# Direct X – Direct way to Microsoft Windows Kernel

Nikita Tarakanov

Zeronights, Moscow

26th of November

# Agenda

- Windows Display Driver Model
- D3DKMTEscape reverse engineering
- D3DKMTEscape fuzzing & demo
- D3DKMTEscape Crash analysis
- Recommendations
- Other stuff
- Q&A

## Windows Display Driver Model

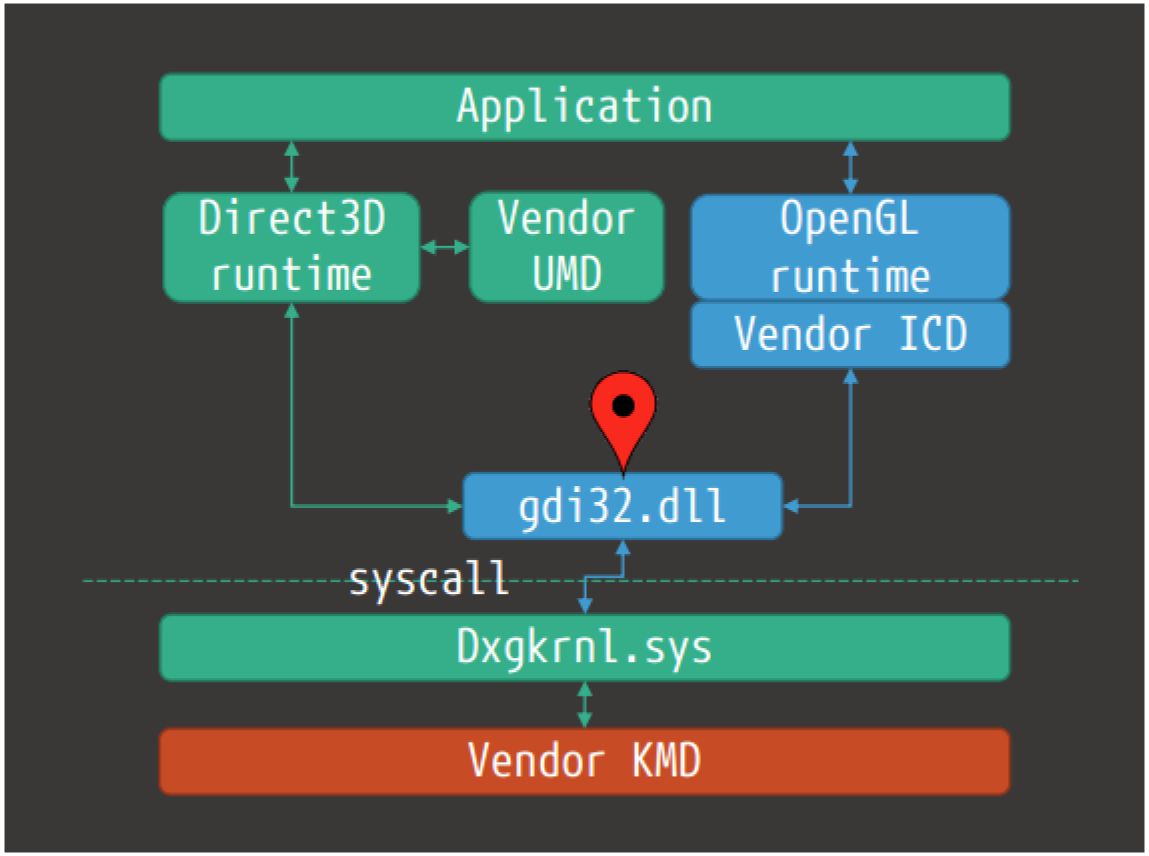
# Windows Display Driver Model

- WDDM is the graphics architecture for video card
- drivers on windows vista+
- Dxgkrnl responsibilities:
  - Virtualization of video memory (process
  - isolation)
  - Scheduling of graphics workloads
  - Fault tolerance (Trigger TDR)

