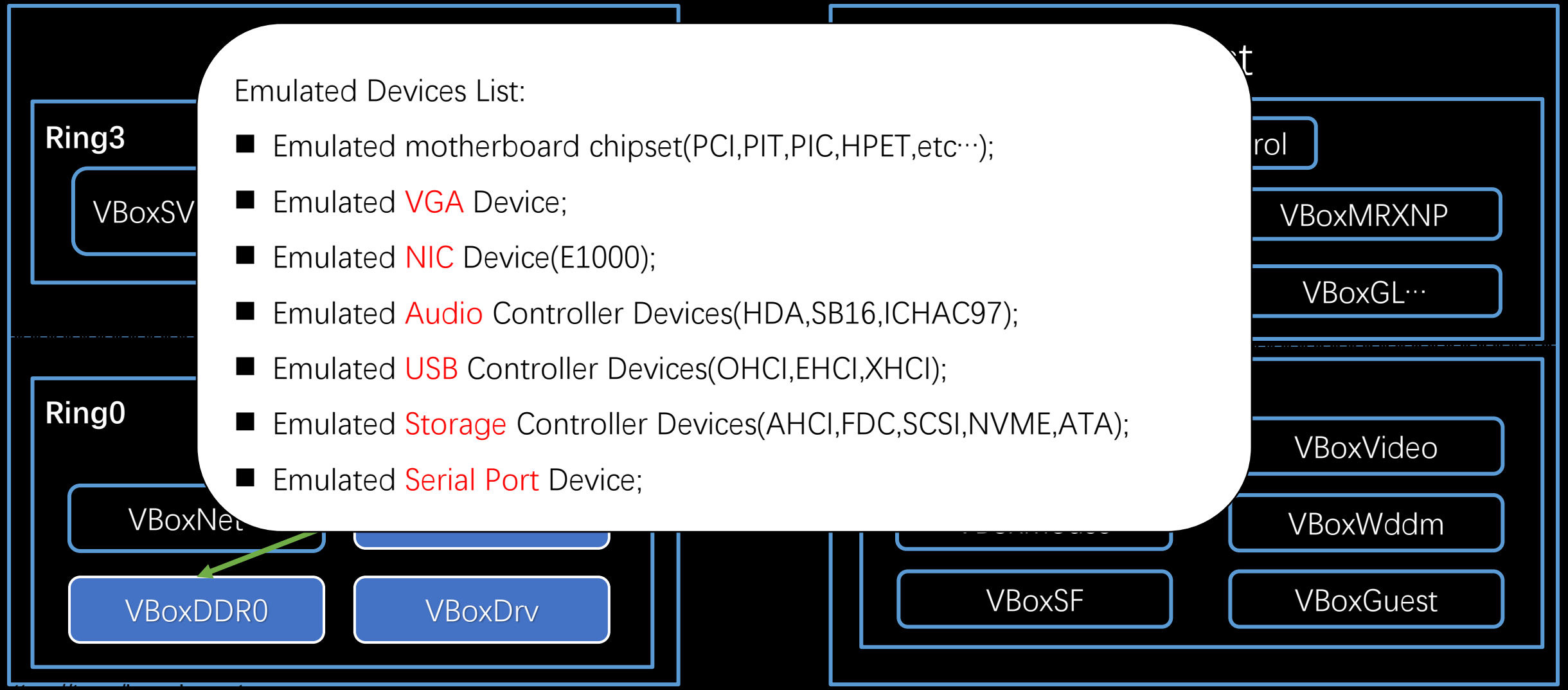
General Communication



General Communication



General Communication

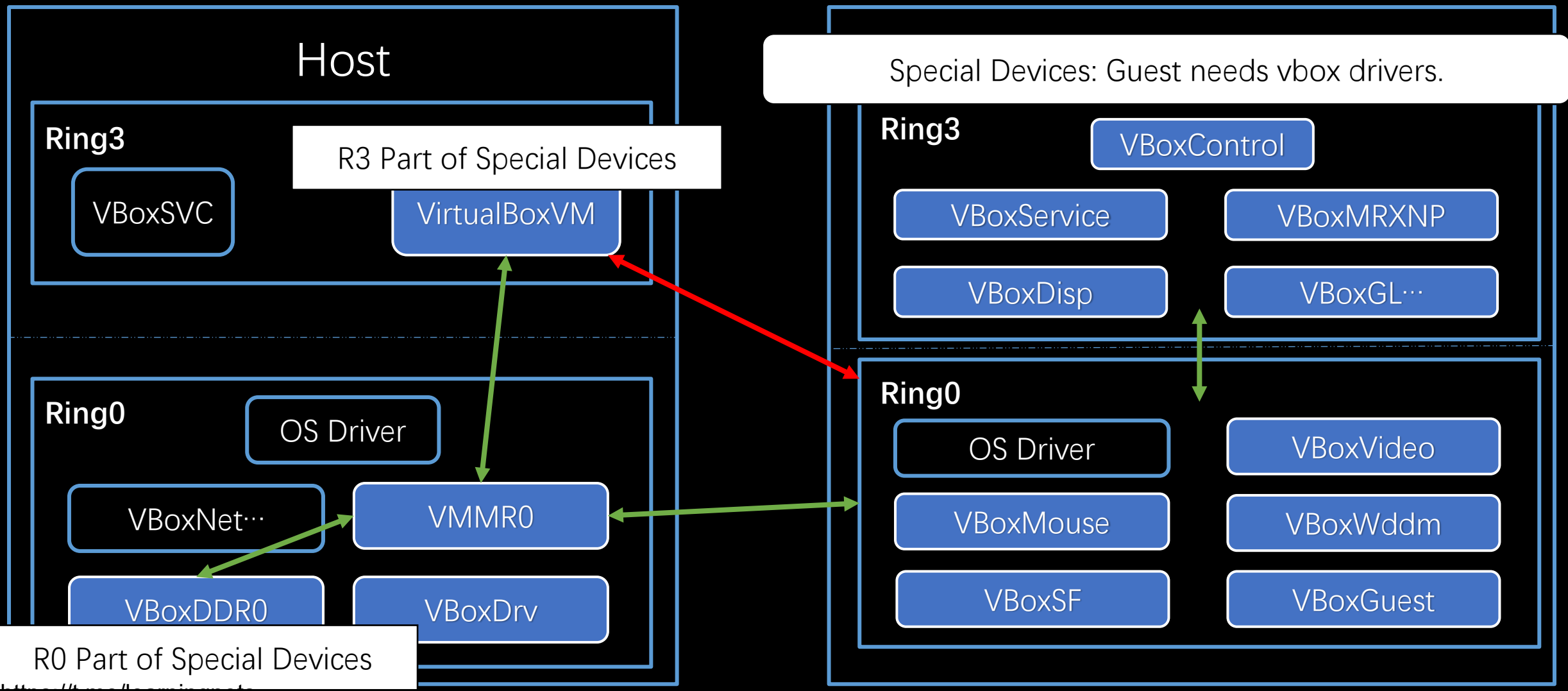


Emulated Devices List:

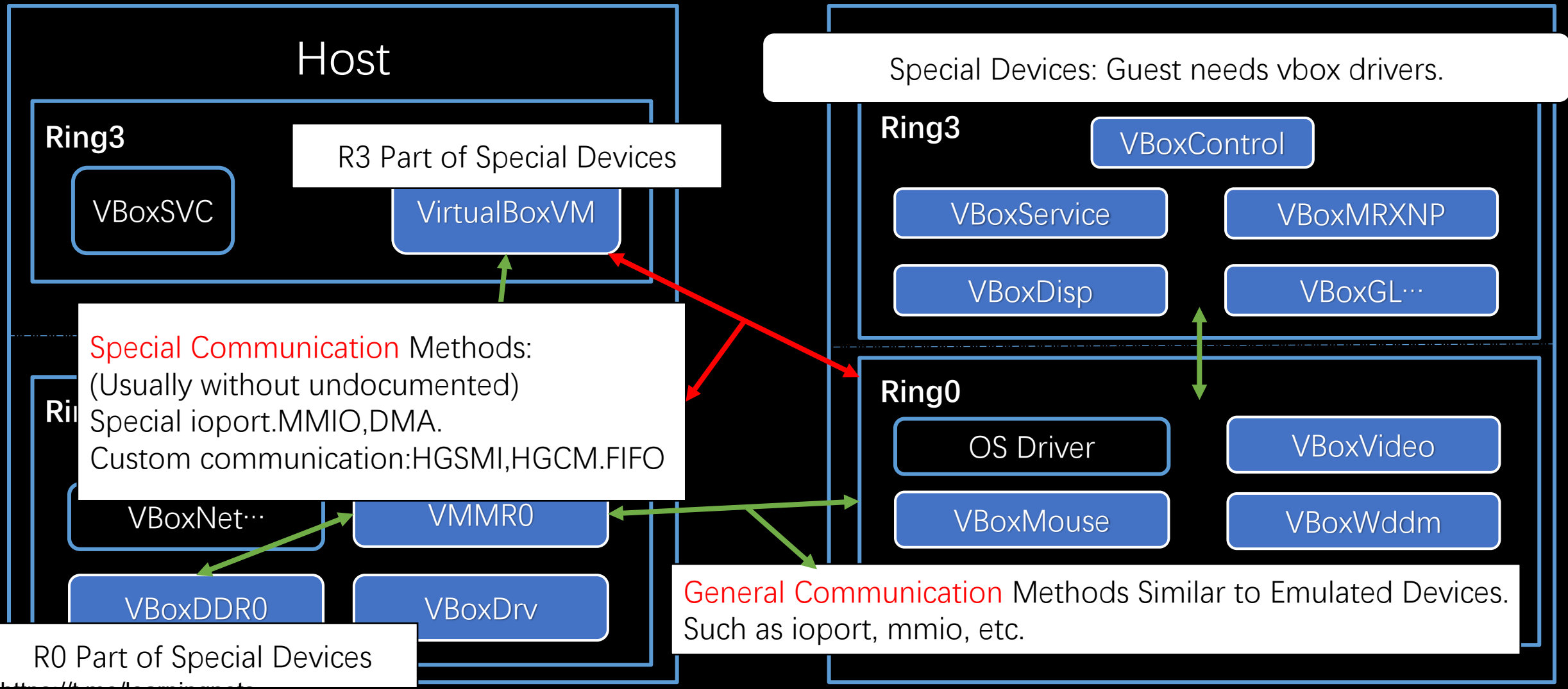
- Emulated motherboard chipset(PCI,PIT,PIC,HPET,etc…);
- Emulated VGA Device;
- Emulated NIC Device(E1000);
- Emulated Audio Controller Devices(HDA,SB16,ICHAC97);
- Emulated USB Controller Devices(OHCI,EHCI,XHCI);
- Emulated Storage Controller Devices(AHCI,FDC,SCSI,NVME,ATA);
- Emulated Serial Port Device;

