Box Escape: Discovering 10+ Vulnerabilities in VirtualBox

ChenNan
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://chaitin.cn/en)
  [https://realworldctf.com/](https://realworldctf.com/)

- Chaitin Security Research Lab
  - Pwn2Own 2017 3rd place
    - 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
    - 4th 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

Host

- Ring3
  - VBoxSVC
  - VirtualBoxVM

- Ring0
  - OS Driver
  - VBoxNet...
  - VBoxDDR0
  - VBoxDrv
  - VMMR0

Guest

- Ring3
  - VBoxControl
  - VBoxService
  - VBoxMRXNP

- Ring0
  - OS Driver
  - VBoxMouse
  - VBoxDisp
  - VBoxVRAT
  - VBoxGL...
  - VBoxVideo
  - VBoxWddm
  - VBoxSF
  - VBoxGuest
General Communication
General Communication

Host
- Ring3
  - VBoxSVC
  - VBoxCow
  - VirtualBoxVM
  - R3 Part of Emulation Devices
- Ring0
  - OS Driver
  - VBoxNet...
  - VBoxDDR0
  - VBoxDrv
  - VMMR0
- Call the real hardware driver
- R0 Part of Emulation Devices

Guest
- Ring3
  - VBoxControl
  - VBoxService
  - VBoxMRXNP
  - VBoxDisp
  - VBoxGL...
- Ring0
  - Device Drivers
  - OS Driver
  - VBoxVideo
  - VBoxMouse
  - VBoxWddm
  - VBoxGuest
  - VBoxSF
  - VBoxGuest

Emulation Devices: Guest does not require vbox drivers.
General Communication

Host

- Ring3
  - VBoxSVC
  - VirtualBoxVM
  - R3 Part of Emulation Devices
  - Call the real hardware driver

- Ring0
  - OS Driver
  - VMMR0
  - VBoxDrv
  - VBoxDDR0
  - R0 Part of Emulation Devices

Guest

- Ring3
  - VBoxControl
  - VBoxService
  - VBoxMRXNP

- Ring0
  - Device Drivers
  - VBoxDisp
  - VBoxGL...
  - VBoxGuest
  - VBoxMouse
  - VBoxVideo
  - VBoxWddm
  - VBoxSF
  - OS Driver

Emulation Devices: Guest does not require vbox drivers.

General Communication Methods (Usually Documented In Datasheet):
- IO: IN, OUT, REP IN, REP OUT
- MMIO or DMA: MOV
- Special instructions: CPUID, INT, MSR, HyperCall, etc...

Call the real hardware driver
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

Host

<table>
<thead>
<tr>
<th>Ring3</th>
<th>VBoxSVC</th>
<th>VirtualBoxVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ring0</td>
<td>OS Driver</td>
<td>VBoxNet...</td>
</tr>
<tr>
<td></td>
<td>VBoxDDR0</td>
<td>VBoxDrv</td>
</tr>
</tbody>
</table>

Guest

<table>
<thead>
<tr>
<th>Ring3</th>
<th>VBoxControl</th>
</tr>
</thead>
<tbody>
<tr>
<td>VBoxService</td>
<td>VBoxMRXNP</td>
</tr>
<tr>
<td>VBoxDisp</td>
<td>VBoxGL...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Ring0</th>
<th>OS Driver</th>
</tr>
</thead>
<tbody>
<tr>
<td>VBoxMouse</td>
<td>VBoxVideo</td>
</tr>
<tr>
<td>VBoxWddm</td>
<td>VBoxGuest</td>
</tr>
<tr>
<td>VBoxSF</td>
<td></td>
</tr>
</tbody>
</table>

Special Devices: Guest needs vbox drivers.
Special Communication

Host

Ring3
- VBoxSVC
- VirtualBoxVM

R3 Part of Special Devices

Special Communication Methods:
(Usually without undocumented)
Special ioport, MMIO, DMA.
Custom communication: HGSMI, HGCM, FIFO

Ring0
- VBoxDrv
- VBoxDDR0
- VMMR0

R0 Part of Special Devices

General Communication Methods Similar to Emulated Devices.
Such as ioport, mmio, etc.

Special Devices: Guest needs vbox drivers.

Ring3
- VBoxControl
- VBoxService
- VBoxMRXNP
- VBoxDisp
- VBoxGL...

