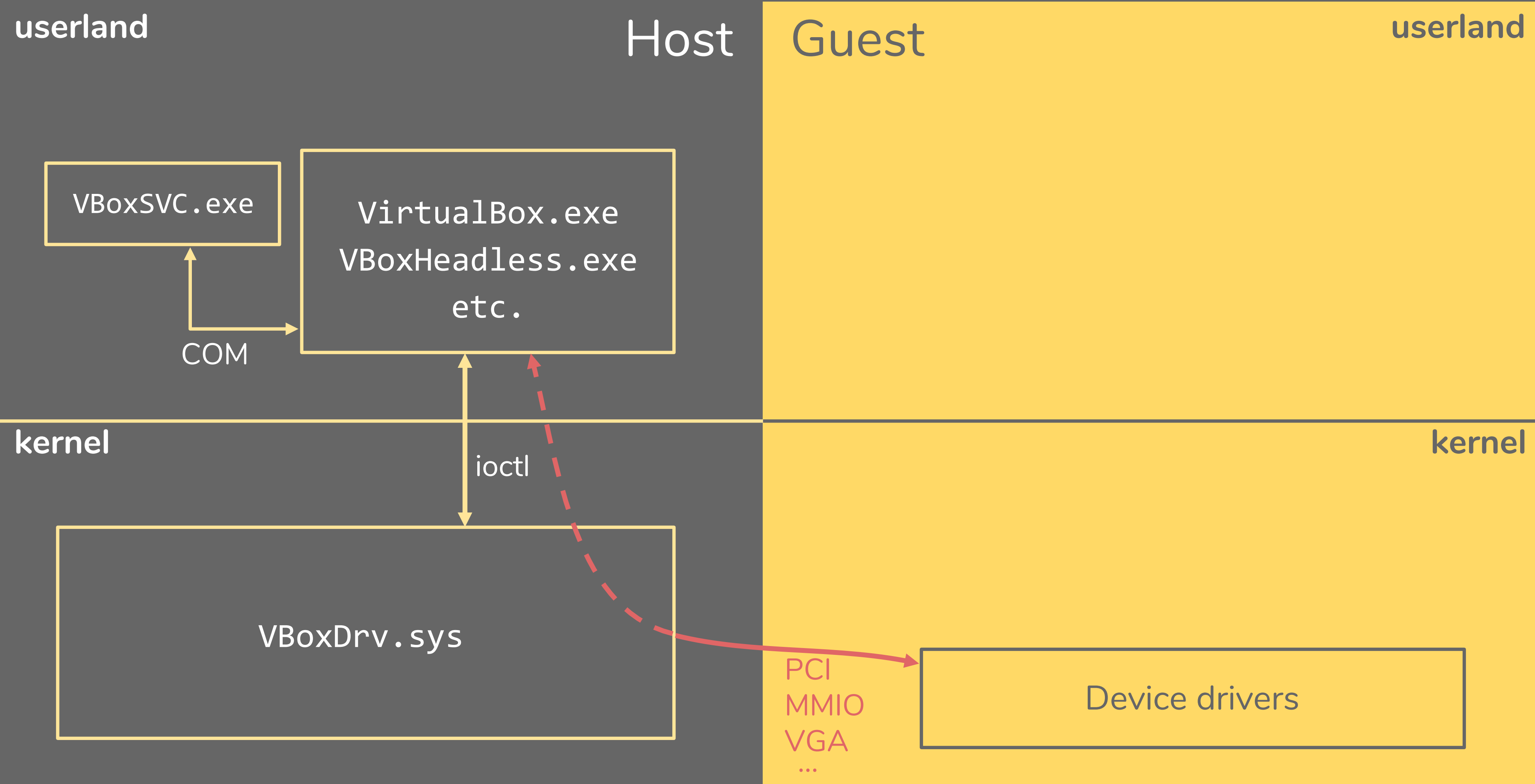
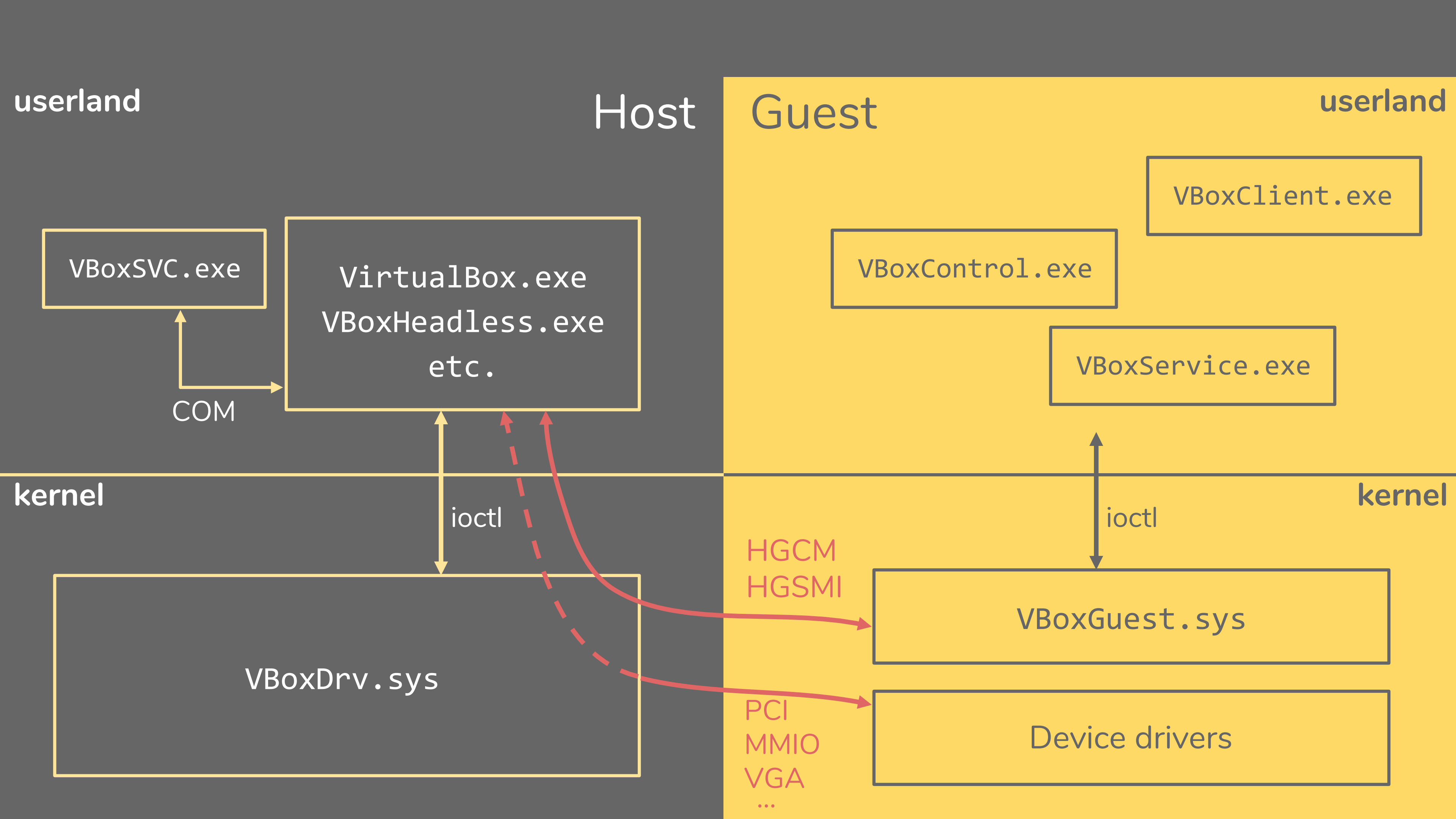
Thinking outside the (Virtual)Box