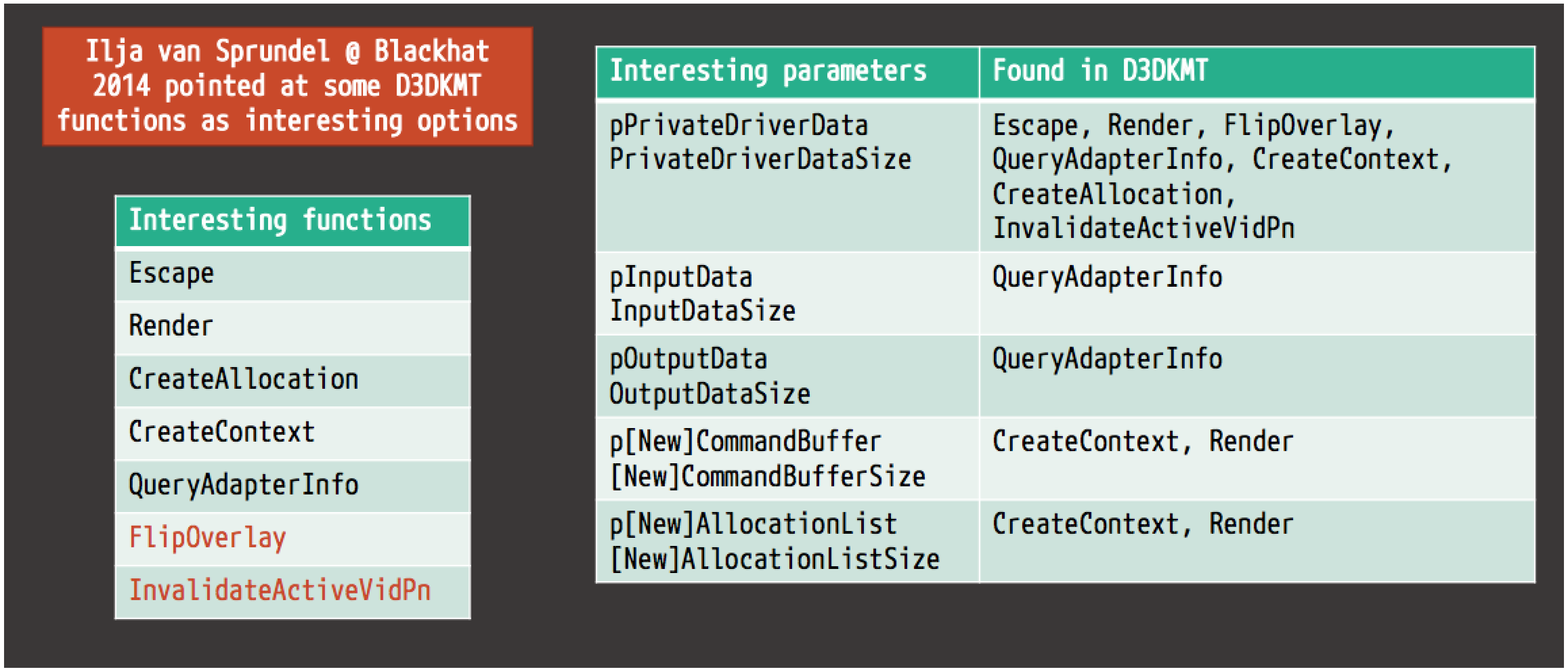
# WDDM vendor-specific functionality

- Thin layer between gdi32.dll and dxgkrnl.sys, mostly just proxy for syscall.
- Depending on how you test you might end up using d3d functionality to get valid resources before call anyway.
  - Resources
  - Handles
  - Contexts
  - Surfaces

