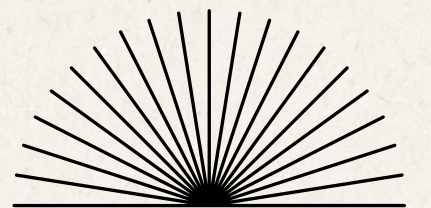


KOREAN ROOKIE HACKER'S JOURNEY

Road to Pwn2Own with a VirtualBox Exploit

EMAIL
creds1337@gmail.com

PRESENTED BY:
Hanseon Kim



Agenda

00	Introduction
01	Korean Rookies
02	Pwn2Own && Virtualization
03	VirtualBox Virtio-net Out-Of-Bounds
04	VMSVGA Integer Overflow (CVE-2025-53024)
05	Information Leak(ZDI-CAN-27241)
06	Exploit
07	Conclusion

About Myself

Hanseon Kim

Software Engineering Student - Ajou University

Intern at Hspace

Research Interests

- OS Internals & Virtualization VR, Red Teaming



Team PrisonBreak

Hanseon Kim, Gangmin Kim, Wonjoon Hwang, Sanghoon Lee, Sangwon Oh, Sangbin Kim

Korean Rookies

Rookie Hacker Life in Korea

Student Hackers

Some start in specialized high schools, others at university. Distinctive clubs at each school and active exchanges build a vibrant hacking community.

Bootcamp... Literally

Mandatory military service often creates a research gap for male students. Assignments in cyber units or CERT roles can provide relevant experience, but they rarely substitute for continuous academic progress.

Elite Program

The government-supported program Best Of the Best (BOB) is one that most students studying cybersecurity aspire to join.



BOB?

01/07

Government-supported program in Korea



BOB?

01/07

Government-supported program in Korea



Vulnerability Research



Security Consulting



Digital Forensic



Secure Product Development

BOB?

VR Track

01/07



BOB?

VR Track

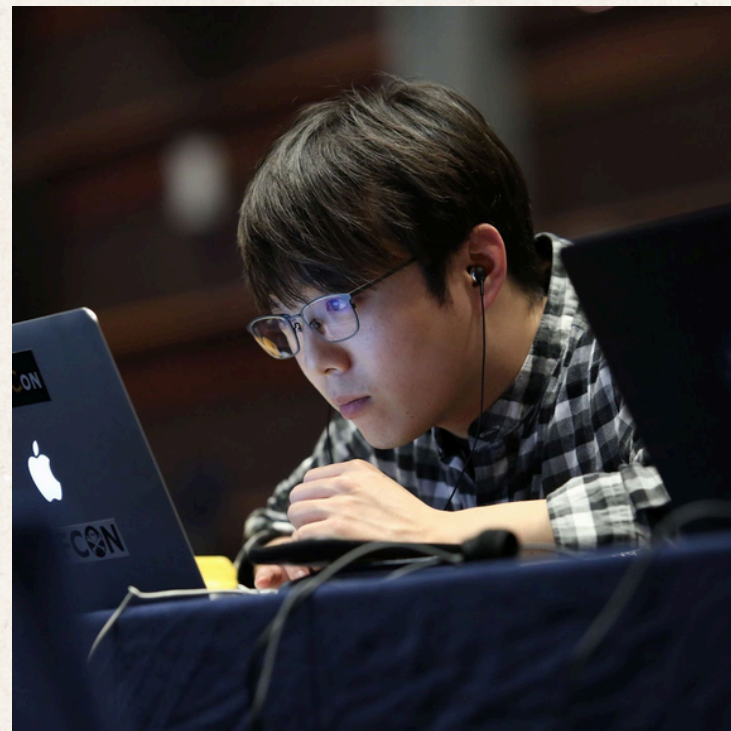
01/07



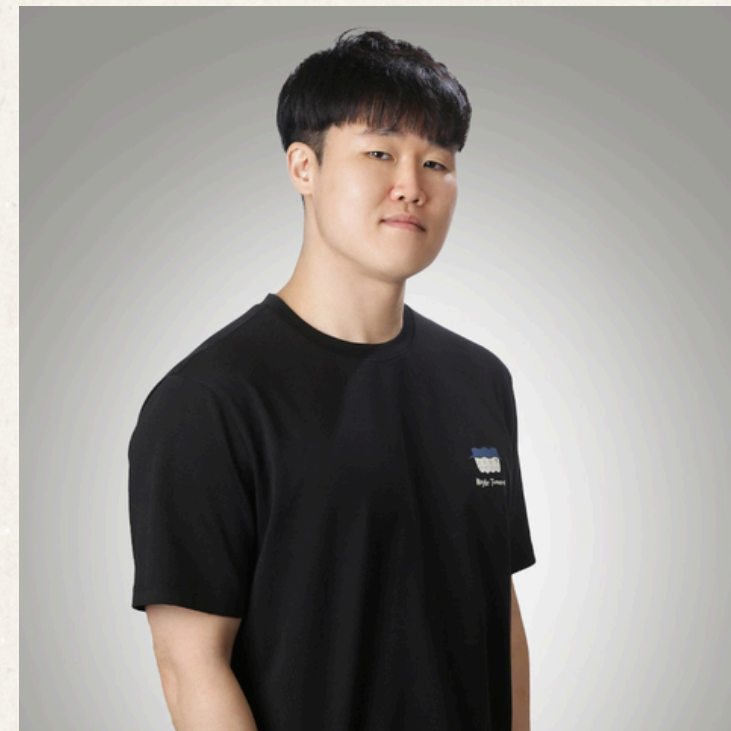
BOB?

01/07

Team Project



Singi Labs
Jeonghoon Shin



Theori
Gwangun Jung

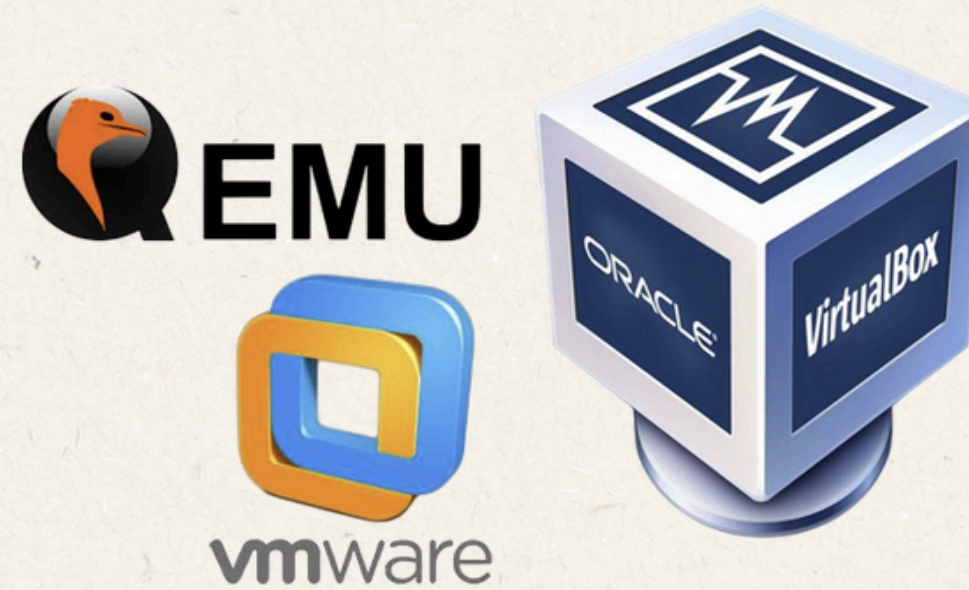


BOB?

01/07

Team Project

Virtualization



Pwn2Own

02/07

Virtualization Category

Target	Prize	Master of Pwn Points	Eligible for Add-on Prize
Oracle VirtualBox	\$40,000	4	Yes
VMware Workstation	\$80,000	8	Yes
VMware ESXi	\$150,000	15	No
Microsoft Hyper-V Client	\$250,000	25	Yes

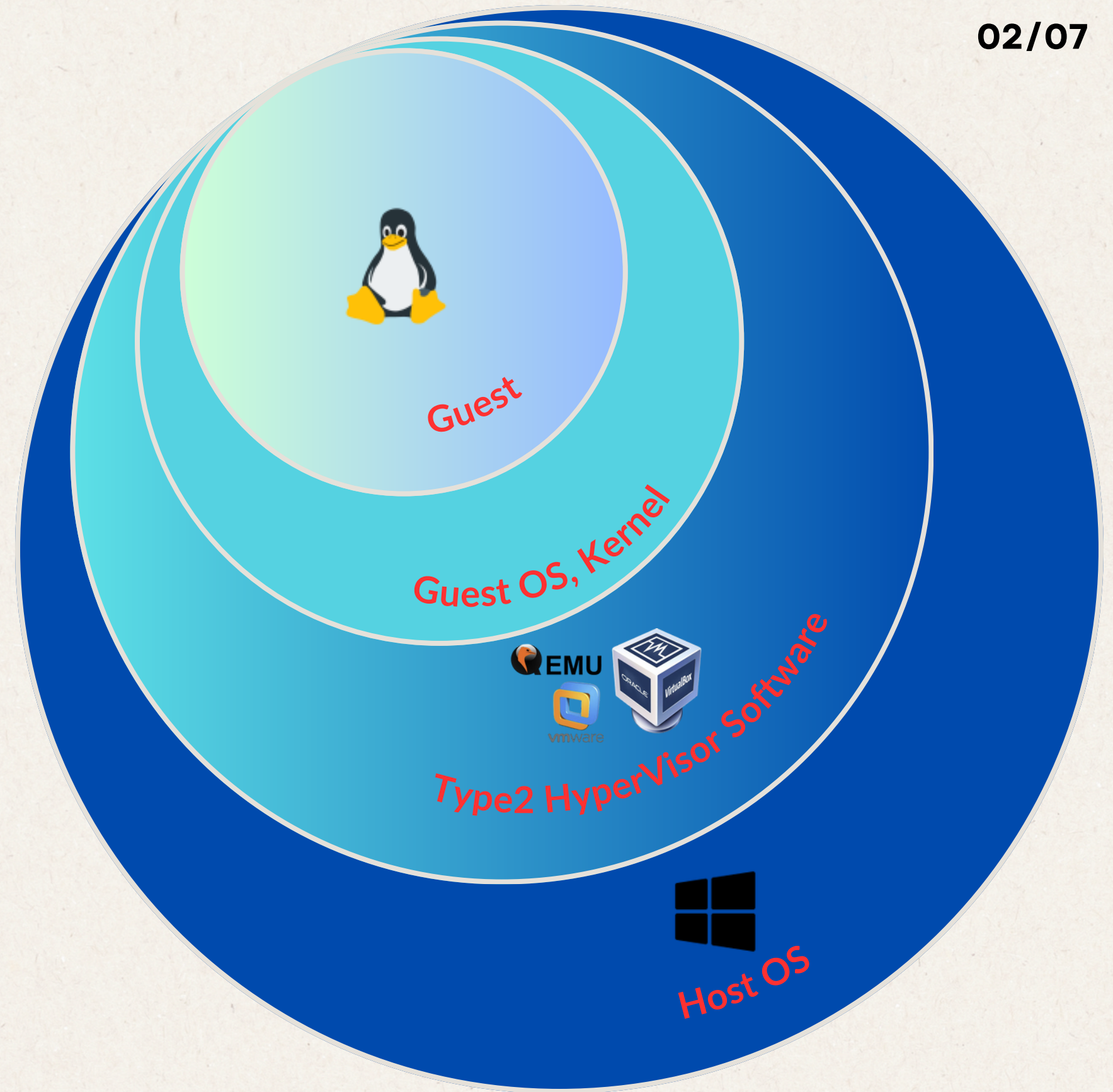
Add-on Prize	Prize	Master of Pwn Points
Escalation of privilege leveraging a Windows kernel vulnerability on the host operating system.	\$50,000	5

Virtualization

Type 2 Hypervisor

A Type 2 hypervisor operates like an application on top of the host operating system, managing virtual machines within that environment.

02/07



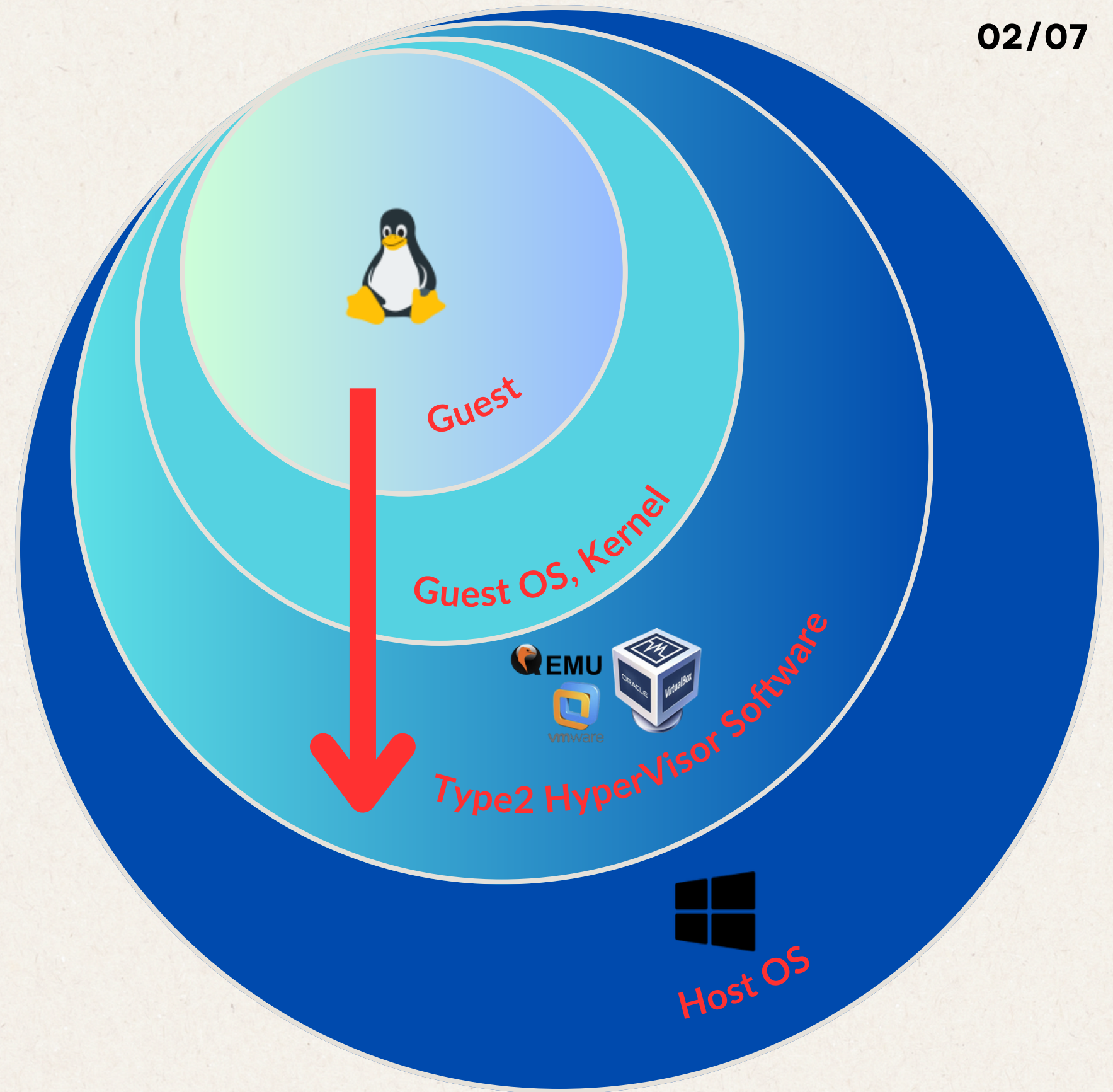
Virtualization

02/07

VM Escape

VM Escape is a vulnerability from indirect communication between a guest OS and a Type-2 hypervisor.

- Impact: Guest-only privileges can lead to host RCE, DoS, or file read/write.
- Attack Vector: Bugs in input handling, memory/integer overflows.

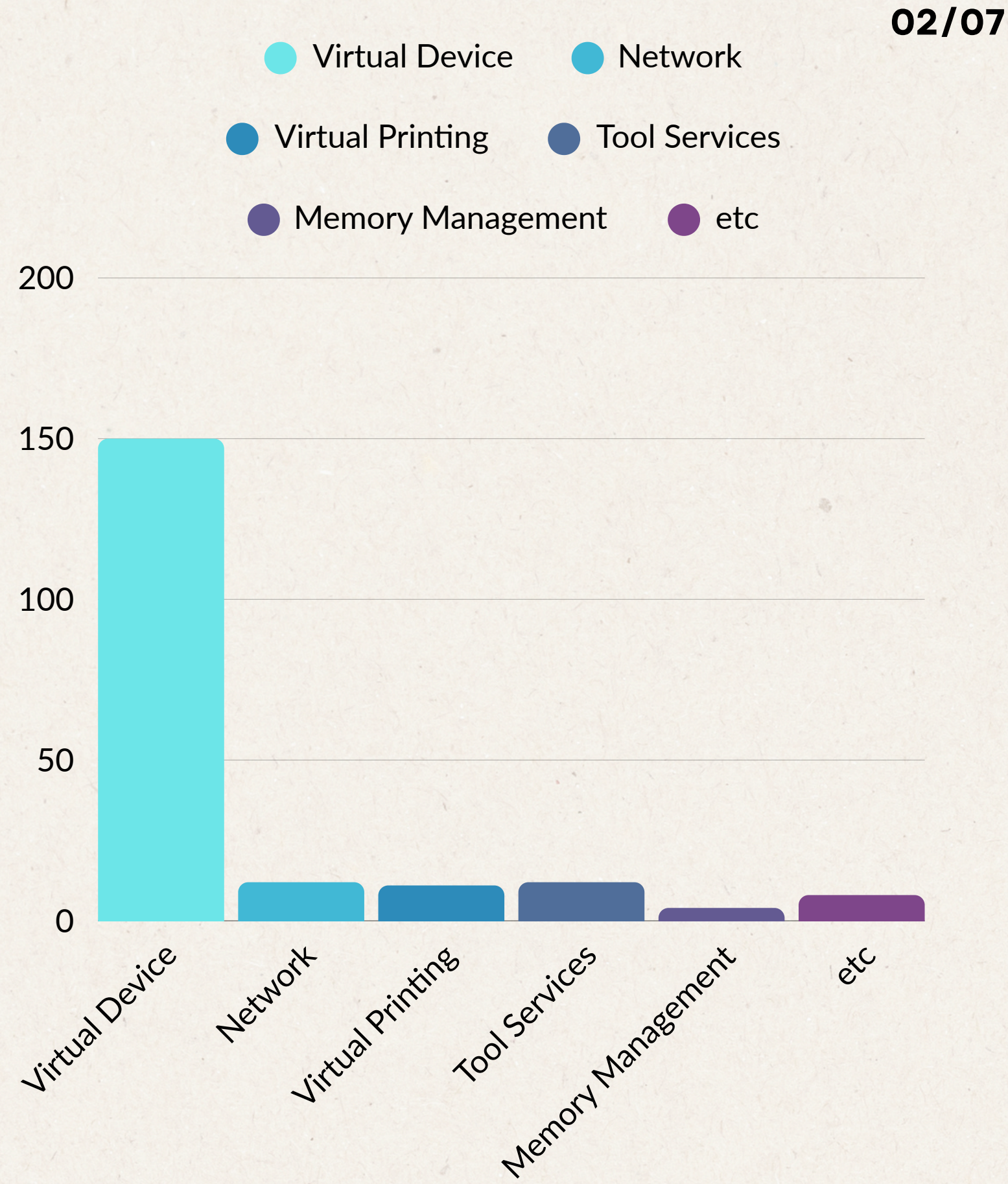


Attack Surfaces

Type 2 Hypervisor

Reported Vulnerabilities by Attack Vector
(2020–2024)

Over 90% of vulnerabilities are found in virtual devices.