Special Communication

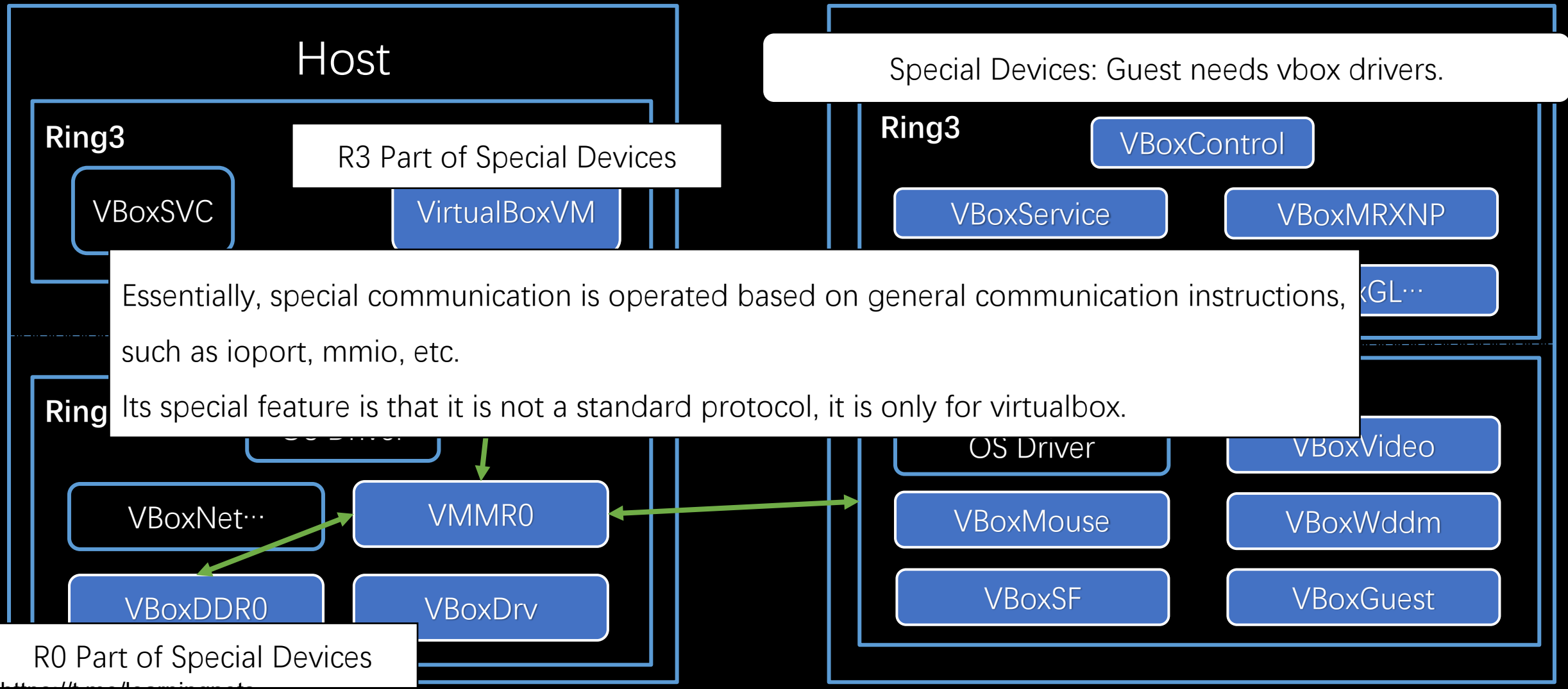
Special Communication



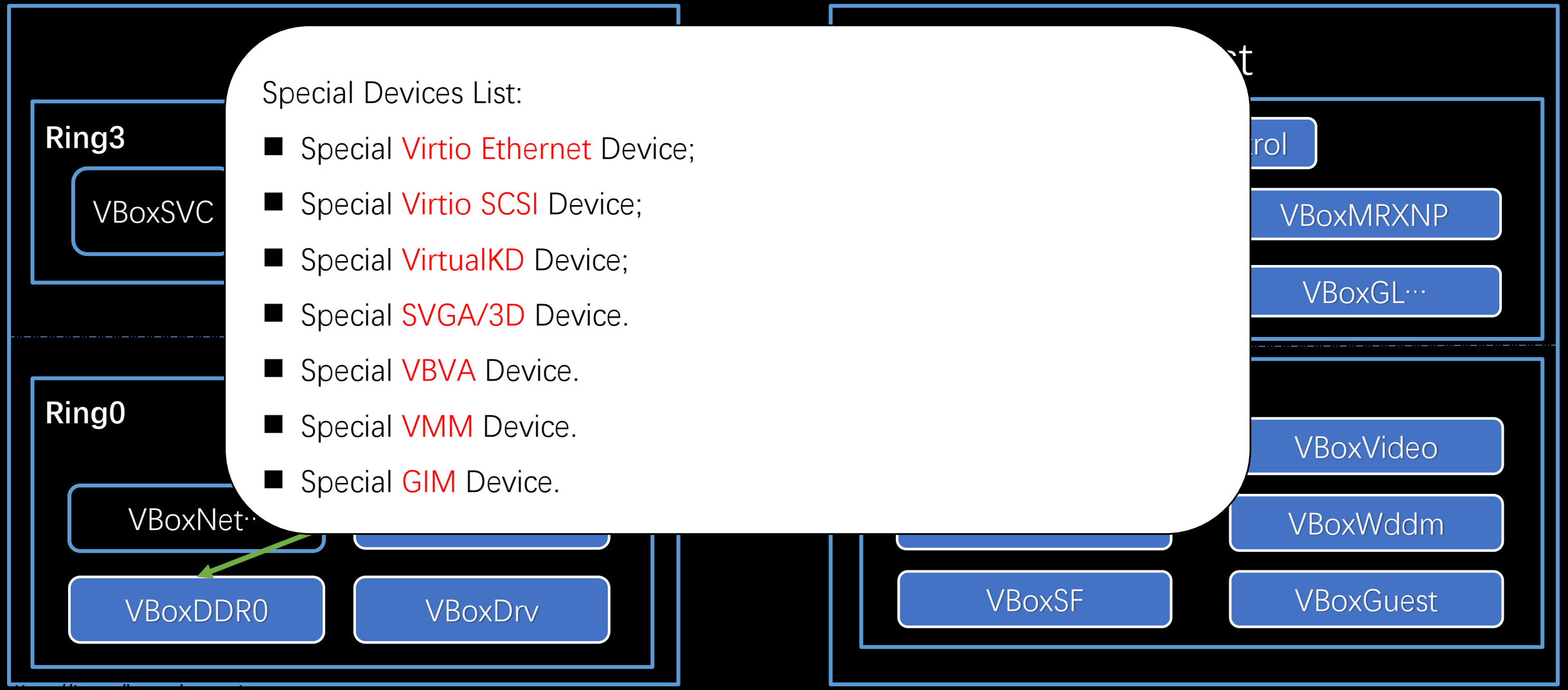
Special Communication



Special Communication



Special Communication



Special Devices List:

- Special Virtio Ethernet Device;
- Special Virtio SCSI Device;
- Special VirtualKD Device;
- Special SVGA/3D Device.
- Special VBVA Device.
- Special VMM Device.
- Special GIM Device.

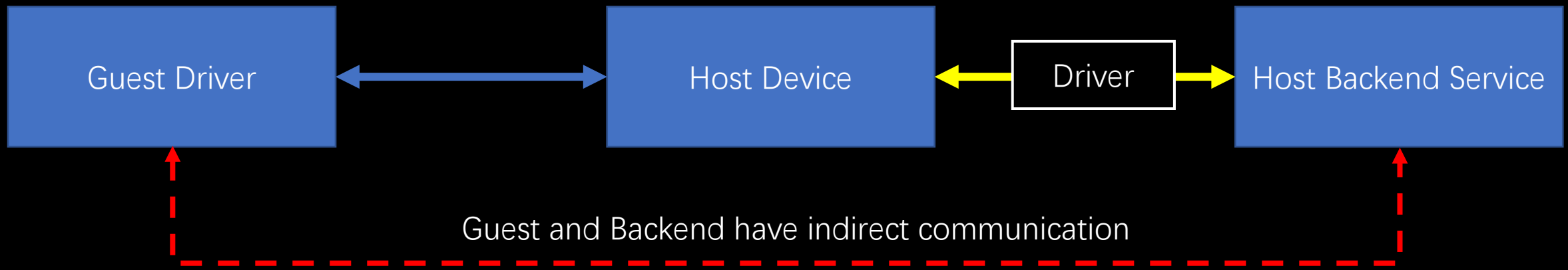
Backend Communication

Backend Communication

Backend refers to the components called by the device to implement specific functions.

Usually it depends on the host operating system and physical devices.