Niklas Baumstark
@_niklasb


userland
Host
Guest
userland
VBoxSVC.exe
VirtualBox.exe
VBoxHeadless.exe
etc.
COM
kernel
ioctl
kernel
VBoxDrv.sys
PCI
MMIO
VGA
...
Device drivers


userland
Host
Guest
userland
VBoxClient.exe
VBoxSVC.exe
VirtualBox.exe
VBoxHeadless.exe
etc.
VBoxControl.exe
VBoxService.exe
COM
kernel
ioctl
ioctl
kernel
HGCM
HGSMI
VBoxGuest.sys
VBoxDrv.sys
PCI
MMIO
VGA
...
Device drivers

Guest-to-host interfaces

- Hypervisor `/src/VBox/VMM`
  - Memory manager
  - x86 emulation
- Emulated devices `/src/VBox/Devices`
  - Audio
  - Networking
  - Graphics (VGA)
  - AHCI
  - ACPI
  - USB
  - Virtual Machine Monitor device
  - Paravirtualization interface (KVM/Hyper-V)
- HGCM services `/src/VBox/HostServices`
  - Shared OpenGL
  - Drag & Drop
  - Shared folders
  - Shared clipboard
- HGSMI services
  - VirtualBox Video Acceleration (VBVA)

Guest-to-host interfaces

- Hypervisor `/src/VBox/VMM`
  - Memory manager
  - x86 emulation
- Emulated devices `/src/VBox/Devices`
  - Audio
  - Networking
  - Graphics (VGA)
  - AHCI
  - ACPI
  - USB
  - Virtual Machine Monitor device
  - Paravirtualization interface (KVM/Hyper-V)

- HGCM services `/src/VBox/HostServices`
  - Shared OpenGL (20+ CVEs...)
  - Drag & Drop
  - Shared folders
  - Shared clipboard
- HGSMI services
  - VirtualBox Video Acceleration (VBVA)

Comparison to VMware Workstation

- Many features are disabled (= secure :) by default
  - 3D support
  - Drag & drop
  - Clipboard sharing
  - USB 2.0 & 3.0
- Some vectors do not exist
  - ThinPrint
- No userland RPC backdoor
- VirtualBox userland parts are privileged, privesc to host kernel is trivial

Host-Guest Communication Manager

- Simple RPC protocol, handled by the VMM PCI device
- Guest allocates request buffer of type
  - `VMMDevHGCMConnect` or
  - `VMMDevHGCMDisconnect` or
  - `VMMDevHGCMCall`
- Physical address of request is written to I/O port
- Call request specifies function ID and parameters
  - Integers
  - Buffers

`/src/VBox/HostServices`

HGCM - Services

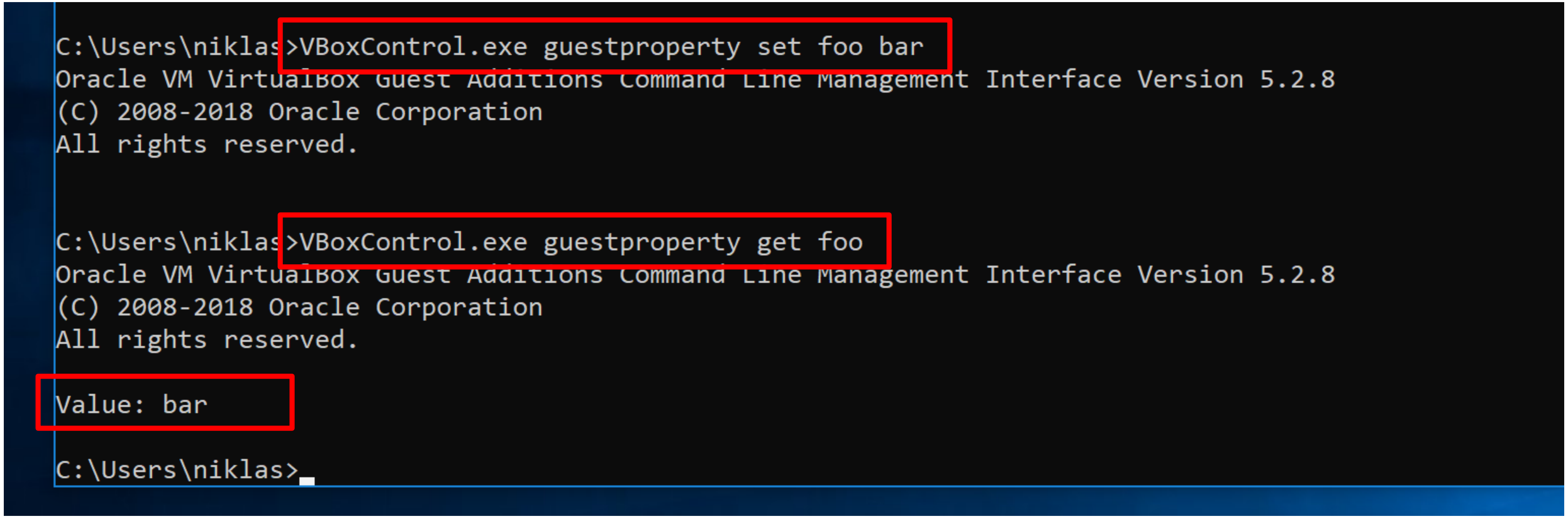
- `VBoxSharedClipboard`
- `VBoxDragAndDropSvc`
- `VBoxGuestPropSvc`
- `VBoxGuestControlSvc`
- `VBoxSharedFolders`
- `VBoxSharedCrOpenGL`

HGCM - Services

- `VBoxSharedClipboard`
- `VBoxDragAndDropSvc`
- `VBoxGuestPropSvc`
- `VBoxGuestControlSvc`
- `VBoxSharedFolders`
- `VBoxSharedCrOpenGL`

/src/VBox/HostServices/GuestProperties/service.cpp

HGCM - Example



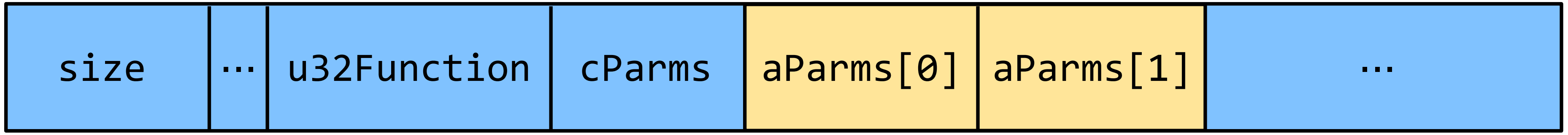
```
C:\Users\niklas>VBoxControl.exe guestproperty set foo bar
Oracle VM VirtualBox Guest Additions Command Line Management Interface Version 5.2.8
(C) 2008-2018 Oracle Corporation
All rights reserved.


C:\Users\niklas>VBoxControl.exe guestproperty get foo
Oracle VM VirtualBox Guest Additions Command Line Management Interface Version 5.2.8
(C) 2008-2018 Oracle Corporation
All rights reserved.

Value: bar

C:\Users\niklas>_
```

```
VMMDevHGCMConnect("VBoxGuestPropSvc") = 42
VMMDevHGCMCall(42, SET_PROP, "foo", "bar") = VERR_SUCCESS
VMMDevHGCMCall(42, GET_PROP, "foo", <result buffer>, ...) = VERR_SUCCESS
```

