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

Presented By: Debasish Mandal & Yakun Zhang

#BHEU / @BLACKHATEVENTS
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/](http://www.debasish.in/)
- [https://securingtomorrow.mcafee.com/author/debasish-mandal/](https://securingtomorrow.mcafee.com/author/debasish-mandal/)
- [@debasishm89](https://twitter.com/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.

#BHEU / @BLACKHATEVENTS
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(..);`
VM Backdoor

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

```assembly
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](https://sites.google.com/site/chitchatvmback/backdoor)
- In vmtools, `vmtools!Backdoor()` function takes care of this.

#BHEU / @BLACKHATEVENTS
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`.

```
#BHEU / @BLACKHATEVENTS
```
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.
Using vmtools.dll API(s) we can send and receive RPC packets from guest to host.

```python
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(spc_cmd+spc_pkt)), addressof(result), addressof(result_len))
            if res_code:
                print('[+] RpcOut_send() successful with code', hex(res_code))
                req_recv = vmtools_dll.RpcOut_Message_Receive(channel, addressof(result), addressof(result_len))
                if req_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('[+] 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.

<table>
<thead>
<tr>
<th>VMware Advisory</th>
<th>Unpatched Version</th>
<th>Patched Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>VMSA-2016-0019</td>
<td>12.5.1</td>
<td>12.5.2</td>
</tr>
<tr>
<td>VMSA-2017-0005</td>
<td>12.5.3</td>
<td>12.5.4</td>
</tr>
</tbody>
</table>
RPC Bug 1: OOB in Drag and Drop

```
typedef struct DnDCPMgHdrV4 {
    DnDCPMgHdrV4 hdr;
    uint32 addrId;
    uint8 *binary;
} DnDCPMgV4;
```

```
struct DnDCPMgHdrV4 {
    uint32 cmd; /* Dn/CP message command. */
    uint32 type; /* Dn/CP message type. */
    uint32 src; /* Message sender. */
    uint32 sessionID; /* Dn/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_144D0D898(*(_DWORD *)(a1 + 44));
payloadSize = *(DWORD *)(v4 + 0x34);
*(DWORD *)(v4 + 0x40) = new_allocation;
memcpy(new_allocation, (const void *)rpc_pkt + 0x80), payloadSize);
```
Achieving OOB Write

- We have to send at least two RPC packets to the host with the same session ID.
- 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.

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

---

#BHEU / @BLACKHATEVENTS
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;
};

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](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.
VMware Virtual Printer

Guest
VM
vmware-vmx.exe

Host
COM1
Named Pipes
vprintproxy.exe
tpview.dll

Print preview

#BHEU / @BLACKHATEVENTS
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.
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:

```c
typedef enum
{
    EMR_HEADER = 0x00000001,
    EMR_POLYBEZIER = 0x00000002,
    EMR_POLYGON = 0x00000003,
    EMR_POLYLINE = 0x00000004,
    EMR_POLYBEZIERTO = 0x00000005,
    EMR_POLYLINEETO = 0x00000006,
    EMR_POLYPOLYLINE = 0x00000007,
    EMR_POLYPOLYGON = 0x00000008,
    EMR_SETWINDOWEXT = 0x00000009,
    EMR_SETWINDOWORGEX = 0x0000000A,
    EMR_SETVIEWPORTEXT = 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.

<table>
<thead>
<tr>
<th>VMware Advisory</th>
<th>Unpatched Version</th>
<th>Patched Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>VMSA-2016-0014</td>
<td>12.1.1</td>
<td>12.5.0</td>
</tr>
</tbody>
</table>
Double Free in EMR_SMALLTEXOUTW (CVE-2016-7082)

- Present in tpview.dll EMR_SMALLTEXOUTW 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".