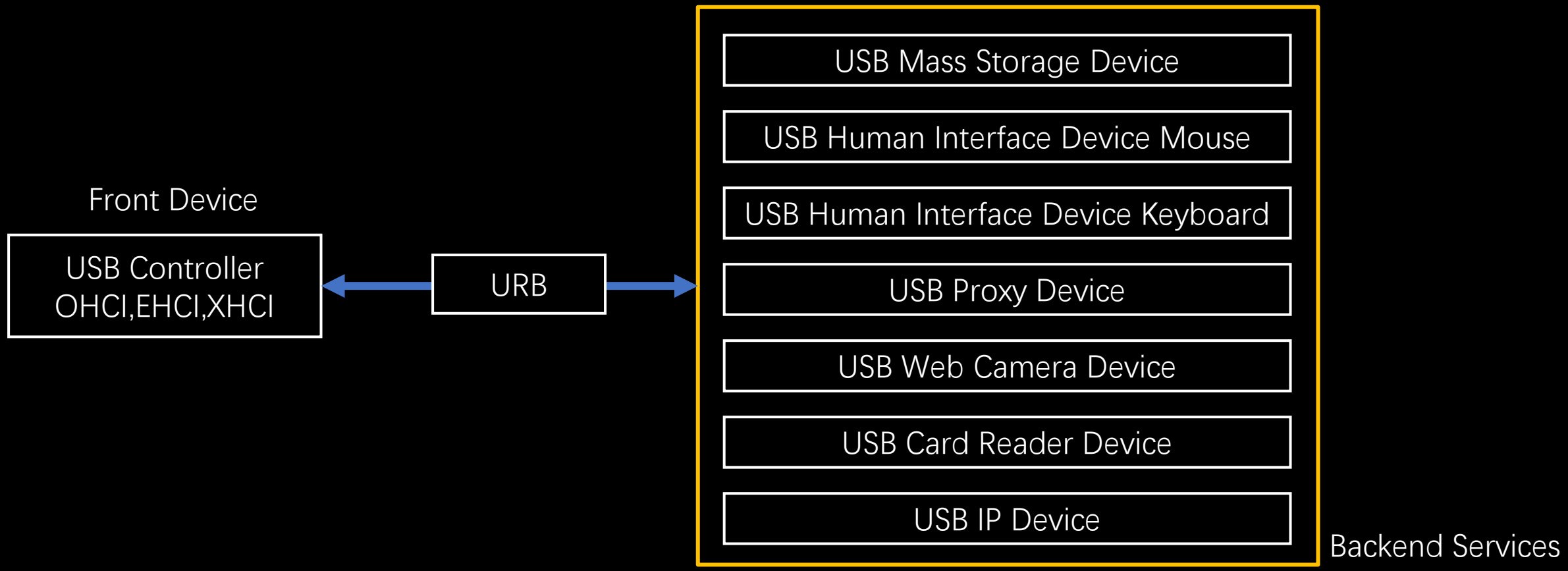
The communication between the device and the backend usually has an intermediate driver component, which is an important attack surface.



Backend is a big attack surface.

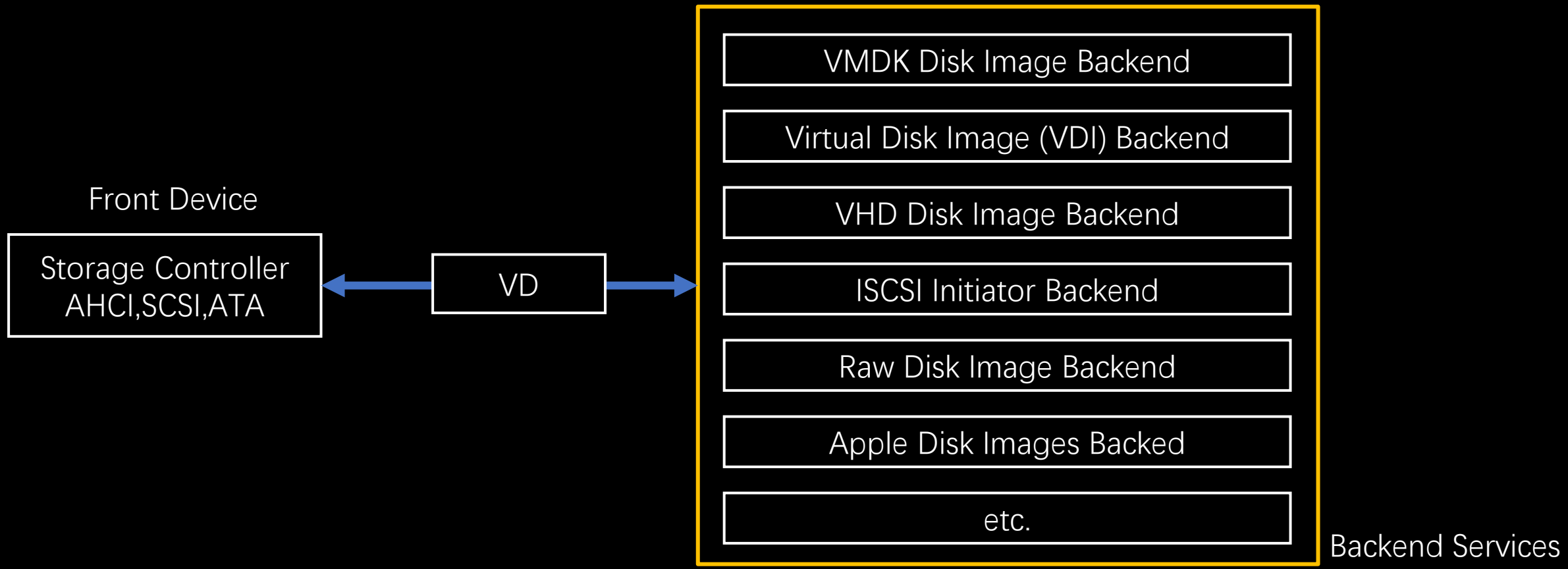
USB Backend

The USB controller device and backend will communicate through a driver component called URB.



Storage Backend

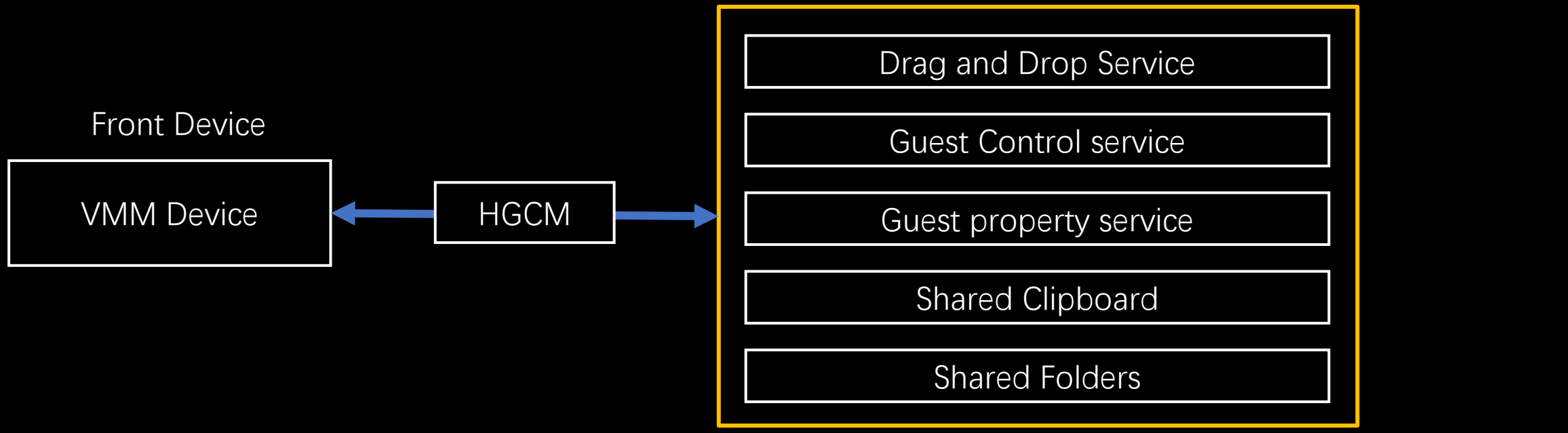
The storage controller device and backend will communicate through a driver component called VD(Virtual Disk).



VMMDev Backend

The VMM device is a custom hardware device emulation for communicating with the guest additions.

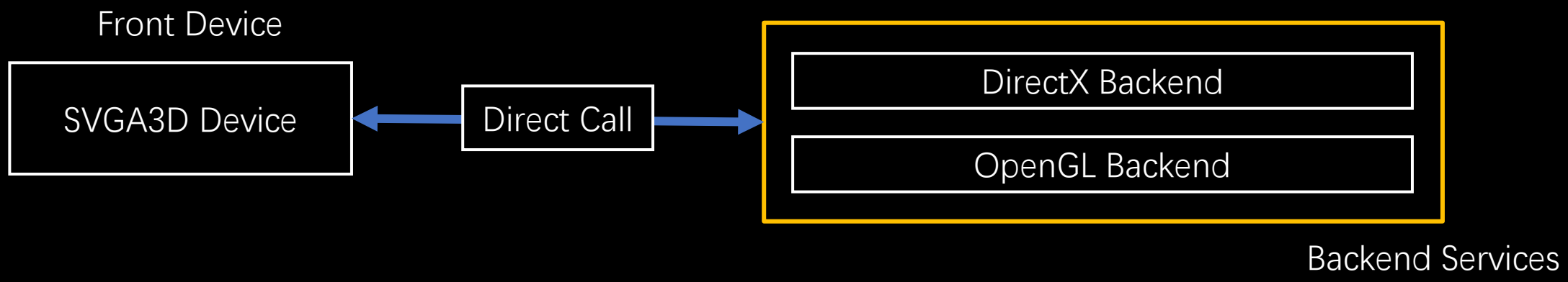
It uses HGCM to communicate with the backend.



SVGA3D Backend

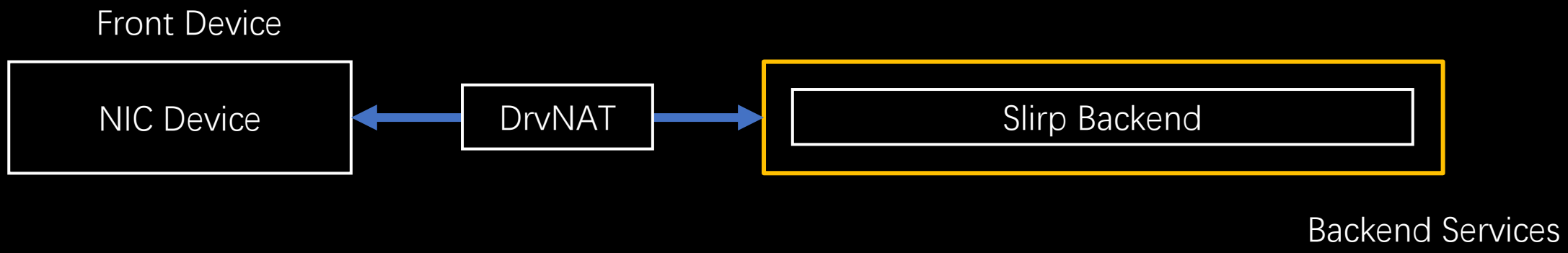
SVGA3D is a device used to support hardware GPU accelerated 3D rendering.

