Direct3D Hooking

by NullBr4in
NullBr4in@gmail.com
2012-1-20

※ 목차

1. 개요

Direct3D(이하 D3D) Hooking은 이전부터 온라인 게임분야, 특히 FPS 게임에서 Wall Hack 이라는 것으로 게임 진행에서 일반 유저들에게 심각한 피해를 입히는 Game Hack으로 사용된 기술입니다.

보안 분야를 공부하고 연구하는 입장에서 해당 D3D Hooking 에 대해 방어할 방법을 찾기 위해 D3D Hooking 소스를 분석해 보았으며 본 문서에선 d3d 관련 파일들을 분석하며 알아낸 내용들을 다른 분들과 정보공유를 통해 좀 더 깊은 내용을 배우고 방어 방법에 대해 같이 생각해보기 위해 부족한 실력에도 불구하고 이렇게 문서를 작성하게 되었습니다. 끝으로 D3D Hooking을 연구 하면서 얻은 기술이나 지식을 공유하고자 최대한 노력하여 본 문서를 작성하였습니다.

D3D Hooking에 대해 분석할 때 도움을 준 영근이에게 고맙다는 말을 끝으로 시작하겠습니다.

1-1. Direct3D Hooking에 필요한 선행 지식

- Assembly
- C++
- Hooking에 대한 이해
- Debugger
- DirectX SDK
- dll

1. Assembly
 - dll이나 타겟 프로그램을 분석할 때 Assembly로 출력되기 때문에 해당 지식 필요

2. C++
 - D3DX는 클래스로 이루어져 있기 때문에 C++지식이 필요하다.

3. Hooking에 대한 이해
 - 이 문서에서 D3D Hooking은 수많은 Hooking 중 하나이기 때문에 만약 Hooking에 대한 이해를 하고 있다면 좀 더 수월하게 이해를 할 수 있다.

4. Debugger
 - d3dx.dll 에 대해 분석하기 위해서는 Debugger 경험이 있으면 좀 더 쉽게 이해할 수 있다.

5. DirectX SDK
 - 이건 D3D 이기 때문에 DirectX SDK 를 사용해봤다면 코드 부분이나 함수 활용부분에서 매우 쉽게 이해 할 수 있긴 하겠지만, 필수는 아니다. 필자도 이번에 분석하면서 처음 D3D를 접했으며 분석하며 모르는 부분은 인터넷을 찾으며 정보를 수집했다.

6. Dll
 - d3dx.dll을 분석하는 것이기 때문에 dll이 윈도우에서 어떤 역할을 하는지 어떻게 동작하는지 정도는 알아야 이해하기가 쉬울 것이다.

2. Direct3D

2-1. Direct3D 란?

Direct3D는 마이크로소프트의 DirectX API에서 3차원 그래픽 연산과 출력을 담당하는 부분이다. 마이크로소프트의 윈도 운영체제(윈도 95이상)에서만 작동하며, Xbox와 Xbox360 게임 콘솔의 그래픽 API로 사용되고 있다. Direct3D와 비슷한 역할을 하는 API로는 OpenGL이 있으며 역할은 같지만 각자가 서로 다른 장단점을 가지고 있다.

출처 : http://ko.wikipedia.org/wiki/Direct3D

요약해보자면 3D게임이나 3D를 요구하는 작업에서 뭔가를 그릴 때 사용하는 게 Direct3D이다.

```
void init( void )
{
    g_pD3D = Direct3DCreate9( D3D_SDK_VERSION );

        if( g_pD3D == NULL )
        {
                // TO DO: Respond to failure of Direct3DCreate8
                return;
        }

    D3DDISPLAYMODE d3ddm;

    if( FAILED( g_pD3D->GetAdapterDisplayMode( D3DADAPTER_DEFAULT, &d3ddm ) ) )
        {
                // TO DO: Respond to failure of GetAdapterDisplayMode
                return;
        }

        HRESULT hr;

        if( FAILED( hr = g_pD3D->CheckDeviceFormat( D3DADAPTER_DEFAULT, D3DDEVTYPE_HAL,
                                                    d3ddm.Format, D3DUSAGE_DEPTHSTENCIL,
                                                    D3DRTYPE_SURFACE, D3DFMT_D16 ) ) )
        {
                if( hr == D3DERR_NOTAVAILABLE )
                        // POTENTIAL PROBLEM: We need at least a 16-bit z-buffer!
                        return;
        }

        //
```

```
        // Do we support hardware vertex processing? if so, use it.
        // If not, downgrade to software.
        //

        D3DCAPS9 d3dCaps;

        if( FAILED( g_pD3D->GetDeviceCaps( D3DADAPTER_DEFAULT,
                                           D3DDEVTYPE_HAL, &d3dCaps ) ) )
        {
                // TO DO: Respond to failure of GetDeviceCaps
                return;
        }

        DWORD dwBehaviorFlags = 0;

        if( d3dCaps.VertexProcessingCaps != 0 )
                dwBehaviorFlags |= D3DCREATE_HARDWARE_VERTEXPROCESSING;
        else
                dwBehaviorFlags |= D3DCREATE_SOFTWARE_VERTEXPROCESSING;

        //
        // Everything checks out - create a simple, windowed device.
        //

        D3DPRESENT_PARAMETERS d3dpp;
        memset(&d3dpp, 0, sizeof(d3dpp));

    d3dpp.BackBufferFormat       = d3ddm.Format;
    d3dpp.SwapEffect             = D3DSWAPEFFECT_DISCARD;
    d3dpp.Windowed               = TRUE;
    d3dpp.EnableAutoDepthStencil = TRUE;
    d3dpp.AutoDepthStencilFormat = D3DFMT_D16;
    d3dpp.PresentationInterval   = D3DPRESENT_INTERVAL_IMMEDIATE;

    if( FAILED( g_pD3D->CreateDevice( D3DADAPTER_DEFAULT, D3DDEVTYPE_HAL, g_hWnd,
                                      dwBehaviorFlags, &d3dpp, &g_pd3dDevice ) ) )
        {
                return;
        }
}
```

```
//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------
void shutDown( void )
{
    if( g_pd3dDevice != NULL )
        g_pd3dDevice->Release();

    if( g_pD3D != NULL )
        g_pD3D->Release();
}
//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------
void render( void )
{
    g_pd3dDevice->Clear( 0, NULL, D3DCLEAR_TARGET | D3DCLEAR_ZBUFFER,
                         D3DCOLOR_COLORVALUE(0.0f,1.0f,0.0f,1.0f), 1.0f, 0 );

    g_pd3dDevice->BeginScene();

    g_pd3dDevice->EndScene();

    g_pd3dDevice->Present( NULL, NULL, NULL, NULL );
}
```

[출처] http://www.codesampler.com/dx9src.htm

위의 sample source는 D3D 관련 함수 부분만 첨부했다.
D3D 코딩 경험이 없다면 한번 보는 것이 좋다. 이 소스코드를 후킹 관점에서 본다면 Direct3DCreate9(), CreateDevice(), EndScene() 이므로 다른 함수들은 물론 이 함수들이 낯설다면 msdn을 통해 함수에 대해 꼭 숙지하고 넘어가자.

3. Direct3D Hooking

3-1. What is Direct3D Hooking?

말 그대로 D3D에 대해 Hooking 하는 것이다. 이 문서에서 다루는 D3D Hooking은 원래 프로세스에서 D3D 관련 함수를 호출할 때 D3D 함수 대신 내가 제작한 함수가 호출되게 만드는 것이다.

이 문서에서 할 D3D Hooking의 원리를 설명해드리면 타겟 프로세스가 그려주는 함수를 호출할 때 내가 만든 함수를 호출 한 후 그리고 싶은 것을 그리고 원래의 그려주는 함수를 호출하면 프로세스는 정상적으로 작동하면서 내가 원하는 부분이 추가된 것이다. 이 문서에서는 d3dx의 vTable을 참고하여 타겟 함수를 설정하여 Detours 방식의 Hooking을 시도할 것이다.

D3D Hooking이 쓰이는 대표적인 것으로는 아직까지는 악용되는 것밖에 보질 못했다. FPS 게임에서 Wall Hack, RPG 게임에서 Map Hack 등등 그래픽을 가진 게임에서는 무궁무진하게 D3D Hooking 악용될 수 있다. 이 문서를 통해 FPS 게임이나 기타 프로그램에서 D3D에 대한 후킹을 연구하여 방어 방법에 대해 생각 할 수 있는 기회가 되었으면 좋겠다.

이 문서에서 D3D Hooking 하는데 사용될 방법은 Detours 라는 후킹 기법이다.
Detours에 대한 자세한 내용은 뒤에서 설명하겠다.

악용되는 예 - FPS Game Wall Hack

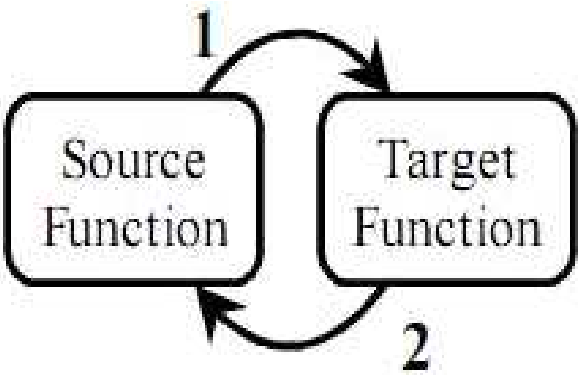
출처 : 구글 이미지 (본 문서와 해당 그림은 상관없습니다.)

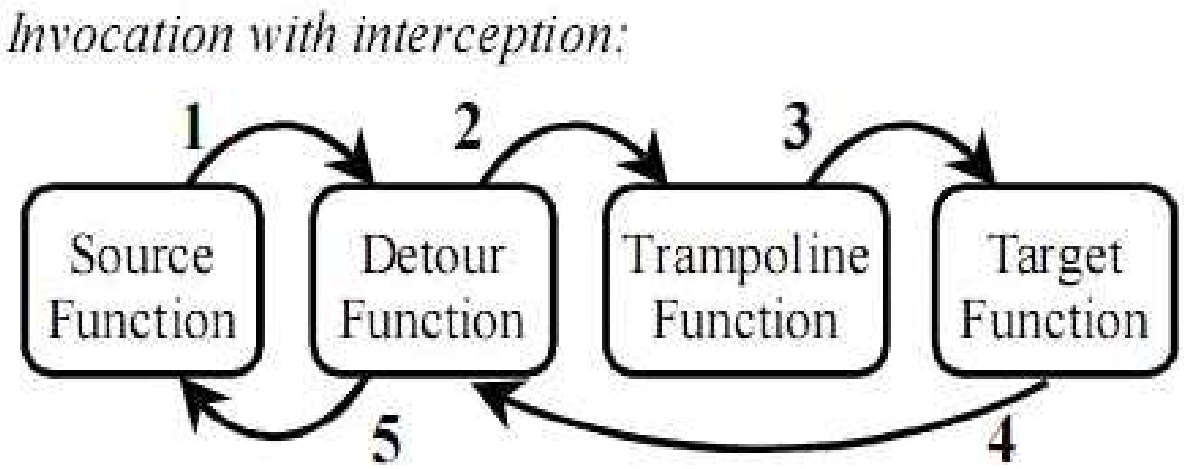
3-2. D3D Hooking 원리

D3D Hooking 원리는 함수를 후킹 할 때 사용하는 방법인 "Detours Hook"와 후킹 할 함수의 주소를 찾고자 할 때 참고하는 "vTable"를 사용하여 D3D에서 사용하는 함수를 후킹 한다.
"Detours Hook" 와 "vTable" 에 대한 설명을 간단하게 하고 넘어가겠다.