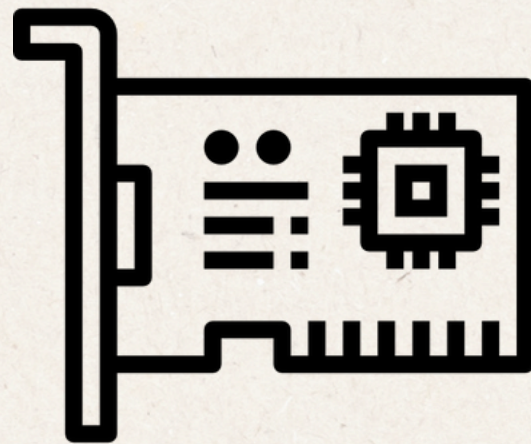


Virtual devices

02/07

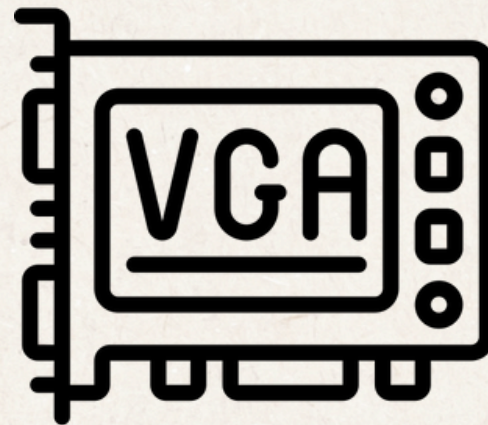
Software-emulated device

Network



E1000, Virtio-Net

Graphic



SVGA, SVGA3D ...

Storage

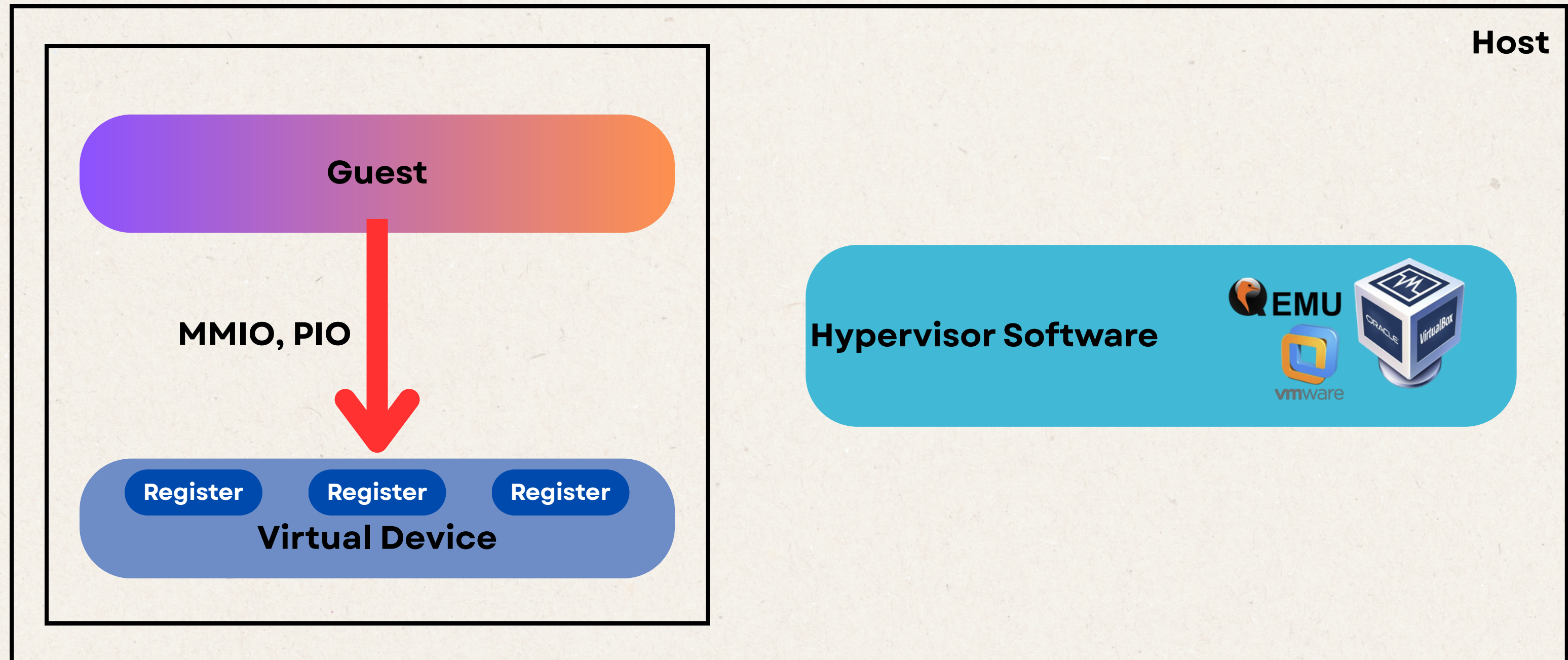


AHCI, Vitio-BLK ...

PIO, MMIO

02/07

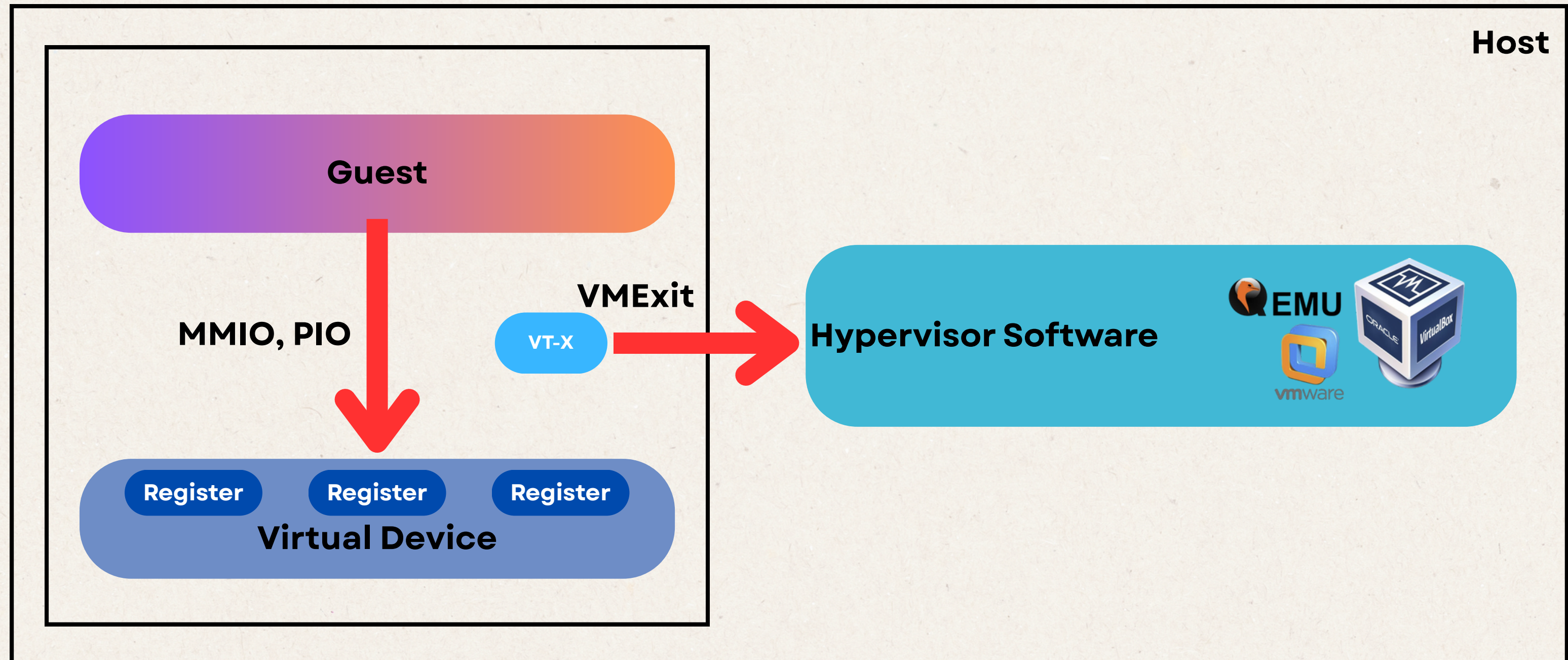
[Port / Memory-mapped] device access



PIO, MMIO

02/07

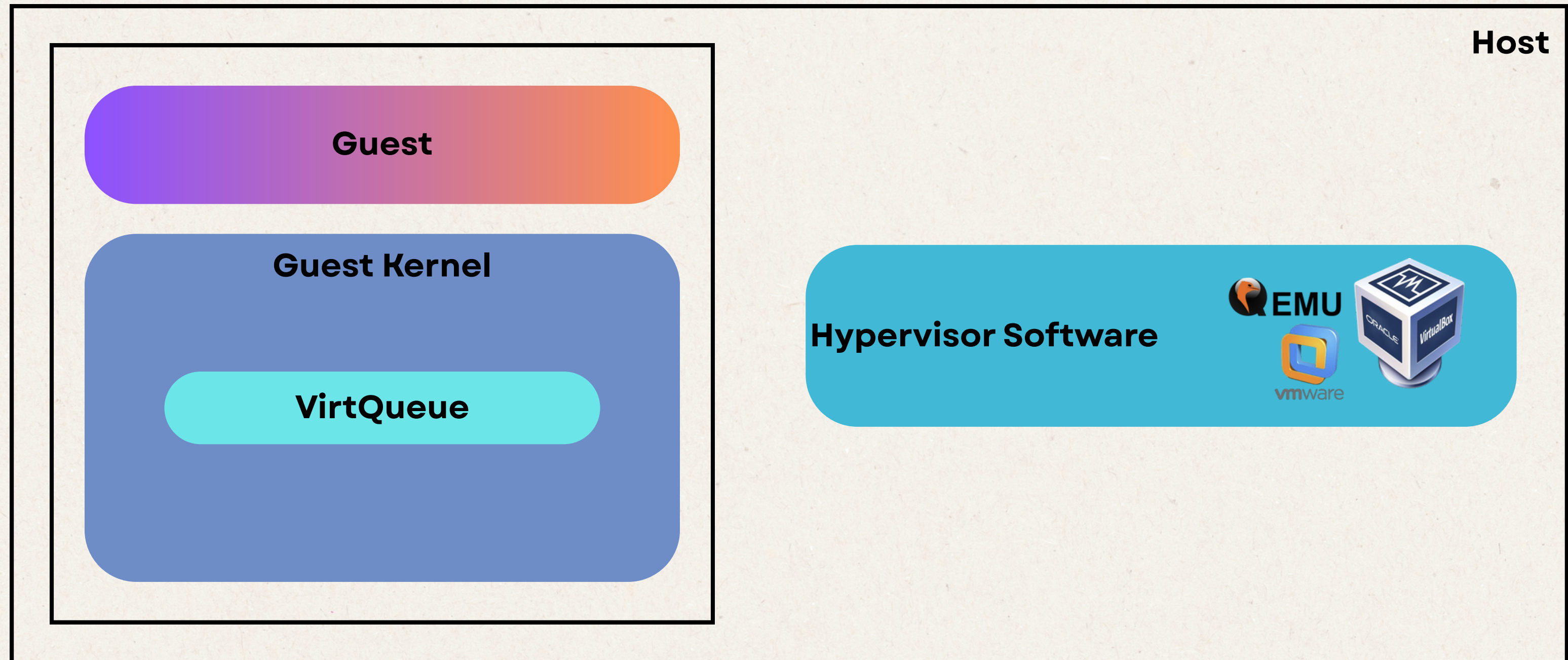
[Port / Memory-mapped] device access



VirtIO

02/07

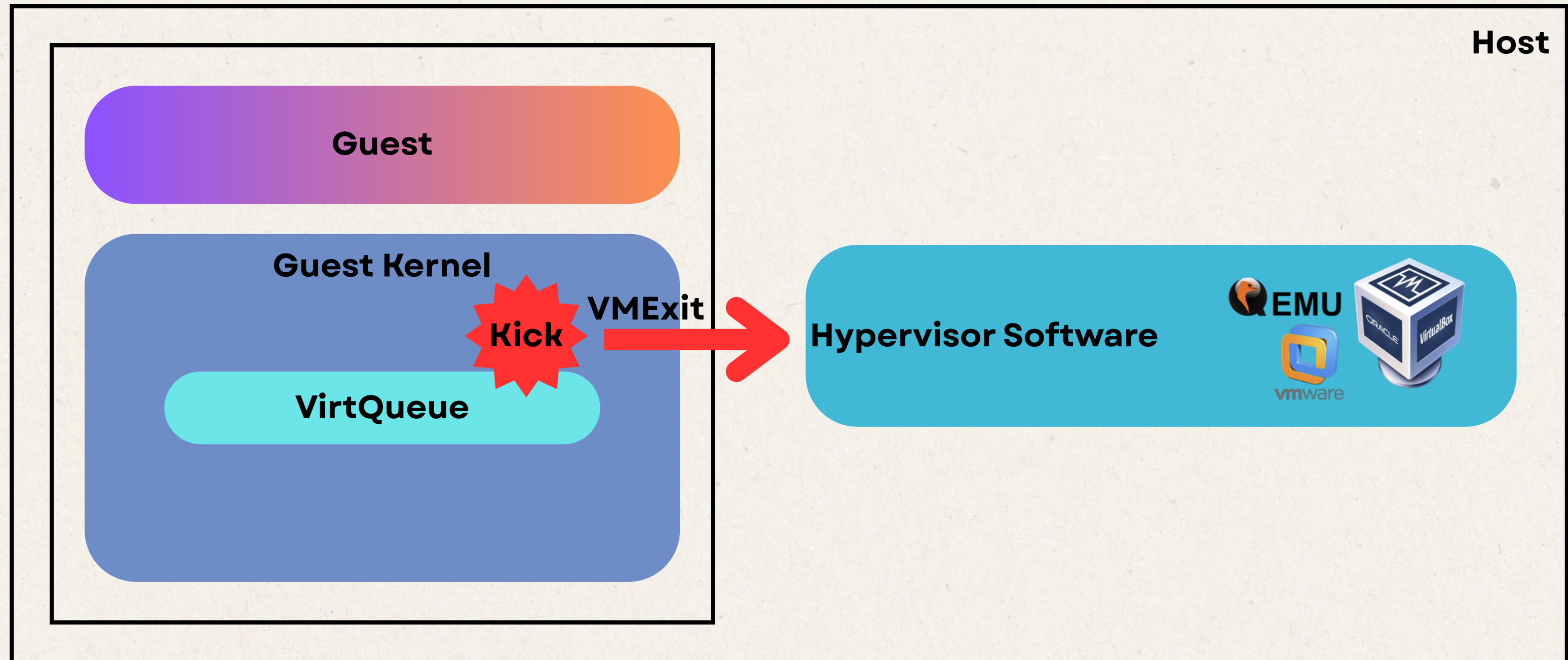
Paravirtualized I/O Interface



VirtIO

02/07

Paravirtualized I/O Interface



VGA-SVGA

03/07

Graphic Device

Guests can communicate with VGA/SVGA devices in two ways :