# WDDM vendor-specific functionality



# D3DKMT attack surface

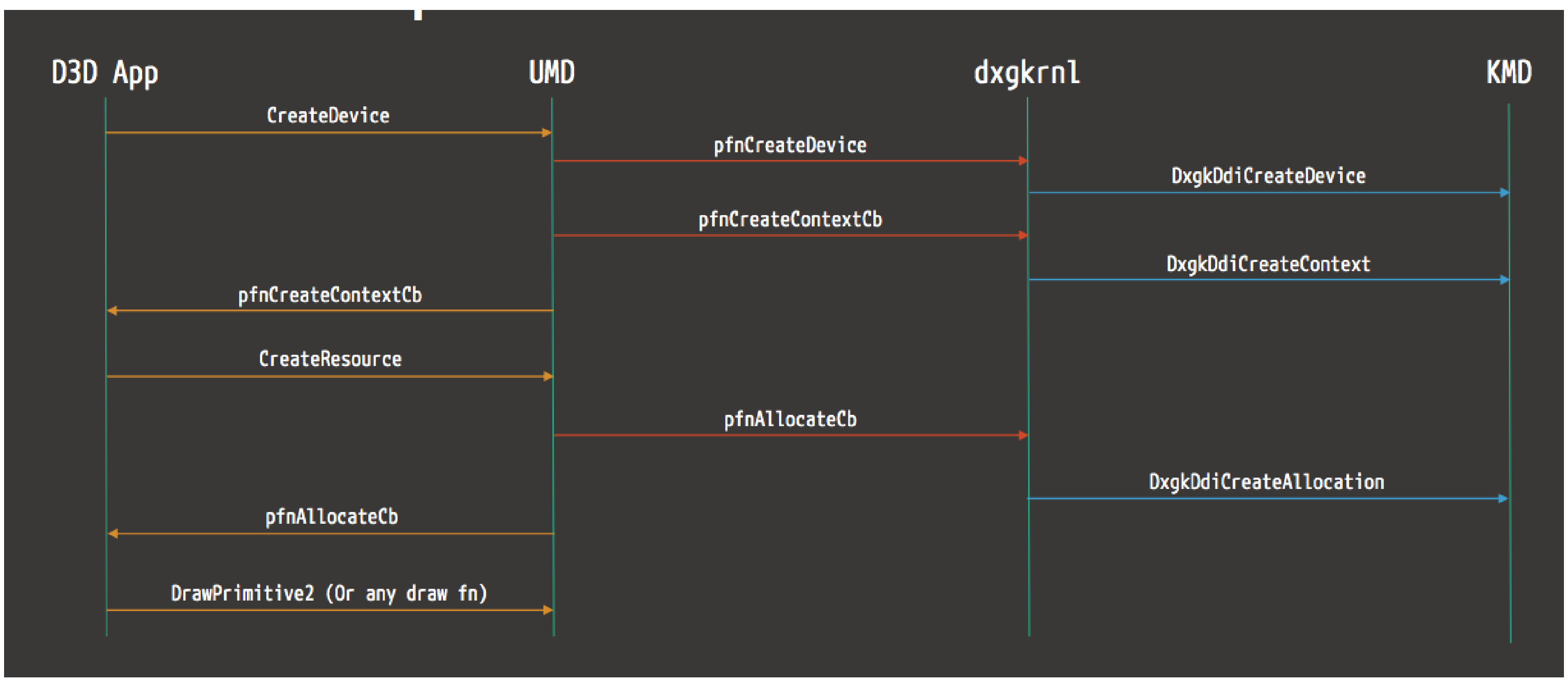


Ilja van Sprundel @ Blackhat 2014 pointed at some D3DKMT functions as interesting options

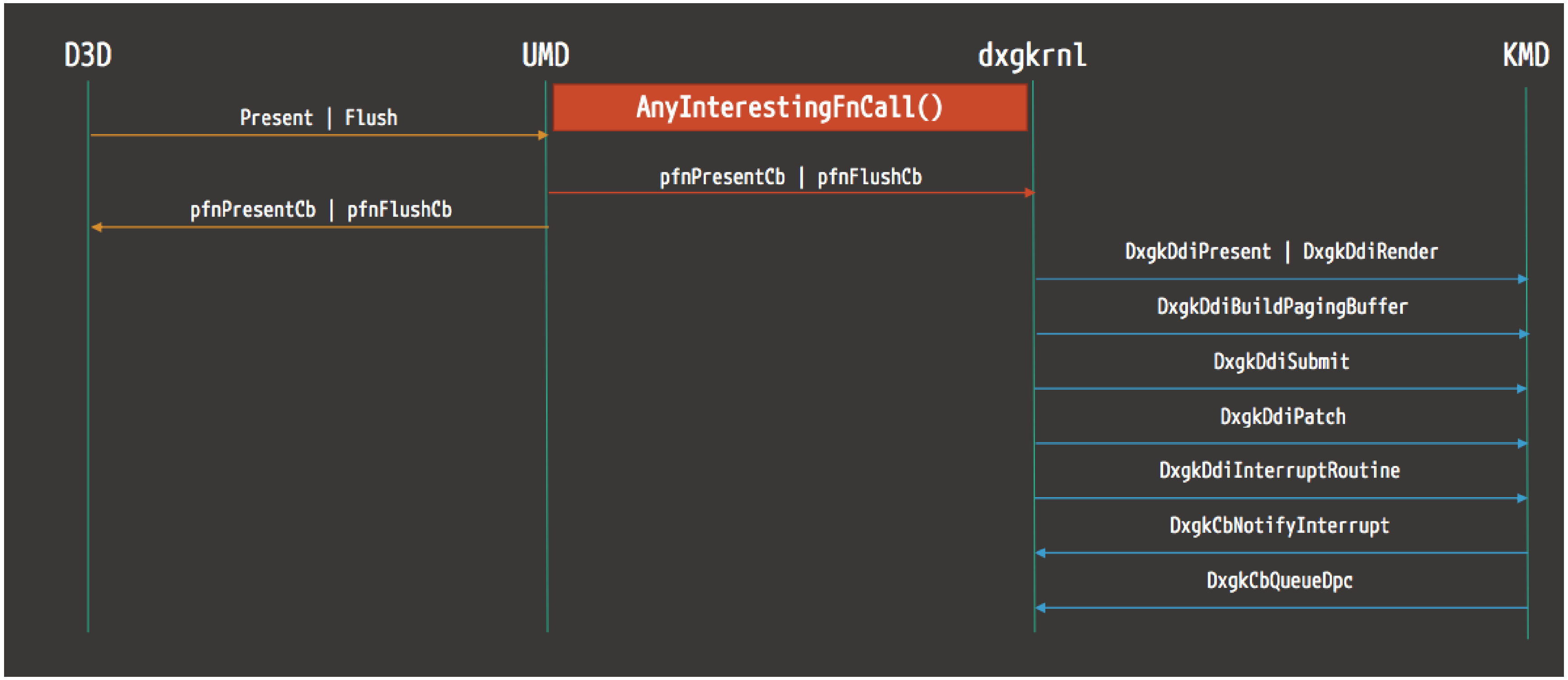
| Interesting functions |
|---|
| Escape |
| Render |
| CreateAllocation |
| CreateContext |
| QueryAdapterInfo |
| FlipOverlay |
| InvalidateActiveVidPn |