It uses direct calls to communicate with backend.



Network Backend

Slirp is a general purpose TCP-IP emulator used by virtual machine hypervisors to provide virtual networking services.



PART 2

Bug Hunting

Code Review

Device Constructor

PDM Device Registration Structure

```
struct PDMDEVREG  {
    …
    PFNPDMDEVCONSTRUCT  pfnConstruct; //Device Constructor
    …
};

static DECLCALLBACK(int) xxxConstruct(PPDMDEVINS pDevIns, int iInstance, PCFGMNODE pCfg)  {
    …
    PDMDevHlpDriverAttach(…); // Binding device back-end components.
    PDMDevHlpPCIRegister(…); // Register the device as a PCI device.
    IoPortCreate(…); // Create ioport and set io callback function.
    MmioCreate(…); // Create mmio and set io callback function.
    ThreadCreate(…); // Create worker thread.
    …
}
```

Device Constructor

```
struct PDMDEVREG  {
    …
    PFNPDMDEVCONSTRU
    …
};

static DECLCALLBACK(int)
    …
    PDMDevHlpDriverAttac
    PDMDevHlpPCIRegister
    IoPortCreate(…); // Cre
    MmioCreate(…); // Cre
    ThreadCreate(…); // Cre
    …
}
```

FGMNODE pCfg) {

src > VBox > Devices > build > VBoxDD.cpp > VBoxDevicesRegister(PPDMDEVREGCB, uint32_t)

```
60      if (RT_FAILURE(rc))
61          return rc;
62      rc = pCallbacks->pfnRegister(pCallbacks, &g_DevicePcArch);
63      if (RT_FAILURE(rc))
64          return rc;
65      rc = pCallbacks->pfnRegister(pCallbacks, &g_DevicePcBios);
66      if (RT_FAILURE(rc))
67          return rc;
68      rc = pCallbacks->pfnRegister(pCallbacks, &g_DeviceIOAPIC);
69      if (RT_FAILURE(rc))
70          return rc;
71      rc = pCallbacks->pfnRegister(pCallbacks, &g_DevicePS2KeyboardMouse);
72      if (RT_FAILURE(rc))
73          return rc;
74      rc = pCallbacks->pfnRegister(pCallbacks, &g_DevicePIIX3IDE);
75      if (RT_FAILURE(rc))
76          return rc;
77      rc = pCallbacks->pfnRegister(pCallbacks, &g_DeviceI8254);
78      if (RT_FAILURE(rc))
79          return rc;
80      rc = pCallbacks->pfnRegister(pCallbacks, &g_DeviceI8259);
81      if (RT_FAILURE(rc))
82          return rc;
83      rc = pCallbacks->pfnRegister(pCallbacks, &g_DeviceHPET);
84      if (RT_FAILURE(rc))
85          return rc;
86      rc = pCallbacks->pfnRegister(pCallbacks, &g_DeviceSmc);
87      if (RT_FAILURE(rc))
88          return rc;
89      rc = pCallbacks->pfnRegister(pCallbacks, &g_DeviceFlash);
90      if (RT_FAILURE(rc))
91          return rc;
```

Device Constructor

src > VBox > Devices > Graphics > DevVGA.cpp > vgaR3Construct(PPDMDEVINS, int, PCFGMNODE)

```
7652  # endif /* CONFIG_BOCHS_VBE */
7653
7654  # ifdef VBOX_WITH_HGSMI
7655      /* Use reserved VGA IO ports for HGSMI. */
7656      REG_PORT(VGA_PORT_HGSMI_HOST,  4, vgaR3IOPortHgsmiWrite, vgaR3IOPortHgmsiRead, "HGSMI host (3b0-3b3)",  &pThis-
7657      REG_PORT(VGA_PORT_HGSMI_GUEST, 4, vgaR3IOPortHgsmiWrite, vgaR3IOPortHgmsiRead, "HGSMI guest (3d0-3d3)", &pThis-
7658  # endif /* VBOX_WITH_HGSMI */
7659
7660  # undef REG_PORT
7661
7662      /* vga bios */
7663      rc = PDMDevHlpIoPortCreateAndMap(pDevIns, VBE_PRINTF_PORT, 1 /*cPorts*/, vgaIoPortWriteBios, vgaIoPortReadBios,
7664                                       "VGA BIOS debug/panic", NULL /*paExtDescs*/, &pThis->hIoPortBios);
7665      AssertRCReturn(rc, rc);
7666
7667      /*
7668       * The MDA/CGA/EGA/VGA/whatever fixed MMIO area.
7669       */
7670      rc = PDMDevHlpMmioCreateExAndMap(pDevIns, 0x000a0000, 0x00020000,
7671                                       IOMMMIO_FLAGS_READ_PASSTHRU | IOMMMIO_FLAGS_WRITE_PASSTHRU | IOMMMIO_FLAGS_ABS
7672                                       NULL /*pPciDev*/, UINT32_MAX /*iPciRegion*/,
7673                                       vgaMmioWrite, vgaMmioRead, vgaMmioFill, NULL /*pvUser*/,
7674                                       "VGA - VGA Video Buffer", &pThis->hMmioLegacy);
7675      AssertRCReturn(rc, rc);
7676
7677      /*
```

struct P
...
PFNI
...
};

static D
...
PDM
PDM
IoPo
Mmi
Thre
...
}

IO/MMIO

Port I/O Handler and Memory Mapped I/O Handler

```
typedef DECLCALLBACK(VBOXSTRICTRC) FNIOMIOPORTNEWOUT(PPDMDEVINS pDevIns, void *pvUser,
RTIOPORT offPort, uint32_t u32, unsigned cb); //out dx, rax
typedef DECLCALLBACK(VBOXSTRICTRC) FNIOMIOPORTNEWOUTSTRING(PPDMDEVINS pDevIns, void *pvUser,
RTIOPORT offPort, const uint8_t *pbSrc,uint32_t *pcTransfers, unsigned cb);//rep outsb

typedef DECLCALLBACK(VBOXSTRICTRC) FNIOMIOPORTNEWIN(PPDMDEVINS pDevIns, void *pvUser, RTIOPORT
offPort, uint32_t *pu32, unsigned cb);//in  eax, dx
typedef DECLCALLBACK(VBOXSTRICTRC) FNIOMIOPORTNEWINSTRING(PPDMDEVINS pDevIns, void *pvUser,
RTIOPORT offPort, uint8_t *pbDst,uint32_t *pcTransfers, unsigned cb);//rep insb

typedef DECLCALLBACK(VBOXSTRICTRC) FNIOMMMIONEWWRITE(PPDMDEVINS pDevIns, void *pvUser,
RTGCPHYS off, void const *pv, uint32_t cb);//mov [rdx],rax
typedef DECLCALLBACK(VBOXSTRICTRC) FNIOMMMIONEWREAD(PPDMDEVINS pDevIns, void *pvUser,
RTGCPHYS off, void *pv, uint32_t cb);//mov rax,[rdx]
```

IO/MMIO

Port I/O Handler Case

IO/MMIO

Memory Mapped I/O Handler Case

Direct Physical Memory Read/Write

Guest's Physical Memory Read/Write Handlers

This is an operation by an emulation device to quickly share a large amount of data with the guest.
It directly reads and writes the physical memory of the guest, just like the host and the guest perform memmove, so this is an important attack point.

DECLINLINE(int) PDMDevHlpPhysWrite(PPDMDEVINS pDevIns, RTGCPHYS GCPhys, const void *pvBuf, size_t cbWrite)

DECLINLINE(int) PDMDevHlpPhysRead(PPDMDEVINS pDevIns, RTGCPHYS GCPhys, void *pvBuf, size_t cbRead)

Direct Physical Memory Read/Write

Guest's Physical Memory Read/Write Case

USB Backend

PDM USB Device Registration Structure and USB request descriptor

```
struct PDMUSBREG{
   ...
   pfnUrbQueue(PPDMUSBINS pUsbIns, PVUSBURB pUrb); //Queues an URB for processing
   ...
}


struct VUSBURB{
   ...
   uint32_tcbData;
   uint8_t abData[8*_1K]; //Guest Controllable
   ...
}
```

USB Backend

```
struct PDM
   ...
   pfnUrbQ
   ...
}

struct VUS
   ...
   uint32_t
   uint8_t a
   ...
}
```

```
407 extern "C" DECLEXPORT(int) VBoxUsbRegister(PCPDMUSBREGCB pCallbacks, uint32_t u32Version)
408 {
409     int rc = VINF_SUCCESS;
410     RT_NOREF1(u32Version);
411
412 #ifdef VBOX_WITH_USB
413     rc = pCallbacks->pfnRegister(pCallbacks, &g_UsbDevProxy);
414     if (RT_FAILURE(rc))
415         return rc;
416 # ifdef VBOX_WITH_SCSI
417     rc = pCallbacks->pfnRegister(pCallbacks, &g_UsbMsd);
418     if (RT_FAILURE(rc))
419         return rc;
420 # endif
421 #endif
422 #ifdef VBOX_WITH_VUSB
423     rc = pCallbacks->pfnRegister(pCallbacks, &g_UsbHidKbd);
424     if (RT_FAILURE(rc))
425         return rc;
426     rc = pCallbacks->pfnRegister(pCallbacks, &g_UsbHidMou);
427     if (RT_FAILURE(rc))
428         return rc;
429 #endif
430 #ifdef VBOX_WITH_USB_VIDEO_IMPL
431     rc = pCallbacks->pfnRegister(pCallbacks, &g_DevWebcam);
```

hingnets

USB Backend

USB Backend Case
gf

Network Backend

Slirp

gf

Network Backend

Slirp case

Hack with CodeQL

CodeQL Basic

What is this?
CodeQL is the analysis engine used by developers to automate security checks, and by security researchers to perform variant analysis. CodeQL compiles code to a relational database (the snapshot database – a combination of database and source code), which is queried using Semmle QL, a declarative, object-oriented query language designed for program analysis. Related Links https://securitylab.github.com/tools/codeql https://codeql.github.com/docs/

CodeQL Basic

How to use

The basic workflow is that based on the analysis of historical vulnerabilities, we write queries to find code patterns that are semantically similar to them.