Ring0
- OS Driver
- VBoxMouse
- VBoxVideo
- VBoxWddm

Guest

Ring3
- VBoxService
- VBoxControl
- VBoxMRXNP
- VBoxDisp
- VBoxGL...

Special Devices: Guest needs vbox drivers.

Special Communication Methods:
(Usually without undocumented)
Special ioport, MMIO, DMA.
Custom communication: HGSMI, HGCM, FIFO

Ring0
- OS Driver
- VBoxMouse
- VBoxVideo
- VBoxWddm

R0 Part of Special Devices

General Communication Methods Similar to Emulated Devices.
Such as ioport, mmio, etc.
Special Communication

Essentially, special communication is operated based on general communication instructions, such as ioport, mmio, etc. Its special feature is that it is not a standard protocol, it is only for virtualbox.
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.

Guest and Backend have indirect communication. **Backend** is a big attack surface.
The USB controller device and backend will communicate through a driver component called **URB**.
The storage controller device and backend will communicate through a driver component called VD (Virtual Disk).
The VMM device is a custom hardware device emulation for communicating with the guest additions. It uses HGCM to communicate with the backend.
SVGA3D is a device used to support hardware GPU accelerated 3D rendering. It uses **direct calls** to communicate with 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

```c
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.
    ...
}
```
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.
...
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(...);
  ...
}

/* Use reserved VGA IO ports for HGSMI. */
REG_PORT(VGA_PORT_HGSMI_HOST, 4, vga3IOPortHgsmiWrite, vga3IOPortHgsmiRead, "HGSMI host (3b0-3b3)", &This-
REG_PORT(VGA_PORT_HGSMI_GUEST, 4, vga3IOPortHgsmiWrite, vga3IOPortHgsmiRead, "HGSMI guest (3d0-3d3)", &This-
/* VBOX_WITH_HGSMI */

/* VGA BIOS */
r = PDMDevHlpIoPortCreateAndMap(pDevIns, VBE_PRINTF_PORT, 1 /*cPorts*/, vgaIoPortWriteBios, vgaIoPortReadBios,
  "VGA BIOS debug/panic", NULL /*pExtDescs*/, &pThis->hIoPortBios);

/* The MDA/CGA/EGA/VGA/whatever fixed MMIO area. */
r = PDMDevHlpMmioCreateExAndMap(pDevIns, 0x000a0000, 0x00020000,
  IOMMIO_FLAGS_READ_PASSTHRU | IOMMIO_FLAGS_WRITE_PASSTHRU | IOMMIO_FLAGS_ABS
  NULL /*pPciDev*/, UINT32_MAX /*iPciRegion*/,
  vgaMmioWrite, vgaMmioRead, vgaMmioFill, NULL /*pvUser*/,
  "VGA - VGA Video Buffer", &pThis->hMmioLegacy);

AssertRCCReturn(rc, rc);
*/
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]
| Port I/O Handler Case |
| Memory Mapped I/O Handler Case |
Direct Physical Memory Read/Write

<table>
<thead>
<tr>
<th>Guest’s Physical Memory Read/Write Handlers</th>
</tr>
</thead>
<tbody>
<tr>
<td>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 <code>memmove</code>, so this is an important attack point.</td>
</tr>
</tbody>
</table>

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)
<table>
<thead>
<tr>
<th>Guest’s Physical Memory Read/Write Case</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
struct PDMUSBREG{
    ...
    pfnUrbQueue(PPDMUSBINS pUsbIns, PVUSBURB pUrb); //Queues an URB for processing
    ...
}

struct VUSBURB{
    ...
    uint32_t cbData;
    uint8_t abData[8*_1K]; //Guest Controllable
    ...
}
USB Backend

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

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

```
407 extern "C" DECLEXPORT(int) VBoxUsbRegister(PCPDMUSBREGCB pCallbacks, uint32_t u32Version)
408 {
    int rc = VINF_SUCCESS;
    RT_NOREF1(u32Version);
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;
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;
430 #endif #ifdef VBOX_WITH_USB_VIDEO_IMPL
431    rc = pCallbacks->pfnRegister(pCallbacks, &g_DevWebcam);
```
<table>
<thead>
<tr>
<th>USB Backend Case</th>
</tr>
</thead>
</table>

gf
Network Backend

Slirp case
Hack with CodeQL
<table>
<thead>
<tr>
<th>What is this?</th>
</tr>
</thead>
<tbody>
<tr>
<td>CodeQL is the analysis engine used by developers to automate security checks, and by security researchers to perform variant analysis.</td>
</tr>
<tr>
<td>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.</td>
</tr>
</tbody>
</table>

Related Links
- [https://securitylab.github.com/tools/codeql](https://securitylab.github.com/tools/codeql)
- [https://codeql.github.com/docs/](https://codeql.github.com/docs/)
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

```bash
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.