| Interesting parameters | Found in D3DKMT |
|---|---|
| pPrivateDriverData<br>PrivateDriverDataSize | Escape, Render, FlipOverlay, QueryAdapterInfo, CreateContext, CreateAllocation, InvalidateActiveVidPn |
| pInputData<br>InputDataSize | QueryAdapterInfo |
| pOutputData<br>OutputDataSize | QueryAdapterInfo |
| p[New]CommandBuffer<br>[New]CommandBufferSize | CreateContext, Render |
| p[New]AllocationList<br>[New]AllocationListSize | CreateContext, Render |

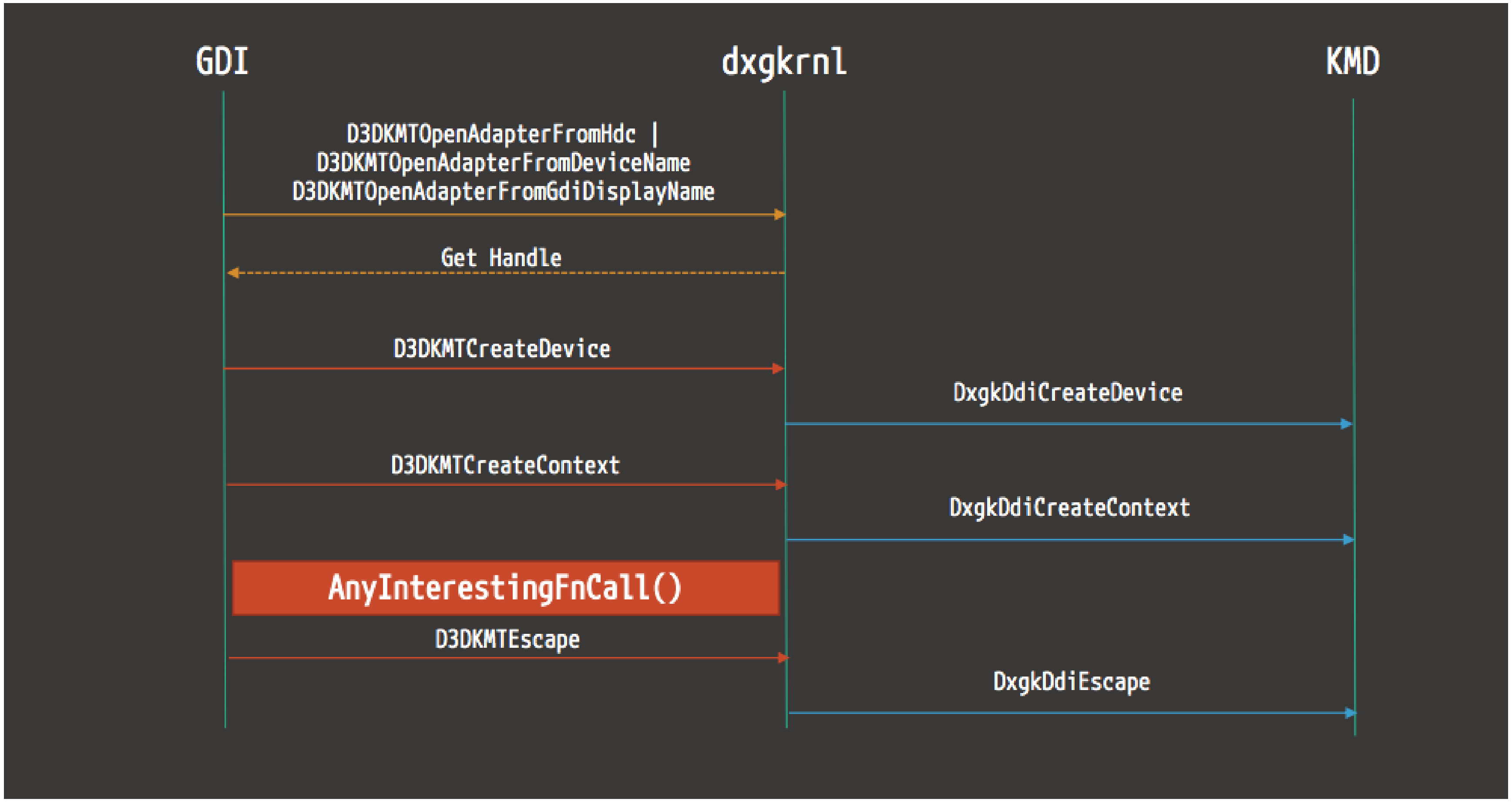
# Render operation flow from D3D #1



# Render operation flow from D3D #2



# Escape operation flow from GDI



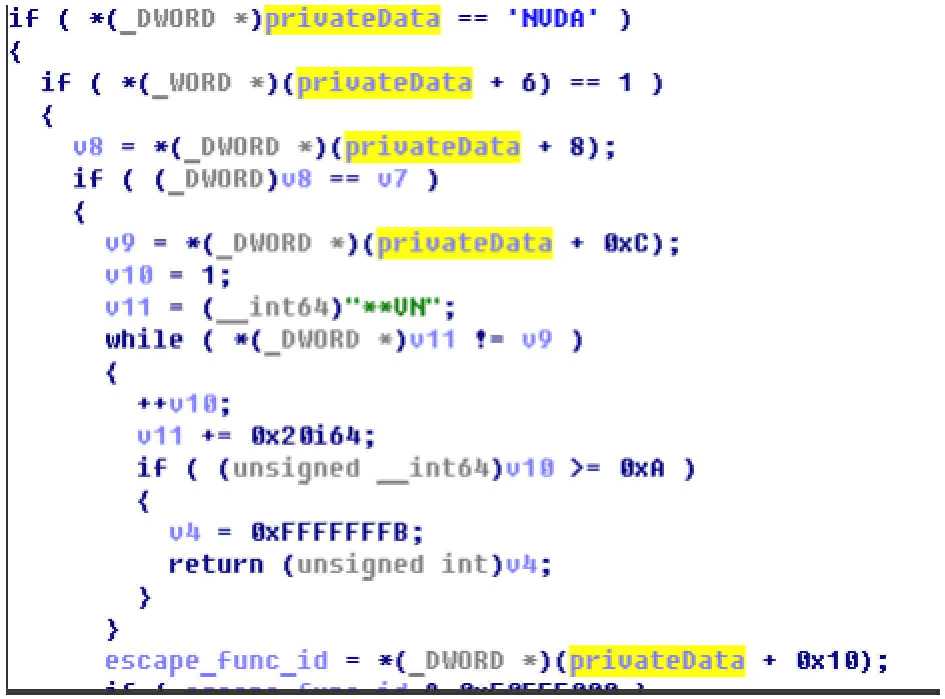
## Dxgkrnl checks and operations

- dxgkrnl copies buffer to kernel space and provides kernel-space pointer to vendor KMD
- dxgkrnl CANNOT check structure of internal buffer, so the driver has to perform the above checks
- Structure of such internal buffer/parameter is created in vendor UMD components

## D3DKMTEscape reverse engineering

## NVIDIA kernel escape

# NVIDIA pPrivateData format



```