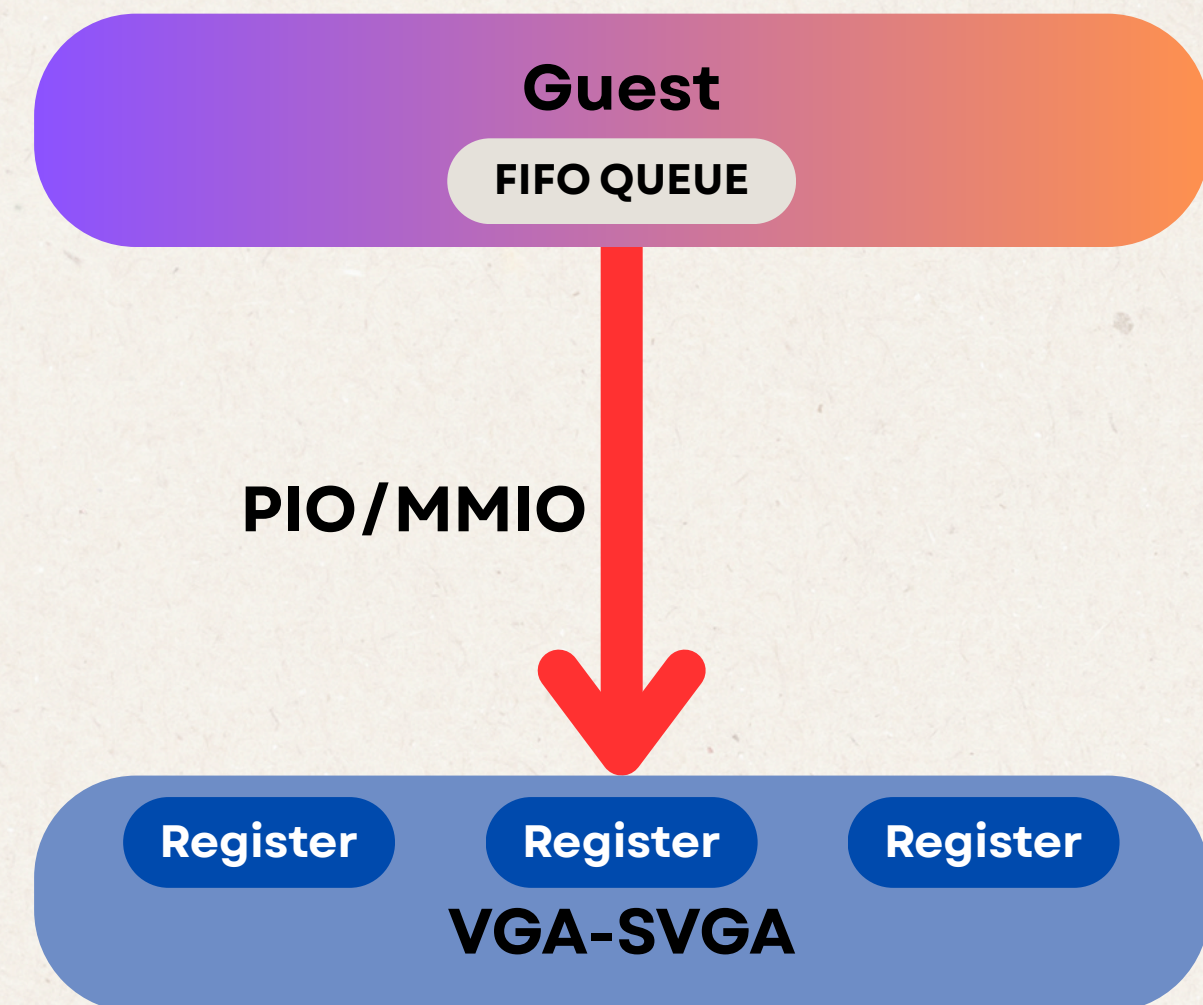
- 1) Simple control: Direct register access via MMIO/PIO
- 2) Complex graphics: Placing commands in FIFO queue and sending execution signals via MMIO/PIO

- **DevVGA-SVGA.cpp**

Core VGA/SVGA device structure and
PIO/MMIO register access handling

- **DevVGA-SVGA-cmd.cpp**

FIFO command interpretation and register-base
graphics operations



VGA-SVGA

03/07

Register Read/Write Example

If you need functionality to copy a rectangular pixel area within guest VRAM

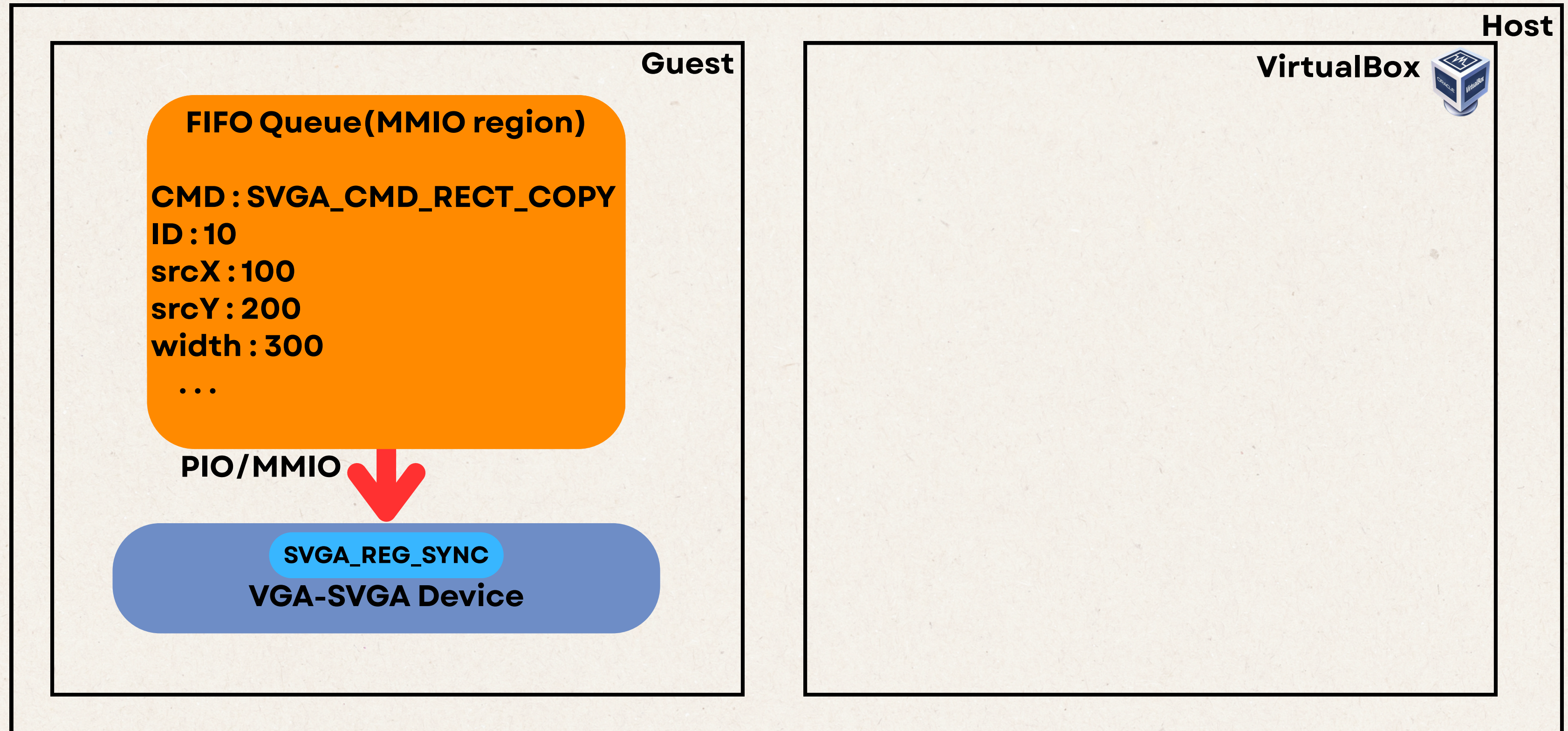
```
/**
 * Copy a rectangle of pixels within guest VRAM.
 */
static void vmsvgaR3RectCopy(PVGASTATECC pThisCC, VMSVGASCREENOBJECT const *pScreen, uint32_t srcX, uint32_t srcY,
                             uint32_t dstX, uint32_t dstY, uint32_t width, uint32_t height, unsigned cbFramebuffer)
{
    if (!width || !height)
        return; /* Nothing to do, don't even bother. */

    /*
     * The guest VRAM (aka GFB) is considered to be a bitmap in the format
     * corresponding to the current display mode.
     */
    uint32_t const cbPixel = RT_ALIGN(pScreen->cbBpp, 8) / 8;
    uint32_t const cbScanline = pScreen->cbPitch ? pScreen->cbPitch : width * cbPixel;
    uint8_t const *pSrc;
    uint8_t *pDst;
    unsigned const cbRectWidth = width * cbPixel;
    unsigned uMaxOffset;

    uMaxOffset = (RT_MAX(srcY, dstY) + height) * cbScanline + (RT_MAX(srcX, dstX) + width) * cbPixel;
    if (uMaxOffset >= cbFramebuffer)
    {
        Log(("Max offset (%u) too big for framebuffer (%u bytes), ignoring!\n", uMaxOffset, cbFramebuffer));
        return; /* Just don't listen to a bad guest. */
    }
}
```