3-2-1. Detours Hook 이란? (Detours hook 원리에 대해 알고 있다면 넘어가도 됩니다.)

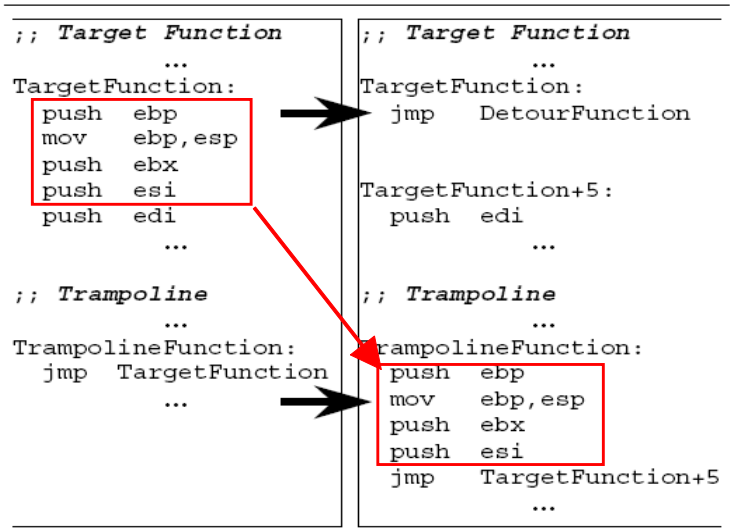
Detour Patching 또는 inline Function Hooking 이라고도 불리는 이 후킹 방법은 여러 방법들 중에 하나이며 그 중에서도 매우 강력한 기능을 가진 후킹 방법이라고 할 수 있습니다.
Target Function에 Detour Function 으로 점프하는 코드를 심어 Detour Function 이 실행되고 실행이 끝나면 Trampoline Function에서 다시 Target Function을 실행시키고 다시 Detour Function으로 return할 수 있도록 하는 후킹 기술입니다.





출처 : http://research.microsoft.com/apps/pubs/default.aspx?id=68568

위의 그림처럼 Target Function 보다 Detour Function 이 먼저 실행되고 또 Detour Function으로 return되기 때문에 어떤 작업을 하거나 결과 값을 수정 할 수 있습니다.
실제 함수 전, 후로 전부 후킹이 되기 때문에 매우 효과적인 조작이 가능합니다.



출처 : http://research.microsoft.com/apps/pubs/default.aspx?id=68568

후킹 방식은 위의 그림과 같이 Trampoline Function에서 Target Function에서 jmp 코드가 덮어쓴 Function Instruction을 수행하고 Target Function에서 해당 Function Instruction이 실행된 이후 번지로 점프하여 Target Function 함수가 정상적으로 호출 된 것처럼 실행되도록 합니다. 이 문서에서는 D3D Hooking에서 함수 후킹 할 때 Detours Hook을 사용합니다. Detours Hook 에 관련된 내용은 다음 문서를 통해 자세히 다루겠습니다.

```
void *DetourFunc(BYTE *src, const BYTE *dst, const int len)
{
	BYTE *jmp = (BYTE*)malloc(len+5);
	DWORD dwback;
	VirtualProtect(src, len, PAGE_READWRITE, &dwback);
	memcpy(jmp, src, len); jmp += len;
	jmp[0] = 0xE9;
	*(DWORD*)(jmp+1) = (DWORD)(src+len - jmp) - 5;
	src[0] = 0xE9;
	*(DWORD*)(src+1) = (DWORD)(dst - src) - 5;
	VirtualProtect(src, len, dwback, &dwback);
	return (jmp-len);
}
```

- 이 문서에서 사용하는 Detours function 소스 (자세한 설명은 뒤에서 하겠습니다.)

3-2-1. vTable 이란? (vTable에 대해 알고 있다면 넘어가도 됩니다.)

vTable(virtual table)이란 가상 함수의 번지 목록을 가지는 일종의 포인터 배열이다. 즉, 이 클래스에 소속된 가상 함수들이 어떤 번지에 저장되어 있는지를 표 형태로 저장해 놓은 목록이다. 컴파일러는 가상 함수를 단 하나라도 가진 클래스에 대해 vTable을 작성하는데, 이 테이블에는 클래스에 소속된 가상 함수들의 실제 번지들이 선언된 순서대로 기록되어 있다. 그리고 이 테이블 타입의 객체가 생성될 때, 각 객체의 선두에 vTable의 번지인 vptr을 기록한다. 간단한 소스와 그림을 통해 vTable의 구조와 원리를 간단하게 살펴보고 넘어가도록 하자.

```
#include <iostream.h>

class B{
private:
        int memB;
public:
        b() : memB(0x11111111){}
        virtual void f1() { puts("B::f1");}
        virtual void f2() { puts("B::f2");}
        virtual void f3() { puts("B::f3");}
        void normal() { puts("non virtual");}
};

class D : public B
{
private:
        int memD;
public:
        D() : memD(0x22222222){}
        virtual void f1() { puts("D::f1");}
        virtual void f2() { puts("D::f2");}
};

void main()
{
        B *pB;
        B b;
        D d;
        pB=&b;
        pB->f2();
        pB=&d;
        pB->f2();
        pB->f3();
}
```

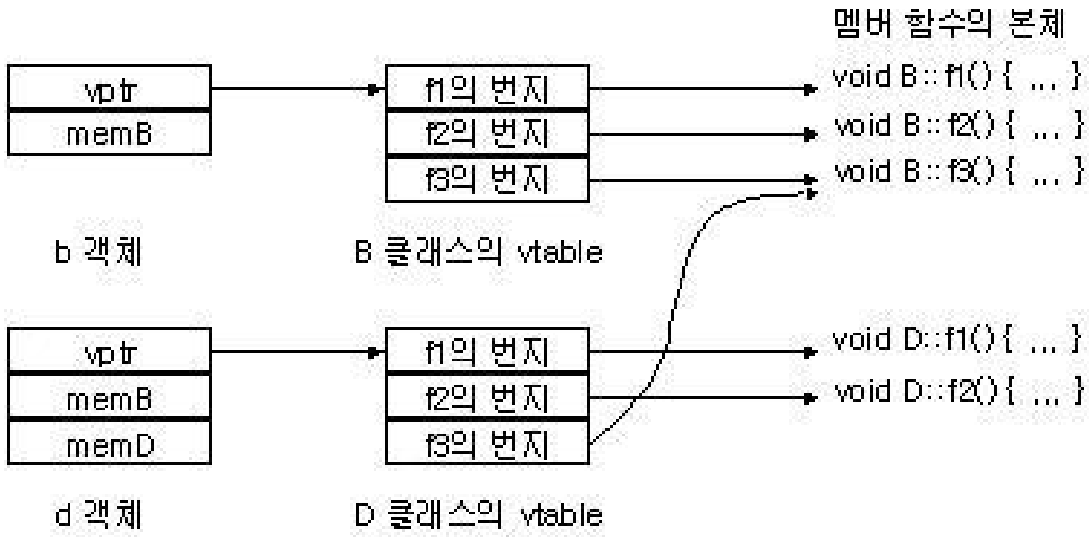
실행 결과는 다음과 같다.

B::f2

D::f2

B::f3

B가 세 개의 가상 함수와 하나의 비가상 함수를 정의하고 있으며 이를 상속 받는 D는 그 중 f1, f2를 재정의하고 있다. 테스트의 편의를 위해 멤버 변수도 하나씩 선언했다. B와 D의 객체 b와 d가 생성되었을 때 이 객체들이 메모리에 구현된 모양을 그려 보면 다음과 같다.



컴파일러는 B 클래스를 위해 B 클래스에 속한 가상 함수의 번지를 vTable로 작성한다. vTable은 가상 함수들의 포인터 배열이라고 할 수 있는데 비가상 함수의 번지는 목록에서 제외된다. 그래서 vTable에 normal 함수의 번지는 없는데 이 함수는 정적으로 결합되므로 테이블에 있을 필요가 없다. B 타입의 b객체에는 자신의 멤버 변수 memB앞에 B클래스의 vTable에 대한 포인터 vptr이 먼저 배치되고 이 포인터가 가리키는 vTable에는 자신이 호출할 수 있는 가상 함수들에 대한 실제 번지들이 기록되어 있다. B 클래스는 모든 가상 함수의 코드를 정의하고 있으므로 vTable에는 자신의 멤버 함수들에 대한 번지만 있다.

D클래스도 가상 함수를 가지고 있으므로 컴파일러는 D에 대해서도 vTable을 작성한다. 이 테이블의 f1, f2는 D가 재 정의한 함수를 가리키고 있으며 f3은 B로부터 상속받은 B::f3을 가리키고 있다. 이 표에 의해 D타입의 객체가 f1, f2를 호출하면 D::f1, D::f2가 호출되지만 f3에 대해서는 상속받은 B::f3이 호출되어야 한다는 것을 알 수 있다. D타입의 객체 d에는 상속받은 memB와 memD 앞에 D클래스의 vTable을 가리키는 포인터 vptr이 배치되어 있다.

즉, vTable의 주소를 알면 선언된 순서대로 위치하기 때문에 vTable의 주소 + 함수 offset을 통해 Target Function 위치를 알 수 있다.

참고 : http://winapi.co.kr/clec/cpp3/30-1-4.png

3-3. D3D Hooking 실습

3-3-1. D3D Hooking에 대한 실습과 설명

이제 위에서 언급한 "vTable"을 이용하여 EndScene 함수의 주소를 얻어서 "Detours Hook“을 사용해 EndScene 함수를 후킹 하여 내가 원하는 것을 그리는 것을 실습 해보자.

먼저 D3D Hooking 의 흐름을 봐보자.
D3DHook.dll 이 삽입되면 Target Process에서 dll 메인이 실행되고, dll 메인에서 D3D Hook을 위해 만든 함수를 호출하고, 해당 함수에서 d3d9.dll의 vTable의 주소를 넘겨주는 부분의 패턴을 검색하여 vTable의 주소를 얻는다. vTable의 시작 주소를 얻으면 EndScene 함수의 offset을 vTable 주소에 더해서 해당 함수 주소를 찾아낸다. 해당 함수(원래의 EndScene)를 후킹 할 때 사용할 함수의 주소로 Detour Hook 방법으로 패치 한다.
패치가 되고나서는 해당 프로세스가 작동하면서 EndScene 함수를 호출 할때 마다 Hook 함수가 먼저 호출되고 난 후 원래의 EndScene이 호출되는 게 반복 될 것이다.