if ( *(_DWORD *)privateData == 'NVDA' )
{
  if ( *(_WORD *)(privateData + 6) == 1 )
  {
    v8 = *(_DWORD *)(privateData + 8);
    if ( (_DWORD)v8 == v7 )
    {
      v9 = *(_DWORD *)(privateData + 0xC);
      v10 = 1;
      v11 = (__int64)"**VN";
      while ( *(_DWORD *)v11 != v9 )
      {
        ++v10;
        v11 += 0x20i64;
        if ( (unsigned __int64)v10 >= 0xA )
        {
          v4 = 0xFFFFFFFB;
          return (unsigned int)v4;
        }
      }
      escape_func_id = *(_DWORD *)(privateData + 0x10);
```

# NVIDIA pPrivateData format

- typedef struct {

```
DWORD nvidiaMagicDWORD;
DWORD dword2;
DWORD privateDriverDataSize;
DWORD escapeAction;
DWORD escapeFuncId;
UINT Data[0];
} NVIDIA_PRIVATE_DRIVER_DATA,
*PNVIDIA_PRIVATE_DRIVER_DATA;
```

# NVIDIA Magic DWORD

- #define NVIDIA_ESCAPE_ID 0x4e564441 // NVDA
- #define NVIDIA_ESCAPE_ACTION_DX 0x4e564458 // NVDX
- #define NVIDIA_ESCAPE_ACTION_STAR 0x4e562a2a // NV**
- #define NVIDIA_ESCAPE_ACTION_GL 0x4e56474c // NVGL
- #define NVIDIA_ESCAPE_ACTION_CP 0x4e564350 // NVCP