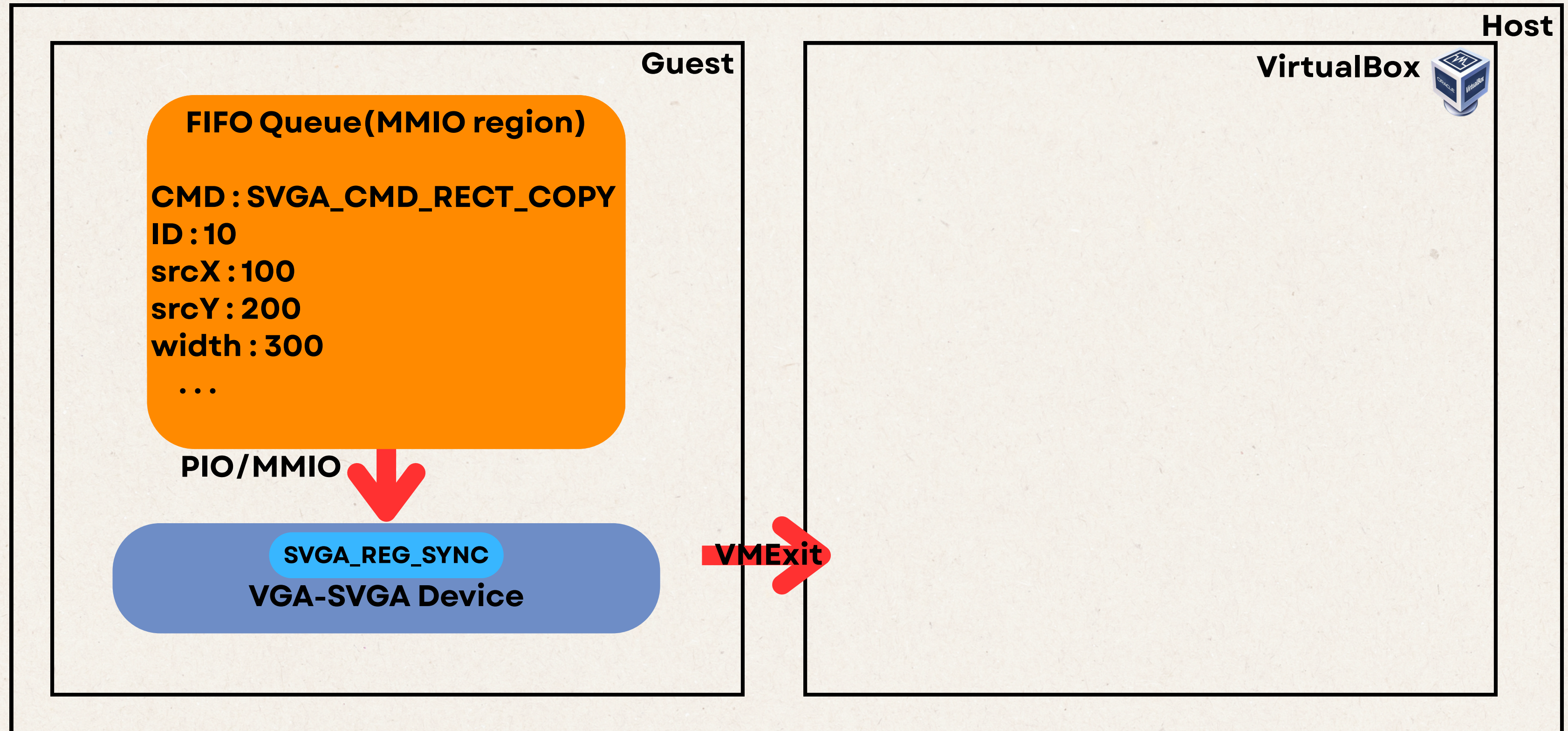

VGA-SVGA

03/07



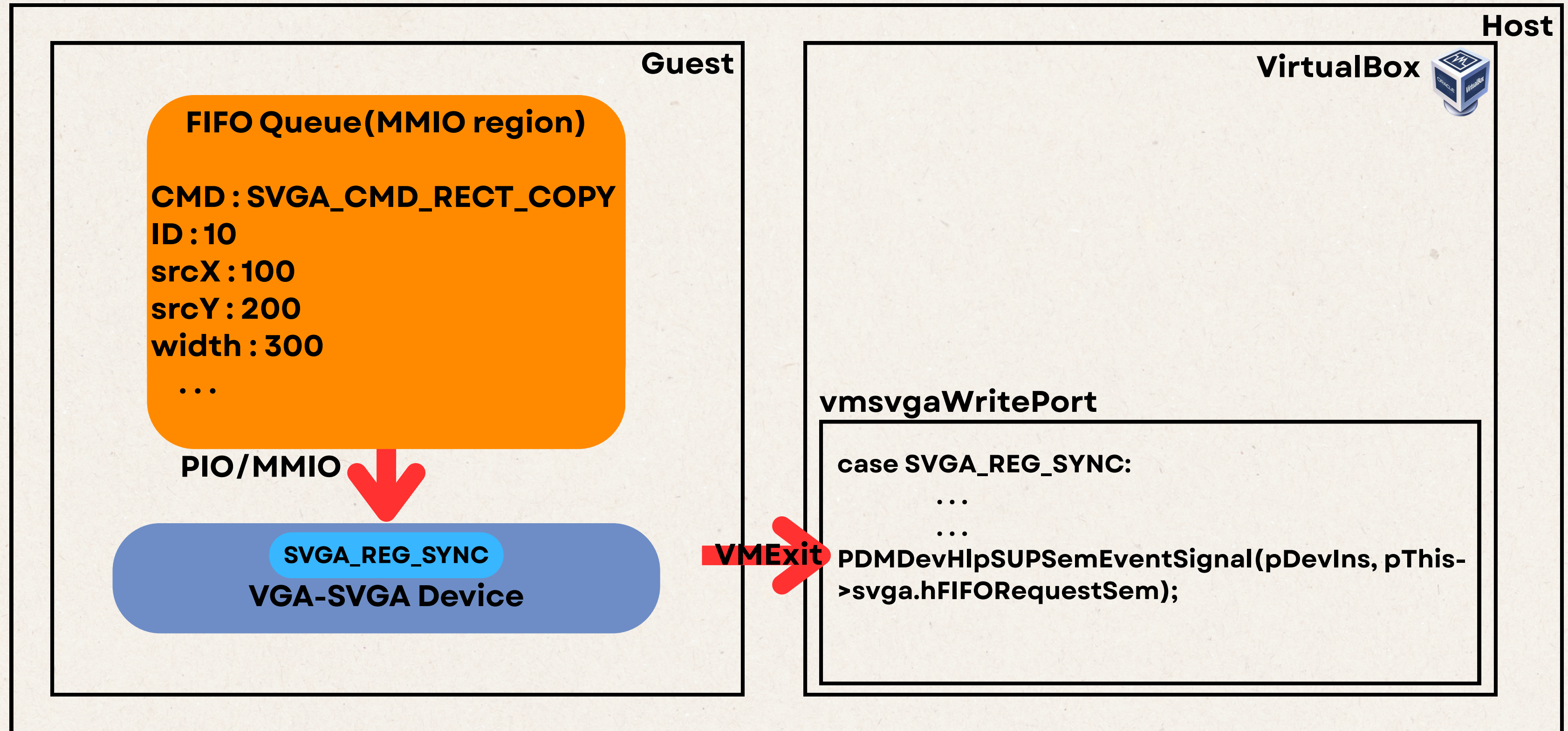
VGA-SVGA

03/07



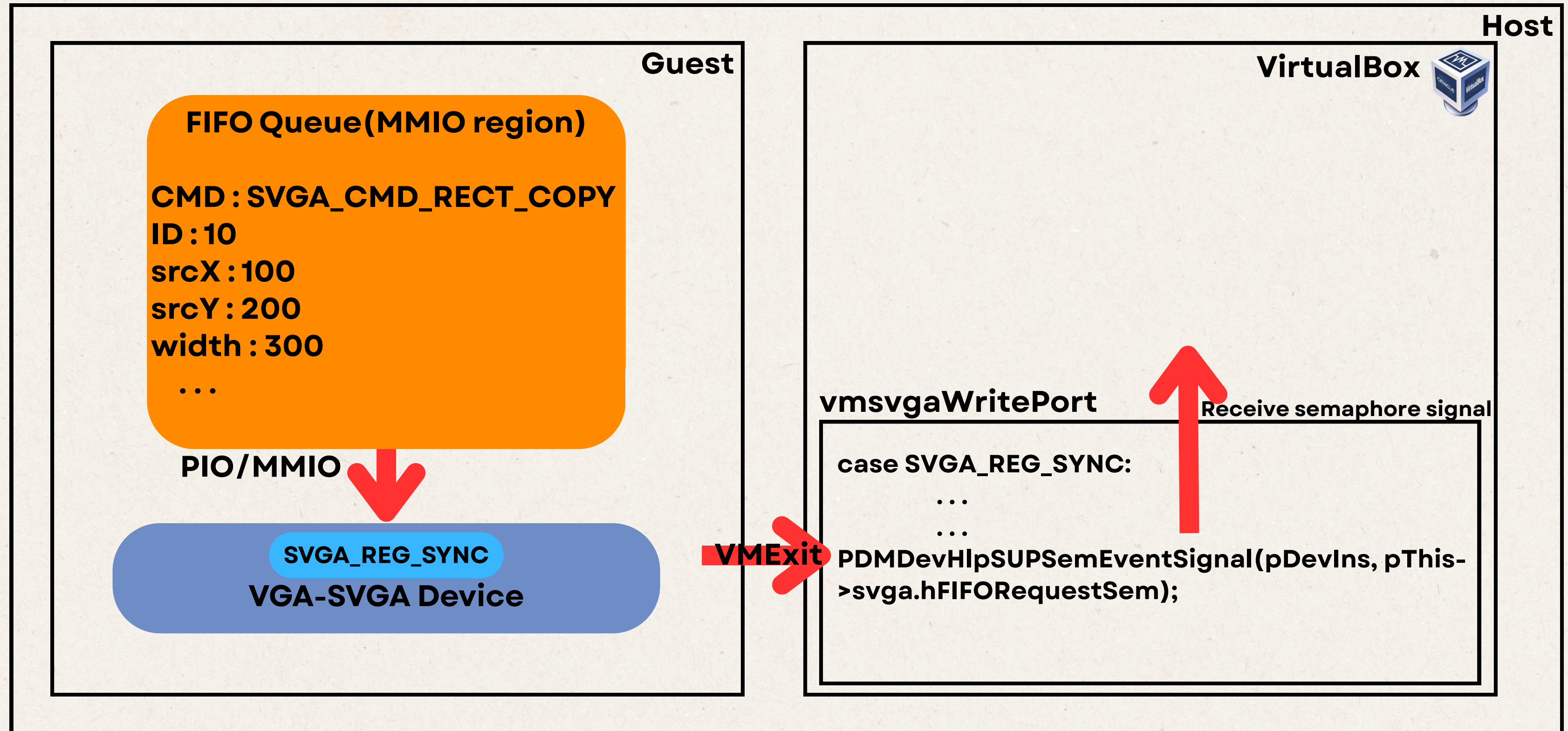
VGA-SVGA

03/07



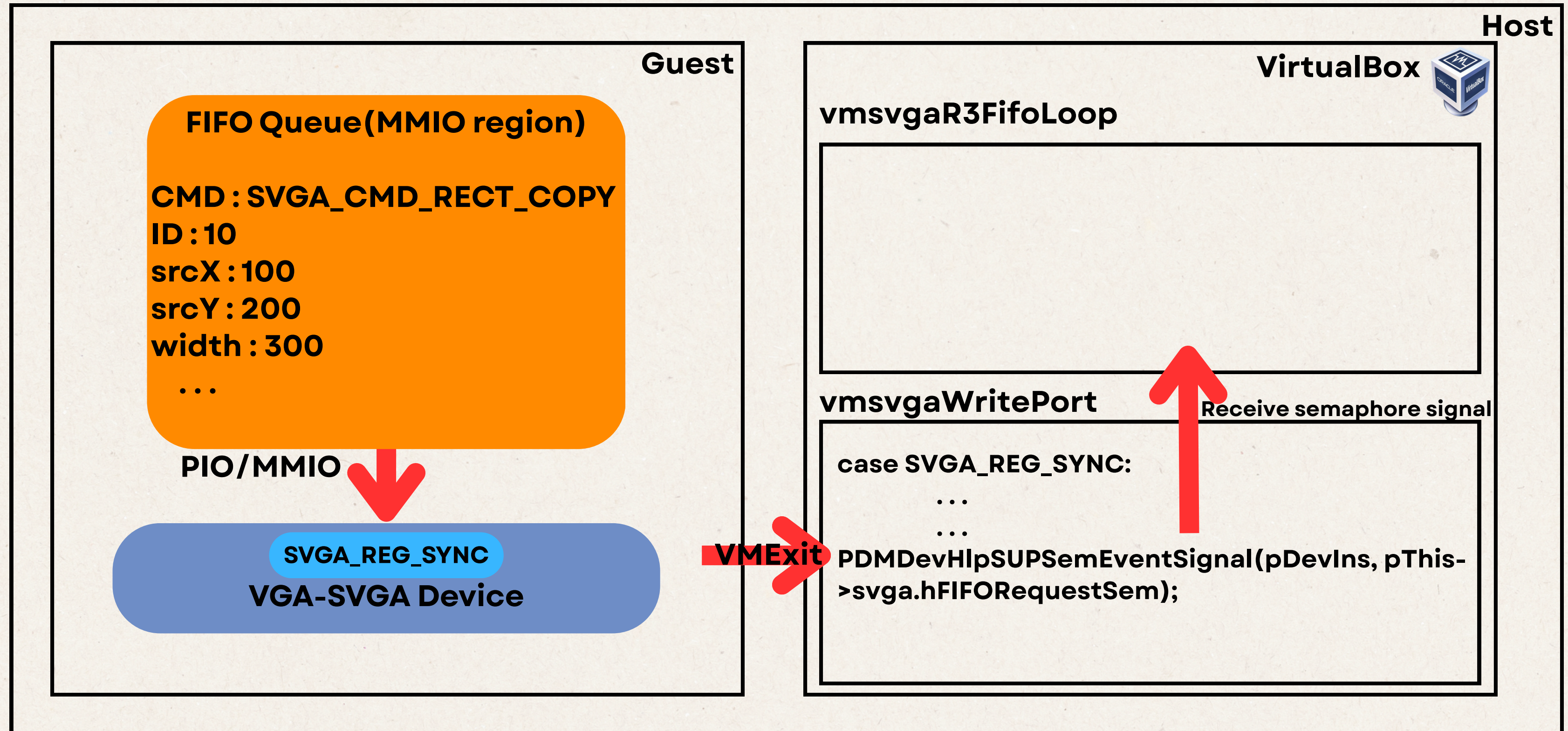
VGA-SVGA

03/07



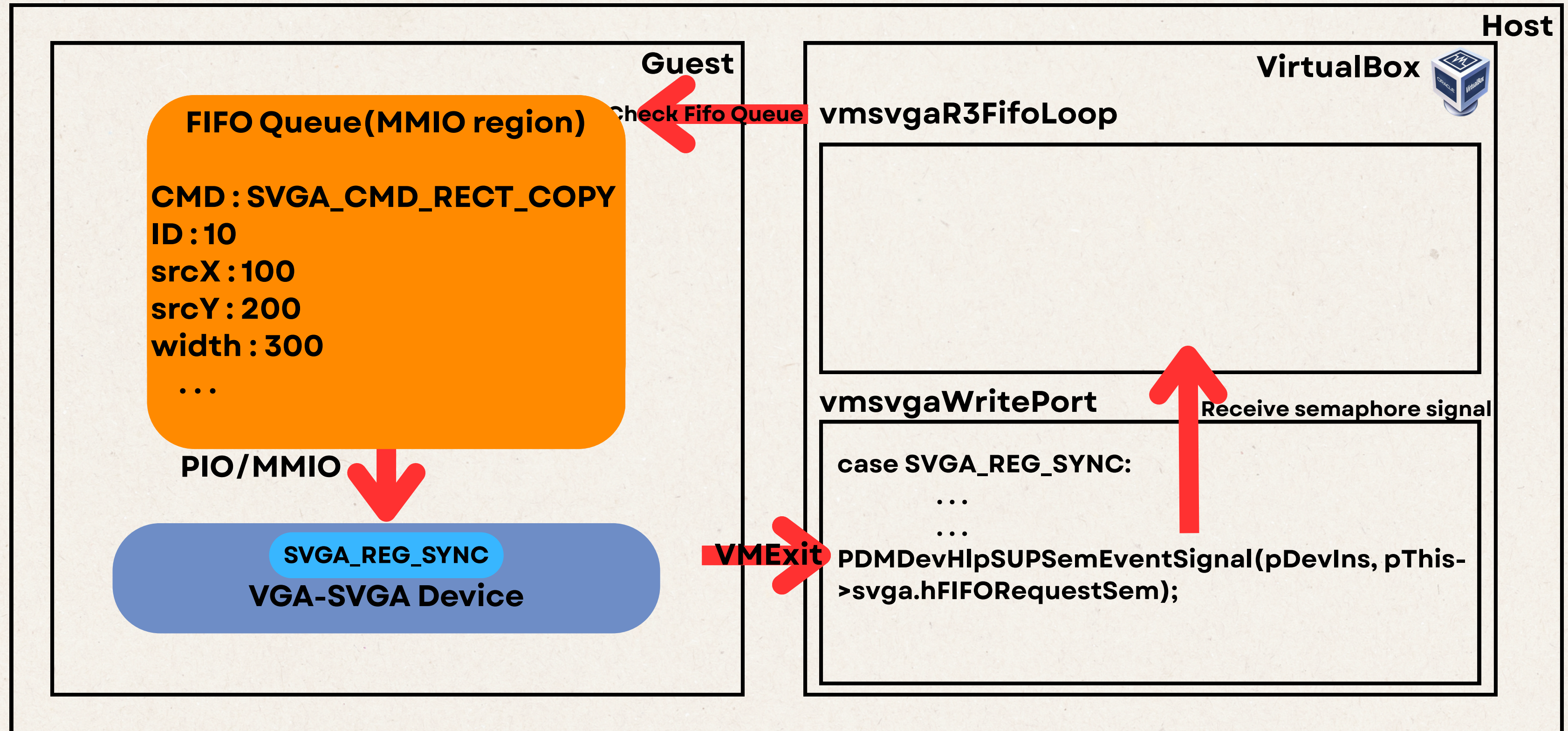
VGA-SVGA

03/07



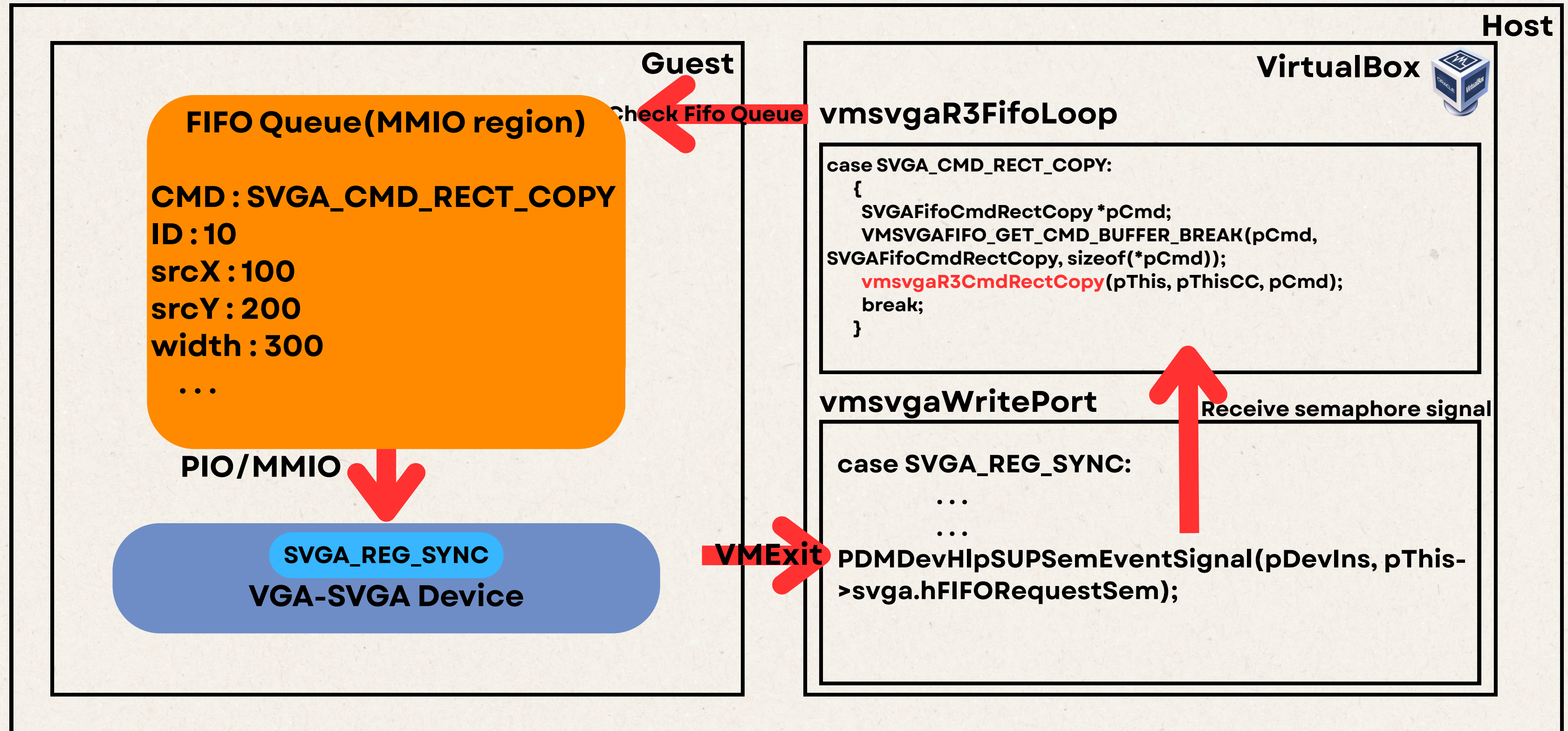
VGA-SVGA

03/07



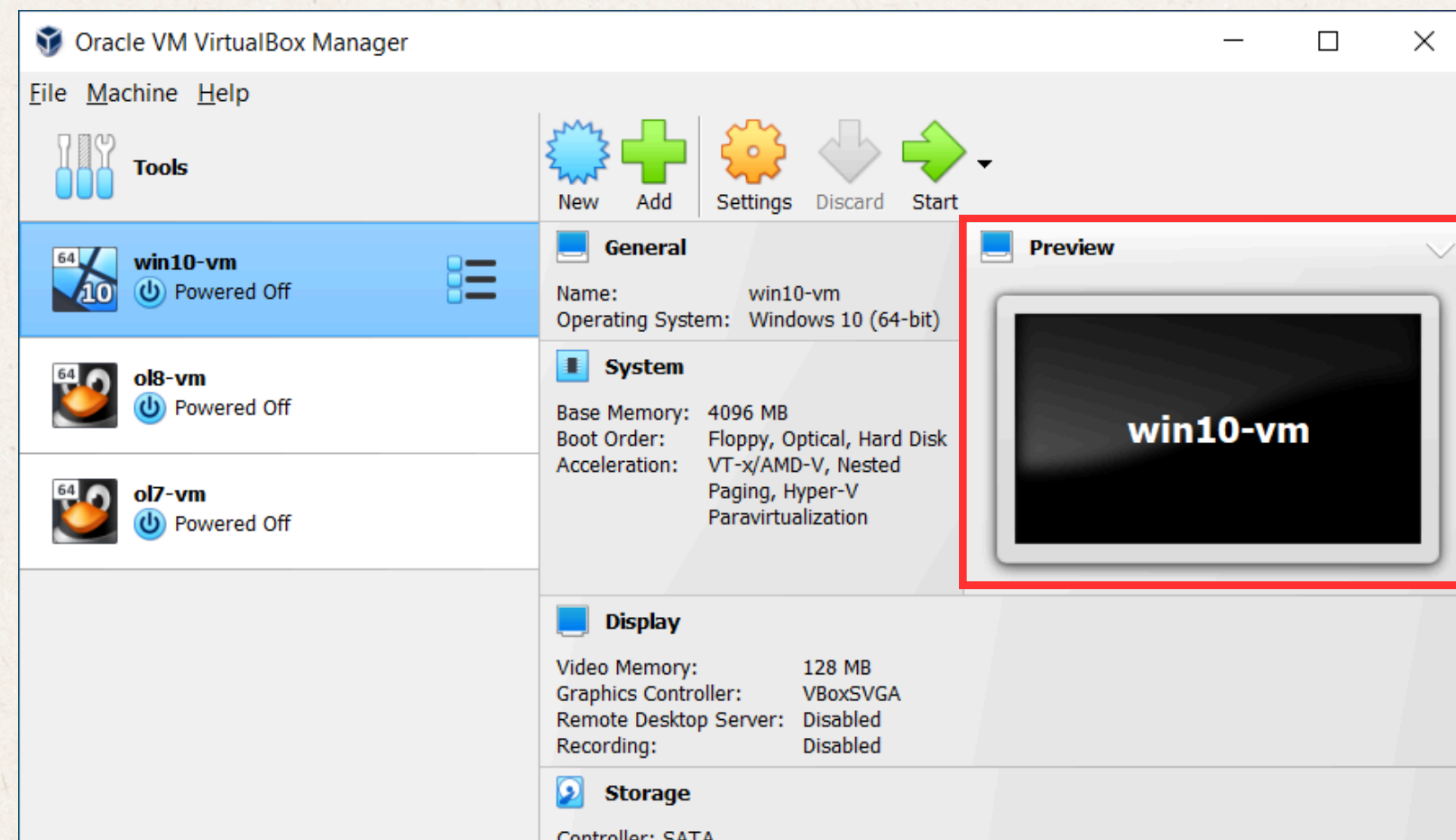
VGA-SVGA

03/07



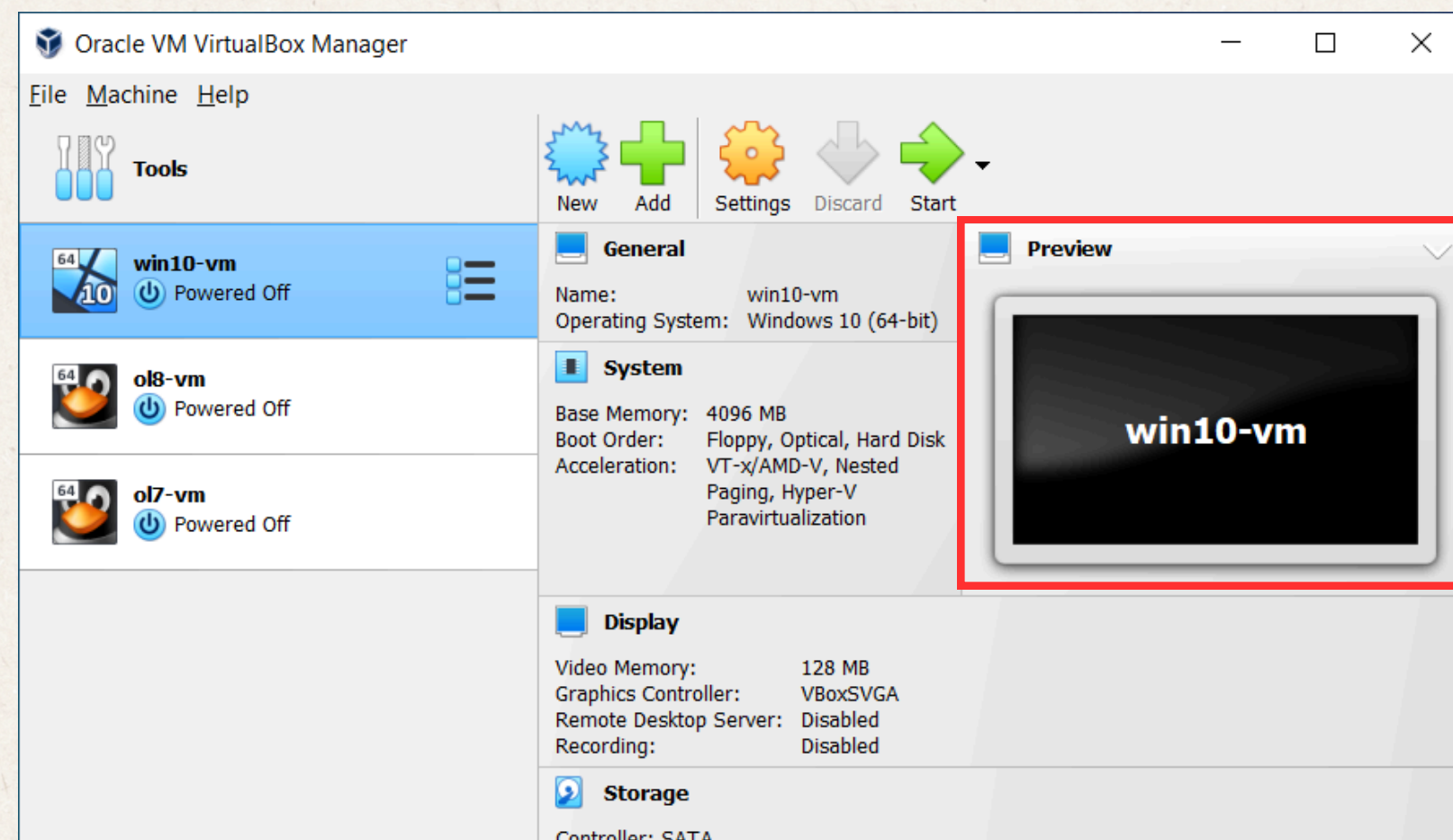
CVE-2025-53024

03/07



CVE-2025-53024

03/07



- Manipulated **uHeight** and **cbScanline** values → crashes
- Root cause: OOB reads beyond pbVram boundary
- Data inaccessible from guest, but promising lead discovered
- Next: Analyze these variables further