DllMain 함수를 봐보자.

```
#include "d3dhooks.h"
#include "common.h"

BOOL WINAPI DllMain(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpReserved )
{
	switch( fdwReason )
	{
	case DLL_PROCESS_ATTACH:
		{
			DisableThreadLibraryCalls(hinstDLL);
			StartD3DHooks();
				return true
			break
		}
	case DLL_PROCESS_DETACH:
		{
			MessageBox(NULL,L"detach dll!", L"ok", MB_OK);
			break
		}
	}
	return TRUE;
}
```

DllMain() 이며, 삽입되자마자 실행될 함수이다. 삽입이 되면 DisableThreadLibraryCalls()을 호출하게 되는데, 이 함수는 "DLL_THREAD_ATTACH" 및 "DLL_THREAD_DETACH" 통지를 방지하게 되는데, “DLL_THREAD_ATTACH” 통지를 방지함으로써 TLS callback 함수 실행을 방지하여 TLS callback 함수에서 Debugger을 검출하는 코드로부터 숨는(stealth) 용도로 사용된다. 그 뒤로 Start3DHook(); 함수가 실행된다.

```
int StartD3DHooks()
{
	DWORD D3DPattern,*vTable, DXBase=NULL;
	DXBase = (DWORD)LoadLibraryA("d3d9.dll");

	while(!DXBase);
	{
		D3DPattern = FindPattern(DXBase, 0x128000,
		   (PBYTE)"₩xC7₩x06₩x00₩x00₩x00₩x00₩x89₩x86₩x00₩x00₩x00₩x00₩x89₩x86", "xx????xx????xx");
	}

	if(D3DPattern)
	{
		memcpy(&vTable,(void *)(D3DPattern+2),4);
		org_EndScene = (EndScene_t)DetourFunc((PBYTE)vTable[ENDSCENE],
		                                      (PBYTE)DXGameHook.h_EndScene,5);
	}
	return 0;
}
```

하나하나 차근차근 봐보자.

```
int StartD3DHooks()
{
	DWORD D3DPattern,*vTable, DXBase=NULL;
	DXBase = (DWORD)LoadLibraryA("d3d9.dll");
```

D3DPattern -> FindPattern함수를 통해 패턴위치를 찾아낸 주소를 저장하는 변수다.
(FindPattern 은 메모리상에서 vTable 주소를 넘겨주는 부분의
Hex값 패턴이다. FindPattern 부분 설명할 때 자세히 설명 하겠다.)

*vTable -> vTable 의 시작 번지를 저장하는 용도로 사용된다.

DXBase -> LoadLibraryA("d3d9.dll") 리턴값인 d3d9.dll 모듈의 핸들(HMODULE)을 저장한다.

```
	while(!DXBase);
	{
		D3DPattern = FindPattern(DXBase, 0x128000,
		   (PBYTE)"₩xC7₩x06₩x00₩x00₩x00₩x00₩x89₩x86₩x00₩x00₩x00₩x00₩x89₩x86", "xx????xx????xx");
	}
```

d3d9.dll 모듈의 핸들을 성공적으로 가져왔다면, FindPattern 함수를 통해서 패턴 검색을 통해서 d3d9 모듈에서 vTable 주소를 가지고 있는 부분을 찾아낸다.

그 다음 코드로 넘어가기 전에 FindPattern 함수에 대해 보고 가자.

```
DWORD FindPattern(DWORD dwAddress,DWORD dwLen,BYTE *bMask,char * szMask)
{
        for(DWORD i=0; i < dwLen; i++)
                if( bDataCompare( (BYTE*)( dwAddress+i ),bMask,szMask) )
                        return (DWORD)(dwAddress+i);

        return 0;
}

bool bDataCompare(const BYTE* pData, const BYTE* bMask, const char* szMask)
{
        for(;*szMask;++szMask,++pData,++bMask)
                if(*szMask=='x' && *pData!=*bMask )
                        return false
        return (*szMask) == NULL;
}
```

FindPattern의 함수를 봐보면, 첫 번째 인자인 dwAddress는 검색을 시작할 주소이고, 두 번째인 dwLen은 검색할 범위를 지정해준다. 세 번째 인자 bMask 는 패턴을 저장하는 변수이고, 마지막 szMask 는 패턴에서 마스크 지정을 저장하는 변수이다.

bDataCompare는 FindPattern에서 받은 szMask 값을 이용하여 마스크 검색을 하는 함수이다. 여기서 마스크 검색이란, 예를 들어 검색하고자 하는 값이 12 34 56 78 이고 마스크가 xx??xx?? 이면 12 ?? 56 ?? 형태를 가진 모든 값을 검색하는 것이다. 여기서 ?? 부분은 무슨 값이 들어가도 12 ?? 56 ?? 형태라면 해당 부분을 찾아낸다. 12 FF 56 FF 라든지, 12 3C 56 A1 이라든지 말이다.

```
        while(!DXBase);
        {
                D3DPattern = FindPattern(DXBase, 0x128000,
                   (PBYTE)"₩xC7₩x06₩x00₩x00₩x00₩x00₩x89₩x86₩x00₩x00₩x00₩x00₩x89₩x86", "xx????xx????xx");
        }
```

자, 이제 이 소스를 다시 보면 DXBase(d3d9.dll 모듈의 시작 주소)부터 0x128000 범위안에서 "₩xC7₩x06₩x00₩x00₩x00₩x00₩x89₩x86₩x00₩x00₩x00₩x00₩x89₩x86" 패턴에 "xx????xx????xx"의 마스크를 적용하여 해당 하는 값을 찾아내라. 라는 말이다.

자 이제 다시 원래 코드의 흐름으로 돌아가보자.

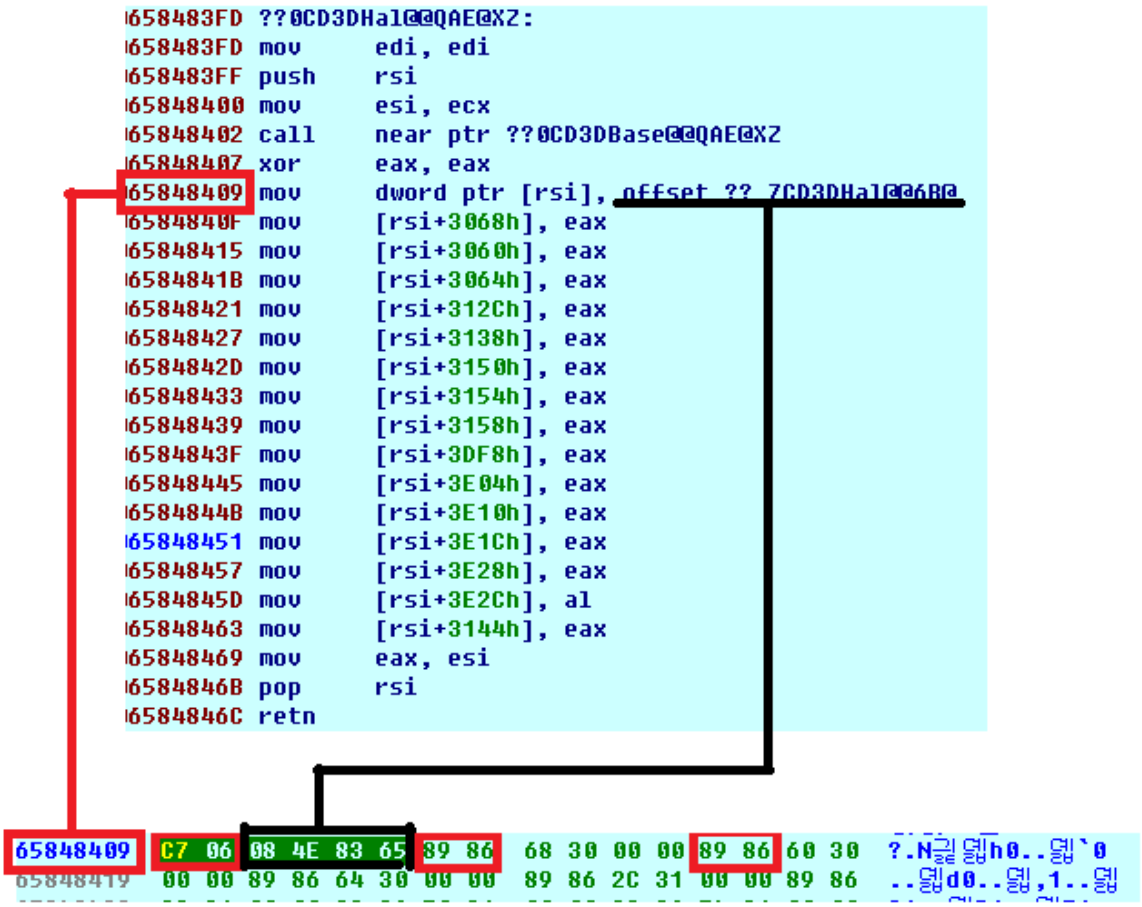
```
        if(D3DPattern)
        {

                memcpy(&vTable,(void *)(D3DPattern+2),4);
```

D3DPattern 을 찾았다면 그 아래 두 개의 함수가 실행이 된다. 첫 번째는 memcpy이고 두 번째는 DetourFunc라는 함수이다. 먼저 memcpy 라는 함수는 메모리를 주어진 인자 값을 참고로 src 번지에서 dst 번지로 len만큼 복사하는 함수이다.

첫 번째 인자 vTable에다가 D3DPattern(FindPattern함수를 이용하여 패턴을 발견한 위치)의 주소에서 +2 한 곳에서 4 Byte 를 복사해라. 라는 말이다. 여기서 찾아낸 D3Dpattern에서 +2를 한 이유는 0xC7 0x06 이후에 나오는 4 Byte에 vTable의 주소 값이 들어가 있기 때문에 맨 앞에 0xC7, 0x06의 2 byte를 제외한 그 다음 4 Byte를 vTable 변수에 저장하라는 의미다.

해당 패턴을 CheatEngine으로 검색하여(CheatEngine에서 마스크 검색을 지원한다.)찾았다.
해당 번지를 ida로 따라 가보았다.



위의 그림에서 아래에 Hex값을 보면 위에서 사용했던 패턴을 볼 수 있다. 이것은 C7 06 이후 4 Byte 값인 08 4E 83 65 값(vTable의 시작주소)를 vTable변수에 저장하는 것을 알 수 있다.
(패턴은 필자가 처음부터 찾아내서 사용한 게 아니다. 문서와 인터넷들을 뒤져보았지만 해당 패턴을 사용하는 곳은 많았으나 이 위치가 무슨 역할을 하는 위치인지에 대한 정보는 없었다.
해당 위치에 BP(Break Point)도 걸어 보았지만 dll 이 인젝션 된 다음에는 멈추지 않았다.
여기에 대한 내용은 뒤에서 언급하겠다.)

```
                org_EndScene = (EndScene_t)DetourFunc((PBYTE)vTable[ENDSCENE],
                                                      (PBYTE)DXGameHook.h_EndScene,5);
        }
        return 0;
}
```

org_EndScene -> typedef HRESULT(__stdcall* EndScene_t)(LPDIRECT3DDEVICE9);
EndScene_t org_EndScene;
즉, EndScene 함수 포인터이다.