## NVIDIA Escape Function ID ranges

- There are two major ranges:
  - 0x1000000 - 0x1000122
  - 0x7000000- 0x70001A2

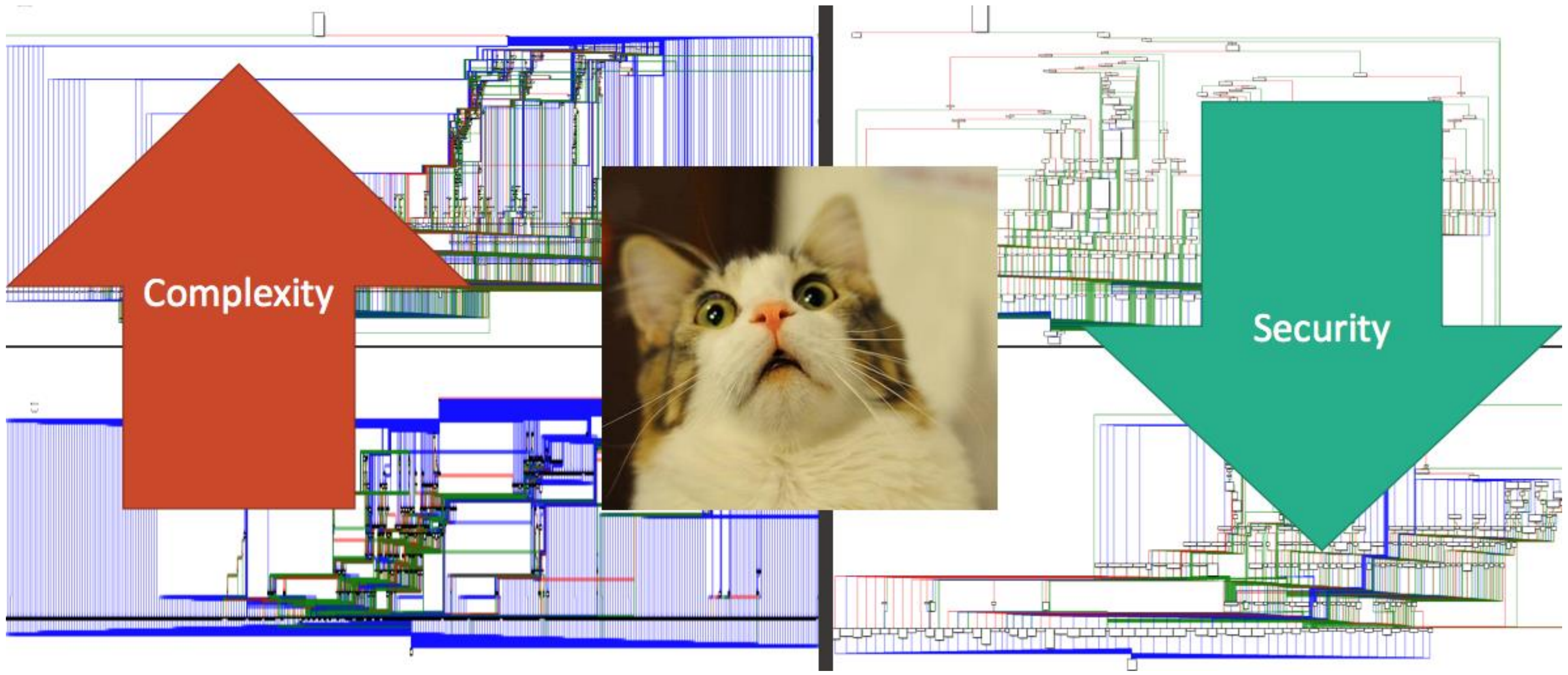
## ATI kernel Escape

```
• typedef struct {
  DWORD first_dword_const_0x80;
  DWORD
  second_dword_const_0x10000_or_0x10002;
  DWORD func_id;
  UINT Data[0];
  } ATI_PRIVATE_DRIVER_DATA,
  *PATI_PRIVATE_DRIVER_DATA;
```

## ATI Escape Function ID ranges

- Several ranges
- 0x01000000 - 0x0100XX03
- 0x02000000 - 0x0200XX00

# KMD DxgkDdiEscape handlers



## D3DKMTEscape fuzzing

# Adapter initialization

- typedef struct DRIVER_INFO {
- HDC hDC;
- D3DKMT_HANDLE hAdapter;
- D3DKMT_CREATEDEVICE device;
- D3DKMT_CREATECONTEXT context;
- } DRIVER_INFO, *PDRIVER_INFO;

# Adapter initialization

- D3DKMT_OPENADAPTERFROMHDC oafh;
- // Get current adapter
- **pDriverInfo->hDC = GetDC(NULL);**
- // Get adapter from hDC
- oafh.hDc = pDriverInfo->hDC;
- **status = D3DKMTOpenAdapterFromHdc(&oafh);**
- pDriverInfo->hAdapter = oafh.hAdapter;
- // Create device
- pDriverInfo->device.hAdapter = pDriverInfo->hAdapter;
- **D3DKMTCreateDevice(&pDriverInfo->device);**
- // Create context
- pDriverInfo->context.NodeOrdinal = 0;
- pDriverInfo->context.hDevice = pDriverInfo-
- >device.hDevice;
- **D3DKMTCreateContext(&pDriverInfo->context);**

## DEMO

# Nvidia vulns

- escapeID 0x7000094
- nvlddmkm+0x20f50b:
- fffff800`78c8450b 41ff542420      call    qword ptr [r12+20h] ds:002b:00000000`00000020=????????????????