CVE-2025-53024

03/07

```
static void vmsvgaR3RectCopy(PVGASTATECC pThisCC, VMSVGASCREENOBJECT const *pScreen, uint32_t srcX, uint32_t srcY,
                             uint32_t dstX, uint32_t dstY, uint32_t width, uint32_t height, unsigned cbFramebuffer)
{
    ...
    uint32_t const  cbPixel = RT_ALIGN(pScreen->cbBpp, 8) / 8;
    uint32_t const  cbScanline = pScreen->cbPitch ? pScreen->cbPitch : width * cbPixel;
    ...

    uMaxOffset = (RT_MAX(srcY, dstY) + height) * cbScanline + (RT_MAX(srcX, dstX) + width) * cbPixel;
    if (uMaxOffset >= cbFramebuffer)
    {
        Log(("Max offset (%u) too big for framebuffer (%u bytes), ignoring!\n", uMaxOffset, cbFramebuffer));
        return; /* Just don't listen to a bad guest. */
    }
}
```


CVE-2025-53024

03/07

```
static void vmsvgaR3RectCopy(PVGASTATECC pThisCC, VMSVGASCREENOBJECT const *pScreen, uint32_t srcX, uint32_t srcY,
                             uint32_t dstX, uint32_t dstY, uint32_t width, uint32_t height, unsigned cbFramebuffer)
{
    ...
    uint32_t const  cbPixel = RT_ALIGN(pScreen->cbBpp, 8) / 8;
    uint32_t const  cbScanline = pScreen->cbPitch ? pScreen->cbPitch : width * cbPixel;
    ...

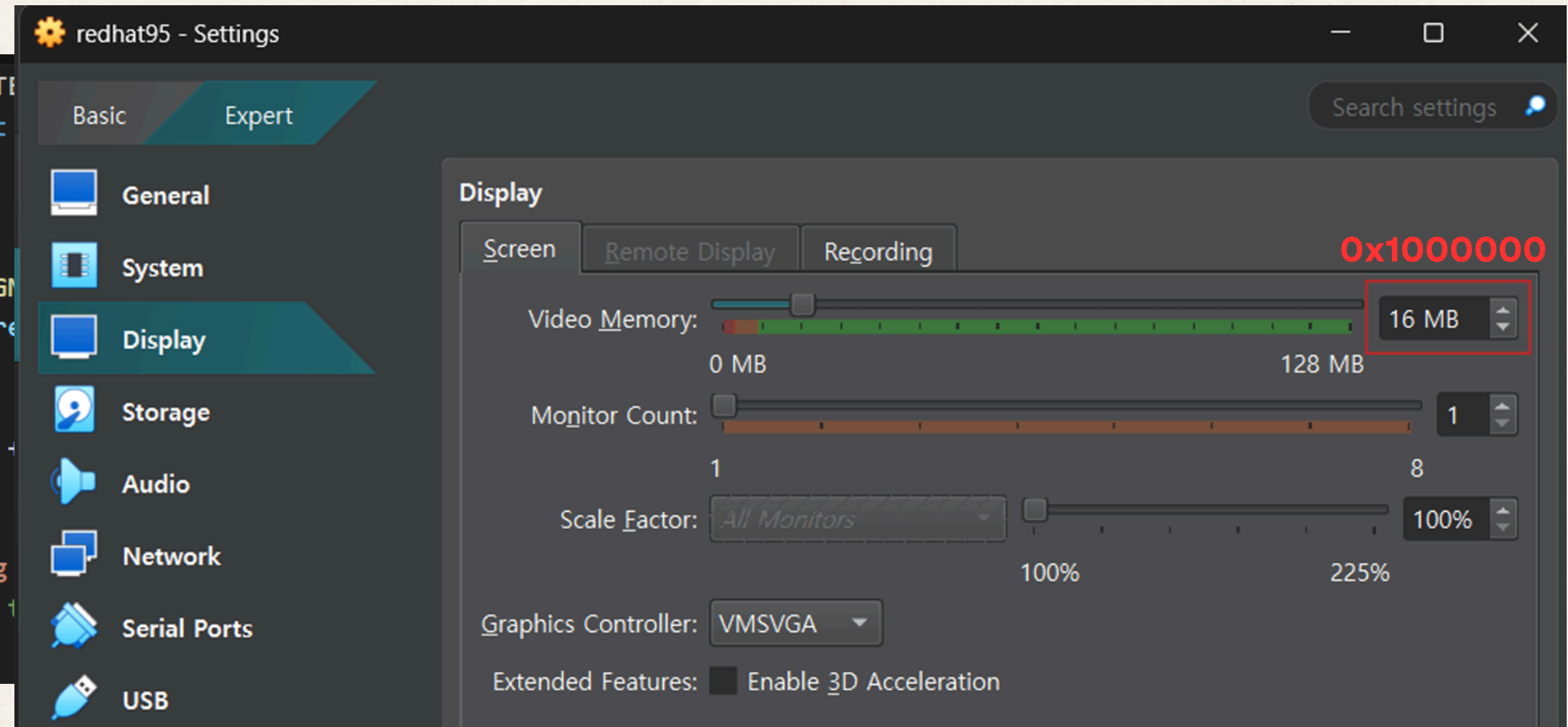
    uMaxOffset = (RT_MAX(srcY, dstY) + height) * cbScanline + (RT_MAX(srcX, dstX) + width) * cbPixel;
    if (uMaxOffset >= cbFramebuffer)
    {
        Log(("Max offset (%u) too big for framebuffer (%u bytes), ignoring!\n", uMaxOffset, cbFramebuffer));
        return; /* Just don't listen to a bad guest. */
    }
}
```


CVE-2025-53024

03/07

```
static void vmsvgaR3RectCopy(PVGASTATE
                                uint32_t
{
    ...
    uint32_t const  cbPixel = RT_ALIGN
    uint32_t const  cbScanline = pScre
    ...

    uMaxOffset = (RT_MAX(srcY, dstY) +
    if (uMaxOffset >= cbFrameBuffer)
    {
        Log(("Max offset (%u) too big
        return; /* Just don't listen t
    }
```



CVE-2025-53024

03/07

SrcX, destX, width, srcY, destY, height - these 6 values cannot be larger than `pThis->svga.u32MaxWidth` and `pThis->svga.u32MaxHeight`.

These 2 values are set to **0x800** in the settings.

```
void vmsvgaR3CmdRectCopy(PVGASTATE pThis, PVGASTATECC pThisCC, SVGAFifoCmdRectCopy const *pCmd)
{
    ...
    ASSERT_GUEST_RETURN_VOID(pCmd->srcX < pThis->svga.u32MaxWidth);
    ASSERT_GUEST_RETURN_VOID(pCmd->destX < pThis->svga.u32MaxWidth);
    ASSERT_GUEST_RETURN_VOID(pCmd->width < pThis->svga.u32MaxWidth);
    ASSERT_GUEST_RETURN_VOID(pCmd->srcY < pThis->svga.u32MaxHeight);
    ASSERT_GUEST_RETURN_VOID(pCmd->destY < pThis->svga.u32MaxHeight);
    ASSERT_GUEST_RETURN_VOID(pCmd->height < pThis->svga.u32MaxHeight);

    vmsvgaR3RectCopy(pThisCC, pScreen, pCmd->srcX, pCmd->srcY, pCmd->destX, pCmd->destY,
        pCmd->width, pCmd->height, pThis->vram_size);
    vmsvgaR3UpdateScreen(pThisCC, pScreen, pCmd->destX, pCmd->destY, pCmd->width, pCmd->height);
}
```


CVE-2025-53024

03/07

```
static void vmsvgaR3RectCopy(PVGASTATECC pThisCC, VMSVGASCREENOBJECT const *pScreen, uint32_t srcX, uint32_t srcY,  
                             uint32_t dstX, uint32_t dstY, uint32_t width, uint32_t height, unsigned cbFramebuffer)  
{  
  
    ...  
    uint32_t const cbPixel = RT_ALIGN(pScreen->cBpp, 8) / 8;  
    uint32_t const cbScanline = pScreen->cbPitch ? pScreen->cbPitch : width * cbPixel;  
    ...  
  
    uMaxOffset = (RT_MAX(srcY, dstY) + height) * cbScanline + (RT_MAX(srcX, dstX) + width) * cbPixel;  
    if (uMaxOffset >= cbFramebuffer)  
    {  
        Log(("Max offset (%u) too big for framebuffer (%u bytes), ignoring!\n", uMaxOffset, cbFramebuffer));  
        return; /* Just don't listen to a bad guest. */  
    }  
}
```



```
uint32_t const cbPixel = RT_ALIGN(pScreen->cBpp, 8) / 8;
uint32_t const cbScanline = pScreen->cbPitch ? pScreen->cbPitch : width * cbPixel;
```

```
/* SVGA_CMD_DEFINE_SCREEN */
void vmsvgaR3CmdDefineScreen(PVGASTATE pThis, PVGASTATECC pThisCC, SVGA FifoCmdDefineScreen const *pCmd)
{
    ...
    uint32_t const idScreen = pCmd->screen.id;
    ASSERT_GUEST_RETURN_VOID(idScreen < RT_ELEMENTS(pSvgaR3State->aScreens));
    uint32_t const uWidth = pCmd->screen.size.width;
    ASSERT_GUEST_RETURN_VOID(uWidth <= pThis->svga.u32MaxWidth);
    uint32_t const uHeight = pCmd->screen.size.height;
    ASSERT_GUEST_RETURN_VOID(uHeight <= pThis->svga.u32MaxHeight);
    uint32_t const cbWidth = uWidth * ((32 + 7) / 8); /** @todo 32? */
    uint32_t const cbPitch = pCmd->screen.backingStore.pitch ? pCmd->screen.backingStore.pitch : cbWidth;
    ASSERT_GUEST_RETURN_VOID(cbWidth <= cbPitch);
    uint32_t const uScreenOffset = pCmd->screen.backingStore.ptr.offset;
    ASSERT_GUEST_RETURN_VOID(uScreenOffset < pThis->vram_size);
    uint32_t const cbVram = pThis->vram_size - uScreenOffset;

    /* If we have a not zero pitch, then height can't exceed the available VRAM. */
    ASSERT_GUEST_RETURN_VOID( (uHeight == 0 && cbPitch == 0)
        || (cbPitch > 0 && uHeight <= cbVram / cbPitch));

    if (!RT_BOOL(pCmd->screen.flags & (SVGA_SCREEN_DEACTIVATE | SVGA_SCREEN_BLANKING)))
    {
        ...
        pScreen->cWidth = uWidth;
        pScreen->cHeight = uHeight;
        pScreen->offVRAM = uScreenOffset;
        pScreen->cbPitch = cbPitch;
        pScreen->cBpp = 32;
    }
}
```



```
uint32_t const cbPixel = RT_ALIGN(pScreen->cBpp, 8) / 8;
uint32_t const cbScanline = pScreen->cbPitch ? pScreen->cbPitch : width * cbPixel;
```

```
/* SVGA_CMD_DEFINE_SCREEN */
void vmsvgaR3CmdDefineScreen(PVGASTATE pThis, PVGASTATECC pThisCC, SVGA_FIFO_CmdDefineScreen const *pCmd)
{
    ...
    uint32_t const idScreen = pCmd->screen.id;
    ASSERT_GUEST_RETURN_VOID(idScreen < RT_ELEMENTS(pSvgaR3State->aScreens));
    uint32_t const uWidth = pCmd->screen.size.width;
    ASSERT_GUEST_RETURN_VOID(uWidth <= pThis->svga.u32MaxWidth);
    uint32_t const uHeight = pCmd->screen.size.height;
    ASSERT_GUEST_RETURN_VOID(uHeight <= pThis->svga.u32MaxHeight);
    uint32_t const cbWidth = uWidth * ((32 + 7) / 8); /** @todo 32? */
    uint32_t const cbPitch = pCmd->screen.backingStore.pitch ? pCmd->screen.backingStore.pitch : cbWidth;
    ASSERT_GUEST_RETURN_VOID(cbWidth <= cbPitch);
    uint32_t const uScreenOffset = pCmd->screen.backingStore.ptr.offset;
    ASSERT_GUEST_RETURN_VOID(uScreenOffset < pThis->vram_size);
    uint32_t const cbVram = pThis->vram_size - uScreenOffset;

    /* If we have a not zero pitch, then height can't exceed the available VRAM. */
    ASSERT_GUEST_RETURN_VOID( (uHeight == 0 && cbPitch == 0)
        || (cbPitch > 0 && uHeight <= cbVram / cbPitch));

    if (!RT_BOOL(pCmd->screen.flags & (SVGA_SCREEN_DEACTIVATE | SVGA_SCREEN_BLANKING)))
    {
        ...
        pScreen->cWidth = uWidth;
        pScreen->cHeight = uHeight;
        pScreen->offVRAM = uScreenOffset;
        pScreen->cbPitch = cbPitch;
        pScreen->cBpp = 32;
    }
}
```



```
uint32_t const cbPixel = RT_ALIGN(pScreen->cBpp, 8) / 8;
uint32_t const cbScanline = pScreen->cbPitch ? pScreen->cbPitch : width * cbPixel;
```

```
/* SVGA_CMD_DEFINE_SCREEN */
void vmsvgaR3CmdDefineScreen(PVGASTATE pThis, PVGASTATECC pThisCC, SVGA_FIFO_CmdDefineScreen const *pCmd)
{
    ...
    uint32_t const idScreen = pCmd->screen.id;
    ASSERT_GUEST_RETURN_VOID(idScreen < RT_ELEMENTS(pSvgaR3State->aScreens));
    uint32_t const uWidth = pCmd->screen.size.width;
    ASSERT_GUEST_RETURN_VOID(uWidth <= pThis->svga.u32MaxWidth);
    uint32_t const uHeight = pCmd->screen.size.height;
    ASSERT_GUEST_RETURN_VOID(uHeight <= pThis->svga.u32MaxHeight);
    uint32_t const cbWidth = uWidth * ((32 + 7) / 8); /** @todo 32? */
    uint32_t const cbPitch = pCmd->screen.backingStore.pitch ? pCmd->screen.backingStore.pitch : cbWidth;
    ASSERT_GUEST_RETURN_VOID(cbWidth <= cbPitch);
    uint32_t const uScreenOffset = pCmd->screen.backingStore.ptr.offset;
    ASSERT_GUEST_RETURN_VOID(uScreenOffset < pThis->vram_size);
    uint32_t const cbVram = pThis->vram_size - uScreenOffset;

    /* If we have a not zero pitch, then height can't exceed the available VRAM. */
    ASSERT_GUEST_RETURN_VOID( (uHeight == 0 && cbPitch == 0)
        || (cbPitch > 0 && uHeight <= cbVram / cbPitch));

    if (!RT_BOOL(pCmd->screen.flags & (SVGA_SCREEN_DEACTIVATE | SVGA_SCREEN_BLANKING)))
    {
        ...
        pScreen->cWidth = uWidth;
        pScreen->cHeight = uHeight;
        pScreen->offVRAM = uScreenOffset;
        pScreen->cbPitch = cbPitch;
        pScreen->cBpp = 32;
    }
}
```



```
uint32_t const cbPixel = RT_ALIGN(pScreen->cBpp, 8) / 8;
uint32_t const cbScanline = pScreen->cbPitch ? pScreen->cbPitch : width * cbPixel;
```

```
/* SVGA_CMD_DEFINE_SCREEN */
void vmsvgaR3CmdDefineScreen(PVGASTATE pThis, PVGASTATECC pThisCC, SVGA FifoCmdDefineScreen const *pCmd)
{
    ...
    uint32_t const idScreen = pCmd->screen.id;
    ASSERT_GUEST_RETURN_VOID(idScreen < RT_ELEMENTS(pSvgaR3State->aScreens));
    uint32_t const uWidth = pCmd->screen.size.width;
    ASSERT_GUEST_RETURN_VOID(uWidth <= pThis->svga.u32MaxWidth);
    uint32_t const uHeight = pCmd->screen.size.height;
    ASSERT_GUEST_RETURN_VOID(uHeight <= pThis->svga.u32MaxHeight);
    uint32_t const cbWidth = uWidth * ((32 + 7) / 8); /** @todo 32? */
    uint32_t const cbPitch = pCmd->screen.backingStore.pitch ? pCmd->screen.backingStore.pitch : cbWidth;
    ASSERT_GUEST_RETURN_VOID(cbWidth <= cbPitch);
    uint32_t const uScreenOffset = pCmd->screen.backingStore.ptr.offset;
    ASSERT_GUEST_RETURN_VOID(uScreenOffset < pThis->vram_size);
    uint32_t const cbVram = pThis->vram_size - uScreenOffset;

    /* If we have a not zero pitch, then height can't exceed the available VRAM. */
    ASSERT_GUEST_RETURN_VOID( (uHeight == 0 && cbPitch == 0)
        || (cbPitch > 0 && uHeight <= cbVram / cbPitch));

    if (!RT_BOOL(pCmd->screen.flags & (SVGA_SCREEN_DEACTIVATE | SVGA_SCREEN_BLANKING)))
    {
        ...
        pScreen->cWidth = uWidth;
        pScreen->cHeight = uHeight;
        pScreen->offVRAM = uScreenOffset;
        pScreen->cbPitch = cbPitch;
        pScreen->cBpp = 32;
    }
}
```



```
uint32_t const cbPixel = RT_ALIGN(pScreen->cBpp, 8) / 8;
uint32_t const cbScanline = pScreen->cbPitch ? pScreen->cbPitch : width * cbPixel;
```

```
/* SVGA_CMD_DEFINE_SCREEN */
void vmsvgaR3CmdDefineScreen(PVGASTATE pThis, PVGASTATECC pThisCC, SVGA FifoCmdDefineScreen const *pCmd)
{
    ...
    uint32_t const idScreen = pCmd->screen.id;
    ASSERT_GUEST_RETURN_VOID(idScreen < RT_ELEMENTS(pSvgaR3State->aScreens));
    uint32_t const uWidth = pCmd->screen.size.width;
    ASSERT_GUEST_RETURN_VOID(uWidth <= pThis->svga.u32MaxWidth);
    uint32_t const uHeight = pCmd->screen.size.height;
    ASSERT_GUEST_RETURN_VOID(uHeight <= pThis->svga.u32MaxHeight); pCmd fields are Guest Controllable
    uint32_t const cbWidth = uWidth * ((32 + 7) / 8); /** @todo 32? */
    uint32_t const cbPitch = pCmd->screen.backingStore.pitch ? pCmd->screen.backingStore.pitch : cbWidth;
    ASSERT_GUEST_RETURN_VOID(cbWidth <= cbPitch);
    uint32_t const uScreenOffset = pCmd->screen.backingStore.ptr.offset;
    ASSERT_GUEST_RETURN_VOID(uScreenOffset < pThis->vram_size);
    uint32_t const cbVram = pThis->vram_size - uScreenOffset;

    /* If we have a not zero pitch, then height can't exceed the available VRAM. */
    ASSERT_GUEST_RETURN_VOID( (uHeight == 0 && cbPitch == 0)
    || (cbPitch > 0 && uHeight <= cbVram / cbPitch));

    if (!RT_BOOL(pCmd->screen.flags & (SVGA_SCREEN_DEACTIVATE | SVGA_SCREEN_BLANKING)))
    {
        ...
        pScreen->cWidth = uWidth;
        pScreen->cHeight = uHeight;
        pScreen->offVRAM = uScreenOffset;
        pScreen->cbPitch = cbPitch;
        pScreen->cBpp = 32;
    }
}
```



```
uint32_t const cbPixel = RT_ALIGN(pScreen->cBpp, 8) / 8;
uint32_t const cbScanline = pScreen->cbPitch ? pScreen->cbPitch : width * cbPixel;
```

```
/* SVGA_CMD_DEFINE_SCREEN */
void vmsvgaR3CmdDefineScreen(PVGASTATE pThis, PVGASTATECC pThisCC, SVGA FifoCmdDefineScreen const *pCmd)
{
    ...
    uint32_t const idScreen = pCmd->screen.id;
    ASSERT_GUEST_RETURN_VOID(idScreen < RT_ELEMENTS(pSvgaR3State->aScreens));
    uint32_t const uWidth = pCmd->screen.size.width;
    ASSERT_GUEST_RETURN_VOID(uWidth <= pThis->svga.u32MaxWidth);
    uint32_t const uHeight = pCmd->screen.size.height;
    ASSERT_GUEST_RETURN_VOID(uHeight <= pThis->svga.u32MaxHeight);
    uint32_t const cbWidth = uWidth * ((32 + 7) / 8); /** @todo 32? */
    uint32_t const cbPitch = pCmd->screen.backingStore.pitch ? pCmd->screen.backingStore.pitch : cbWidth;
    ASSERT_GUEST_RETURN_VOID(cbWidth <= cbPitch);
    uint32_t const uScreenOffset = pCmd->screen.backingStore.ptr.offset;
    ASSERT_GUEST_RETURN_VOID(uScreenOffset < pThis->vram_size);
    uint32_t const cbVram = pThis->vram_size - uScreenOffset;

    /* If we have a not zero pitch, then height can't exceed the available VRAM. */
    ASSERT_GUEST_RETURN_VOID( (uHeight == 0 && cbPitch == 0)
    || (cbPitch > 0 && uHeight <= cbVram / cbPitch));

    if (!RT_BOOL(pCmd->screen.flags & (SVGA_SCREEN_DEACTIVATE | SVGA_SCREEN_BLANKING)))
    {
        ...
        pScreen->cWidth = uWidth;
        pScreen->cHeight = uHeight;
        pScreen->offVRAM = uScreenOffset;
        pScreen->cbPitch = cbPitch;
        pScreen->cBpp = 32;
    }
}
```