DetourFunc() -> DetourFunc 함수는 Detour Hook 원리를 토대로 만들어진 함수이며 앞에서 말한 방식으로 메모리 패치를 하는 함수이다. (3-2-1. Detours Hook 이란? 참고)

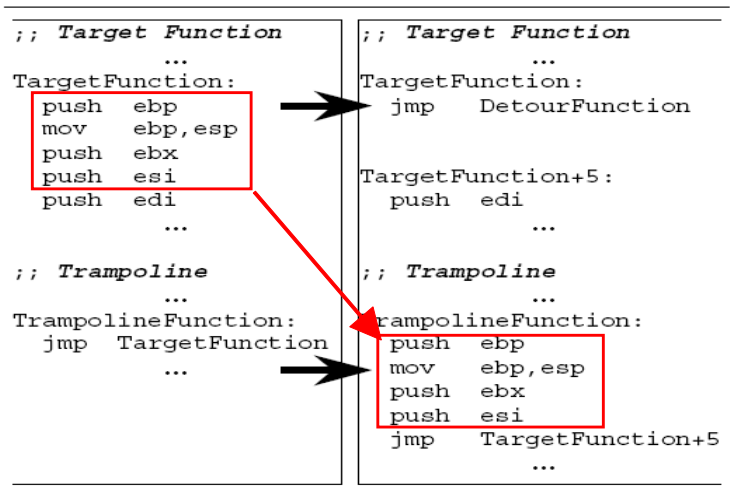
```
void *DetourFunc(BYTE *src, const BYTE *dst, const int len)
{
        BYTE *jmp = (BYTE*)malloc(len+5);
        DWORD dwback;
        VirtualProtect(src, len, PAGE_READWRITE, &dwback);
        memcpy(jmp, src, len); jmp += len;
        jmp[0] = 0xE9;
        *(DWORD*)(jmp+1) = (DWORD)(src+len - jmp) - 5;
        src[0] = 0xE9;
        *(DWORD*)(src+1) = (DWORD)(dst - src) - 5;
        VirtualProtect(src, len, dwback, &dwback);
        return (jmp-len);
}
```

DetourFunc 함수를 보면, 첫 번째 인자는 Target Function의 위치이고, 두 번째 인자는 Detours Function의 위치이고, 세 번째 인자인 len 은 뒤에서 복사할 때 사용될 값인데, Hex 값 복사할 때 몇 바이트를 복사할건지 결정된다.

```
void *DetourFunc(BYTE *src, const BYTE *dst, const int len)
{
        BYTE *jmp = (BYTE*)malloc(len+5);
        DWORD dwback;
```

jmp -> Trampoline Function 용도로 사용될 메모리 공간을 할당한 곳의 포인터를 저장한다.
dwback -> 바로 뒤에서 사용할 VirtualProtect에서 메모리 영역의 접근권한이 설정할 때 새로운 접근권한 이전의 접근권한을 저장하는데 사용되는 변수다.

(BYTE*)malloc(len + 5); -> 왜 len+5 만큼 공간을 할당할까?



위의 그림을 보면, function prologue 부분을 Trampoline Function 영역으로 복사하고 그 뒤에 “jmp TargetFunction+5“ 를 복사한 것을 볼 수 있다.

TargetFunction에서 상위 5byte를 따로 복사한 이유는 Target Function의 상위 5 byte를 “jmp DetourFunction”으로 덮어쓰기 때문이다. 이 부분을 덮어쓰기 전에 원래 5 byte만큼의 명령어들을 TrampolineFunction 에 복사해두는것이다. 그래야 TrampolineFunction에서 TargetFunction을 호출하기 전에 원래의 TargetFunction 코드를 실행하고 넘어가기 때문에 정상적인 함수처럼 작동이 가능하는것과, "jmp DetourFunction"을 TargetFunction 함수에서 가장 앞에 둔 이유는 TargetFunction이 실행하기전 우리가 만든 함수가 실행된 후에 TargetFunction이 실행되게 하려고 했기 때문이다.

"jmp TargetFunction+5"가 왜 5 byte 이냐면, 저 위에 소스에서도 볼 수 있듯이 jmp 는 0xE9 를 사용하고 뒤에 주소는 0x00000000 식의 4바이트이기 때문에 총 5 byte가 필요하다.

```
VirtualProtect(src, len, PAGE_READWRITE, &dwback);
memcpy(jmp, src, len); jmp += len;
```

VirtualProtect(src, len, PAGE_READWRITE, &dwback);

- Virtualprotect라는 함수는 메모리에서 영역의 접근 권한을 설정하는 함수이다. 첫 번째 인수에는 변경할 주소, 두 번째에는 변경할 주소부터의 범위, 세 번째는 접근권한, 네 번째는 위에서 말한 것과 같이 새로운 접근권한을 적용할 때 적용 전의 접근권한

memcpy(jmp, src, len); jmp += len;

- Trampoline 역할을 하는 메모리 위치(jmp) 에다가 src, 즉 Target Function의 주소에서 상위 5byte를 복사한다. 그리고 jmp 포인터를 len 만큼 이동시킨다.

```
jmp[0] = 0xE9;
*(DWORD*)(jmp+1) = (DWORD)(src+len - jmp) - 5;
```

jmp[0] = 0xE9 -> jmp의 범위는 총 len+5 byte 인데, 바로 위에서 memcpy로 jmp의 시작 번지부터 len 만큼 Target Function에서 값을 복사해왔고, 그 뒤 jmp+=len 을 통해 jmp 포인터를 이동시킨 다음에 jmp[0] = 0xE9 를 하면 앞에서 memcpy 한 부분에 덮어쓰는 것이 아니라 memcpy에서 카피한 부분 뒤에 "0xE9"를 넣는 것이다.

여기서 다음 줄 코드를 설명하기 전에 어셈블리어 명령어 중 JMP에 대해 간단하게 알아보고 넘어가자. JMP에는 두 가지가 있다. long jmp 와 short jmp 가 있는데, 이중에서 0xE9는 long jmp이고 0xEB는 short jmp인데 후킹할 때 short jmp는 한번에 255byte 정도밖에 점프할 수 없으니 보통 long jmp를 사용합니다. 보통 E9 XXXXXXXX 의 형태를 가지는데 뒤에 XXXXXXXX는 주소입니다. 근데 이게 그냥 주소가 아닌 상대 주소입니다.

예를 들어 E9 12 34 56 78 이라면 0x78563412 위치로 점프해라가 아니라 해당 명령어가 있는 주소를 기준으로 더해주는겁니다. 예를 들어 11110000번지에서 E9 00 01 00 00 명령어가 있다면 0x00000100로 이동하는게 아니라 0x11110000 + 0x00000100 +0x5 에 해당하는 위치로 이동합니다. 여기서 마지막 0x5는 5byte 명령어 E9 XXXXXXXX 의 길이도 더하는 겁니다.

(DWORD)(jmp+1) = (DWORD)(scr+len - jmp) -5;

- 앞에서 jmp[0] 위치에 E9를 넣었으니 jmp의 남은 4byte 부분에는 점프 시킬 주소가 들어갑니다. 바로 위에서 설명한 상대주소가 들어갑니다. src, 즉 Target Function의 주소에서 앞에서 "jmp DetourFunction"의 명령어 길이인 len 을 더해준 값에 jmp의 위치를 빼준다. 그리고 마지막에는 E9 XXXXXXXX의 명령어 길이까지 빼서 정확한 위치를 계산한다.
  상대 번지 계산 = (점프하고자 하는 번지) - (jmp 명령어가 있는 위치) - 5(명령어 길이)

```
src[0] = 0xE9;
*(DWORD*)(src+1) = (DWORD)(dst - src) - 5;
```

src[0] = 0xE9; -> 이제 Target Function 의 5 byte Trampoline 위치로 옮겼으니 Target Function에 jmp "DetourFunction+5" 를 삽입하는 것만 남았다. 먼저 src[0], 즉 src 맨 앞에 0xE9 를 삽입함으로써 jmp 명령어를 넣는다.

(DWORD)(src+1) = (DWORD)(dst - src) - 5;

- 위에서 말한 방법과 똑같이 상대주소를 계산하여 src(TargetFunction)에 DetourFunction으로 갈 수 있는 상대 주소를 넣어준다.

```
        VirtualProtect(src, len, dwback, &dwback);
        return (jmp-len);
}
```

VirtualProtect(src, len, dwback, &dwback);
- 이 부분은 앞에서 원래의 접근 권한을 READWRITE로 바꾸어준 것을 다시 원래대로 돌려주는 부분이다. 원래의 접근 권한을 넣어둔 dwback를 이번엔 "PAGE_READWRITE"이 있던 자리에 넣음으로써 접근권한을 원상태로 돌려놓는다.

return (jmp-len);
- Trampoline 역할을 하는 메모리영역을 리턴 한다.

이제 다시 StartD3DHook() 로 돌아가 보자.

```
int StartD3DHooks()
{
        DWORD D3DPattern,*vTable, DXBase=NULL;
        DXBase = (DWORD)LoadLibraryA("d3d9.dll");

        while(!DXBase);
        {
                D3DPattern = FindPattern(DXBase, 0x128000,
                   (PBYTE)"₩xC7₩x06₩x00₩x00₩x00₩x00₩x89₩x86₩x00₩x00₩x00₩x00₩x89₩x86", "xx????xx????xx");
        }

        if(D3DPattern)
        {
                memcpy(&vTable,(void *)(D3DPattern+2),4);
                org_EndScene = (EndScene_t)DetourFunc((PBYTE)vTable[ENDSCENE],
                                                      (PBYTE)DXGameHook.h_EndScene,5);
        }
        return 0;
}
```

자 이제 org_EndScene에는 Trampoline 메모리영역의 주소가 들어가 있다. 이것은 DetourFunction에서 작업을 끝내고 EndScene의 원래 함수를 호출할 때 사용되는 주소이다. 여기서 vTable[ENDSCENE] 이란 부분이 있는데, ENDSCENE 값은 42이다. 왜냐하면 EndScene 함수는 D3D9 vTable에서 42번째에 위치하기 때문이다. vTable은 함수를 선언한 순서대로 만들어지기 때문에 d3d9.h 파일을 참고하면 알 수 있다. vTable[X] 란 vTable이 DWORD형 포인터이기 때문에 vTable + (X * 0x4)를 가독성 있게 효율적으로 작성 한 것이다. 이제 마지막으로 EndScene Hooking 함수를 봐보자.

```
static HRESULT WINAPI h_EndScene(LPDIRECT3DDEVICE9 pDevice);
typedef HRESULT(__stdcall* EndScene_t)(LPDIRECT3DDEVICE9);
EndScene_t org_EndScene;

HRESULT WINAPI DXGH::h_EndScene(LPDIRECT3DDEVICE9 pDevice)
{
	DXGameHook.DrawRect(pDevice, 40, 110, 50, 50, txtPink);
	return org_EndScene(pDevice);
}
```

여기서 중요한건 함수 형태와 return 이다.
후킹 함수는 타겟 함수의 형태와 동일해야하며 return에서 org_EndScene을 호출하는 것으로 원래의 EndScene함수를 호출하면서 D3D Hooking은 끝난다. DrawRect함수는 D3D Hooking와 관련 없다고 생각되어 여기서 설명은 안하고 뒤에 전체 소스를 첨부 하겠다.