CodeQL analysis consists of three steps:

1. Preparing the code, by creating a CodeQL database
2. Running CodeQL queries against the database
3. Interpreting the query results

Create Vbox CodeQL Database

```
codeql database create vbox-database --language=cpp --command=kmk
```

```
ccc@ubuntu:~/vbox/codeql/VirtualBox-6.1.16/vbox-database$ ls -l
total 75420
-rw-r--r-- 1 root root      133 Dec 24 00:33 codeql-database.yml
drwxr-xr-x 3 root root     4096 Dec 24 00:33 db-cpp
drwxr-xr-x 2 root root     4096 Dec 24 00:15 log
-rw------- 1 root root 77215260 Dec 24 00:33 src.zip
```

Write CodeQL

For example, we now want to find out the length parameter of memcpy controllable by the Guest.

First, we determine the source of the data.

Take URB as an example here.

```
override predicate isSource(DataFlow::Node source) {
    exists(Parameter p,Variable v |
        (
            v.hasName("pUrb") and
            source.asExpr() = v.getInitializer().getExpr()
        )
        or
        (
            p.hasName("pUrb") and
            source.asParameter() = p
        )
    )
}
```

Write CodeQL

Second, We need to determine the destination.

The destination here is the third length parameter of memcpy.

```
class MemFunctionCall extends FunctionCall {
    int argToCheck;
    MemFunctionCall() {
      ( this.getTarget().hasName("memcpy") and argToCheck = 2 )
    }
    Expr getArgumentToCheck() { result = this.getArgument(argToCheck) }
}

override predicate isSink(DataFlow::Node sink) {
    exists (MemFunctionCall fc |
        fc.getArgumentToCheck() = sink.asExpr())
}
```

Write CodeQL

Third, add data polluted by source data to query.

e.g. in expression a->b.c the data flows from a to c.

```
override predicate isAdditionalFlowStep(DataFlow::Node node1, DataFlow::Node node2) {
    exists(Expr e, FieldAccess fa |
    node1.asExpr() = e and node2.asExpr() = fa |
    fa.getQualifier*() = e and not (fa.getParent() instanceof FieldAccess)
    )
}
```

Execute CodeQL

Finally, execute the query.

```
from Config cfg,DataFlow::PathNode source,DataFlow::PathNode sink
where cfg.hasFlowPath(source, sink)
select source.getNode().getLocation(),sink.getNode().getLocation()
```

Check Result

Check the result and find vulnerabilities.

Fuzz with AFL++

AFL++ Basic

What is this?
AFL(american fuzzy lop) is a fuzzer that employs genetic algorithms in order to efficiently increase code coverage of the test cases. AFL++ is a superior fork to Google's AFL - more speed, more and better mutations, more and better instrumentation, custom module support, etc. Related Links https://github.com/AFLplusplus/AFLplusplus

AFL++ Basic

How to use?

There are three steps to fuzzing source code.

1. Compile the target with a special compiler that prepares the target to be fuzzed efficiently. This step is called "instrumenting a target".
2. Prepare the fuzzing by selecting and optimizing the input corpus for the target.
3. Perform the fuzzing of the target by randomly mutating input and assessing if a generated input was processed in a new path in the target binary.

Four modes

afl-clang-lto > afl-clang-fast > afl-gcc-fast > afl-gcc

Vbox with AFL++

Some Difficulties

1. Compile problems.
2. The choice between the whole and the split.
3. Where is the fuzz cycle?
4. Performance problems.

Compile problems

Compile VBox with afl-clang-fast

The first thing I thought of was to modify Configure file to change the CC and CXX to afl-clang-fast and afl-clang-fast++.

But there were a lot of errors when compiling, and even c grammar that clang could not recognize, so that I couldn't fix it easily.

```
/home/ccc/vbox/vboxshow/VirtualBox-6.1.16/src/VBox/Devices/PC/ipxe/src/arch/i386/core/setjmp.S:6:2: error: unknown directive
 .arch i386
```

```
/home/ccc/vbox/vboxshow/VirtualBox-6.1.16/src/VBox/Devices/PC/ipxe/src/core/settings.c:289:8: error: fields must have a constant size: 'variable length array in structure' extension will never be supported
          char name[ strlen ( name ) + 1 /* NUL */ ];
```

Solve Compilation Problems

Compile VBox with afl-clang-fast

After research, I found that the source code that failed to compile is not related to the component we want to fuzz.

Such as UI-related, kernel-related, and some assembly files.

Therefore, I consider mixed compilation. The source code of fuzz needs to be compiled with afl, and the others are compiled with original gcc.

Fortunately, the compilation framework of virtualbox is very easy to do this. Kmk is a template-based compilation framework.

Virtualbox defines a template for each module. We only need to modify the specified template.

Solve Compilation Problems

Compile VBox with afl-clang-fast

Create an AFL.kmk file in /kBuild/tools/, which can be copied from /kBuild/tools/GXX64.kmk, and write the afl compilation tool into it.

```
# Tool Specific Properties
TOOL_AFL_CC  ?= afl-clang-fast$(HOSTSUFF_EXE) -m64
TOOL_AFL_CXX ?= afl-clang-fast++$(HOSTSUFF_EXE) -m64
TOOL_AFL_PCH ?= $(TOOL_AFL_CXX)
TOOL_AFL_AS  ?= afl-clang-fast$(HOSTSUFF_EXE) -m64
TOOL_AFL_AR  ?= ar$(HOSTSUFF_EXE)
TOOL_AFL_LD  ?= afl-clang-fast++$(HOSTSUFF_EXE) -m64
TOOL_AFL_LD_SYSMOD ?= ld$(HOSTSUFF_EXE)
ifndef TOOL_AFL_LDFLAGS.$(KBUILD_TARGET)
TOOL_AFL_LDFLAGS.dll ?= -shared
else
TOOL_AFL_LDFLAGS.dll ?= $(TOOL_AFL_LDFLAGS.$(KBUILD_TARGET))
endif
```

Solve Compilation Problems

Compile VBox with afl-clang-fast

Next, modify the Config.kmk file, and change the compilation tool of the fuzz module you need to AFL.

For example:

```
TEMPLATE_VBOXMAINEXE_TOOL          = AFL
TEMPLATE_VBOXR3EXE_TOOL            = AFL
```

Then you can compile it.

There may still be some errors, but they can be easily resolved, so I won't repeat them here.

Whole or Split ?

Selected

The most common use of AFL is to fuzz a separate library, such as libxml2.so.

There are also some libraries that can be fuzzed in VirtualBox, such as slirp, shaderlib.

But this is only a small part of the virtualbox, most of the others are strongly coupled with the core, such as the devices and backends.

These components are difficult to separate and run independently.

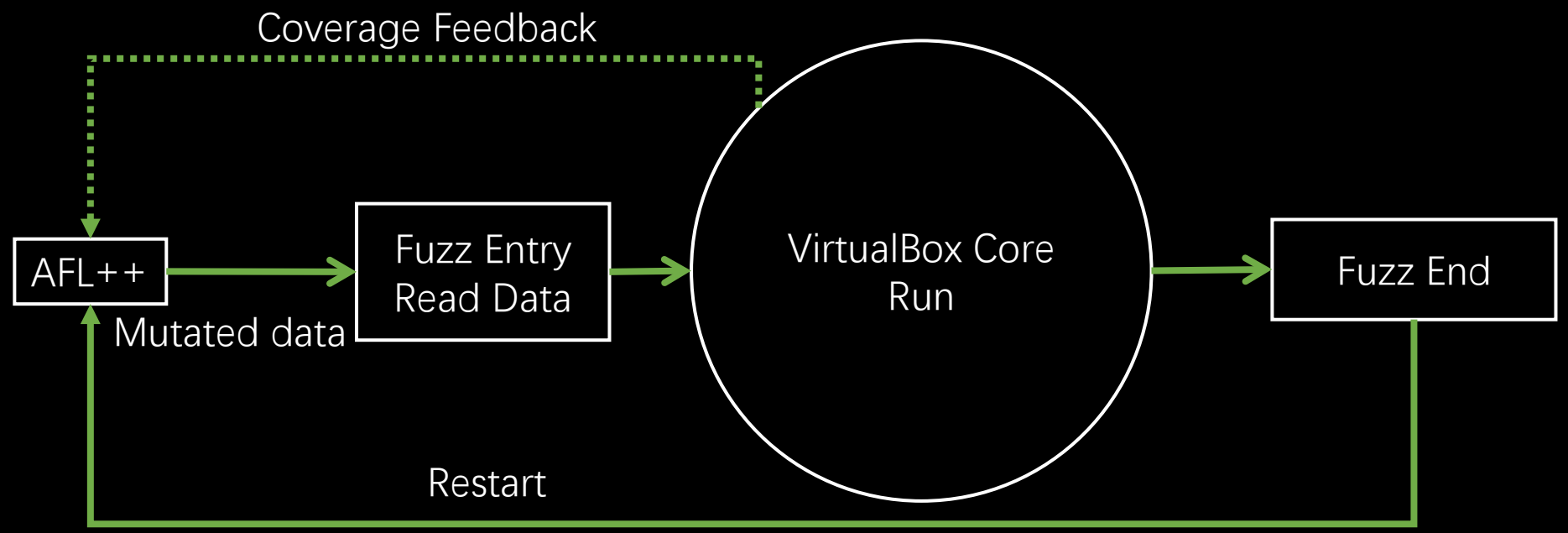
Therefore, split fuzz will not be able to cover completely.

In the end, I chose to fuzz the complete virtualbox.

Fuzz Cycle

Fuzz Cycle

Because virtualbox is not a standard input and output program, and it is not a one-time execution program. So we need to make it accept input and end after processing.



Fuzz Cycle

Fuzz Cycle Details

1. Using VBoxHeadless as the startup program, it will avoid loading the UI program.
2. Add the pfnFuzzEntry callback function to the PDMDEVREG structure, which is the fuzz entry for each device.
3. After PowerUp(Initialization of vbox), call the pfnFuzzEntry function of each device.
4. Finally close virtualbox.
5. The main fuzz code is in pfnFuzzEntry, we can directly call the ioport handler or mmio handler, and its parameters are taken from the input of afl.
6. In addition, for direct physical memory read operations, we also need to modify it to obtain from the afl input.