```c
v12 = *((DWORD *)v8 + 5);
if (v12 & 0x100 & !(v12 & 4) 1) *(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 )
    ExtTextOut(hdc, v13, *((DWORD *)v8 + 3), a6 & v12, 0, v16, v17, 0);
else
    ExtTextOut(hdc, v13, *((DWORD *)v8 + 3), a6 & v12, 0, LPCWSTR)v16, v17, 0);
free(v8);
v14 = 0;
} free(v8);
return v14;
```
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.

#BHEU / @BLACKHATEVENTS
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
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().
Out of Bounds memset() in TrueTypeFont Embedded EMF (CVE-2016-7083)

No check for v7.

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

memset() triggers crash.

memset(eax, 0, 0xffffffff)
Added necessary checks before `memset()`. 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.
- \((0xB0 - 0x8 - 0x8) = 0xA0 = 0x28 \times 4.\)
- The loop count from 0x0 to 0x27. Should be less than 0x28.
- No check for the loop count.
- OOB write to next heap entry.

#BHEU / @BLACKHATEVENTS
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].

```c
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.

#BHEU / @BLACKHATEVENTS
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
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.
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, <scanlines> */

#define SVGA_CMD_DEFINE_PIXMAP 6
/* FIFO layout: Pixmap ID, Width, Height, Line $, scanline */

#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_SETZ RANGE 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
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
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;
}
```

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.xyyy
mov o0.zw, l(0.0,0.0,1.000000)
mov o1.xyzw, v1.xyzw
ret
// Approximately 4 instruction slots used
```
Shader inside VMware Workstation

Guest Operating System (User)
- d3d11.dll
- shader bytecode
- App.
- vm3dum_**.dll

Guest OS (Kernel)
- SVGA 3D CMD Buffer
- VMware Virtual SVGA Device
- VMWare Virtual SVGA Device
- vmware_vmx.exe
- shader translation to Host

Host (OS)
- Host GPU Driver
- Guest shader to Host shader translation

---

#BHEU / @BLACKHATEVENTS
Passing Shader bytecode from guest to host via ‘SVGA3D’ Protocol

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

SVGA3dCmdDefineShader *cmd;

uint32 cid = cid;
uint32 shid = shid;
uint32 type = type;

memcpy(&cmd[1], bytecode, bytecodeLen);
SVGA_PIFOCommitAll();

typedef struct {
    uint32 id;
    uint32 shid;
    SVGA3dShaderType type;
    /* Followed by variable number of DWORDs for shader bytecode */
} PACKED
SVGA3dCmdHeader;

void *SVGA3D_FIFOReserve(uint32 cmd, uint32 cmdSize)
{
    SVGA3dCmdHeader *header;
    header = SVGA3D_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. 😊

```
goto LABEL_65;
```

<table>
<thead>
<tr>
<th>Case 0x62:</th>
<th>goto LABEL_65;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case 0x64:</td>
<td>goto LABEL_65;</td>
</tr>
</tbody>
</table>

```
v22 = 1;
```

```
*(_DWORD *)(u28 + 0x10) = (u32 >> 11) & 0xFF;
```

<table>
<thead>
<tr>
<th>Case 0x5E:</th>
<th>goto LABEL_65;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case 0x58:</td>
<td>goto LABEL_65;</td>
</tr>
</tbody>
</table>

```
v40 -= *(_DWORD *)v18;
```

```
v18 += 4;
```

<table>
<thead>
<tr>
<th>Case 0x58:</th>
<th>goto LABEL_65;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case 0x58:</td>
<td>goto LABEL_65;</td>
</tr>
</tbody>
</table>

```
v22 = 1;
```

```
*(_DWORD *)(u28 + 0x10) = (unsigned __int16)v30 >> 11;
```

<table>
<thead>
<tr>
<th>Case 0x58:</th>
<th>goto LABEL_65;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case 0x58:</td>
<td>goto LABEL_65;</td>
</tr>
</tbody>
</table>

```
v22 = 1;
```

```
*(_DWORD *)(u28 + 0x10) = (u32 >> 11) & 0xFF;
```