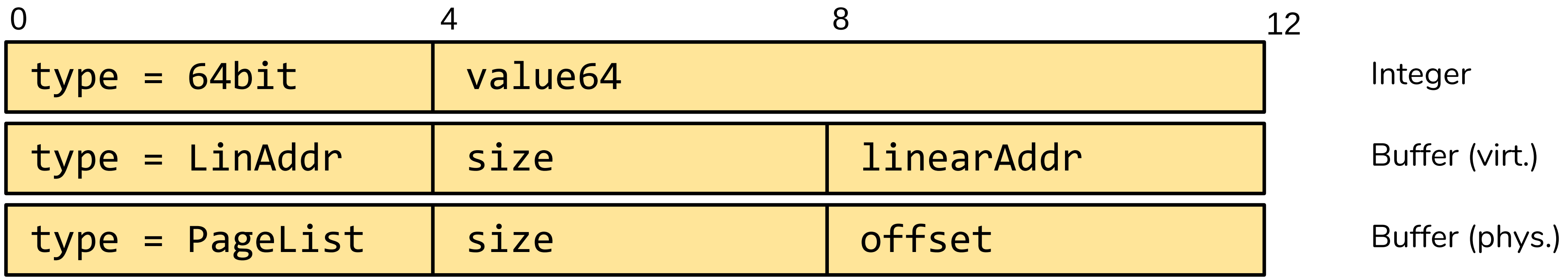
/src/VBox/Devices/VMMDev/VMMDevHGCM.cpp

HGCM - Call request

VMMDevHGCMCall

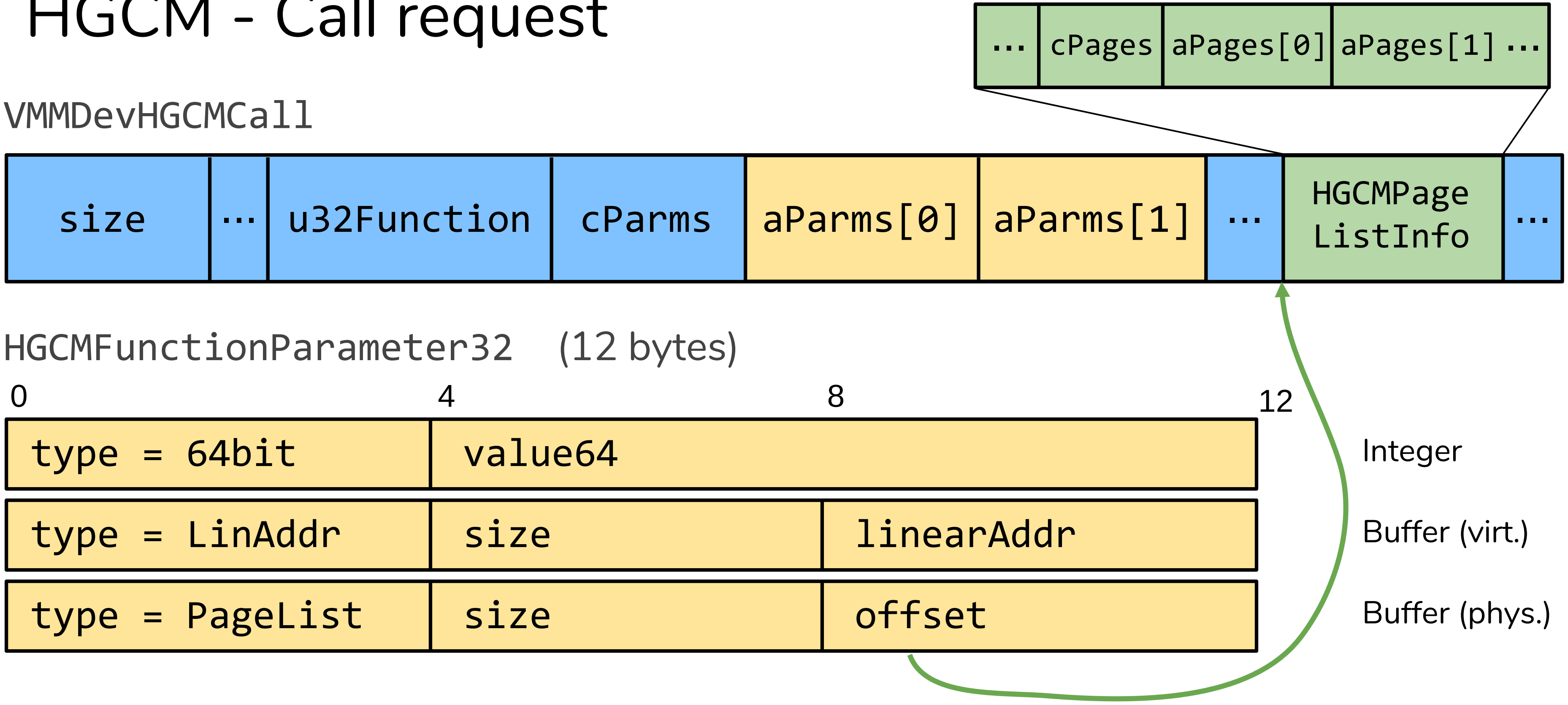


HGCMFunctionParameter32 (12 bytes)



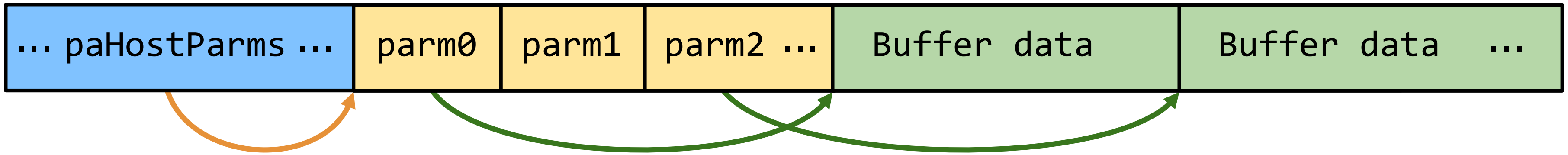
/src/VBox/Devices/VMMDev/VMMDevHGCM.cpp

HGCM - Call request



HGCM - Call handling (< 5.2.10)

1. Copy `VMMDevHGCMCall` to host heap
2. Allocate `VBOXHGCMCMD` large enough to hold copy of parameters (host params) & buffer data
3. Copy buffer data from the guest into the `VBOXHGCMCMD`



Bug #1: Double fetch on buffer write-back

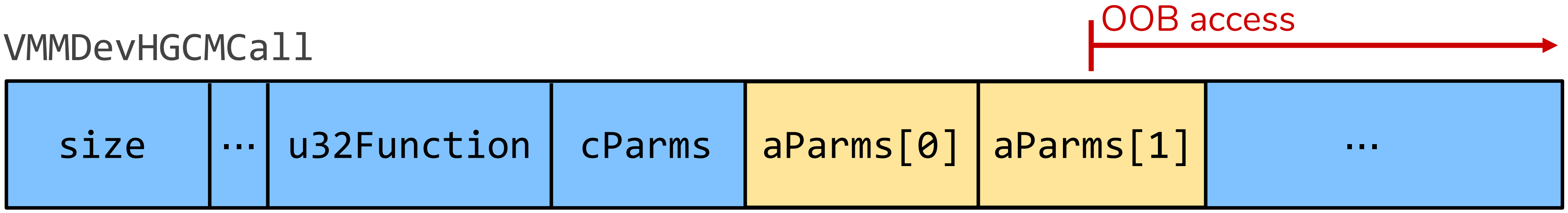
ZERO DAY INITIATIVE

- Most HGCM functions return data
- Implemented by writing back VBOXHGCMCMD buffers to guest memory
- hgcmCompletedWorker re-fetches the request to determine sizes
- Disclose heap memory by increasing the size during dispatch!

```
case VMMDevHGCMParmType_LinAddr: {
   /* Copy buffer back to guest memory. */
   uint32_t size = pGuestParm->u.Pointer.size;
   ...
       /* Use the saved page list to write data back to the guest RAM. */
       rc = vmmdevHGCMWriteLinPtr (...,
                                   pHostParm->u.pointer.addr,
                                   size, ...);
```