When width and height are appropriately small,
cbPitch can reach VRAM size


```

static void vmsvgaR3RectCopy(PVGASTATECC pThisCC, VMSVGASCREENOBJECT const *pScreen, uint32_t srcX, uint32_t srcY,
uint32_t dstX, uint32_t dstY, uint32_t width, uint32_t height, unsigned cbFramebuffer)
{
    uint32_t const cbScanline = pScreen->cbPitch ? pScreen->cbPitch : width * cbPixel;
    uMaxOffset = (RT_MAX(srcY, dstY) + height) * cbScanline + (RT_MAX(srcX, dstX) + width) * cbPixel;
    if (uMaxOffset >= cbFramebuffer)
    {
        Log(("Max offset (%u) too big for framebuffer (%u bytes), ignoring!\n", uMaxOffset, cbFramebuffer));
        return; /* Just don't listen to a bad guest. */
    }

    pSrc = pDst = pThisCC->pbVRam;
    pSrc += srcY * cbScanline + srcX * cbPixel;
    pDst += dstY * cbScanline + dstX * cbPixel;

    if (srcY >= dstY)
    {
        /* Source below destination, copy top to bottom. */
        for (; height > 0; height--)
        {
            memmove(pDst, pSrc, cbRectWidth);
            pSrc += cbScanline;
            pDst += cbScanline;
        }
    }
    else
    {
        /* Source above destination, copy bottom to top. */
        pSrc += cbScanline * (height - 1);
        pDst += cbScanline * (height - 1);
        for (; height > 0; height--)
        {
            memmove(pDst, pSrc, cbRectWidth);
            pSrc -= cbScanline;
            pDst -= cbScanline;
        }
    }
}

```

03/07

0x100000400 0 0x1ff 1 0x800000 0 0 0x100 4
srcY = 0, dstY = 0x1ff, height = 1; srcX = 0, dstX = 0, width = 0x100
pScreen->cBpp = 32, pScreen->cbPitch = 0x800000;


```

static void vmsvgaR3RectCopy(PVGASTATECC pThisCC, VMSVGASCREENOBJECT const *pScreen, uint32_t srcX, uint32_t srcY,
                             uint32_t dstX, uint32_t dstY, uint32_t width, uint32_t height, unsigned cbFramebuffer)
{
    uint32_t const  cbScanline = pScreen->cbPitch ? pScreen->cbPitch : width * cbPixel;

    uMaxOffset = (RT_MAX(srcY, dstY) + height) * cbScanline + (RT_MAX(srcX, dstX) + width) * cbPixel;
    if (uMaxOffset >= cbFramebuffer)
    {
        Log("Max offset (%u) too big for framebuffer (%u bytes), ignoring!\n", uMaxOffset, cbFramebuffer));
        return; /* Just don't listen to a bad guest. */
    }

    pSrc = pDst = pThisCC->pbVRam;
    pSrc += srcY * cbScanline + srcX * cbPixel;
    pDst += dstY * cbScanline + dstX * cbPixel;

    if (srcY >= dstY)
    {
        /* Source below destination, copy top to bottom. */
        for (; height > 0; height--)
        {
            memmove(pDst, pSrc, cbRectWidth);
            pSrc += cbScanline;
            pDst += cbScanline;
        }
    }
    else
    {
        /* Source above destination, copy bottom to top. */
        pSrc += cbScanline * (height - 1);
        pDst += cbScanline * (height - 1);
        for (; height > 0; height--)
        {
            memmove(pDst, pSrc, cbRectWidth);
            pSrc -= cbScanline;
            pDst -= cbScanline;
        }
    }
}

```

03/07

Integer Overflow!
0x1000000400 >= 0x1000000000


```

static void vmsvgaR3RectCopy(PVGASTATECC pThisCC, VMSVGASCREENOBJECT const *pScreen, uint32_t srcX, uint32_t srcY,
                             uint32_t dstX, uint32_t dstY, uint32_t width, uint32_t height, unsigned cbFramebuffer)
{
    uint32_t const cbScanline = pScreen->cbPitch ? pScreen->cbPitch : width * cbPixel;

    uMaxOffset = (RT_MAX(srcY, dstY) + height) * cbScanline + (RT_MAX(srcX, dstX) + width) * cbPixel;
    if (uMaxOffset >= cbFramebuffer)
    {
        Log(("Max offset (%u) too big for framebuffer (%u bytes), ignoring!\n", uMaxOffset, cbFramebuffer));
        return; /* Just don't listen to a bad guest. */
    }

    pSrc = pDst = pThisCC->pbVRam;
    pSrc += srcY * cbScanline + srcX * cbPixel;
    pDst += dstY * cbScanline + dstX * cbPixel;

    if (srcY >= dstY)
    {
        /* Source below destination, copy top to bottom. */
        for (; height > 0; height--)
        {
            memmove(pDst, pSrc, cbRectWidth);
            pSrc += cbScanline;
            pDst += cbScanline;
        }
    }
    else
    {
        /* Source above destination, copy bottom to top. */
        pSrc += cbScanline * (height - 1);
        pDst += cbScanline * (height - 1);
        for (; height > 0; height--)
        {
            memmove(pDst, pSrc, cbRectWidth);
            pSrc -= cbScanline;
            pDst -= cbScanline;
        }
    }
}

```

pDst = pThisCC->pbVRam

pSrc = pThisCC->pbVRam + 0xff800000

*** pbVRam is Guest read/writable Memory ***


```

static void vmsvgaR3RectCopy(PVGASTATECC pThisCC, VMSVGASCREENOBJECT const *pScreen, uint32_t srcX, uint32_t srcY,
                             uint32_t dstX, uint32_t dstY, uint32_t width, uint32_t height, unsigned cbFramebuffer)
{
    uint32_t const cbScanline = pScreen->cbPitch ? pScreen->cbPitch : width * cbPixel;

    uMaxOffset = (RT_MAX(srcY, dstY) + height) * cbScanline + (RT_MAX(srcX, dstX) + width) * cbPixel;
    if (uMaxOffset >= cbFramebuffer)
    {
        Log(("Max offset (%u) too big for framebuffer (%u bytes), ignoring!\n", uMaxOffset, cbFramebuffer));
        return; /* Just don't listen to a bad guest. */
    }

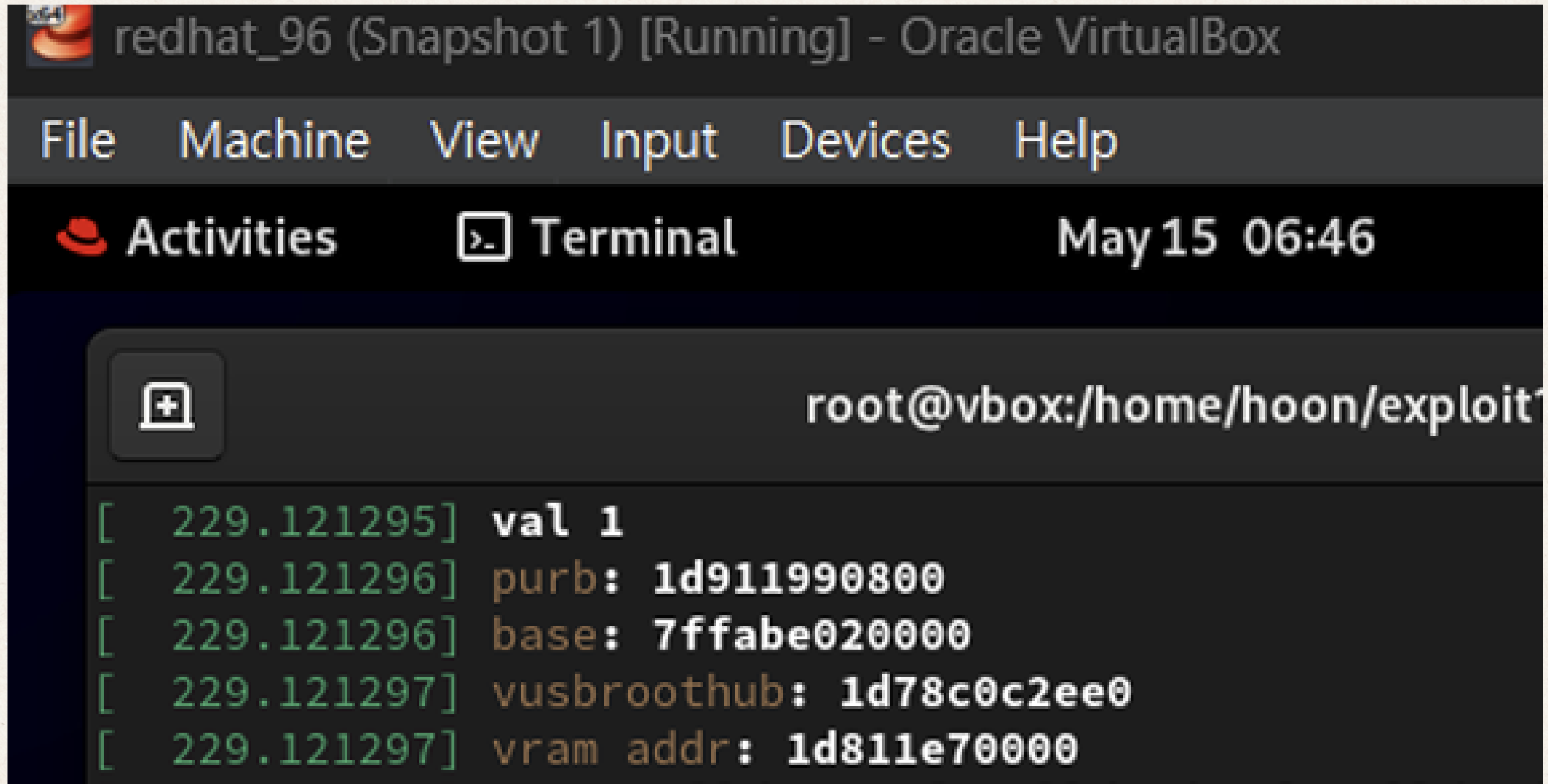
    pSrc = pDst = pThisCC->pbVRam;
    pSrc += srcY * cbScanline + srcX * cbPixel;
    pDst += dstY * cbScanline + dstX * cbPixel;

    if (srcY >= dstY)
    {
        /* Source below destination, copy top to bottom. */
        for (; height > 0; height--)
        {
            memmove(pDst, pSrc, cbRectWidth);
            pSrc += cbScanline;
            pDst += cbScanline;
        }
    }
    else
    {
        /* Source above destination, copy bottom to top. */
        pSrc += cbScanline * (height - 1);
        pDst += cbScanline * (height - 1);
        for (; height > 0; height--)
        {
            memmove(pDst, pSrc, cbRectWidth);
            pSrc -= cbScanline;
            pDst -= cbScanline;
        }
    }
}

```

03/07

[pSrc in pbVRAM && pDst out pbVRAM] → OOB write
[pSrc out pbVRAM && pDst in pbVRAM] → OOB Read



```
redhat_96 (Snapshot 1) [Running] - Oracle VirtualBox
File Machine View Input Devices Help
Activities Terminal May 15 06:46
root@vbox:/home/hoon/exploit
[ 229.121295] val 1
[ 229.121296] purb: 1d911990800
[ 229.121296] base: 7ffabe020000
[ 229.121297] vusbroot hub: 1d78c0c2ee0
[ 229.121297] vram addr: 1d811e70000
```


VirtIO OOB/W

03/07

A structural flaw in the validation logic during
GSO (Generic Segmentation Offload) processing

```
static int virtioNetR3TransmitPkts(PPDMDEVINS pDevIns, PVIRTIONET pThis, PVIRTIONETCC pThisCC,  
                                   PVIRTIONETVIRTQ pTxVirtq, bool fOnWorkerThread)  
{  
    ...  
    rc = pThisCC->pDrv->pfnAllocBuf(pThisCC->pDrv, uFrameSize, pGso, &pSgBufToPdmLeafDevice);  
    ...  
    rc = virtioNetR3TransmitFrame(pThis, pThisCC, pSgBufToPdmLeafDevice, pGso, pPktHdr);  
    ...  
}
```