여기서 EndScene함수뿐만 아니라 다른 함수이거나, 이 함수안에 DrawRect 라는 함수가 아니라 이 기술을 사용하는 사람의 의도에 따라 유용하게 쓰일 수 있다.

예를 들어 메모리에서 어느 값을 실시간으로 체크할 수 있게 해준다든지, D3D 관련 프로그램을 가지고 있지만 소스가 없을 때 오류가 발생했을 때 로그 기록이라든지, 새로운 기능 추가 등등 다양하다.

- D3D Hooking (EndScene)한 화면 -

3-3-2. 패턴은 어떻게 찾아낼까?

"₩xC7₩x06₩x00₩x00₩x00₩x00₩x89₩x86₩x00₩x00₩x00₩x00₩x89₩x86"

위에서 한 D3D Hooking 의 핵심은 이 패턴이다. 이 패턴은 vTable과 연관이 있다는 건 알겠지만 과연 어디에 위치하였고 어떤 함수가 호출하는지, 이 패턴이 바뀌면 어떻게 찾아낼 것인지 등등 여러 호기심을 자극하여 한번 찾아보았다.

처음에 시도한 것은 해당 패턴이 있는 위치에 Break Point(이하 BP)를 걸어 보았지만 반응이 없다. 그렇다면 해당 패턴을 가지고 있는 함수는 한 번만 호출되거나 어떠한 함수를 호출 할 때 내부적으로 해당 패턴을 지나간다고 가정해보았다.

함수가 한 번만 호출될 경우, D3D 관련 객체 생성자에서 이 작업을 처리하거나 어느 함수의 내부적으로 호출하는 것으로 생각할 수 있는데, 먼저 ida에서 해당 패턴 위치를 알아보았다.

```
15E3483FD ??0CD3DHal@@QAE@XZ:
15E3483FD mov     edi, edi
15E3483FF push    rsi
15E348400 mov     esi, ecx
15E348402 call    ??0CD3DBase@@QAE@XZ
15E348407 xor     eax, eax
15E348409 mov     dword ptr [rsi], offset ??_7CD3DHal@@6B@
15E34840F mov     [rsi+3068h], eax
15E348415 mov     [rsi+3060h], eax
15E34841B mov     [rsi+3064h], eax
15E348421 mov     [rsi+312Ch], eax
15E348427 mov     [rsi+3138h], eax
15E34842D mov     [rsi+3150h], eax
15E348433 mov     [rsi+3154h], eax
15E348439 mov     [rsi+3158h], eax
15E34843F mov     [rsi+3DF8h], eax
15E348445 mov     [rsi+3E04h], eax
15E34844B mov     [rsi+3E10h], eax
15E348451 mov     [rsi+3E1Ch], eax
15E348457 mov     [rsi+3E28h], eax
15E34845D mov     [rsi+3E2Ch], al
15E348463 mov     [rsi+3144h], eax
15E348469 mov     eax, esi
15E34846B pop     rsi
15E34846C retn
```

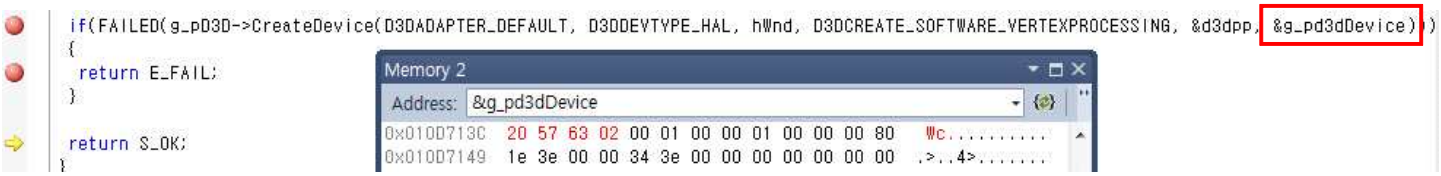
해당 위치로 가보니 ??0CD3DHal@@QAE@XZ로 네임맹글링 된 함수를 볼 수 있다.
하지만 BP가 걸리지도 않고 "Chart of xrefs to" 기능을 사용하여 해당 함수 호출 부분을 찾아보려 했으나 나오지 않아서 다른 방법을 생각해 보게 되었다.

먼저 D3D 샘플을 하나 만들고 함수들 하나씩 정보를 찾아보는 중에 흥미로운 사실을 알 수 있었다. IDirect3D9::CreateDevice 함수에 대해 msdn에서 찾아보았다.

```
HRESULT CreateDevice(
  [in]           UINT Adapter,
  [in]           D3DDEVTYPE DeviceType,
  [in]           HWND hFocusWindow,
  [in]           DWORD BehaviorFlags,
  [in, out]      D3DPRESENT_PARAMETERS *pPresentationParameters,
  [out, retval]  IDirect3DDevice9 **ppReturnedDeviceInterface
);
```

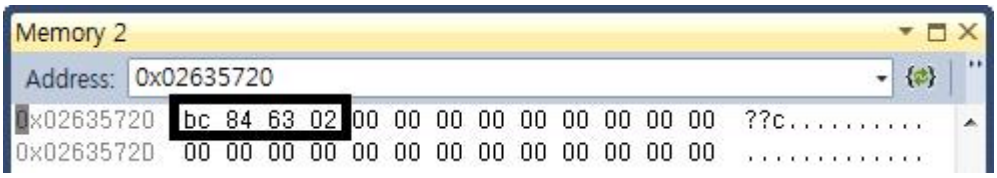
CreateDevice의 인자 값 중 마지막 인자 값에 IDirect3DDevice9 인터페이스의 더블 포인터가 들어오는 것 을 알 수 있다. IDirect3DDevice9 인터페이스의 메소드에 d3d에서 사용 하는 함수들이 들어있다. 저 마지막 인자에 리턴 되는 주소가 의심스러워 디버깅 해보았다.

위에 게시하였던 D3D Sample Source 로 샘플을 만들고 CreateDevice에 BP를 걸어 마지막 인자값에 들어가는 주소를 얻었다.

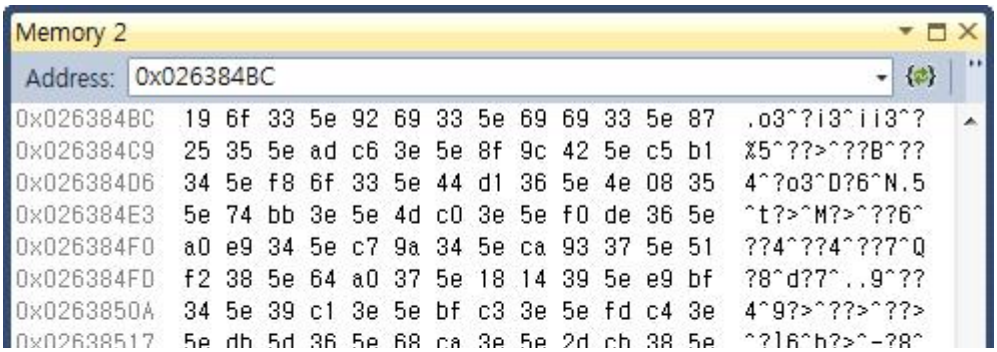


리턴 값은 위에 보이듯이 **ppReturnedDeviceInterface, 더블 포인터이기 때문에 2번을 따라가야 IDirect3DDevice9 Interface가 위치 할 것이다.

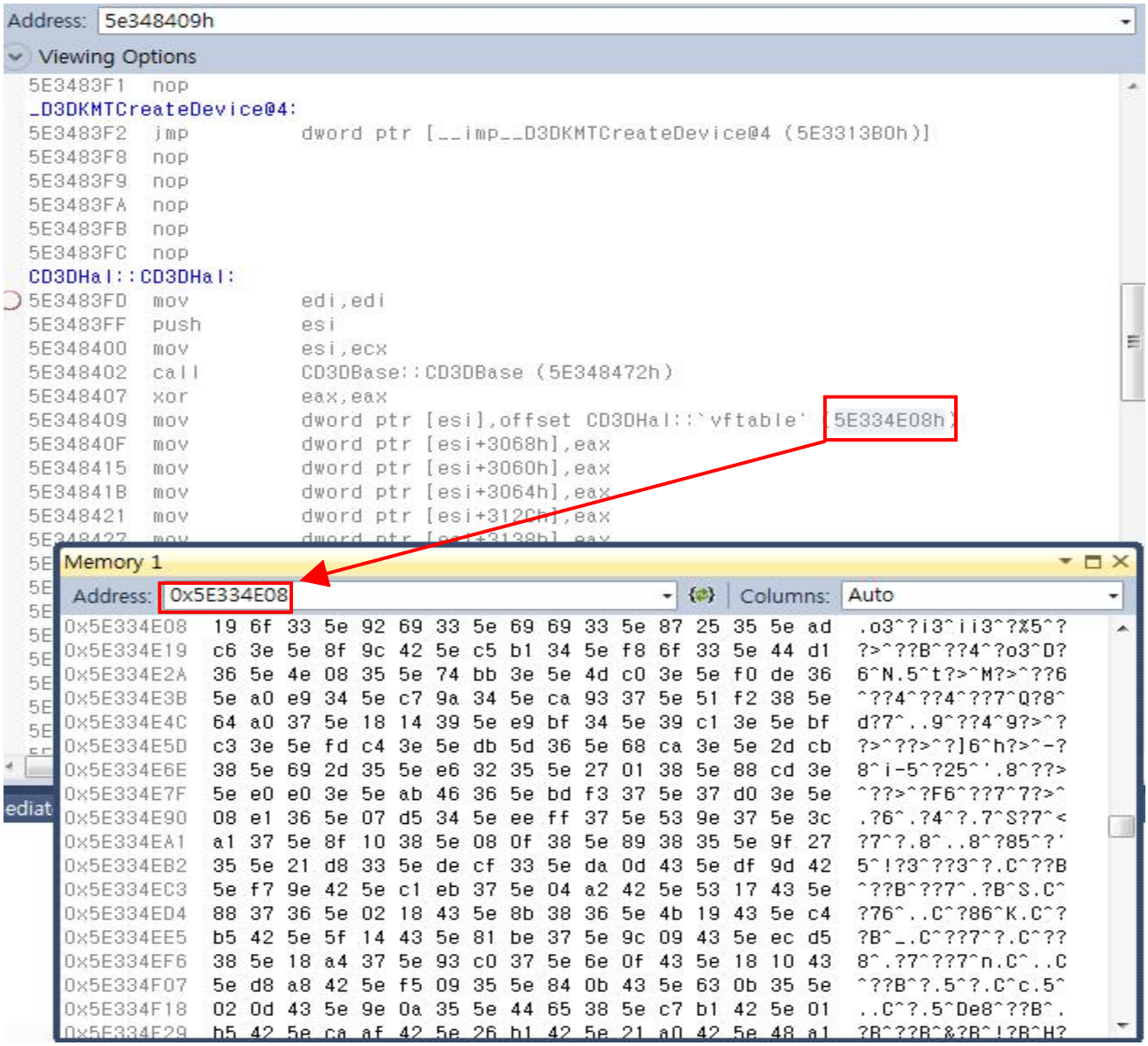
g_pd3dDevice 포인터가 가르키는 곳을 이동해보았다. (0x02635720)



이 역시 포인터이기 때문에 한 번 더 가리키는 곳을 가봤다. (0x026484BC)