```cpp
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)
}
```
Second, we need to determine the destination. The destination here is the third length parameter of `memcpy`.

```java
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())
}
```
Third, add data polluted by source data to query.

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

```java
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)
    )
}
```
Finally, execute the query.

```java
from Config cfg, DataFlow::PathNode source, DataFlow::PathNode sink
where cfg.hasFlowPath(source, sink)
select source.getNode().getLocation(), sink.getNode().getLocation()
```
Check the result and find vulnerabilities.
Fuzz with AFL++
**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](https://github.com/AFLplusplus/AFLplusplus)
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?
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.

.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 /* NULL */ ];
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.

```bash
# 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
```
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.
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.
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]) - 1];
    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]); i++)
            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

Fuzz PCNET Device

static DECLCALLBACK(void) PCNET3FuzzEntry(PPDMDEVINS pDevIns)
{
    uint32_t readlen = 0x2000;
    Fuzz_Init(g_PCIE_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);
    }
}

Fuzz PCNET Device
DECLINE(int) PDMDevHlpPhysRead(PPDMDEVINS pDevIns, RTGCPHYSC GPhys, 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, GPhys, pvBuf, cbRead);
}
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.
The biggest reason for the slowness is the startup process. So we use Persistent Mode, a cool feature of AFL++. 


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.

```
american fuzzy lop ++3.00a (default) [explore] {}

overall results
  cycles done: 0
total paths: 40
uniq crashes: 0
uniq hangs: 0

map coverage: 2.39% / 13.00%
map density: 7.82 bits/tuple

findings in depth
  favored paths: 1 (2.50%)
  new edges on: 22 (55.00%)
  total crashes: 0 (0 unique)
  total tmouts: 0 (0 unique)

exec speed: 354.6/sec
```

- fuzzer:
  - levels: 2
  - pending: 40
  - pend fav: 1
  - own finds: 39
  - imported: 0
  - stability: 2.68%

- test harness:
  - trim: 0.66%/1263, n/a

[cpu000: 18%]
Results

10+ vulnerabilities

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  ...
MEMORY CORRUPTION (OOW) IN USBMSD FROM GUEST TO HOST

**Summary**

**Start OHCI Device**

Set OHCIED and OHCITD through ioport. These two structures contain the physical address of the data packet and the id of the backend.

**Bulk List Enable. Send data to URB.**

URBCore sends data to usb backend.

**usbMsdQueue parses the input data into CUSBCBW, and lacks checks on pCbw->bCBWCBLength, resulting in OOW.**

**usbMsdQueue of USBMAD Backend**
CVE-2021-2112

**Summary**

MEMORY CORRUPTION (OOW) IN USBMSD FROM GUEST TO HOST

```c
static void usbMsdReqPrepare(PUSBMSDREQ pReq, PCUSBCBW pCbw)
{
    /* Copy the CBW */
    size_t cbCopy = RT_OFFSETOF_DYN(USBCBW, CBWC[pCbw->bCBWCBLen]);
    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 (OWO) IN USBMSD FROM GUEST TO HOST
CVE-2021-2131

(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\vusurb.cpp @ 1001]
02 00000000`43c6f520 00007ffa`db1e2d46 VBoxDD\vusbUrbSubmit+0x1ae [c:\virtualbox-src\src\vbox\devices\usb\vusurb.cpp @ 1205]
03 00000000`43c6f560 00007ffa`f1c6fa5f VBoxDD\vusbRhSubmitUrb+0xa6 [c:\virtualbox-src\src\vbox\devices\usb\drvusbroothub.cpp @ 714]
04 00000000`43c6f5a0 00007ffa`f1c627e1 VBoxEhciR3+0x1fa5
05 00000000`43c6f630 00007ffa`f1c62a1f VBoxEhciR3+0x27e1
06 00000000`43c6f6e0 00007ffa`f1c6265 VBoxEhciR3+0x2a1f
07 00000000`43c6f7b0 00007ffa`f1c62da9 VBoxEhciR3+0x2c65
08 00000000`43c6f850 00007ffa`f1c62fbb VBoxEhciR3+0x2da9
09 00000000`43c6f890 00007ffa`f1c67ef9 VBoxEhciR3+0x2fbb
0a 00000000`43c6f900 00007ffa`f1c67ef9 VBoxVM\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`f1c22d41 KERNEL32!BaseThreadInitThunk+0x14
10 00000000`43c6fa50 00000000`00000000 ntdll!RtlUserThreadStart+0x21
**CVE-2021-2120**