VirtIO OOB/W

03/07

```
static DECLCALLBACK(int) drvNATNetworkUp_AllocBuf(PPDMINETWORKUP pInterface, size_t cbMin,  
PCPDMNETWORKGSO pGso, PPPDMSCATTERGATHER ppSgBuf)  
{  
    ...  
    /*  
    * Allocate a scatter/gather buffer and an mbuf.  
    */  
    PPDMSCATTERGATHER pSgBuf = (PPDMSCATTERGATHER)RTMemAlloc(sizeof(*pSgBuf));  
  
    ...  
    check sbHdrsTotal Size 0x4000  
    if (pGso->cbHdrsTotal + pGso->cbMaxSeg >= DRVNAT_MAXFRAMESIZE)  
    {  
        Log(("drvNATNetowrkUp_AllocBuf: drops over-sized frame (%u bytes), returns VERR_INVALID_PARAMETER\n",  
            pGso->cbHdrsTotal + pGso->cbMaxSeg));  
        RTMemFree(pSgBuf);  
        return VERR_INVALID_PARAMETER;  
    }  
}
```


VirtIO OOB/W

03/07

```
static int virtioNetR3TransmitFrame(PVIRTIONET pThis, PVIRTIONETCC pThisCC, PPDMSCATTERGATHER pSgBuf,  
                                     PPDMNETWORKGSO pGso, PVIRTIONETPKTHDR pPktHdr)  
{  
    ...  
    switch (pGso->u8Type)  
    {  
        case PDMNETWORKGSOTYPE_IPV4_TCP:  
        case PDMNETWORKGSOTYPE_IPV6_TCP:  
            pGso->cbHdrsTotal = pPktHdr->uChecksumStart +  
                ((PRTNETTCP)((uint8_t*)pSgBuf->aSegs[0].pvSeg) + pPktHdr->uChecksumStart)->th_off * 4;  
    }
```


VirtIO OOB/W

03/07

```
static DECLCALLBACK(void) drvNATSendWorker(PDRVNAT pThis, PPDMSCATTERGATHER pSgBuf)
{
    ...
    if (PDMNetGsoIsValid(pGso, sizeof(*pGso), pSgBuf->cbUsed))
    {
        uint32_t const cSegs = PDMNetGsoCalcSegmentCount(pGso, pSgBuf->cbUsed); Assert(cSegs > 1);
        for (uint32_t iSeg = 0; iSeg < cSegs; iSeg++)
        {
            size_t cbSeg;
            void *pvSeg;

            //pvSeg = RTMEMAlloc(DRVNAT_MAXFRAME_SIZE);
            m = slirp_ext_m_get(pThis->pNATState, pGso->cbHdrsTotal + pGso->cbMaxSeg, &pvSeg, &cbSeg);
            if (!m)
                break;

            uint32_t cbPayload, cbHdrs;
            uint32_t offPayload = PDMNetGsoCarveSegment(pGso, pbFrame, pSgBuf->cbUsed,
                iSeg, cSegs, (uint8_t *)pvSeg, &cbHdrs, &cbPayload);
            memcpy((uint8_t *)pvSeg + cbHdrs, pbFrame + offPayload, cbPayload);
            ...
        }
    }
}
```

OOB WRITE!

ZDI-CAN-27241

03/07

Unpatched

EXPLOIT

03/07

Sponsor reserves the right, in its sole discretion,
to **allow non-default configurations** if the Sponsor deems them
to be in the normal use case of the target under test.

VirtIO-Net OOB/W + Info Leak Exploit reliability stabilized at **90%+**
(Guest Ubuntu + Host Windows)

We were waiting for **March**
(Pwn2Own 2024 Vancouver was held in March)



EXPLOIT

03/07

 Oracle January 2025 Patch
✗ Our vulnerabilities: **NOT FIXED**



EXPLOIT

03/07

March was approaching but no [Pwn2Own](#) schedule announced
(Rumors suggested it would be held with [OffensiveCon](#))



EXPLOIT

03/07

Pwn2Own 2025 Berlin with OffensiveCon : **May**

&&

 Oracle April 2025 Patch
✗ VirtIO-NET OOB/W : **FIXED**

&&

Guest OS : **RHEL**



EXPLOIT

03/07

But still we had **VGA R/W** vulnerabilities

Found **new exploitation objects** for RHEL

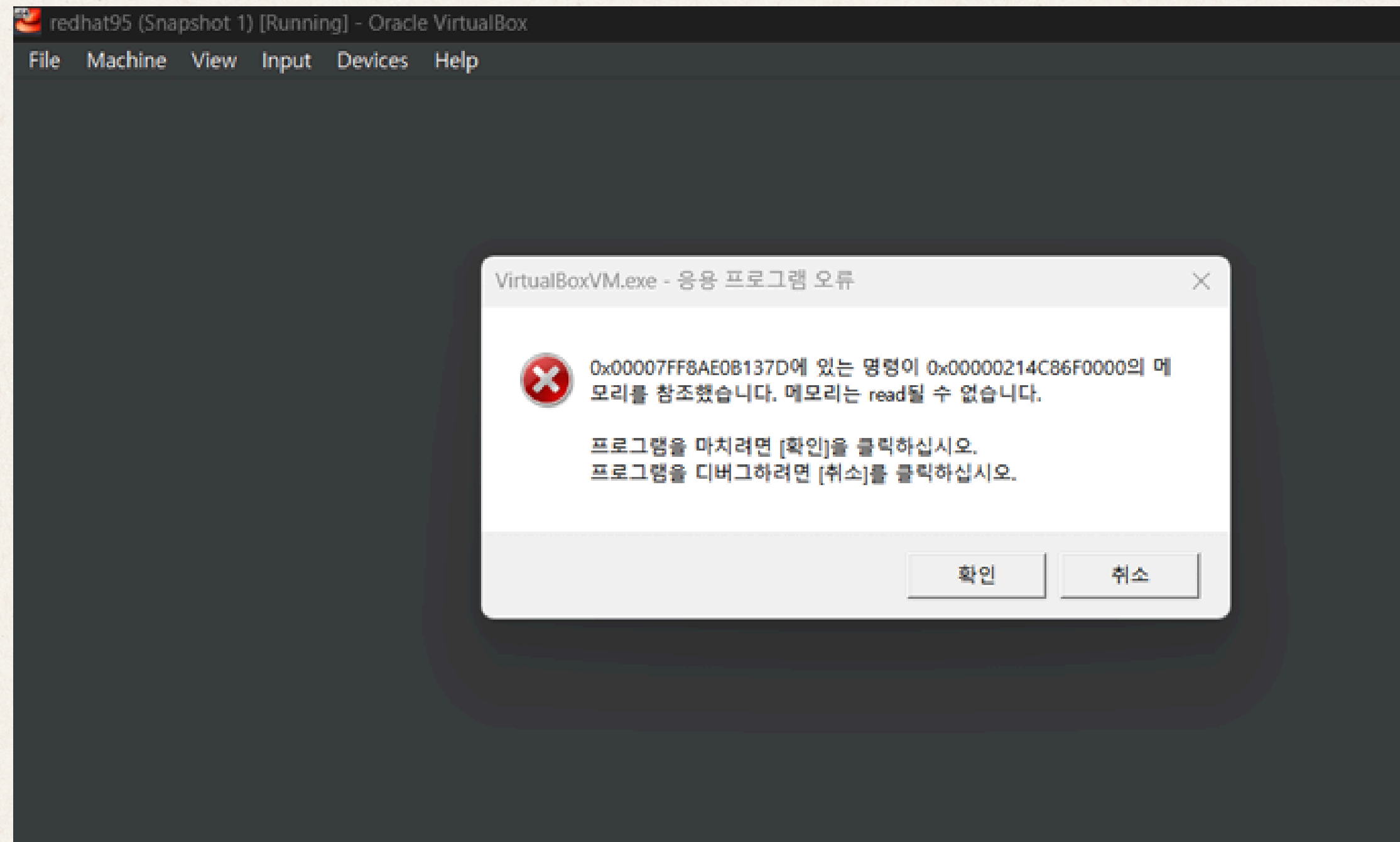
Successfully stabilized exploit reliability to **90%+**



EXPLOIT

03/07

VGA-SVGA OOB R/W Exploit - Heap Spray



Crash occurs: Memory not allocated at pbVRAM+0xffc00000
We need massive heap allocation (Almost 4GB)

EXPLOIT

03/07

VGA-SVGA OOB R/W Exploit - Heap Spray

Exploit Object Requirements

- Massive heap allocation capability
 - Need chunks at ~4GB offset
- Chunk size > 0x4000
 - Avoid LFH guard pages
 - Reduce heap randomness

```
case SVGA_CMD_DEFINE_GMR2:
{
    SVGA FifoCmdDefineGMR2 *pCmd = (SVGA FifoCmdDefineGMR2 *)&pu8Cmd[cbCmd];
    VMSVGA_INC_CMD_SIZE_BREAK(sizeof(*pCmd));
    vmsvgaR3CmdDefineGMR2(pThis, pThisCC, pCmd);
    break;
}
```

```
/* SVGA_CMD_DEFINE_GMR2 */
void vmsvgaR3CmdDefineGMR2(PVGASTATE pThis, PVGASTATECC pThisCC,
                           SVGA FifoCmdDefineGMR2 const *pCmd)
{
    PVMSVGAR3STATE const pSvgaR3State = pThisCC->svga.pSvgaR3State;

    STAM_REL_COUNTER_INC(&pSvgaR3State->StatR3CmdDefineGmr2);
    Log(("SVGA_CMD_DEFINE_GMR2 id=%#x %#x pages\n", pCmd->gmrId, pCmd->numPages));

    /* Validate current GMR id. */
    ASSERT_GUEST_RETURN_VOID(pCmd->gmrId < pThis->svga.cGMR); /* 0x2000 */
    ASSERT_GUEST_RETURN_VOID(pCmd->numPages <= VMSVGA_MAX_GMR_PAGES); /* 0x100000 */
}
```



```

/* SVGA_CMD_REMAP_GMR2 */
void vmsvgaR3CmdRemapGMR2(PVGASTATE pThis, PVGASTATECC pThisCC, SVGAFifoCmdRemapGMR2 const *pCmd)
{
    PVMSVGAR3STATE const pSvgaR3State = pThisCC->svga.pSvgaR3State;

    STAM_REL_COUNTER_INC(&pSvgaR3State->StatR3CmdRemapGmr2);
    Log(("SVGA_CMD_REMAP_GMR2 id=0x%x flags=0x%x offset=0x%x npages=0x%x\n", pCmd->gmrId, pCmd->flags, pCmd->offsetPages, pCmd->numPages));

    /* Validate current GMR id and size. */
    ASSERT_GUEST_RETURN_VOID(pCmd->gmrId < pThis->svga.cGMR);
    RT_UNTRUSTED_VALIDATED_FENCE();
    PGMR pGMR = &pSvgaR3State->paGMR[pCmd->gmrId];
    ...
    /* Allocate the maximum amount possible (everything non-continuous) */
    PVMSVGAGMRDESCRIPTOR paDescs;
    pGMR->paDesc = paDescs = (PVMSVGAGMRDESCRIPTOR)RTMemAllocZ(cNewTotalPages * sizeof(VMSVGAGMRDESCRIPTOR));
    AssertReturnVoidStmt(paDescs, RTMemFree(paNewPage64));
    else
    {
        ...
        uint32_t iDescriptor = 0;
        for (uint32_t i = 1; i < cPages; i++)
        {
            /* cPages <= 0x100000 */
            ...
            else
            {
                iDescriptor++;
                paDescs[iDescriptor].GCPhys = GCPhys; /* Guest Control */
                paDescs[iDescriptor].numPages = 1;
                Log5Func(("Page 0x%x GCPhys=0xRGp\n", i, paDescs[iDescriptor].GCPhys));
            }
        }
        ...
    }
    ...
}

```

0x2000 GMRs × 0x100000 pages → 4GB+ heap allocation

EXPLOIT

03/07

VGA-SVGA OOB R/W Exploit - Heap Spray, Information Leak



EXPLOIT

03/07

VGA-SVGA OOB R/W Exploit - Heap Spray, Information Leak



EXPLOIT

03/07

VGA-SVGA OOB R/W Exploit - Heap Spray, Information Leak



EXPLOIT

VGA-SVGA OOB R/W Exploit - URB

```
/**
 * Worker routine for vusbRhConnNewUrb().
 */
static PVUSBURB vusbRhNewUrb(PVUSBROOTTHUB pRh, uint8_t DstAddress, uint32_t uPort,
                             VUSBXFERTYPE enmType,
                             VUSB_DIRECTION enmDir, uint32_t cbData, uint32_t cTds, const char *pszTag)
{
    ...
    if (RT_LIKELY(pUrb))
    {
        pUrb->pVUsb->pvFreeCtx = pRh;           // roothub
        pUrb->pVUsb->pfnFree   = vusbRhFreeUrb; // function address
        pUrb->DstAddress       = DstAddress;
        pUrb->pVUsb->pDev      = pDev;
        pUrb->pVUsb->pUrb      = pUrb;         // vram_address = pUrb - 0xffc00000 - a
    }
}
```

Information Disclosure via VUSBURB:

- ✓ VboxDD.dll base address
- ✓ Heap addresses
- ✓ VRAM & roothub address derivation

EXPLOIT

VGA-SVGA OOB R/W Exploit - URB

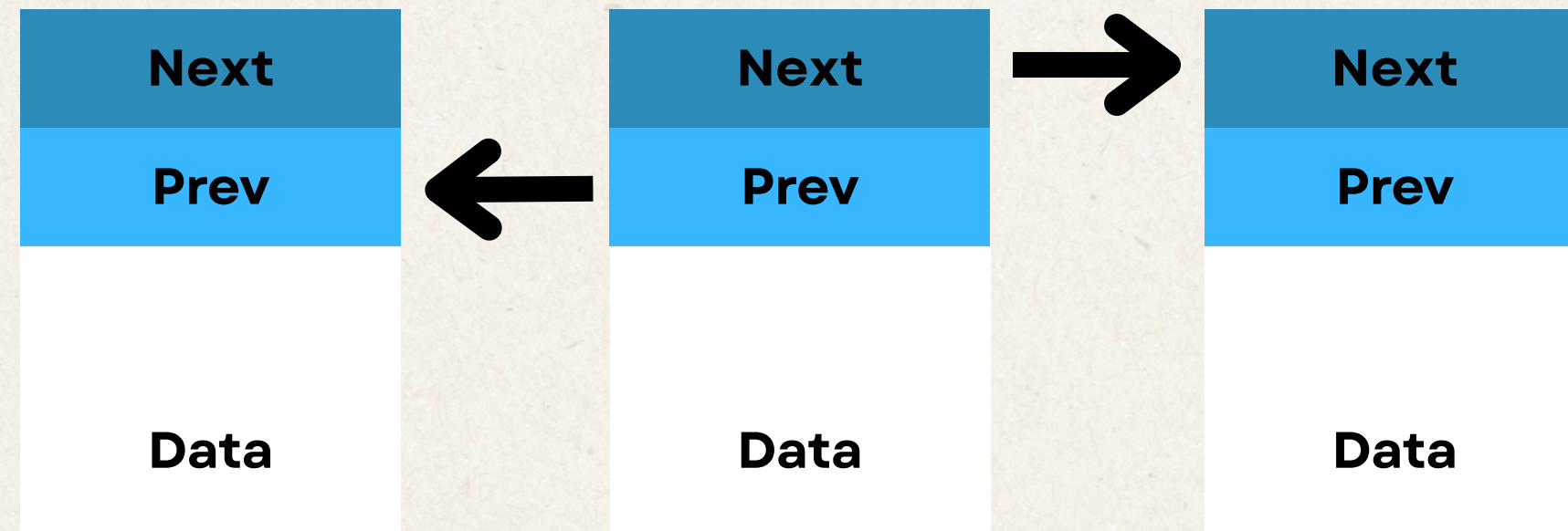
linked list structure

```
typedef struct RTLISTNODE
{
    /** Pointer to the next list node. */
    struct RTLISTNODE *pNext;
    /** Pointer to the previous list node. */
    struct RTLISTNODE *pPrev;
} RTLISTNODE;
```

```
typedef struct VUSBURBHDR
{
    /** List node for keeping the URB in the free list. */
    RTLISTNODE NdFree;
    /** Size of the data allocated for the URB (Only the variable part including the
     * HCI and TDs). */
    size_t cbAllocated;
    /** Age of the URB waiting on the list, if it is waiting for too long without being used
     * again it will be freed. */
    uint32_t cAge;
#ifdef HC_ARCH_BITS == 64
    uint32_t u32Alignment0;
#endif
    /** The embedded URB. */
    VUSBURB Urb;
} VUSBURBHDR;
```