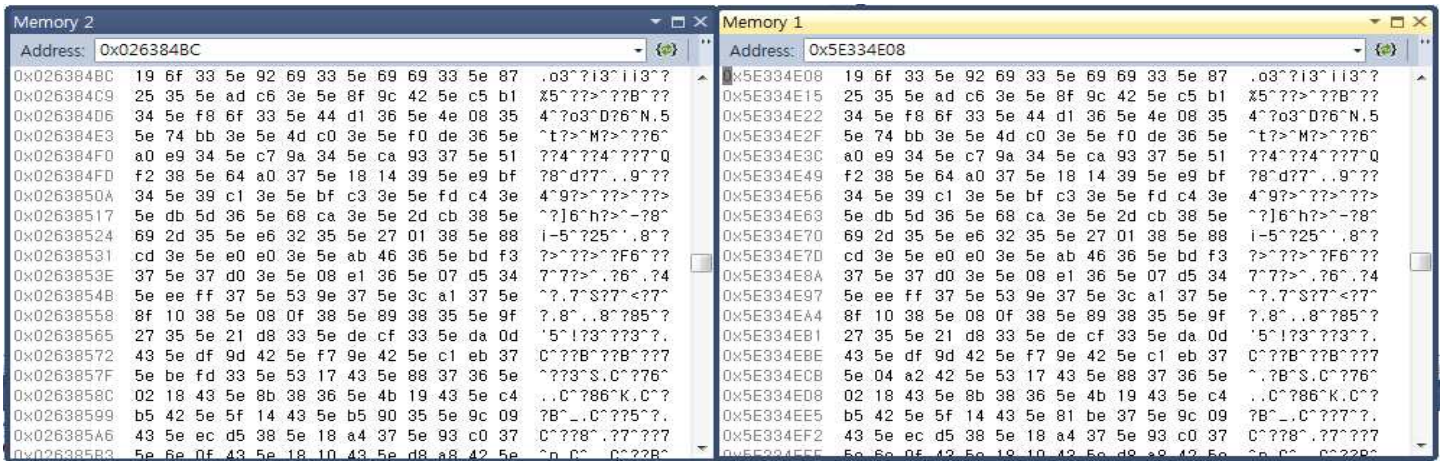


해당 위치로 이동하니 위와 같은 헥스 값이 나왔다. 예상이 맞다면 이 부분은 vTable의 위치이다. 이제 메모리상에서 패턴으로 vTable의 주소를 찾아서 이동해보겠다.



해당 패턴을 통해 찾아가 vTable의 주소(0x5E334E08)의 주소로 이동해보았다.

위에서 더블 포인터로 이동한 위치(0x026384BC)의 메모리와 vTable의 메모리가 일치하는 것을 볼 수 있다.

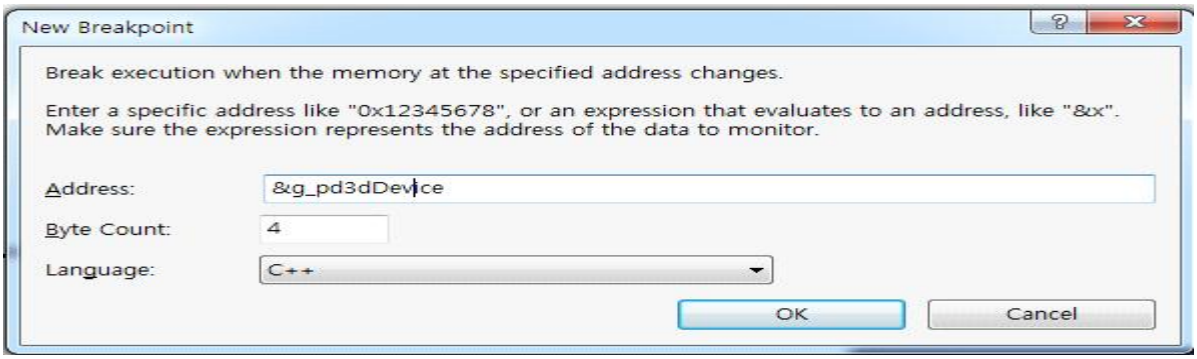


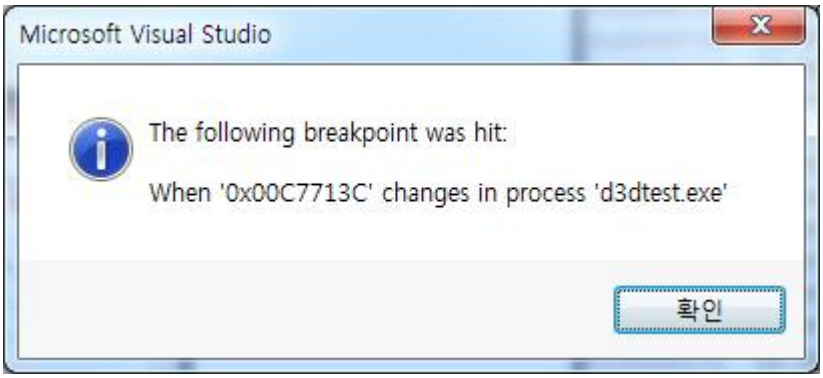
즉, **ppReturnedDeviceInterface 가 vTable의 주소를 가리키고 있다는 말이다.

```
HRESULT CreateDevice(
  [in]           UINT Adapter,
  [in]           D3DDEVTYPE DeviceType,
  [in]           HWND hFocusWindow,
  [in]           DWORD BehaviorFlags,
  [in, out]      D3DPRESENT_PARAMETERS *pPresentationParameters,
  [out, retval]  IDirect3DDevice9 **ppReturnedDeviceInterface
);
```

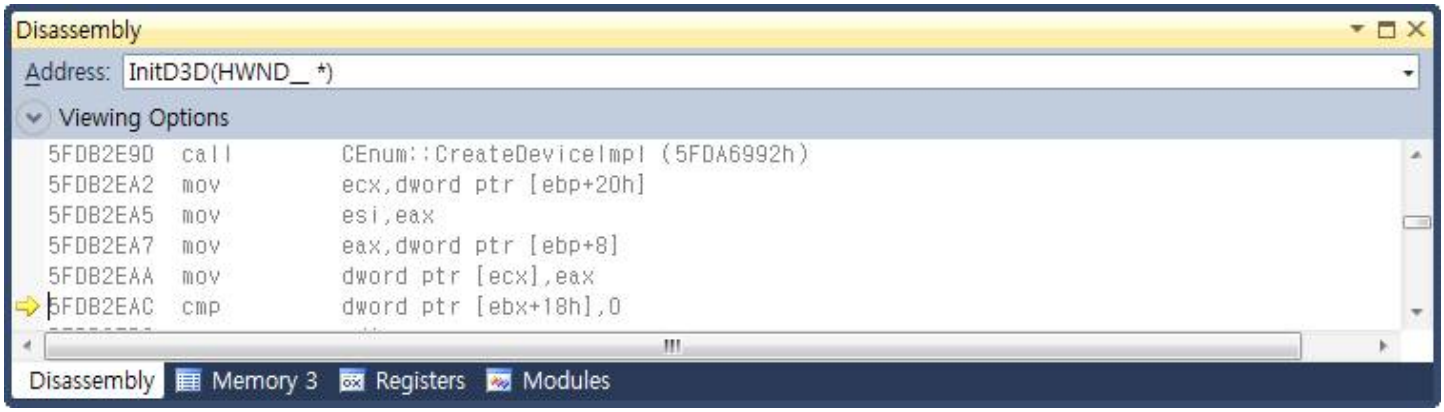
이 말은 CreateDevice 함수 내부 어딘가에서 vTable의 주소를 마지막 인자 값에 넘겨주는 부분이 위치한다는 말이고, 이 말은 vTable의 주소가 위치한 곳을 찾을 수 있다는 말이다.

일단 IDirect3DDevice9 인자 값이 들어가는 변수에 Write BP를 설정하여 해당 변수에 vTable 주소가 들어갈 때 BP가 걸리도록 한 뒤 디버깅을 해보자.





BP를 걸어두고 실행을 하니 해당 번지를 변조하다가 Break가 걸렸다는 메시지 창이 떴다.



해당 위치가 Break 된 위치이다.

0x5FDB2EAC mov dword ptr [ecx], eax

여기가 마지막 인자 값에 IDirect3DDevice9 Interface의 더블 포인터를 집어넣는 곳이다.

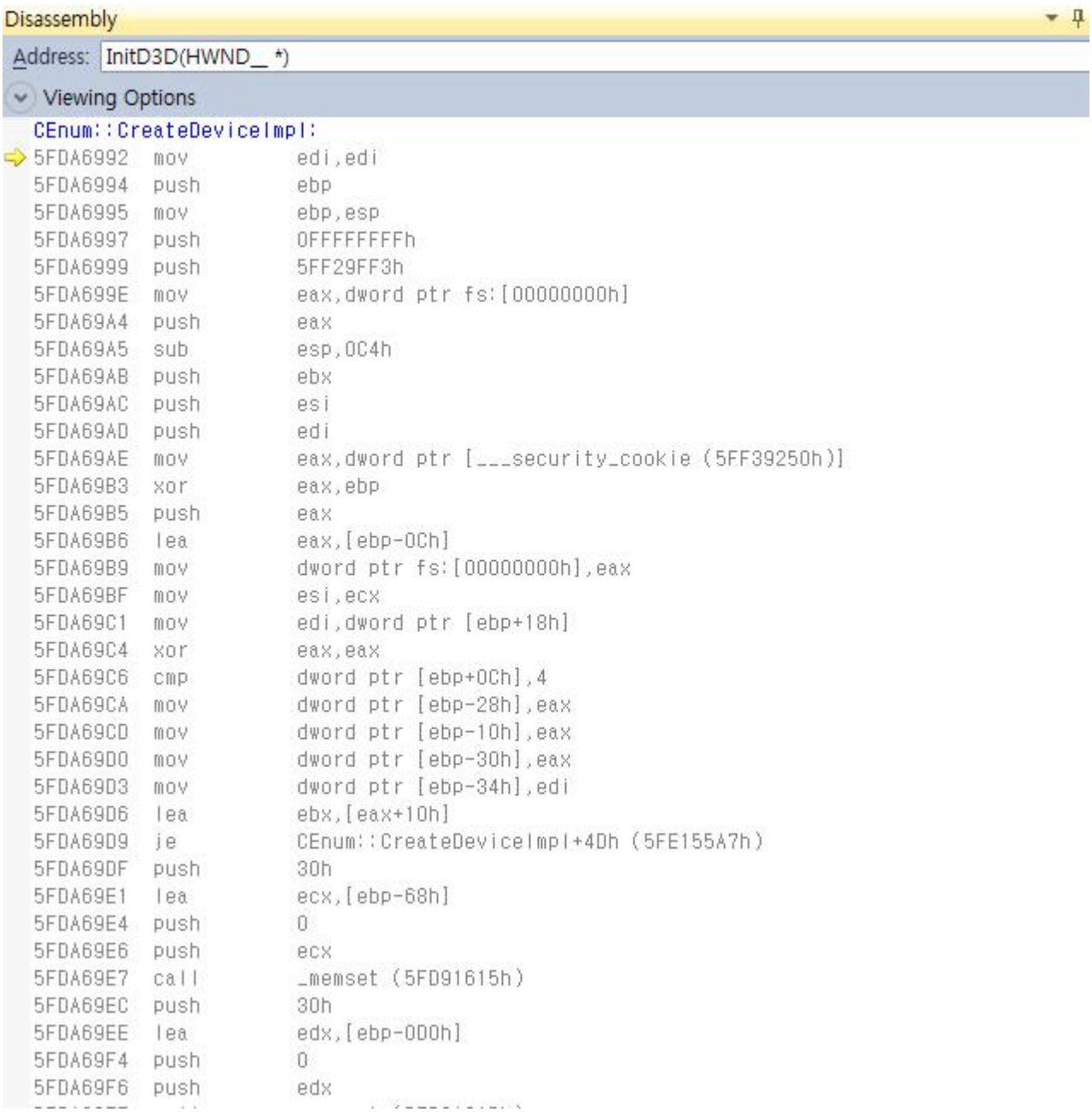
ECX 의 값에는 g_pd3dDevie(마지막 인자)의 주소가 들어가 있다.

eax에는Table의 주소가 들어가 있는 걸로 봐서는 가장 마지막에 호출 한 함수를 의심해볼 필요가 있다.