# Nvidia vulns

- STACK_TEXT:
- ffffd000`217c5e40 fffff800`78cf4561 : ffffe000`866097f0 00000000`00000000 ffffd000`217c68c0 ffffe000`87d49700 : nvlddmkm+0x20f50b
- ffffd000`217c5ec0 fffff800`78c1a6dc : 00000000`00000000 00000000`00000000 ffffd000`217c6280 ffffd000`217c6280 : nvlddmkm+0x27f561
- ffffd000`217c5fd0 fffff800`78c1a354 : ffffd000`217c6280 00000000`00000000 ffffd000`217c6280 00000000`00000008 : nvlddmkm+0x1a56dc
- ffffd000`217c6030 fffff800`78c1a11a : 00000000`c1d00161 fffff800`c1d00161 0000007c`00000000 0000007c`600e61e0 : nvlddmkm+0x1a5354
- ffffd000`217c60d0 fffff800`78bef326 : ffffd000`217c6150 00000000`c1d00161 ffffd000`217c6340 fffff802`24463df5 : nvlddmkm+0x1a511a
- ffffd000`217c6120 fffff800`78ab4b73 : ffffd000`217c6870 fffff800`78ad0053 00000000`c1d00161 00000000`00000803 : nvlddmkm+0x17a326
- ffffd000`217c61f0 fffff800`78ad05bc : ffffd000`217c6870 ffffd000`217c689c ffffd001`00000000 fffff6fb`7dbedd00 : nvlddmkm+0x3fb73
- ffffd000`217c6240 fffff800`78b18257 : ffffd000`217c63b8 fffff800`78f88470 ffffd000`217c6768 00000000`c000000d : nvlddmkm+0x5b5bc
- ffffd000`217c63a0 fffff800`78b20d59 : fffff800`78f88470 ffffd000`217c6489 00000000`00000000 00000000`00000000 : nvlddmkm+0xa3257
- ffffd000`217c63d0 fffff800`7916db6d : ffffe000`86ff6540 ffffd000`217c6559 ffffe000`86ff6540 ffffd000`217c6768 : nvlddmkm+0xabd59
- ffffd000`217c64f0 fffff800`7858c570 : 00000000`00000000 00000000`00000000 00000000`4e562a2a 00000000`07000094 : nvlddmkm!nvDumpConfig+0xf016d
- ffffd000`217c65c0 fffff800`78559a2d : ffffe000`86ff6540 ffffd000`217c6b80 00000000`00000000 00000000`00000000 : dxgkrnl!DXGADAPTER::DdiEscape+0x48
- ffffd000`217c65f0 fffff960`0029010f : 000000ac`220ada70 ffffe000`8891d4c0 000000ac`22110740 00000000`00000000 : dxgkrnl!DxgkEscape+0x57d
- ffffd000`217c6ab0 fffff802`245771b3 : 00000000`00000001 000000ac`22110000 ffffffff`ff676901 00000000`00000000 : win32k!NtGdiDdDDIEscape+0x53
- ffffd000`217c6b00 00007ff8`16ba14aa : 00007ff6`8d5d154d 00007ff6`8d5f1f20 00007ff6`8d7bb030 00007ff6`8d7bd600 : nt!KiSystemServiceCopyEnd+0x13
- 000000ac`220af708 00007ff6`8d5d154d : 00007ff6`8d5f1f20 00007ff6`8d7bb030 00007ff6`8d7bd600 00007ff6`8d7bd600 : GDI32!NtGdiDdDDIEscape+0xa

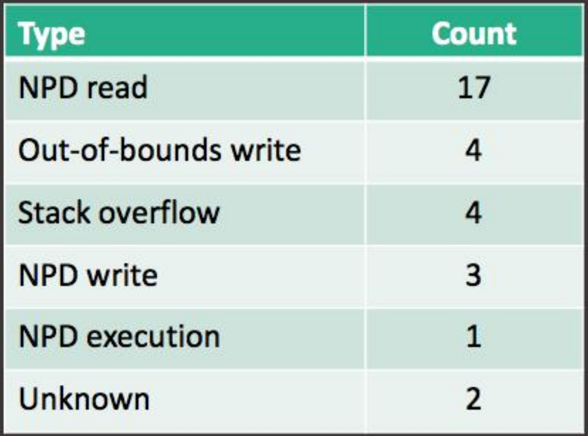
## Nvidia vulns

- escapeID 0x700000C
- nvlddmkm+0x551a7:
- fffff801`eb8e51a7 89848ddc050000  mov dword ptr [rbp+rcx*4+5DCh],eax ss:0018:ffffd000`2632a000=????????