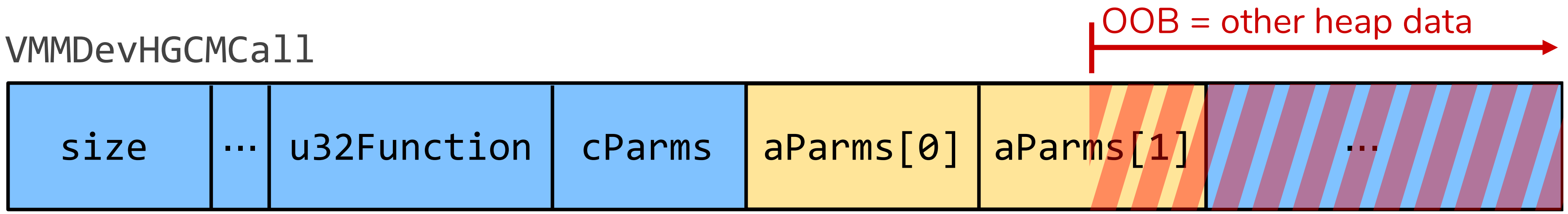
Bug #2: Heap out-of-bounds read

- `VMMDevHGCMCall` is copied from guest to the host heap
  - `size` bytes are copied
  - No check that `size` is large enough to hold all parameters
  - Later: OOB read access on the heap
- Looks harmless, because the guest fully controls the object anyways



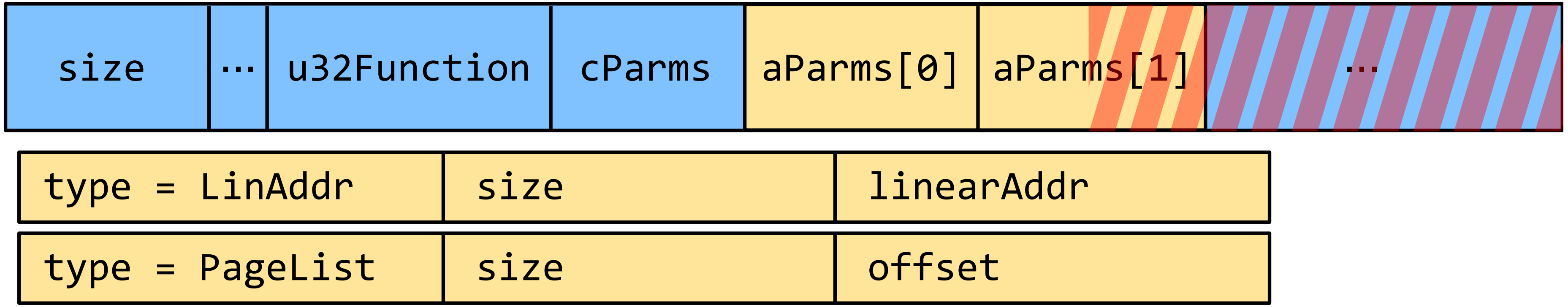
Bug #2: Heap out-of-bounds **double** read

- `VMMDevHGCMCall` is copied from guest to the host heap
    - `size` bytes are copied
    - No check that `size` is large enough to hold all parameters
    - Later: OOB read access on the heap
- Looks harmless, because the guest fully controls the object anyways
- But: **parameters are accessed twice!** TOCTOU issue?

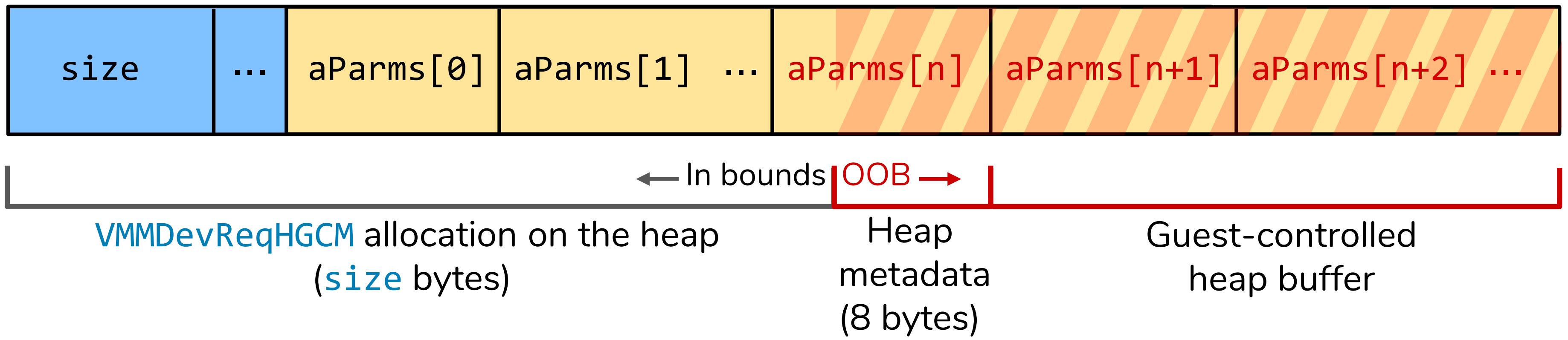


Bug #2: Heap out-of-bounds **double** read

- **Pass 1**: `cbCmdSize` variable sums up `size` values for buffer params
- Allocate `VBOXHGCMCMD` including space for `cbCmdSize` bytes of data
- **Pass 2**: Copy data from guest
    - Due to OOB access we can change `size` values concurrently



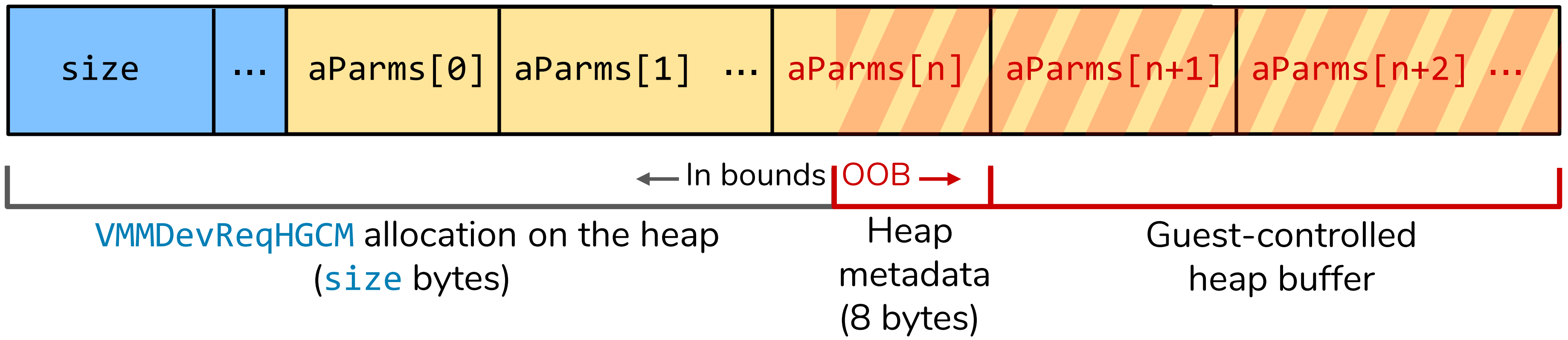
Bug #2: Turn into OOB write



Params *n* + 1 and upwards are in a different heap chunk, we can race them between the two passes