0x5FDB2E9D call CEnum::CreateDeviceImpl (5FDA6992h)

CEnum::CreateDeviceImpl 에 대해 msdn에 찾아보았지만 내용이 없어서 직접 디버깅을 해보았다. 해당 함수에 BP를 설정하고 다시 디버깅을 해보았다.

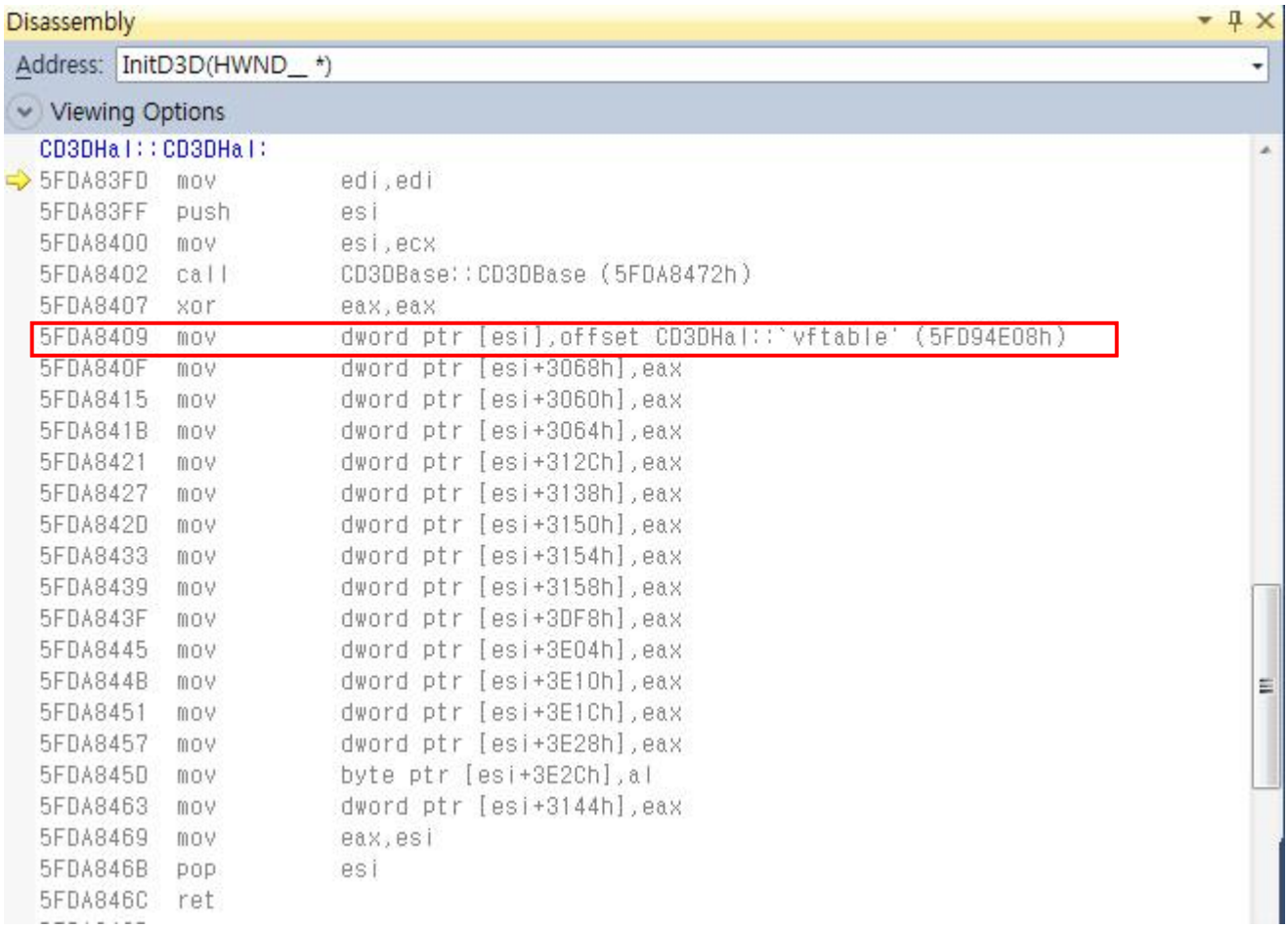
해당 함수 내부로 들어가 보자.



CEnum::CreateDeviceImpl 함수 내부에서 호출 하는 함수들을 보던 도중

```
5FDA6C4D  mov         ecx,eax
5FDA6C4F  call        CD3DHal::CD3DHal (5FDA83FDh)
5FDA6C54  mov         edi,eax
```

위와 같은 부분을 발견했다. CD3DHal::CD3DHal() 해당 함수는 23페이지에서 패턴의 위치를 찾을 때 해당 패턴이 들어가 있던 함수 이름이다. 아마 이 함수 안으로 들어가 보면 패턴의 위치가 나오며 vTable의 주소를 복사하는 부분이 있을 것이다.



23페이지에서 보았던 패턴 위치부분과 일치하며 5FDA8409h 번지를 보면 vTable 주소를 복사하는 부분을 찾을 수 있다. 해당 패턴은 CreateDevice 함수 내부에 존재하였다.

이 주소를 이용하면 IDriect3DDevice9 Interface 이기 때문에 원하는 함수를 호출 할 수 있게 된다. 해당 패턴은 vTable의 주소를 복사하는 부분을 찾아내어 vTable 주소만 저장하여 후킹 대상인 함수의 위치를 찾아내어 후킹 하는 원리였다.

3-3-3. vTable에서 함수의 위치를 어떻게 찾아 낼까?

vTable에는 가상 함수(Virtual Function)들이 선언된 순서대로 저장되어 있다. 그런 이유 때문에 19페이지에서 해당 방법으로 원하는 함수의 주소를 찾아내는 것을 설명 했었는데, 그렇다면 함수의 선언 순서는 어떻게 알 수 있을까? 이거에 대한 대답은 간단하다.
d3d9의 헤더파일을 참고하면 되기 때문이다.