Fuzz Cycle

Fuzz IO/MMIO

```
static DECLCALLBACK(void) hdaR3FuzzEntry(PPDMDEVINS pDevIns)
{
    Fuzz_CMD offary[sizeof(g_aHdaRegMap)/sizeof(g_aHdaRegMap[0]) +
    int i = 0;
    for(i = 0;i<sizeof(g_aHdaRegMap)/sizeof(g_aHdaRegMap[0]);i++)
    for(i = 0;i<sizeof(g_aHdaRegAliases)/sizeof(g_aHdaRegAliases[0
    uint32_t readlen = 0x2000;
    Fuzz_Init(g_HDA_fuzzbuf,readlen);
    while(!Fuzz_GetGlobalBufOver())
    {
        RUN_RAND_FUNCS(pDevIns,HDA_funcs);
        RUN_RAND_FUNCS(pDevIns,HDA_funcs);
        Fuzz_CMD cmd = {0};
        cmd = offary[Fuzz_GetRandRange(0,ARY_NUM(offary))];
        cmd.val = Fuzz_GetRand();
        hdaMmioWrite(pDevIns,NULL,cmd.off,&cmd.val,cmd.cb);
    }
}
```

Fuzz HDA Device

```
static DECLCALLBACK(void) PCNETR3FuzzEntry(PPDMDEVINS pDevIns)
{

    uint32_t readlen = 0x2000;
    Fuzz_Init(g_PCNET_fuzzbuf,readlen);
    while(!Fuzz_GetGlobalBufOver())
    {
        RUN_RAND_FUNCS(pDevIns,pcnet_funcs);

        Fuzz_CMD cmd = {0};
        cmd = g_pcnetIoPortWrite[Fuzz_GetRandRange(0,ARY_NUM(g_pcnetIoPortWrite))];
        cmd.val = Fuzz_GetRand();
        pcnetIoPortWrite(pDevIns,NULL,cmd.off,cmd.val,cmd.cb);

        cmd.cb = 1;
        cmd.off = Fuzz_GetRandRange(0,0x10);
        cmd.val = Fuzz_GetRand();
        pcnetIoPortAPromWrite(pDevIns,NULL,cmd.off,cmd.val,cmd.cb);
    }
}
```

Fuzz PCNET Device

Fuzz Cycle

Fuzz PhyRead

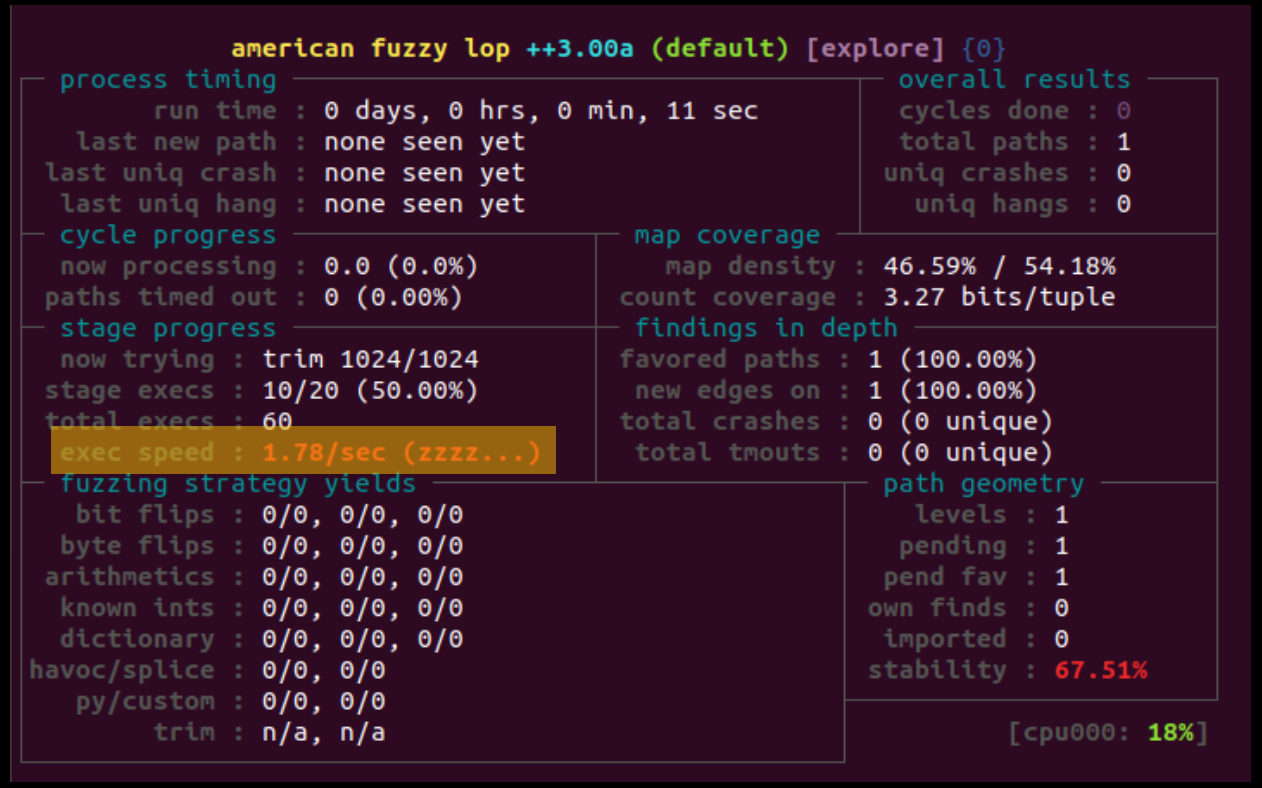
```
DECLINLINE(int) PDMDevHlpPhysRead(PPDMDEVINS pDevIns, RTGCPHYS GCPhys, void *pvBuf, size_t cbRead)
{
    #ifdef IN_RING3
    if(pDevIns->afReserved_Fuzzing[0])
    {
        Fuzz_GetRandBuf(pvBuf,cbRead);
        RTStrmPrintfTrace(g_pFuzzLogOut);
        return 0;
    }
    #endif
    return pDevIns->CTX_SUFF(pHlp)->pfnPhysRead(pDevIns, GCPhys, pvBuf, cbRead);
}
```

Performance Problems

Performance Problems

When we completed the first version of fuzz, it ran very slowly, about xxx times in 1 second, which could not meet our needs at all.

So we started the optimization road.



Performance Optimization

Performance Optimization

The biggest reason for the slowness is the startup process.

So we use Persistent Mode, a cool feature of AFL++.

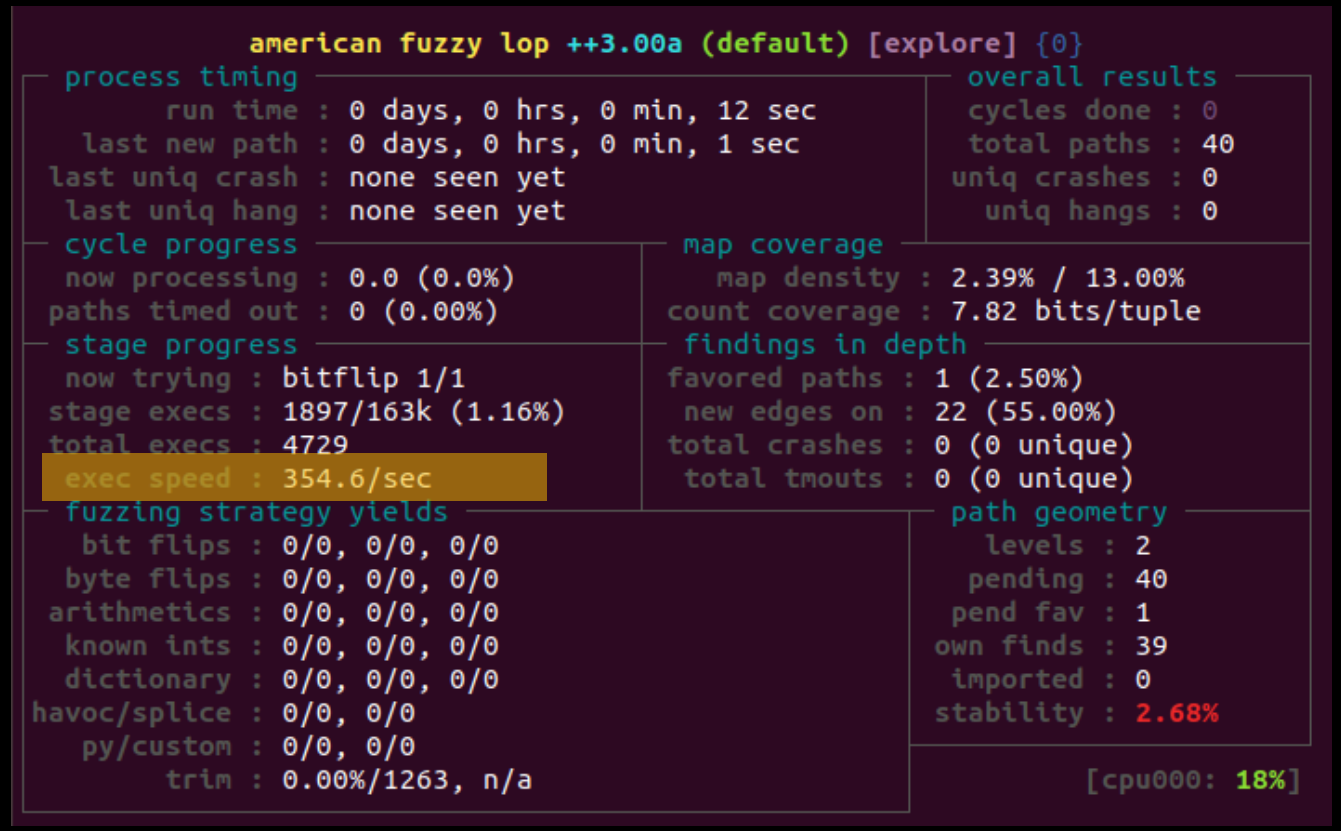
https://github.com/AFLplusplus/AFLplusplus/blob/stable/instrumentation/README.persistent_mode.md

1. First, we load __AFL_LOOP after powerup.
2. Then after the FuzzEntry is completed, manually call device reset to ensure the consistency of the state. But even this is very slow, because the reset of the device is a waste of time, and it cannot be placed outside the loop.
3. Therefore, we need to optimize the reset of each device.