=> Requires two vCPUs

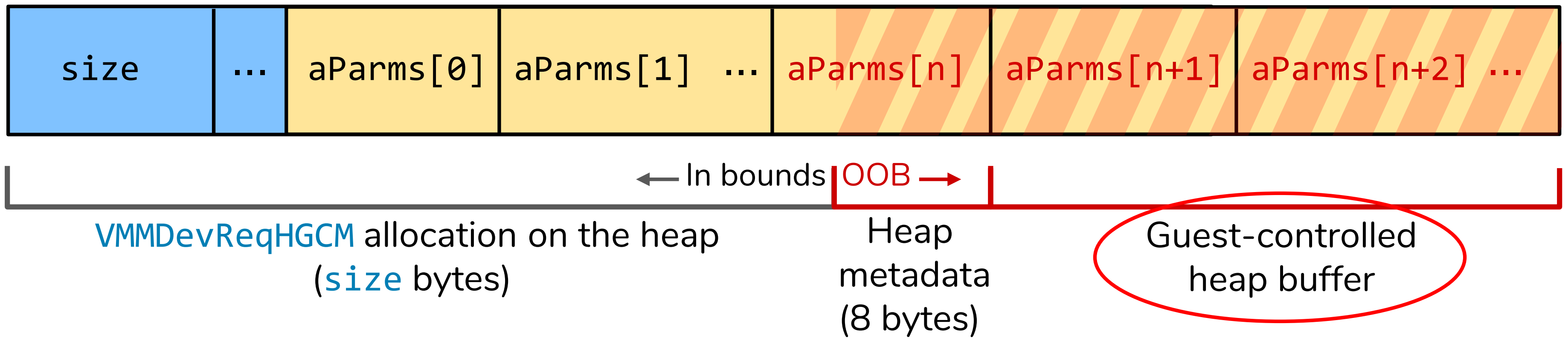
Bug #2: Turn into OOB write



Challenge 1: Find an object on the heap that we can write repeatedly

Challenge 2: Incorporate heap metadata into the request

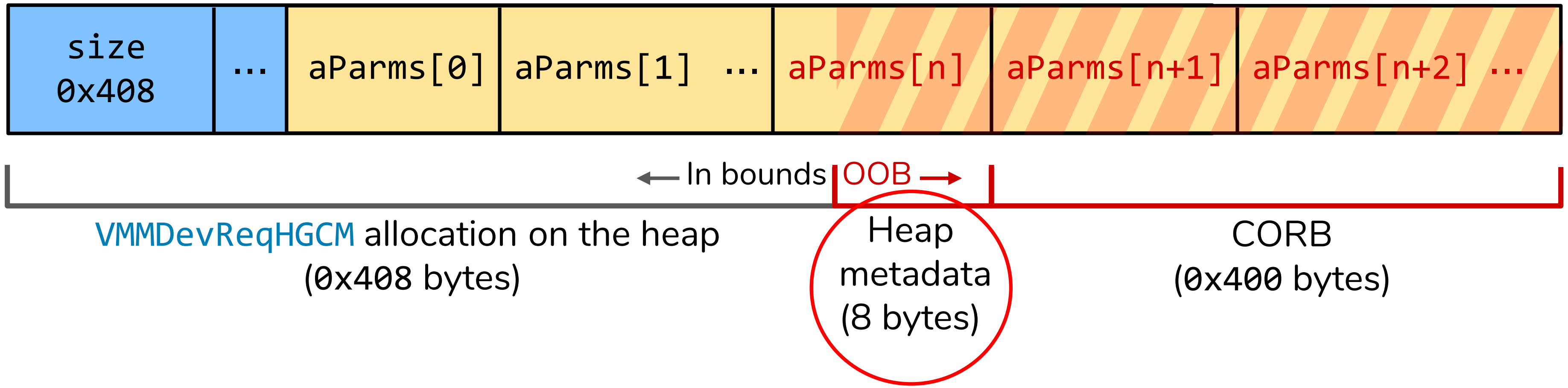
Bug #2: Turn into OOB write



Find an object on the heap that we can write repeatedly

- Intel HD audio device *command output ring buffer* (CORB)
  - `0x400` bytes, can be re-allocated at will

Bug #2: Turn into OOB write



Incorporate heap metadata into request

- CORB size `0x400` => LFH bucket size `0x410` (incl. 8 bytes metadata)
- `VMMDevReqHGCM` with 83 parameters has size `0x410`
    - Last 8 bytes of 83rd parameter are uncontrolled heap metadata
    - This is ok for integer parameters!

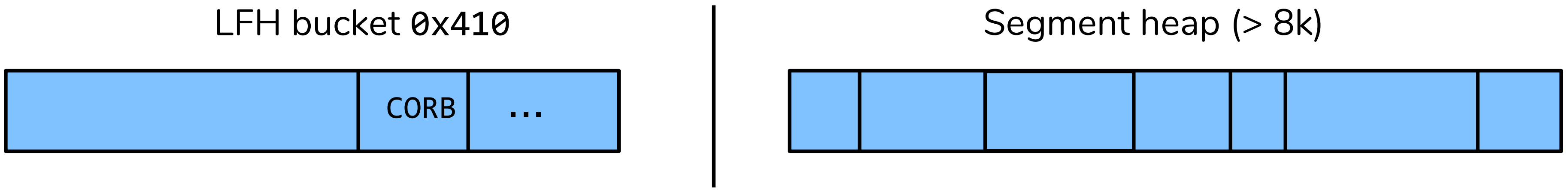
Bug #2: Make it an OOB write

One thread constantly flips a `PageList` parameter size in CORB

```
uint32_t size = pGuestParm->u.PageList.size;  // <- fully controlled!
...
// This will happily read less than size bytes, if page list is smaller
rc = vmmdevHGCMPageListRead(pThis->pDevIns, pcBuf, size, pPageListInfo);
...
pcBuf += size; // <- will be used as destination for the next parameter
```

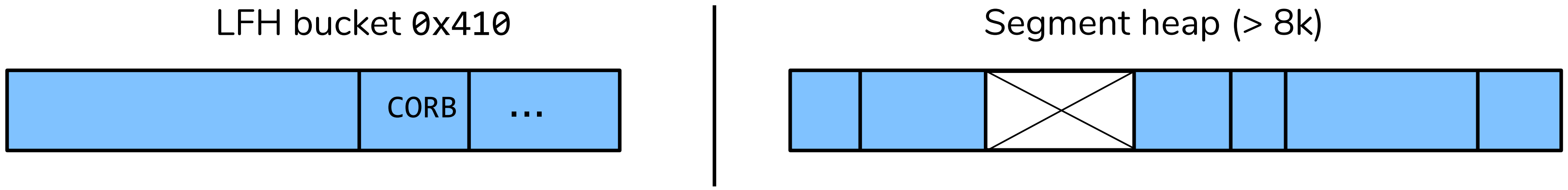
Exploit

- Powerful, heap-based **relative** read and write primitives
- `VBOXHGCMCMD` is variable-size, we can put it on a “nice” heap
  - Allocator fully predictable, with help of bug #1
- Can already leak some vtable pointers from `VBoxC.dll` with bug #1
- Next: want to corrupt
  - a pointer that is read from (for full ASLR break)
  - a function or vtable pointer (for control flow hijack)