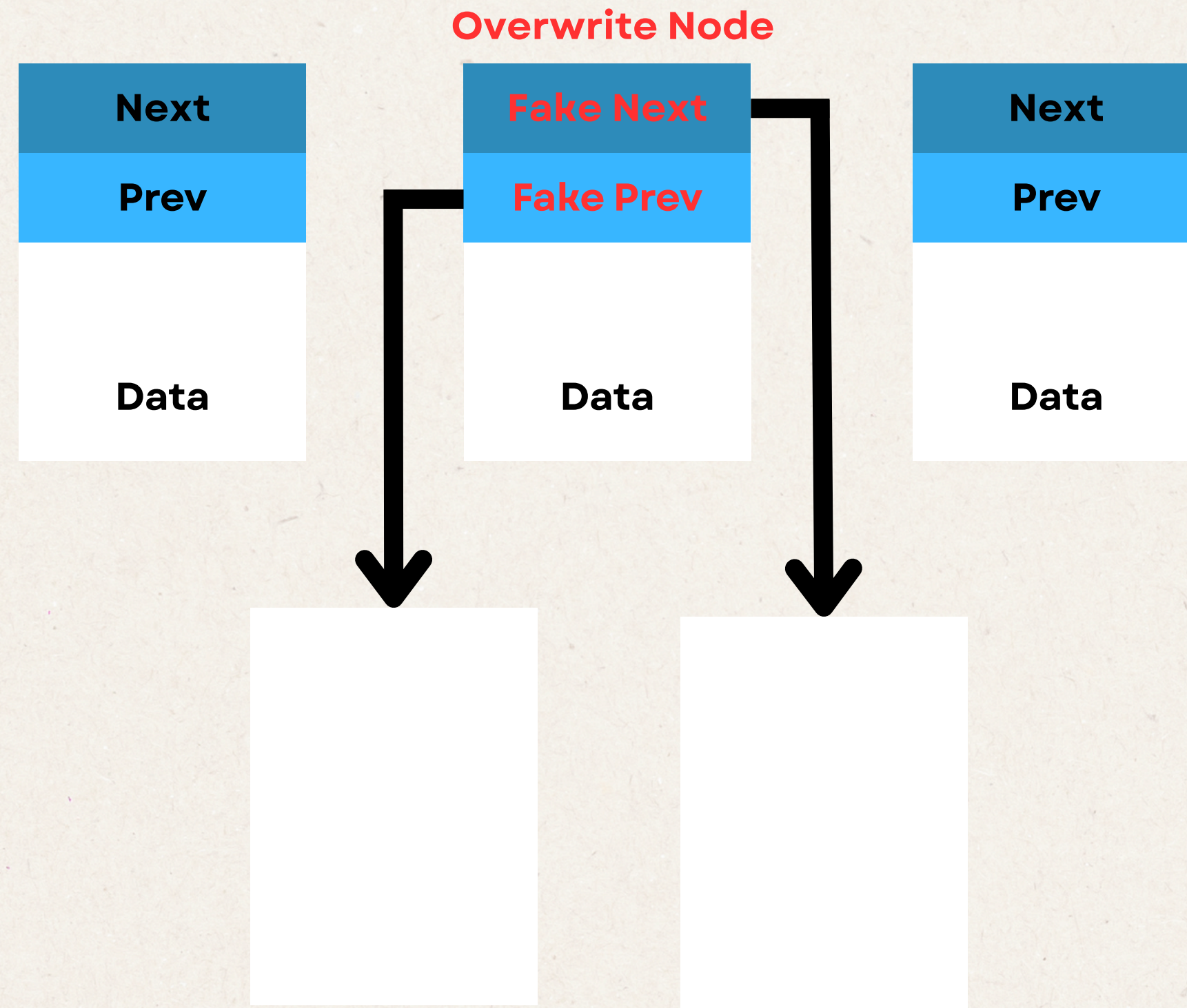

EXPLOIT

VGA-SVGA OOB R/W Exploit - URB



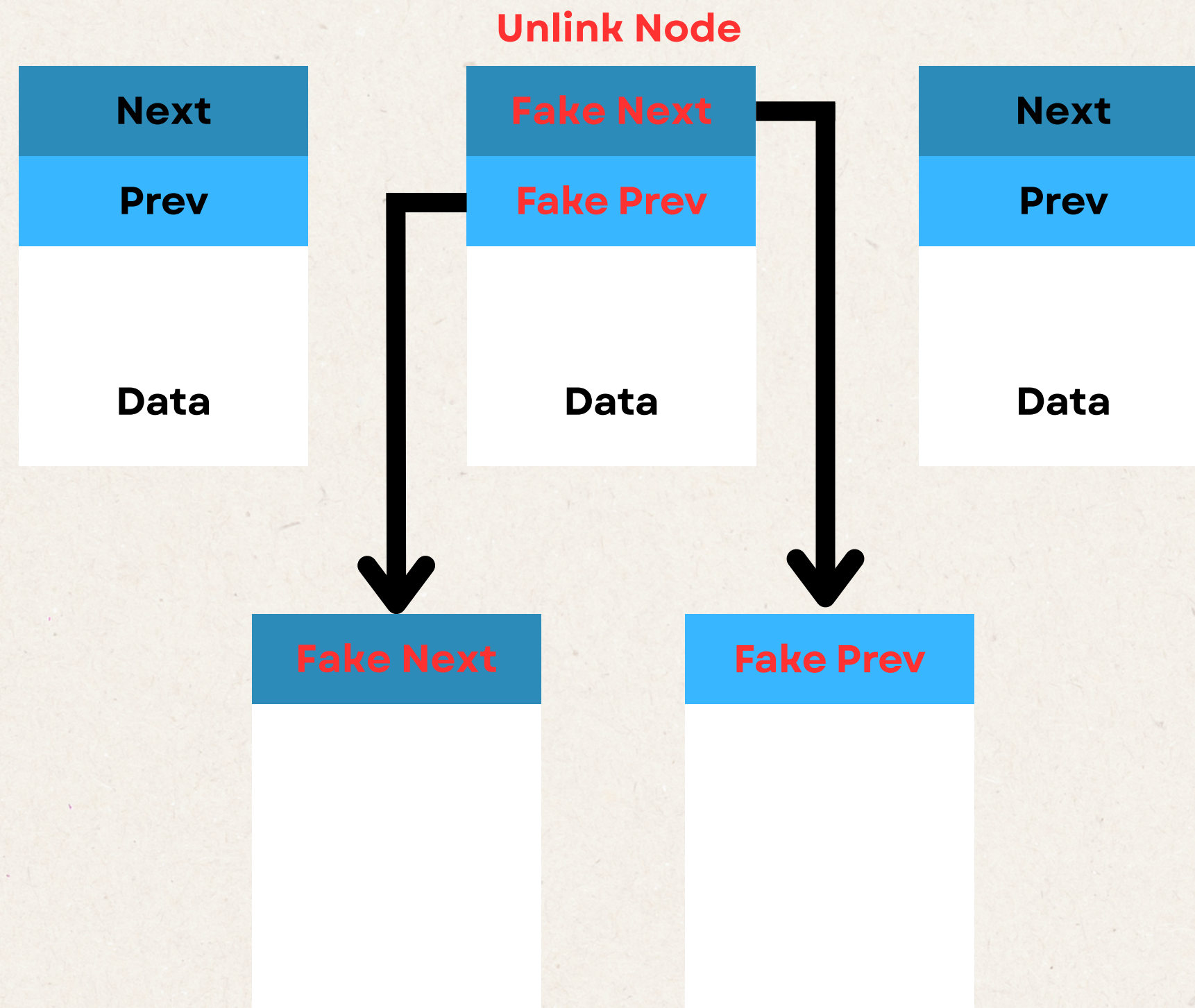
EXPLOIT

VGA-SVGA OOB R/W Exploit - URB



EXPLOIT

VGA-SVGA OOB R/W Exploit - URB



EXPLOIT

VGA-SVGA OOB R/W Exploit - URB

```
typedef struct VUSBROOTHUB
{
    /** Pointer to the driver instance. */
    PPDMDRVINS          pDrvIns;
    /** Pointer to the root hub port interface we're attached to. */
    PVUSBIROOTHUBPORT   pIRhPort;
    /** Connector interface exposed upwards. */
    VUSBIROOTHUBCONNECTOR IRhConnector;
    /** Critical section protecting the device arrays. */
    RTCRITSECT           CritSectDevices;
    /** Array of pointers to USB devices indexed by the port the device is on. */
    PVUSBDEV             apDevByPort[VUSB_DEVICES_MAX];
    /** Array of pointers to USB devices indexed by the address assigned. */
    PVUSBDEV             apDevByAddr[VUSB_DEVICES_MAX];
}
```

**/* overwrite apDevbyAddr[0x10]
apDevbyAddr[0x10] = vram+0x8000000 */**

EXPLOIT

VGA-SVGA OOB R/W Exploit - URB

```
typedef struct VUSBDEV
{
    ...

    /** Pipe/direction -> endpoint descriptor mapping */
    VUSBPIPE          aPipes[VUSB_PIPE_MAX];

    /** ... */

    /** Pointer to the extra state data required to run a control pipe */
    PVUSBCTRLEXTRA     pCtrl;
}

typedef struct vusb_ctrl_extra
{
    ...

    /** Current DATA pointer.
     * This starts at pMsg + 1 and is incremented as we read/write data. */
    uint8_t            *pbCur;
}
```

VUSBCTRLEXTRA's pbCur points to the next read/write address Controlling pbCur enables AAR & AAW through control pipe

```
static int vusbUrbSubmitCtrl(PVUSBURB pUrb)
{
    #ifdef LOG_ENABLED
        vusbUrbTrace(pUrb, "vusbUrbSubmitCtrl", false);
    #endif

    PVUSBDEV          pDev = pUrb->pVUsb->pDev;
    PVUSBPIPE          pPipe = &pDev->aPipes[pUrb->EndPt];

    RTCritSectEnter(&pPipe->CritSectCtrl);
    PVUSBCTRLEXTRA     pExtra = pPipe->pCtrl;

    ...

    memcpy(pUrb->abData, pExtra->pbCur, cbRead);
}
```


EXPLOIT

VGA-SVGA OOB R/W Exploit - Code Execution

```
/**
 * The instance data of a root hub driver.
 *
 * This extends the generic VUSB hub.
 *
 * @implements VUSBIROOTHUBCONNECTOR
 */
typedef struct VUSBR00THUB
{
    ...
    /** Sniffer instance for the root hub. */
    VUSBSNIFFER          hSniffer;
}
```

**When hSniffer is not NULL,
VUSBSnifferRecordEvent gets called**

```
/** @interface_method_impl{VUSBIROOTHUBCONNECTOR,pfnSubmitUrb} */
static DECLCALLBACK(int) vusbRhSubmitUrb(PVUSBIROOTHUBCONNECTOR pInterface, PVUSBURB pUrb, PPDMLLED pLed)
{
    ...

    /* If there is a sniffer on the roothub record the URB there. */
    if (pRh->hSniffer != VUSBSNIFFER_NIL)
    {
        int rc = VUSBSnifferRecordEvent(pRh->hSniffer, pUrb, VUSBSNIFFEREVENT_SUBMIT);
        if (RT_FAILURE(rc))
            LogRel(("VUSB: Capturing URB submit event on the root hub failed with %Rrc\n", rc));
    }
}
```


EXPLOIT

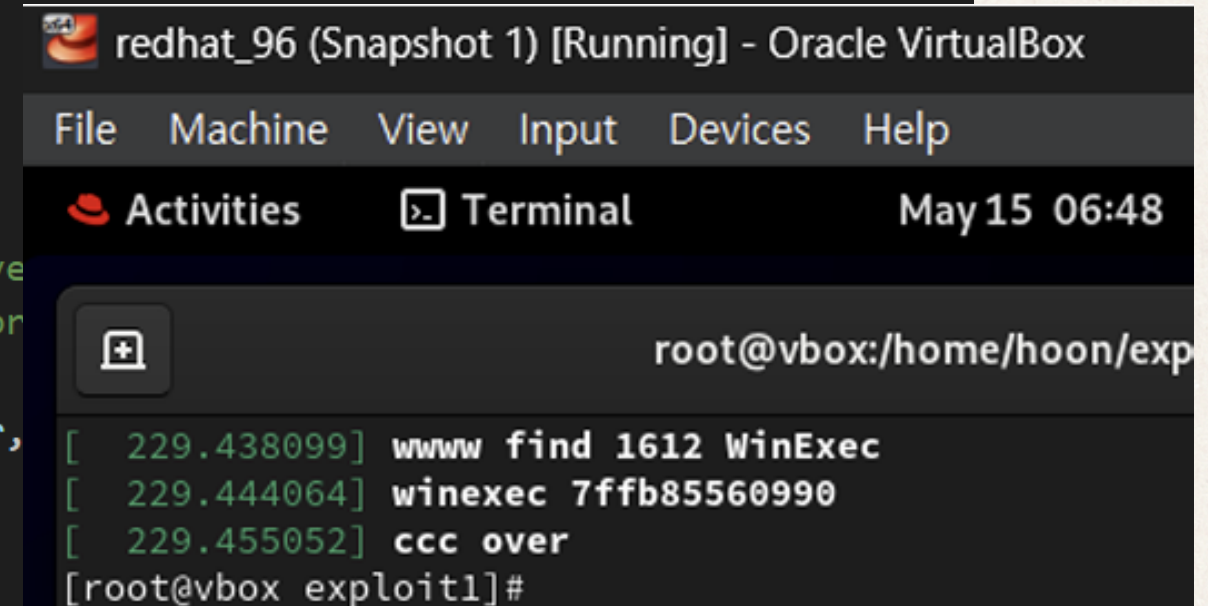
VGA-SVGA OOB R/W Exploit - Code Execution

**Overwrite sniffer with guest-accessible address
and set pfnRecordEvent to target function → arbitrary code execution**

```
/**
 * Records an VUSB event.
 *
 * @returns VBox status code.
 * @param hSniffer      The sniffer instance.
 * @param pUrb          The URB triggering the event.
 * @param enmEvent      The type of event to record.
 */
DECLHIDDEN(int) VUSBSnifferRecordEvent(VUSBSNIFFER hSniffer,
{
    int rc = VINF_SUCCESS;
    PVUSBSNIFFERINT pThis = hSniffer;

    /* Write the packet to the capture file. */
    rc = RTSemFastMutexRequest(pThis->hMtx);
    if (RT_SUCCESS(rc))
    {
        rc = pThis->pFmt->pfnRecordEvent((PVUSBSNIFFERFMTINT)&pThis->abFmt[0], pUrb, enmEvent);
        RTSemFastMutexRelease(pThis->hMtx);
    }

    return rc;
}
```



redhat_96 (Snapshot 1) [Running] - Oracle VirtualBox

File Machine View Input Devices Help

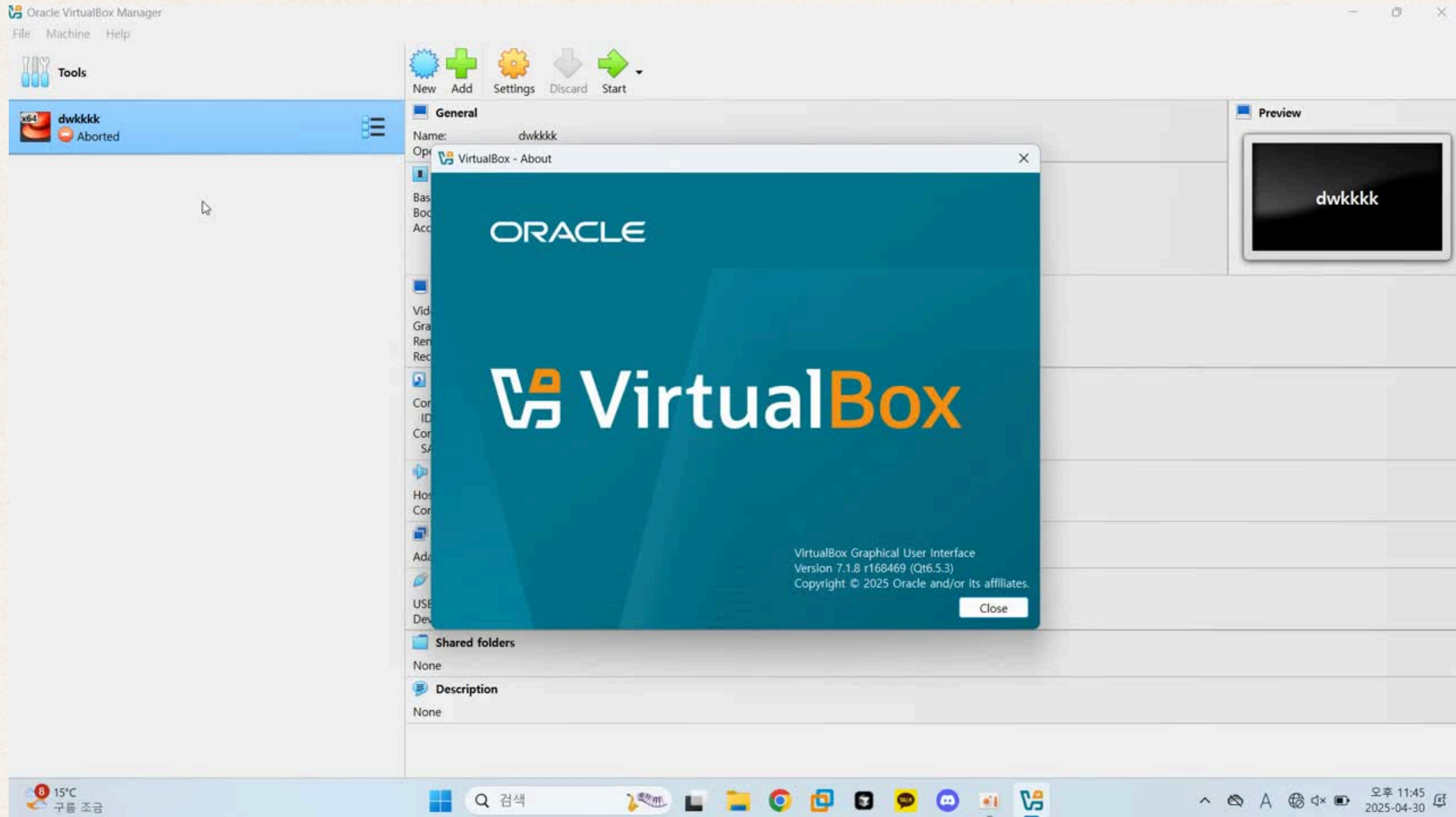
Activities Terminal May 15 06:48

root@vbox:/home/hoon/exploit1#

```
[ 229.438099] www find 1612 WinExec
[ 229.444064] winexec 7ffb85560990
[ 229.455052] ccc over
[root@vbox exploit1]#
```


DEMO

03/07



CONCLUSION

- Due to CFI protection, JOP and ROP remain difficult, necessitating better CFI bypass methods for sophisticated attacks
- Complex virtualization device implementation continues harboring persistent bugs.
- Rookies fear complexity but succeed with proper guidance
Individual growth strengthens entire security community



Thank you

E-mail creds1337@gmail.com