Performance Problems

Compile VBox with afl-clang-fast

After optimization, it finally meets the requirements of fuzz.



Results

10+ vulnerbilities

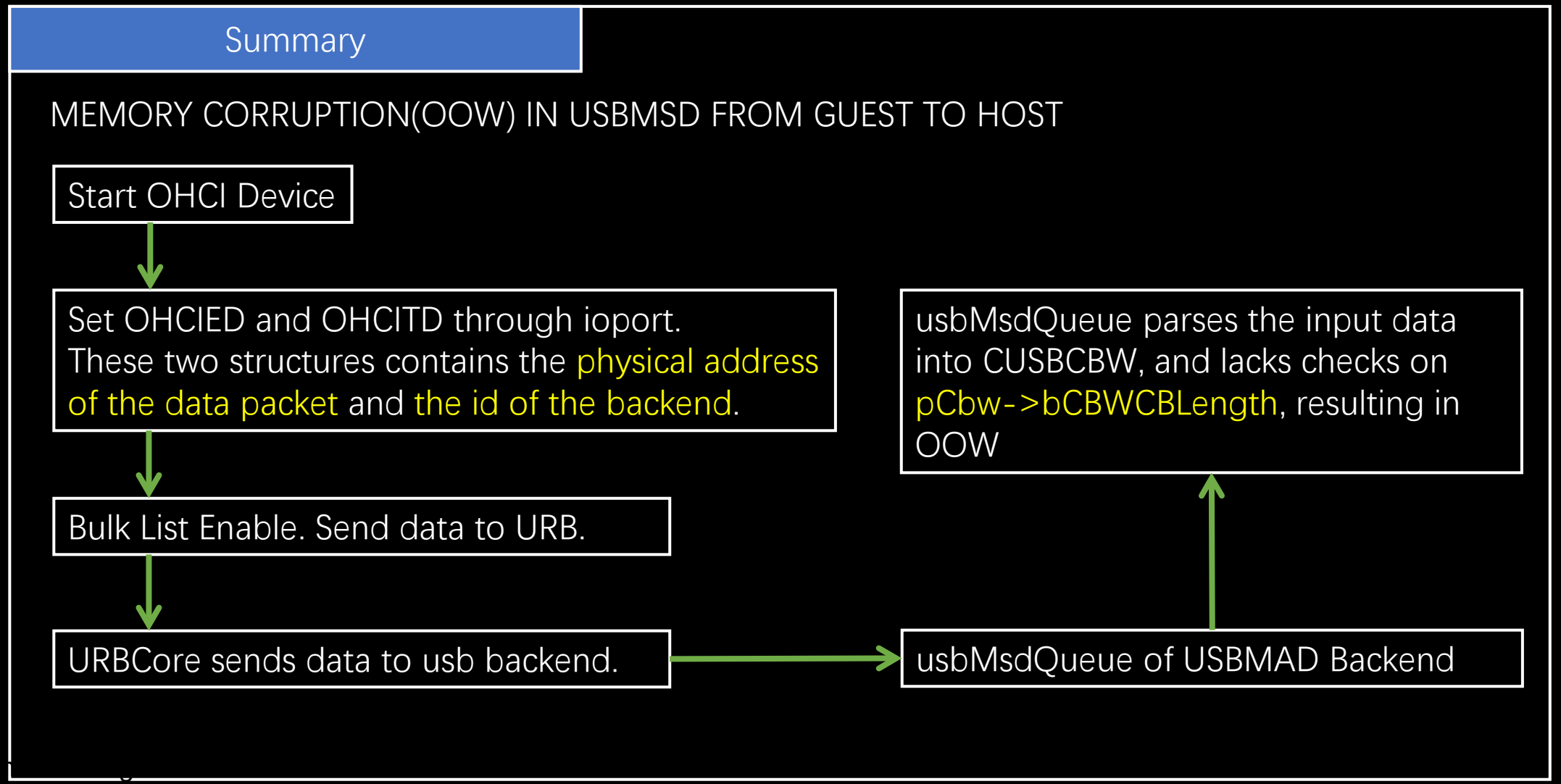
CVE-2021-2086	CVE-2021-2111	CVE-2021-2112	CVE-2021-2119
CVE-2021-2120	CVE-2021-2121	CVE-2021-2125	CVE-2021-2126
CVE-2021-2129	CVE-2021-2131	…	

PART 3

Case Study

CVE-2021-2112

CVE-2021-2112



CVE-2021-2112

Summary

MEMORY CORRUPTION(OOW) IN USBMSD FROM GUEST TO HOST

```
static void usbMsdReqPrepare(PUSBMSDREQ pReq, PCUSBCBW pCbw)
{
    /* Copy the CBW */
    size_t cbCopy = RT_UOFFSETOF_DYN(USBCBW, CBWCB[pCbw->bCBWCBLength]);
    memcpy(&pReq->Cbw, pCbw, cbCopy);
    memset((uint8_t *)&pReq->Cbw + cbCopy, 0, sizeof(pReq->Cbw) - cbCopy);

    /* Setup the SCSI request. */
    pReq->offBuf         = 0;
    pReq->iScsiReqStatus = 0xff;
}
```

CVE-2021-2112

Summary

MEMORY CORRUPTION(OOW) IN USBMSD FROM GUEST TO HOST

```
0:037> g
(4b8.f80): Access violation - code c0000005 (!!! second chance !!!)
rax=0000000000000000 rbx=000000000000010e rcx=00000000190b9018
rdx=000000000313fc28 rsi=000000001c1f8b40 rdi=00000000224edf00
rip=0000000074e1c021 rsp=000000002d93f4d8 rbp=00000000190b8fb0
 r8=000000000000010a  r9=0000000000000007 r10=0000000000000000
r11=00000000190b8fd4 r12=0000000000000048 r13=000000001c1f8e1c
r14=000000001c1f8b40 r15=0000000000000220
iopl=0         nv up ei pl nz na po nc
cs=0033  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00010204
MSVCR100!memcpy+0xc1:
00000000`74e1c021 4c8951e8        mov     qword ptr [rcx-18h],r10 ds:00000000`190b9000=????????????????
0:037> k
 # Child-SP          RetAddr           Call Site
00 00000000`2d93f4d8 000007fe`ea8ac133 MSVCR100!memcpy+0xc1
01 00000000`2d93f4e0 000007fe`ea8ac2b6 VBoxDD!usbMsdHandleBulkHostToDev+0x163 [c:\devel\virtualbox-src\src\vbox\devices\storage\usbmsd.cpp @ 1662]
02 00000000`2d93f510 000007fe`ea8b788b VBoxDD!usbMsdQueue+0x56 [c:\devel\virtualbox-src\src\vbox\devices\storage\usbmsd.cpp @ 1978]
03 00000000`2d93f540 000007fe`ea8b8224 VBoxDD!vusbUrbQueueAsyncRh+0x5b [c:\devel\virtualbox-src\src\vbox\devices\usb\vusburb.cpp @ 456]
04 00000000`2d93f570 000007fe`ea8b2dc6 VBoxDD!vusbUrbSubmit+0x1a4 [c:\devel\virtualbox-src\src\vbox\devices\usb\vusburb.cpp @ 1209]
05 00000000`2d93f5b0 000007fe`ea8aec0e VBoxDD!vusbRhSubmitUrb+0xa6 [c:\devel\virtualbox-src\src\vbox\devices\usb\drvvusbroothub.cpp @ 714]
06 00000000`2d93f5f0 000007fe`ea8af51f VBoxDD!ohciR3ServiceTdMultiple+0x52e [c:\devel\virtualbox-src\src\vbox\devices\usb\devohci.cpp @ 3286]
07 00000000`2d93f6c0 000007fe`ea8afff9 VBoxDD!ohciR3ServiceBulkList+0x14f [c:\devel\virtualbox-src\src\vbox\devices\usb\devohci.cpp @ 3767]
08 00000000`2d93f730 000007fe`ea8b00dd VBoxDD!ohciR3StartOfFrame+0xf9 [c:\devel\virtualbox-src\src\vbox\devices\usb\devohci.cpp @ 4234]
09 00000000`2d93f760 000007fe`ea8b29d9 VBoxDD!ohciR3StartFrame+0xad [c:\devel\virtualbox-src\src\vbox\devices\usb\devohci.cpp @ 4320]
0a 00000000`2d93f790 000007fe`ea8b2b4e VBoxDD!vusbRhR3ProcessFrame+0x89 [c:\devel\virtualbox-src\src\vbox\devices\usb\drvvusbroothub.cpp @ 526]
0b 00000000`2d93f7c0 000007fe`ed377ef9 VBoxDD!vusbRhR3PeriodFrameWorker+0xfe [c:\devel\virtualbox-src\src\vbox\devices\usb\drvvusbroothub.cpp @ 593]
0c 00000000`2d93f820 000007fe`efb9345f VBoxVMM!pdmR3ThreadMain+0x99 [c:\devel\virtualbox-src\src\vbox\vmm\vmmr3\pdmthread.cpp @ 780]
0d 00000000`2d93f880 000007fe`efc449b2 VBoxRT!rtThreadMain+0x2f [c:\devel\virtualbox-src\src\vbox\runtime\common\misc\thread.cpp @ 727]
0e 00000000`2d93f8b0 00000000`74e01d9f VBoxRT!rtThreadNativeMain+0x92 [c:\devel\virtualbox-src\src\vbox\runtime\r3\win\thread-win.cpp @ 256]
0f 00000000`2d93f8e0 00000000`74e01e3b MSVCR100!endthreadex+0x43
10 00000000`2d93f910 00000000`773c652d MSVCR100!endthreadex+0xdf
11 00000000`2d93f940 00000000`775fc521 kernel32!BaseThreadInitThunk+0xd
12 00000000`2d93f970 00000000`00000000 ntdll!RtlUserThreadStart+0x1d
```