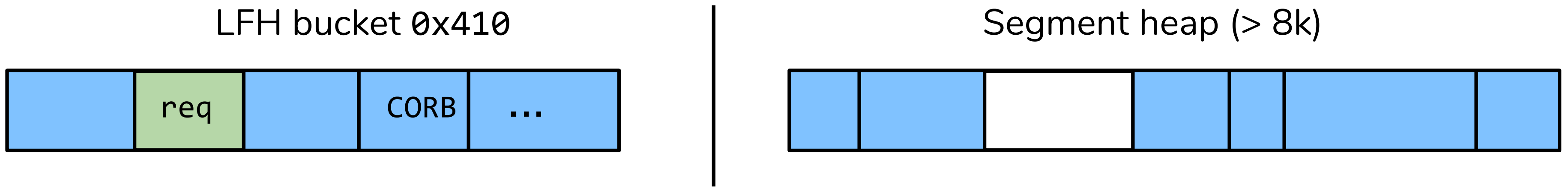
Exploit

- Powerful, heap-based **relative** read and write primitives
- `VBOXHGCMCMD` is variable-size, we can put it on a “nice” heap
  - Allocator fully predictable, with help of bug #1
- Can already leak some vtable pointers from `VBoxC.dll` with bug #1
- Next: want to corrupt
  - a pointer that is read from (for full ASLR break)
  - a function or vtable pointer (for control flow hijack)



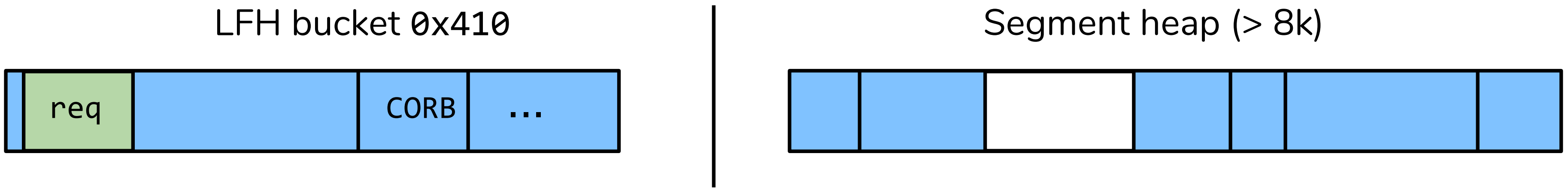
Exploit

- Powerful, heap-based **relative** read and write primitives
- `VBOXHGCMCMD` is variable-size, we can put it on a “nice” heap
  - Allocator fully predictable, with help of bug #1
- Can already leak some vtable pointers from `VBoxC.dll` with bug #1
- Next: want to corrupt
  - a pointer that is read from (for full ASLR break)
  - a function or vtable pointer (for control flow hijack)



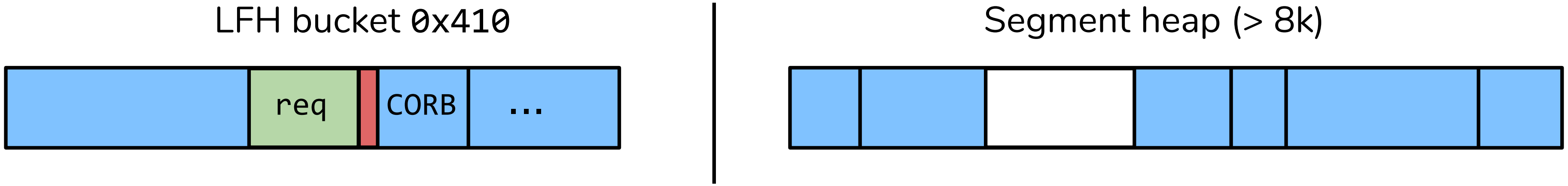
Exploit

- Powerful, heap-based **relative** read and write primitives
- `VBOXHGCMCMD` is variable-size, we can put it on a “nice” heap
  - Allocator fully predictable, with help of bug #1
- Can already leak some vtable pointers from `VBoxC.dll` with bug #1
- Next: want to corrupt
  - a pointer that is read from (for full ASLR break)
  - a function or vtable pointer (for control flow hijack)



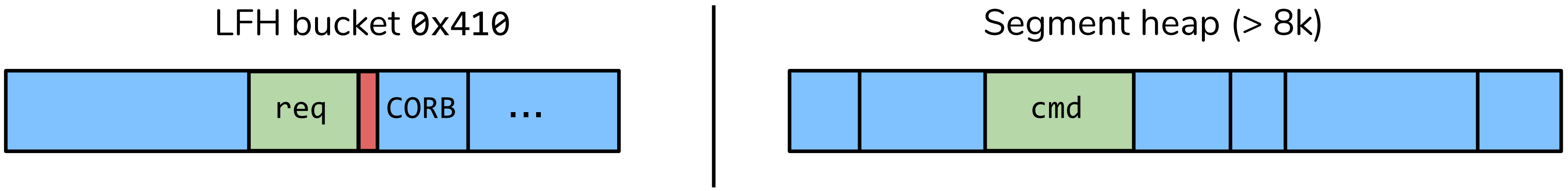
Exploit

- Powerful, heap-based **relative** read and write primitives
- `VBOXHGCMCMD` is variable-size, we can put it on a “nice” heap
    - Allocator fully predictable, with help of bug #1
- Can already leak some vtable pointers from `VBoxC.dll` with bug #1
- Next: want to corrupt
    - a pointer that is read from (for full ASLR break)
    - a function or vtable pointer (for control flow hijack)



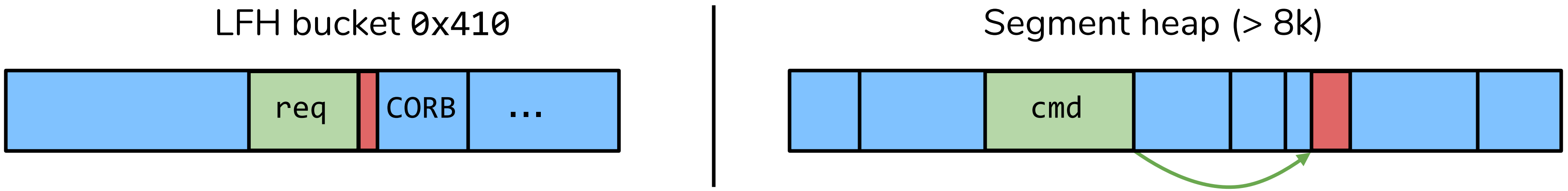
Exploit

- Powerful, heap-based **relative** read and write primitives
- `VBOXHGCMCMD` is variable-size, we can put it on a “nice” heap
  - Allocator fully predictable, with help of bug #1
- Can already leak some vtable pointers from `VBoxC.dll` with bug #1
- Next: want to corrupt
  - a pointer that is read from (for full ASLR break)
  - a function or vtable pointer (for control flow hijack)



Exploit

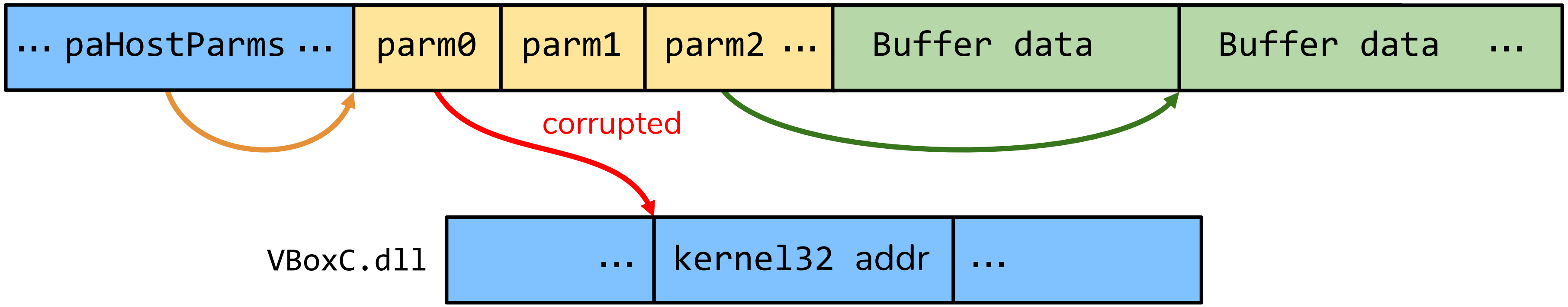
- Powerful, heap-based **relative** read and write primitives
- `VBOXHGCMCMD` is variable-size, we can put it on a “nice” heap
  - Allocator fully predictable, with help of bug #1
- Can already leak some vtable pointers from `VBoxC.dll` with bug #1
- Next: want to corrupt
  - a pointer that is read from (for full ASLR break)
  - a function or vtable pointer (for control flow hijack)