<table>
<thead>
<tr>
<th>Case 0x58:</th>
<th>goto LABEL_65;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case 0x58:</td>
<td>goto LABEL_65;</td>
</tr>
</tbody>
</table>

```
v40 -= *(_DWORD *)v18;
```

```
v18 += 4;
```

<table>
<thead>
<tr>
<th>Case 0x58:</th>
<th>goto LABEL_65;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case 0x58:</td>
<td>goto LABEL_65;</td>
</tr>
<tr>
<td>Case 0x41:</td>
<td>goto LABEL_65;</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>VMware Advisory</th>
<th>Patched Version</th>
<th>Unpatched Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>VMSA-2016-0019</td>
<td>12.5.2</td>
<td>12.5.3</td>
</tr>
<tr>
<td>VMSA-2017-0006</td>
<td>12.5.4</td>
<td>12.5.5</td>
</tr>
<tr>
<td>VMSA-2017-0015.2</td>
<td>12.5.6</td>
<td>12.5.7</td>
</tr>
</tbody>
</table>
SVGA Patch 1 (Workstation 12.5.4 -> 12.5.5):

0x69 is opcode for `dcl_indexableTemp` Instruction
Heap OOB Write

```
sub_14B24B2D0 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
    ret
sub_14B24B2D0 endp
```

OOB Write

```
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 0be58000
End Address: 00000000 0b6ec000
Region Size: 00000000 00034000
State: PMEM
Protect: 00000004 PA
Type: 00020000 MEM
Allocation Base: 00000000 0af70000
Allocation Protect: 00000004 PA
More info: heap owning the address
More info: heap segment
More info: heap entry containing
```
Demo: SVGA Memory Corruption
SM4 ‘dcl_constantbuffer’ Instruction Parsing (0x59) Bug

Fixed In 12.5.5
Other SVGA Issues fixed in 12.5.5

```c
void __fastcall sub_1402488800(__int64 a1, const void *a2, unsigned int a3)
{
    unsigned int v2; // 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;
}

void __fastcall sub_140248A010(__int64 a1, const void *a2, unsigned int a3)
{
    unsigned int v3; // ebx@1
    __int64 v4; // rdi@1
    void *result; // rax@2

    v3 = a3;
    v4 = a1;

    if ( a3 > 0x10000 )
        sub_140008550("VERIFY %s: %d\n", "bora\nmsk\lib\ stateEFP\ unqiEmit.c", 1119164);
    result = memcpy((void *)(a1 + 0x19EF4), a2, 4i64 * a3);
    *(DWORD *)(v4 + 0x1DEF4) = v3;
    return result;
}
```

SM4
dcl_immediateConstantBuffer

Security Patch
Possible Security Issue fixed in SM1 ‘op_call’ instruction parser in version 12.5.3?

```c
char __fastcall sub_1402D1D0(__int64 a1, __int64 a2)
{
    __int64 v2; // rd181
    __int64 v3; // rd81
    __int64 v5; // rwx81
    __int64 v6; // rx85

    v2 = a1;
    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;
    }

    IODWORD(v5) = sub_1401FB930(65545164, 1164, 16164);
    if ( v5 )
    {
        return 0;
    }

    *(DWORD *)v5 = *(DWORD *)(v2 + 12) & 0x7ff;
    *(DWORD *)v5 + 8 = *(DWORD *)(v3 + 8);
    v6 = *(DWORD *)(v3 + 8) - v5;
    sub_1402D7630(v6, v5 + 4);
    return 1;
}
```

#BHEU / @BLACKHATEVENTS
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.
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

#BHEU / @BLACKHATEVENTS
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](mailto:Debasish_Mandal@McAfee.com))
  - Yakun Zhang ([Yakun_Zhang@McAfee.com](mailto:Yakun_Zhang@McAfee.com))