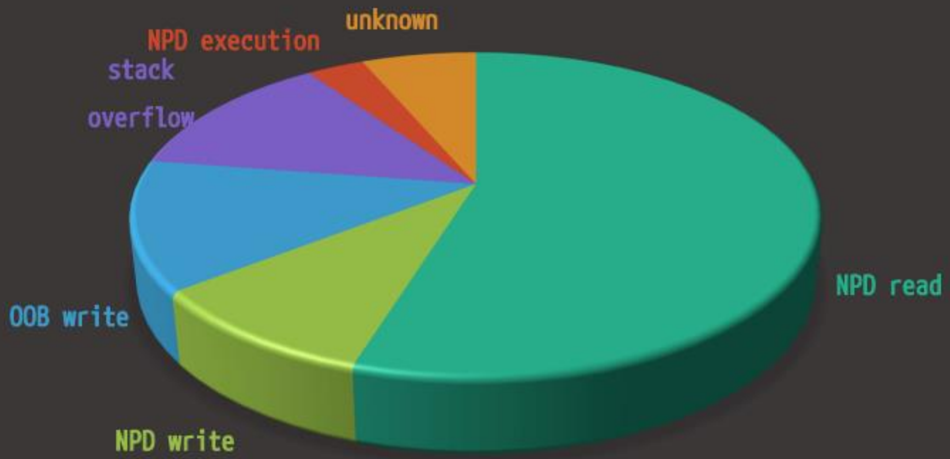
# Nvidia vulns

- ffffd000`26328830 00380033`00660062 : 00640061`00360035 00650034`00360033 0036005f`00350033 0039002e`0033002e : **nvlddmkm+0x551a7**
- ffffd000`26329240 00640061`00360035 : 00650034`00360033 0036005f`00350033 0039002e`0033002e 002e0030`00300036 : **0x00380033`00660062**
- ffffd000`26329248 00650034`00360033 : 0036005f`00350033 0039002e`0033002e 002e0030`00300036 00380033`00360031 : **0x00640061`00360035**
- ffffd000`26329250 0036005f`00350033 : 0039002e`0033002e 002e0030`00300036 00380033`00360031 006f006e`005f0034 : **0x00650034`00360033**
- ffffd000`26329258 0039002e`0033002e : 002e0030`00300036 00380033`00360031 006f006e`005f0034 0062005f`0065006e : 0x0036005f`00350033
- ffffd000`26329260 002e0030`00300036 : 00380033`00360031 006f006e`005f0034 0062005f`0065006e 00340032`00380038 : 0x0039002e`0033002e
- ffffd000`26329268 00380033`00360031 : 006f006e`005f0034 0062005f`0065006e 00340032`00380038 00650061`00380064 : 0x002e0030`00300036
- ffffd000`26329270 006f006e`005f0034 : 0062005f`0065006e 00340032`00380038 00650061`00380064 00350037`00370034 : 0x00380033`00360031
- ffffd000`26329278 0062005f`0065006e : 00340032`00380038 00650061`00380064 00350037`00370034 002e0039`00650062 : 0x006f006e`005f0034

## DxgkDdiEscape crashes (anon vendor )



| Type | Count |
|---|---|
| NPD read | 17 |
| Out-of-bounds write | 4 |
| Stack overflow | 4 |
| NPD write | 3 |
| NPD execution | 1 |
| Unknown | 2 |



## Recommendations

- Check for user-supplied pointers in pPrivateDriverData ProbeFor[Read|Write]
- Copy buffers pointed to by pPrivateDriverData internal members before using them
- Avoid using kernel pointers in your pPrivateDriverData, use handles instead

## What about other buffers?

## D3DKMT*->pCommandBuffer

- On WDDM 1.x
  - Command buffers are used to instruct GPU on how to do its work
  - Generated in userland
  - Passed to kernel driver for parsing and validation
  - Kernel driver sends commands to GPU
  - **Proprietary and vendor specific**

## D3DKMT*->pCommandBuffer

- The following are operations available in command buffers:
  - Manage allocations
  - Paging
  - Context management
  - Execution flow

## Not just Direct X

- CVE-2014-0972

  The kgsl graphics driver for the Linux kernel 3.x, as used in Qualcomm Innovation Center (QuIC) Android contributions for MSM devices and other products, **does not properly prevent write access to IOMMU context registers**, which allows local users to select a custom page table, and consequently **write to arbitrary memory locations**, by using a crafted GPU command stream to modify the contents of a certain register.

## Q&A

- @Ntarakanov
- BIG Thanks to Rodrigo Axel Monroy and Rodrigo Branco for help to make that talk happen
- Thanks!