CVE-2021-2131

CVE-2021-2131

OOW in VUSBURB

```
(a08.26cc): Access violation - code c0000005 (first chance)
MSVCR100!memcpy+0x1ec:
00000000`6057c14c 8901            mov     dword ptr [rcx],eax ds:00000000`22c68fb8=????????
0:036> k
 # Child-SP          RetAddr           Call Site
00 00000000`43c6f4d8 00007ffa`db1e7efa MSVCR100!memcpy+0x1ec
01 00000000`43c6f4e0 00007ffa`db1e81ae VBoxDD!vusbUrbSubmitCtrl+0x1ca [c:\virtualbox-src\src\vbox\devices\usb\vusburb.cpp @ 1001]
02 00000000`43c6f520 00007ffa`db1e2d46 VBoxDD!vusbUrbSubmit+0x1ae [c:\virtualbox-src\src\vbox\devices\usb\vusburb.cpp @ 1205]
03 00000000`43c6f560 00007ffa`f1c61fa5 VBoxDD!vusbRhSubmitUrb+0xa6 [c:\virtualbox-src\src\vbox\devices\usb\drvvusbroothub.cpp @ 714]
04 00000000`43c6f5a0 00007ffa`f1c627e1 VBoxEhciR3+0x1fa5
05 00000000`43c6f630 00007ffa`f1c62a1f VBoxEhciR3+0x27e1
06 00000000`43c6f6e0 00007ffa`f1c62c65 VBoxEhciR3+0x2a1f
07 00000000`43c6f7b0 00007ffa`f1c62da9 VBoxEhciR3+0x2c65
08 00000000`43c6f850 00007ffa`f1c62fbb VBoxEhciR3+0x2da9
09 00000000`43c6f890 00007ffa`dd4c7ef9 VBoxEhciR3+0x2fbb
0a 00000000`43c6f900 00007ffa`e03f345f VBoxVMM!pdmR3ThreadMain+0x99 [c:\virtualbox-src\src\vbox\vmm\vmmr3\pdmthread.cpp @ 780]
0b 00000000`43c6f960 00007ffa`e04a49b2 VBoxRT!rtThreadMain+0x2f [c:\devel\virtualbox-src\src\vbox\runtime\common\misc\thread.cpp @ 727]
0c 00000000`43c6f990 00000000`60561d9f VBoxRT!rtThreadNativeMain+0x92 [c:\virtualbox-src\src\vbox\runtime\r3\win\thread-win.cpp @ 256]
0d 00000000`43c6f9c0 00000000`60561e3b MSVCR100!endthreadex+0x43
0e 00000000`43c6f9f0 00007ffa`f8287c24 MSVCR100!endthreadex+0xdf
0f 00000000`43c6fa20 00007ffa`fa22d4d1 KERNEL32!BaseThreadInitThunk+0x14
10 00000000`43c6fa50 00000000`00000000 ntdll!RtlUserThreadStart+0x21
```

CVE-2021-2120

CVE-2021-2120

OUT-OF-BOUNDS READ IN LSILOGICSCSI

```
(4ac.2378): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
VBoxDD!lsilogicR3ProcessMessageRequest+0x1fe:
00007ffe`1581dbde 0fb744913a      movzx   eax,word ptr [rcx+rdx*4+3Ah] ds:00000000`1177c006=????
0:019> k
 # Child-SP          RetAddr           Call Site
00 00000000`2917f700 00007ffe`1581dfe8 VBoxDD!lsilogicR3ProcessMessageRequest+0x1fe [c:\virtualbox-src\src\vbox\devices\storage\devlsilogicscsi.cpp @ 1139]
01 00000000`2917f750 00007ffe`1581e1c7 VBoxDD!lsilogicRegisterWrite+0x258 [c:\virtualbox-src\src\vbox\devices\storage\devlsilogicscsi.cpp @ 1426]
02 00000000`2917f790 00007ffe`160bb46e VBoxDD!lsilogicIOPortWrite+0x17 [c:\virtualbox-src\src\vbox\devices\storage\devlsilogicscsi.cpp @ 1729]
03 00000000`2917f7c0 00007ffe`15fbb4da VBoxVMM!IOMIOPortWrite+0xae [c:\virtualbox-src\src\vbox\vmm\vmmall\iomall.cpp @ 417]
04 00000000`2917f820 00007ffe`15fb1bc8 VBoxVMM!IOMR3ProcessForceFlag+0x5a [c:\virtualbox-src\src\vbox\vmm\vmmr3\iom.cpp @ 389]
05 00000000`2917f860 00007ffe`15fb5789 VBoxVMM!emR3HighPriorityPostForcedActions+0xf8 [c:\devel\virtualbox-src\src\vbox\vmm\vmmr3\em.cpp @ 1470]
06 00000000`2917f890 00007ffe`15fb3d8a VBoxVMM!emR3HmExecute+0x129 [c:\virtualbox-src\src\vbox\vmm\vmmr3\emhm.cpp @ 437]
07 00000000`2917f8c0 00007ffe`16014fd4 VBoxVMM!EMR3ExecuteVM+0x43a [c:\virtualbox-src\src\vbox\vmm\vmmr3\em.cpp @ 2658]
08 00000000`2917f950 00007ffe`17b9345f VBoxVMM!vmR3EmulationThreadWithId+0x364 [c:\virtualbox-src\src\vbox\vmm\vmmr3\vmemt.cpp @ 243]
09 00000000`2917f9e0 00007ffe`17c449b2 VBoxRT!rtThreadMain+0x2f [c:\virtualbox-src\src\vbox\runtime\common\misc\thread.cpp @ 727]
0a 00000000`2917fa10 00000000`77f41d9f VBoxRT!rtThreadNativeMain+0x92 [c:\virtualbox-src\src\vbox\runtime\r3\win\thread-win.cpp @ 256]
0b 00000000`2917fa40 00000000`77f41e3b MSVCR100!endthreadex+0x43
0c 00000000`2917fa70 00007ffe`42817c24 MSVCR100!endthreadex+0xdf
0d 00000000`2917faa0 00007ffe`4444d4d1 KERNEL32!BaseThreadInitThunk+0x14
0e 00000000`2917fad0 00000000`00000000 ntdll!RtlUserThreadStart+0x21
```

Exploit

CVE-2021-2119

CVE-2021-2119

OUT-OF-BOUNDS READ in SCSI DEVICES

es > Storage > VBoxSCSI.cpp > vboxscsiReadString(PPDMDEVINS, PVBOXSCSI, uint8_t, uint8_t *, uint32_t *, unsigned)

```
 */
uint32_t cbTransfer = *pcTransfers * cb;
if (pVBoxSCSI->cbBufLeft > 0)
{
    Assert(cbTransfer <= pVBoxSCSI->cbBuf);
    if (cbTransfer > pVBoxSCSI->cbBuf)
    {
        memset(pbDst + pVBoxSCSI->cbBuf, 0xff, cbTransfer - pVBoxSCSI->cbBuf);
        cbTransfer = pVBoxSCSI->cbBuf;  /* Ignore excess data (not supposed to happen). */
    }

    /* Copy the data and adance the buffer position. */
    memcpy(pbDst, pVBoxSCSI->pbBuf + pVBoxSCSI->iBuf, cbTransfer);

    /* Advance current buffer position. */
    pVBoxSCSI->iBuf      += cbTransfer;
    pVBoxSCSI->cbBufLeft -= cbTransfer;

    /* When the guest reads the last byte from the data in buffer, clear
       everything and reset command buffer. */
    if (pVBoxSCSI->cbBufLeft == 0)
        vboxscsiReset(pVBoxSCSI, false /*fEverything*/);
}
```

This check can be bypassed

OOR Info Leak

Integer overflow

vboxscsiReadString is the “rep in “ handler of the scsi device.

cbTransfer is controllable by the guest.

The lack of checks on cbTransfer and cbBufLeft in vboxscsiReadString leads to OOR.

CVE-2021-2119

OUT-OF-BOUNDS WRITE in SCSI DEVICES

es > Storage > VBoxSCSI.cpp > vboxscsiWriteString(PPDMDEVINS, PVBOXSCSI, uint8_t, uint8_t const *, uint32_t *, unsigned)

```
 */
int rc = VINF_SUCCESS;
if (pVBoxSCSI->cbBufLeft > 0)
{
    uint32_t cbTransfer = RT_MIN(*pcTransfers * cb, pVBoxSCSI->cbBufLeft);

    /* Copy the data and adance the buffer position. */
    memcpy(pVBoxSCSI->pbBuf + pVBoxSCSI->iBuf, pbSrc, cbTransfer);
    pVBoxSCSI->iBuf      += cbTransfer;
    pVBoxSCSI->cbBufLeft -= cbTransfer;

    /* If we've reached the end, tell the caller to submit the command. */
    if (pVBoxSCSI->cbBufLeft == 0)
    {
        ASMAtomicXchgBool(&pVBoxSCSI->fBusy, true);
        rc = VERR_MORE_DATA;
    }
}
```

由于vboxscsiReadString中的漏洞导致cbBufLeft整型溢出，因此可以绕过cbBufLeft>0的判断，造成OOW。

CVE-2021-2119

Exploit

OOB SCSI DEVICES

该漏洞在RWCTF中已经产生一个版本的利用，链接如下[3]

该版本利用主要使用了HGCM相关的对象，关于HGCM的使用， niklasb已经做了很详细的介绍，链接如下[2]

今天我们将介绍另外一种利用原语，它SVGA3D相关。
VirtualBox在3D之路上走十分艰难，VBox3D的HGCM/Chromium给Vbox带来了很多安全漏洞，因此在6.1版本后删除了该模块，同时也删除了很好用的利用原语CRConnection/CRClient。[3]

Exploit

MEMORY CORRUPTION(OOW) IN USBMSD FROM GUEST TO HOST

PART 4

Demo Time

Demo Time

Acknowledgements

Kelwin of Chaitin Security Research Lab

M4x of Chaitin Security Research Lab

Swing of Chaitin Security Research Lab

explorer of Chaitin Security Research Lab

f1yyy of Chaitin Security Research Lab

Leommxj of Chaitin Security Research Lab

Thanks

@RealWorldCTF

Reference

[1] https://secret.club/2021/01/14/vbox-escape.html

[2] https://github.com/niklasb/sploits/tree/master/virtualbox/hgcm-oob

[3] https://www.coresecurity.com/corelabs-research/publications/breaking-out-virtualbox-through-3d-acceleration

[4] http://blog.paulch.ru/2020-07-26-hunting-for-bugs-in-virtualbox-first-take.html

[5] https://starlabs.sg/blog/2020/04/adventures-in-hypervisor-oracle-virtualbox-research/