(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=

<table>
<thead>
<tr>
<th># Child-SP</th>
<th>RetAddr</th>
<th>Call Site</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 00000000'2917f700 00007ffe 1581afe8 VBoxDD!lsilogicR3ProcessMessageRequest+0x1fe [c:\virtualbox-src\src\vbox\devices\storage\devlsilogicscsi.cpp @ 1139]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01 00000000'2917f750 00007ffe 1581e1c7 VBoxDD!lsilogicRegisterWrite+0x258 [c:\virtualbox-src\src\vbox\devices\storage\devlsilogicscsi.cpp @ 1426]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>02 00000000'2917f790 00007ffe 160b4b6e VBoxDD!lsilogicOPortWrite+0x17 [c:\virtualbox-src\src\vbox\devices\storage\devlsilogicscsi.cpp @ 1729]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>03 00000000'2917f7c0 00007ffe 15fb4da VBoxVMM!OMIOPortWrite+0xae [c:\virtualbox-src\src\vbox\vmm\vmmall\iomall.cpp @ 417]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>04 00000000'2917f820 00007ffe 15fb1bc8 VBoxVMM!OMR3ProcessForceFlag+0x5a [c:\virtualbox-src\src\vbox\vmm\vmmr3\iom.cpp @ 389]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>05 00000000'2917f860 00007ffe 15fb5789 VBoxVMM!emR3HighPriorityPostForcedActions+0xf8 [c:\devel\virtualbox-src\src\vbox\vmm\vmmr3\em.cpp @ 1470]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>06 00000000'2917f890 00007ffe 15fb3d8a VBoxVMM!emR3HmExecute+0x129 [c:\virtualbox-src\src\vbox\vmm\vmmr3\vhmm.cpp @ 437]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>07 00000000'2917f8c0 00007ffe 16014df4 VBoxVMM!EMR3ExecuteVM+0x43a [c:\virtualbox-src\src\vbox\vmm\vmmr3\em.cpp @ 2658]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>08 00000000'2917f950 00007ffe 17b9345f VBoxVMM!vmR3EmulationThreadWithId+0x364 [c:\virtualbox-src\src\vbox\vmm\vmmr3\vmmemt.cpp @ 243]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>09 00000000'2917f9e0 00007ffe 17c44f2 VBoxRT!rtThreadMain+0x2f [c:\virtualbox-src\src\vbox\runtime\common\misc\thread.cpp @ 727]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0a 00000000'2917fa10 00000000 77f41d9f VBoxRT!rtThreadNativeMain+0x92 [c:\virtualbox-src\src\vbox\runtime\r3\win\thread-win.cpp @ 256]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0b 00000000'2917fa40 00000000 77f41e3b MSVCRT100!lendthreadex+0x43</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0c 00000000'2917fa70 00007ffe 42817c24 MSVCRT100!lendthreadex+0xdf</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0d 00000000'2917faa0 00007ffe 44444d4d KERNEL32!BaseThreadInitThunk+0x14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0e 00000000'2917fad0 00000000'00000000 ntdll!RtlUserThreadStart+0x21</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Exploit

CVE-2021-2119
**CVE-2021-2119**

OUT-OF-BOUNDS READ in SCSI DEVICES

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.

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

/* Copy the data and advance the buffer position. */
memcpy(pData + VBoxSCSI->cbBuf + pVBoxSCSI->cbBuf, cbTransfer);

/* Advance current buffer position. */
pVBoxSCSI->cbBuf += 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
OUT-OF-BOUNDS WRITE in SCSI DEVICES

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 (OOO) IN USBMSD FROM GUEST TO HOST
PART 4

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