/src/VBox/HostServices/GuestProperties/service.cpp

Exploit - Absolute read

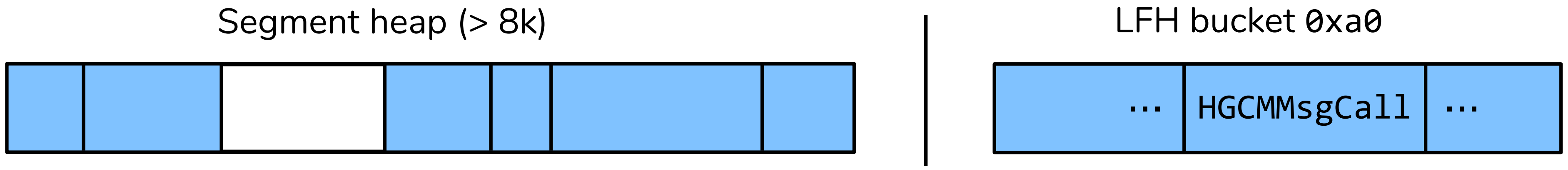
- VBOXHGCMCMD is an interesting data structure with pointers
  - Not all HGCM calls return immediately!
  - Send GET_NOTIFICATION to guest properties service
  - It returns when a property is set that matches the given pattern
  - This will cause a writeback using pointers from VBOXHGCMCMD
  - Used to leak kernel32 and ntdll base addresses



/src/VBox/Main/src-client/HGCM.cpp

Exploit - Nail in the coffin

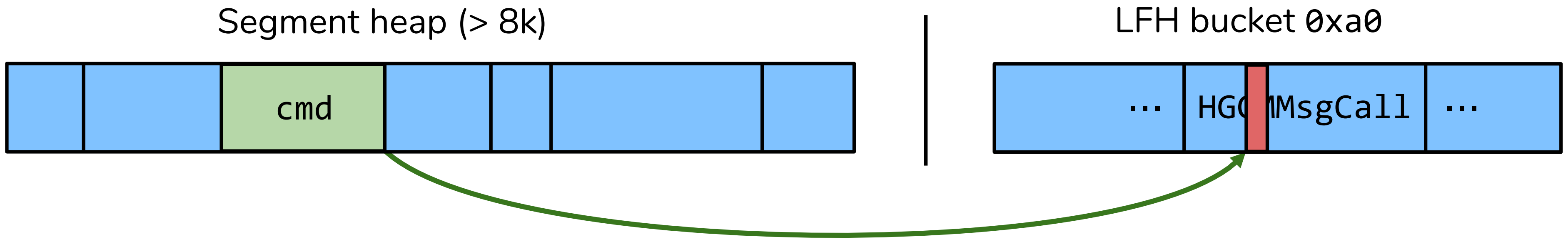
- A `HGCMMsgCall` object is allocated for each HGCM call
- Unlike `VBOXHGCMCMD`, it has a constant size of `0x98` => LFH heap
- Contains a pointer to itself
    - use a small spray and bug #1 to find it
- We corrupt the `pHGCMPort` field



/src/VBox/Main/src-client/HGCM.cpp

Exploit - Nail in the coffin

- A `HGCMMsgCall` object is allocated for each HGCM call
- Unlike `VBOXHGCMCMD`, it has a constant size of `0x98` => LFH heap
- Contains a pointer to itself
  - use a small spray and bug #1 to find it
- We corrupt the `pHGCMPort` field

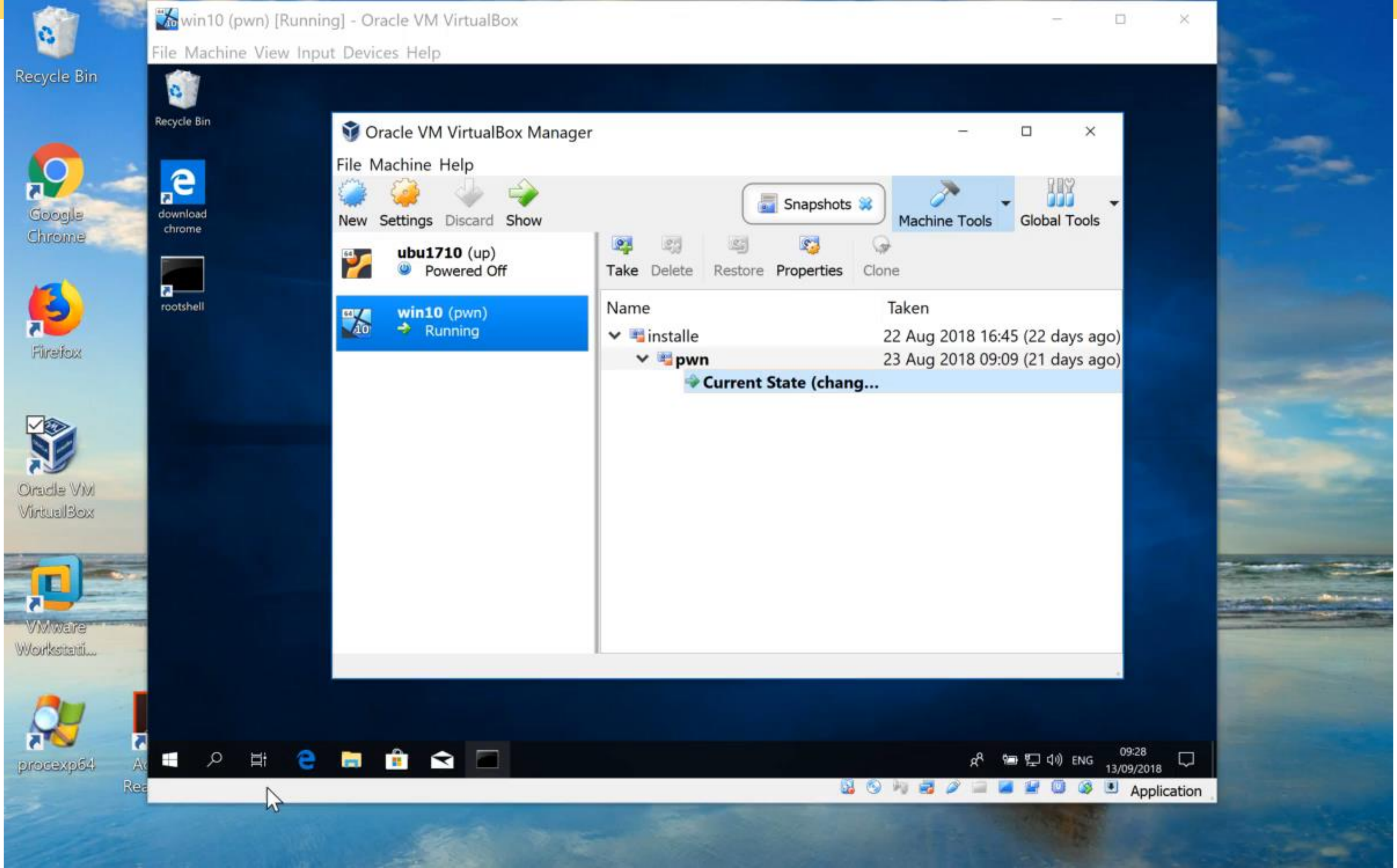


/src/VBox/Main/src-client/HGCM.cpp

Exploit - Nail in the coffin

- A `HGCMMsgCall` object is allocated for each HGCM call
- Unlike `VBOXHGCMCMD`, it has a constant size of `0x98` => LFH heap
- Contains a pointer to itself
  - use a small spray and bug #1 to find it
- We corrupt the `pHGCMPort` field

```
static DECLCALLBACK(void) hgcmMsgCompletionCallback(int32_t result,
                                                     HGCMMsgCore *pMsgCore) {
    /* Call the VMMDev port interface to issue IRQ notification. */
    HGCMMsgHeader *pMsgHdr = (HGCMMsgHeader *)pMsgCore;
    ...
    if (pMsgHdr->pHGCMPort && !g_fResetting) {
        pMsgHdr->pHGCMPort->pfnCompleted(pMsgHdr->pHGCMPort, ...);
```



Name	Taken
installe	22 Aug 2018 16:45 (22 days ago)
pwn	23 Aug 2018 09:09 (21 days ago)
Current State (chang...	

ubu1710 (up)
Powered Off

win10 (pwn)
Running

09:28
13/09/2018

`/src/VBox/HostDrivers/Support/SUPDrv.cpp`

From `VirtualBox.exe` to host kernel

- `VirtualBox.exe` is privileged, since it has access to `VBoxDrv` IOCTLs
- `SUP_IOCTL_LDR_{OPEN,LOAD}` load PE file as kernel plugin
  - Verifies driver signature
- `SUP_IOCTL_CALL_SERVICE` calls into a plugin
  - Full control over 4th argument

    => RIP control via `jmp r9`
- `SUP_IOCTL_PAGE_{ALLOC_EX,MAP_KERNEL,PROTECT}`
  - Map RWX code in the kernel

Because why not

`/src/VBox/Devices/Graphics/DevVGA_VDMA.cpp`

CVE-2018-2698

SSD
SecuriTeam
Secure
Disclosure

- HGSMI = **H**ost-**G**uest **S**hared **M**emory **I**nterface
- Guest allocates request buffer in video RAM, notifies VGA device
- Used for VBVA subsystem (**V**irtual**B**ox **V**ideo **A**cceleration)
- `VBVA_VDMA_CMD` is used for video DMA commands:
  - `VBOXVDMACMD_TYPE_DMA_PRESENT_BLT`
  - `VBOXVDMACMD_TYPE_DMA_BPB_TRANSFER`

/src/VBox/Devices/Graphics/DevVGA_VDMA.cpp

```
int rc = vboxVDMACmdExecBltPerform(pVdma, pvRam + pBlt->offDst, pvRam + pBlt->offSrc,
       &pBlt->dstDesc, &pBlt->srcDesc,
       pDstRectl,
       pSrcRectl);
...
static int vboxVDMACmdExecBltPerform(PVBOXVDMAHOST pVdma,
                uint8_t *pvDstSurf, const uint8_t *pvSrcSurf,
                const PVBOXVDMA_SURF_DESC pDstDesc, const PVBOXVDMA_SURF_DESC pSrcDesc,
                const VBOXVDMA_RECTL * pDstRectl, const VBOXVDMA_RECTL * pSrcRectl)
{
   ...
   if (pDstDesc->width == pDstRectl->width && pSrcDesc->width == pSrcRectl->width
           && pSrcDesc->width == pDstDesc->width) {
       ...
       uint32_t cbOff = pDstDesc->pitch * pDstRectl->top;
       uint32_t cbSize = pDstDesc->pitch * pDstRectl->height;
       memcpy(pvDstSurf + cbOff, pvSrcSurf + cbOff, cbSize);
```

🟥 = guest-controlled

```
int rc = vboxVDMACmdExecBltPerform(pVdma, pvRam + pBlt->offDst, pvRam + pBlt->offSrc,
       &pBlt->dstDesc, &pBlt->srcDesc,
       pDstRectl,
       pSrcRectl);
...
static int vboxVDMACmdExecBltPerform(PVBOXVDMAHOST pVdma,
                uint8_t *pvDstSurf, const uint8_t *pvSrcSurf,
                const PVBOXVDMA_SURF_DESC pDstDesc, const PVBOXVDMA_SURF_DESC pSrcDesc,
                const VBOXVDMA_RECTL * pDstRectl, const VBOXVDMA_RECTL * pSrcRectl)
{
   ...
   if (pDstDesc->width == pDstRectl->width && pSrcDesc->width == pSrcRectl->width
           && pSrcDesc->width == pDstDesc->width) {
       ...
       uint32_t cbOff = pDstDesc->pitch * pDstRectl->top;
       uint32_t cbSize = pDstDesc->pitch * pDstRectl->height;
       memcpy(pvDstSurf + cbOff, pvSrcSurf + cbOff, cbSize);
```

VirtualBox host debugging

- Cannot attach to `VirtualBox.exe` due to *process hardening*
- Exploit dev on Windows: non-hardened debug build
    - Get ready for a nostalgic experience with VS 2010
    - Ideally have a friend do it for you
- Debugging the official Windows build:
    - Run VirtualBox inside VMware Workstation (enable “Virtualize Intel VT-x”)
    - Use a kernel debugger with `!gflag +soe` and `!process`
- Bug hunting & PoCs are much easier on Linux host + guest
    - Configure guest VM according to target

Dig deeper

Advisories for presented bugs https://www.zerodayinitiative.com/advisories/ZDI-18-782/ https://www.zerodayinitiative.com/advisories/ZDI-18-783/ https://blogs.securiteam.com/index.php/archives/3649

Bugs in E1000 network card, NAT & virtio-net (2017) https://github.com/fundacion-sadosky/vbox_cve_2017_10235 https://bugs.chromium.org/p/project-zero/issues/detail?id=1086 https://bugs.chromium.org/p/project-zero/issues/detail?id=1136

VDMA exploit and host-/guest-based privilege escalations (2018) https://www.youtube.com/watch?v=fFaWE3jt7qU https://github.com/phoenhex/files/blob/master/slides/unboxing_your_virtualboxes.pdf

VBVA double fetch (2018) https://www.voidsecurity.in/2018/08/from-compiler-optimization-to-code.html

Windows process hardening https://googleprojectzero.blogspot.com/2017/08/bypassing-virtualbox-process-hardening.html

VirtualBox 3D hacks https://www.coresecurity.com/corelabs-research/publications/breaking-out-virtualbox-through-3d-acceleration https://phoenhex.re/2018-07-27/better-slow-than-sorry https://github.com/niklasb/3dpwn https://www.thezdi.com/blog/2018/8/28/virtualbox-3d-acceleration-an-accelerated-attack-surface

Simple Python HGCM client library: https://github.com/niklasb/3dpwn/blob/master/lib/hgcm.py

VMware Workstation vulnerabilities & exploitation https://keenlab.tencent.com/en/2018/04/23/A-bunch-of-Red-Pills-VMware-Escapes/ https://www.thezdi.com/blog/2018/3/1/vmware-exploitation-through-uninitialized-buffers https://comsecuris.com/blog/posts/vmware_vgpu_shader_vulnerabilities/ https://www.blackhat.com/docs/eu-17/materials/eu-17-Mandal-The-Great-Escapes-Of-Vmware-A-Retrospective-Case-Study-Of-Vmware-G2H-Escape-Vulnerabilities.pdf and many more

`DSEFix` (exploits old VBoxDrv version to disable Driver Signature Enforcement on Windows 10 ≤ RS4) https://github.com/hfiref0x/DSEFix
`fwexpl`: awesome framework for low-level I/O hacking (with ridiculous RWEverything driver) https://github.com/Cr4sh/fwexpl