아래 내용은 d3d9.h에서 IDriect3DDevice9 Interface의 정의 부분이다.
여기서 볼 점은 함수 선언 순서인데, EndScene 함수는 43번째에 선언된 것을 볼 수 있다.
즉, 위에서도 말했듯이 vTable의 구조는 선언된 함수 순서대로 저장되어 있다.

```
DECLARE_INTERFACE_(IDirect3DDevice9, IUnknown)
{
/*** IUnknown methods ***/
    STDMETHOD(QueryInterface)(THIS_ REFIID riid, void** ppvObj) PURE;
STDMETHOD_(ULONG,AddRef)(THIS) PURE;
STDMETHOD_(ULONG,Release)(THIS) PURE;

/*** IDirect3DDevice9 methods ***/
    STDMETHOD(TestCooperativeLevel)(THIS) PURE;
STDMETHOD_(UINT, GetAvailableTextureMem)(THIS) PURE;
STDMETHOD(EvictManagedResources)(THIS) PURE;
STDMETHOD(GetDirect3D)(THIS_ IDirect3D9** ppD3D9) PURE;
STDMETHOD(GetDeviceCaps)(THIS_ D3DCAPS9* pCaps) PURE;
STDMETHOD(GetDisplayMode)(THIS_ UINT iSwapChain,D3DDISPLAYMODE* pMode) PURE;
STDMETHOD(GetCreationParameters)(THIS_ D3DDEVICE_CREATION_PARAMETERS *pParameters) PURE;
STDMETHOD(SetCursorProperties)(THIS_ UINT XHotSpot,UINT YHotSpot,IDirect3DSurface9* pCursorBitmap) PURE;
STDMETHOD_(void, SetCursorPosition)(THIS_ int X,int Y,DWORD Flags) PURE;
STDMETHOD_(BOOL, ShowCursor)(THIS_ BOOL bShow) PURE;
STDMETHOD(CreateAdditionalSwapChain)(THIS_                                    D3DPRESENT_PARAMETERS*
pPresentationParameters,IDirect3DSwapChain9** pSwapChain) PURE;
STDMETHOD(GetSwapChain)(THIS_ UINT iSwapChain,IDirect3DSwapChain9** pSwapChain) PURE;
STDMETHOD_(UINT, GetNumberOfSwapChains)(THIS) PURE;
STDMETHOD(Reset)(THIS_ D3DPRESENT_PARAMETERS* pPresentationParameters) PURE;
STDMETHOD(Present)(THIS_ CONST RECT* pSourceRect,CONST RECT* pDestRect,HWND hDestWindowOverride,CONST
RGNDATA* pDirtyRegion) PURE;
STDMETHOD(GetBackBuffer)(THIS_        UINT        iSwapChain,UINT        iBackBuffer,D3DBACKBUFFER_TYPE
Type,IDirect3DSurface9** ppBackBuffer) PURE;
STDMETHOD(GetRasterStatus)(THIS_ UINT iSwapChain,D3DRASTER_STATUS* pRasterStatus) PURE;
STDMETHOD(SetDialogBoxMode)(THIS_ BOOL bEnableDialogs) PURE;
```

```
STDMETHOD_(void, SetGammaRamp)(THIS_ UINT iSwapChain,DWORD Flags,CONST D3DGAMMARAMP* pRamp) PURE;
STDMETHOD_(void, GetGammaRamp)(THIS_ UINT iSwapChain,D3DGAMMARAMP* pRamp) PURE;
STDMETHOD(CreateTexture)(THIS_ UINT Width,UINT Height,UINT Levels,DWORD Usage,D3DFORMAT Format,D3DPOOL
Pool,IDirect3DTexture9** ppTexture,HANDLE* pSharedHandle) PURE;
STDMETHOD(CreateVolumeTexture)(THIS_ UINT Width,UINT Height,UINT Depth,UINT Levels,DWORD Usage,D3DFORMAT
Format,D3DPOOL Pool,IDirect3DVolumeTexture9** ppVolumeTexture,HANDLE* pSharedHandle) PURE;
STDMETHOD(CreateCubeTexture)(THIS_ UINT EdgeLength,UINT Levels,DWORD Usage,D3DFORMAT Format,D3DPOOL
Pool,IDirect3DCubeTexture9** ppCubeTexture,HANDLE* pSharedHandle) PURE;
STDMETHOD(CreateVertexBuffer)(THIS_ UINT Length,DWORD Usage,DWORD FVF,D3DPOOL
Pool,IDirect3DVertexBuffer9** ppVertexBuffer,HANDLE* pSharedHandle) PURE;
STDMETHOD(CreateIndexBuffer)(THIS_ UINT Length,DWORD Usage,D3DFORMAT Format,D3DPOOL
Pool,IDirect3DIndexBuffer9** ppIndexBuffer,HANDLE* pSharedHandle) PURE;
STDMETHOD(CreateRenderTarget)(THIS_ UINT Width,UINT Height,D3DFORMAT Format,D3DMULTISAMPLE_TYPE
MultiSample,DWORD MultisampleQuality,BOOL Lockable,IDirect3DSurface9** ppSurface,HANDLE* pSharedHandle)
PURE;
STDMETHOD(CreateDepthStencilSurface)(THIS_ UINT Width,UINT Height,D3DFORMAT Format,D3DMULTISAMPLE_TYPE
MultiSample,DWORD MultisampleQuality,BOOL Discard,IDirect3DSurface9** ppSurface,HANDLE* pSharedHandle)
PURE;
STDMETHOD(UpdateSurface)(THIS_ IDirect3DSurface9* pSourceSurface,CONST RECT*
pSourceRect,IDirect3DSurface9* pDestinationSurface,CONST POINT* pDestPoint) PURE;
STDMETHOD(UpdateTexture)(THIS_ IDirect3DBaseTexture9* pSourceTexture,IDirect3DBaseTexture9*
pDestinationTexture) PURE;
STDMETHOD(GetRenderTargetData)(THIS_ IDirect3DSurface9* pRenderTarget,IDirect3DSurface9* pDestSurface)
PURE;
STDMETHOD(GetFrontBufferData)(THIS_ UINT iSwapChain,IDirect3DSurface9* pDestSurface) PURE;
STDMETHOD(StretchRect)(THIS_ IDirect3DSurface9* pSourceSurface,CONST RECT*
pSourceRect,IDirect3DSurface9* pDestSurface,CONST RECT* pDestRect,D3DTEXTUREFILTERTYPE Filter) PURE;
STDMETHOD(ColorFill)(THIS_ IDirect3DSurface9* pSurface,CONST RECT* pRect,D3DCOLOR color) PURE;
STDMETHOD(CreateOffscreenPlainSurface)(THIS_ UINT Width,UINT Height,D3DFORMAT Format,D3DPOOL
Pool,IDirect3DSurface9** ppSurface,HANDLE* pSharedHandle) PURE;
STDMETHOD(SetRenderTarget)(THIS_ DWORD RenderTargetIndex,IDirect3DSurface9* pRenderTarget) PURE;
STDMETHOD(GetRenderTarget)(THIS_ DWORD RenderTargetIndex,IDirect3DSurface9** ppRenderTarget) PURE;
STDMETHOD(SetDepthStencilSurface)(THIS_ IDirect3DSurface9* pNewZStencil) PURE;
STDMETHOD(GetDepthStencilSurface)(THIS_ IDirect3DSurface9** ppZStencilSurface) PURE;
STDMETHOD(BeginScene)(THIS) PURE;
STDMETHOD(EndScene)(THIS) PURE;
STDMETHOD(Clear)(THIS_ DWORD Count,CONST D3DRECT* pRects,DWORD Flags,D3DCOLOR Color,float Z,DWORD
Stencil) PURE;
STDMETHOD(SetTransform)(THIS_ D3DTRANSFORMSTATETYPE State,CONST D3DMATRIX* pMatrix) PURE;
STDMETHOD(GetTransform)(THIS_ D3DTRANSFORMSTATETYPE State,D3DMATRIX* pMatrix) PURE;
STDMETHOD(MultiplyTransform)(THIS_ D3DTRANSFORMSTATETYPE,CONST D3DMATRIX*) PURE;
STDMETHOD(SetViewport)(THIS_ CONST D3DVIEWPORT9* pViewport) PURE;
STDMETHOD(GetViewport)(THIS_ D3DVIEWPORT9* pViewport) PURE;
STDMETHOD(SetMaterial)(THIS_ CONST D3DMATERIAL9* pMaterial) PURE;
```

```
STDMETHOD(GetMaterial)(THIS_ D3DMATERIAL9* pMaterial) PURE;
STDMETHOD(SetLight)(THIS_ DWORD Index,CONST D3DLIGHT9*) PURE;
STDMETHOD(GetLight)(THIS_ DWORD Index,D3DLIGHT9*) PURE;
STDMETHOD(LightEnable)(THIS_ DWORD Index,BOOL Enable) PURE;
STDMETHOD(GetLightEnable)(THIS_ DWORD Index,BOOL* pEnable) PURE;
STDMETHOD(SetClipPlane)(THIS_ DWORD Index,CONST float* pPlane) PURE;
STDMETHOD(GetClipPlane)(THIS_ DWORD Index,float* pPlane) PURE;
STDMETHOD(SetRenderState)(THIS_ D3DRENDERSTATETYPE State,DWORD Value) PURE;
STDMETHOD(GetRenderState)(THIS_ D3DRENDERSTATETYPE State,DWORD* pValue) PURE;
STDMETHOD(CreateStateBlock)(THIS_ D3DSTATEBLOCKTYPE Type,IDirect3DStateBlock9** ppSB) PURE;
STDMETHOD(BeginStateBlock)(THIS) PURE;
STDMETHOD(EndStateBlock)(THIS_ IDirect3DStateBlock9** ppSB) PURE;
STDMETHOD(SetClipStatus)(THIS_ CONST D3DCLIPSTATUS9* pClipStatus) PURE;
STDMETHOD(GetClipStatus)(THIS_ D3DCLIPSTATUS9* pClipStatus) PURE;
STDMETHOD(GetTexture)(THIS_ DWORD Stage,IDirect3DBaseTexture9** ppTexture) PURE;
STDMETHOD(SetTexture)(THIS_ DWORD Stage,IDirect3DBaseTexture9* pTexture) PURE;
STDMETHOD(GetTextureStageState)(THIS_ DWORD Stage,D3DTEXTURESTAGESTATETYPE Type,DWORD* pValue) PURE;
STDMETHOD(SetTextureStageState)(THIS_ DWORD Stage,D3DTEXTURESTAGESTATETYPE Type,DWORD Value) PURE;
STDMETHOD(GetSamplerState)(THIS_ DWORD Sampler,D3DSAMPLERSTATETYPE Type,DWORD* pValue) PURE;
STDMETHOD(SetSamplerState)(THIS_ DWORD Sampler,D3DSAMPLERSTATETYPE Type,DWORD Value) PURE;
STDMETHOD(ValidateDevice)(THIS_ DWORD* pNumPasses) PURE;
STDMETHOD(SetPaletteEntries)(THIS_ UINT PaletteNumber,CONST PALETTEENTRY* pEntries) PURE;
STDMETHOD(GetPaletteEntries)(THIS_ UINT PaletteNumber,PALETTEENTRY* pEntries) PURE;
STDMETHOD(SetCurrentTexturePalette)(THIS_ UINT PaletteNumber) PURE;
STDMETHOD(GetCurrentTexturePalette)(THIS_ UINT *PaletteNumber) PURE;
STDMETHOD(SetScissorRect)(THIS_ CONST RECT* pRect) PURE;
STDMETHOD(GetScissorRect)(THIS_ RECT* pRect) PURE;
STDMETHOD(SetSoftwareVertexProcessing)(THIS_ BOOL bSoftware) PURE;
STDMETHOD_(BOOL, GetSoftwareVertexProcessing)(THIS) PURE;
STDMETHOD(SetNPatchMode)(THIS_ float nSegments) PURE;
STDMETHOD_(float, GetNPatchMode)(THIS) PURE;
STDMETHOD(DrawPrimitive)(THIS_ D3DPRIMITIVETYPE PrimitiveType,UINT StartVertex,UINT PrimitiveCount)
PURE;
STDMETHOD(DrawIndexedPrimitive)(THIS_ D3DPRIMITIVETYPE,INT BaseVertexIndex,UINT MinVertexIndex,UINT
NumVertices,UINT startIndex,UINT primCount) PURE;
STDMETHOD(DrawPrimitiveUP)(THIS_ D3DPRIMITIVETYPE PrimitiveType,UINT PrimitiveCount,CONST void*
pVertexStreamZeroData,UINT VertexStreamZeroStride) PURE;
STDMETHOD(DrawIndexedPrimitiveUP)(THIS_ D3DPRIMITIVETYPE PrimitiveType,UINT MinVertexIndex,UINT
NumVertices,UINT PrimitiveCount,CONST void* pIndexData,D3DFORMAT IndexDataFormat,CONST void*
pVertexStreamZeroData,UINT VertexStreamZeroStride) PURE;
STDMETHOD(ProcessVertices)(THIS_ UINT SrcStartIndex,UINT DestIndex,UINT
VertexCount,IDirect3DVertexBuffer9* pDestBuffer,IDirect3DVertexDeclaration9* pVertexDecl,DWORD Flags)
PURE;
STDMETHOD(CreateVertexDeclaration)(THIS_ CONST D3DVERTEXELEMENT9*
```

```
pVertexElements,IDirect3DVertexDeclaration9** ppDecl) PURE;
STDMETHOD(SetVertexDeclaration)(THIS_ IDirect3DVertexDeclaration9* pDecl) PURE;
STDMETHOD(GetVertexDeclaration)(THIS_ IDirect3DVertexDeclaration9** ppDecl) PURE;
STDMETHOD(SetFVF)(THIS_ DWORD FVF) PURE;
STDMETHOD(GetFVF)(THIS_ DWORD* pFVF) PURE;
STDMETHOD(CreateVertexShader)(THIS_ CONST DWORD* pFunction,IDirect3DVertexShader9** ppShader) PURE;
STDMETHOD(SetVertexShader)(THIS_ IDirect3DVertexShader9* pShader) PURE;
STDMETHOD(GetVertexShader)(THIS_ IDirect3DVertexShader9** ppShader) PURE;
STDMETHOD(SetVertexShaderConstantF)(THIS_ UINT StartRegister,CONST float* pConstantData,UINT
Vector4fCount) PURE;
STDMETHOD(GetVertexShaderConstantF)(THIS_ UINT StartRegister,float* pConstantData,UINT Vector4fCount)
PURE;
STDMETHOD(SetVertexShaderConstantI)(THIS_ UINT StartRegister,CONST int* pConstantData,UINT
Vector4iCount) PURE;
STDMETHOD(GetVertexShaderConstantI)(THIS_ UINT StartRegister,int* pConstantData,UINT Vector4iCount)
PURE;
STDMETHOD(SetVertexShaderConstantB)(THIS_ UINT StartRegister,CONST BOOL* pConstantData,UINT BoolCount)
PURE;
STDMETHOD(GetVertexShaderConstantB)(THIS_ UINT StartRegister,BOOL* pConstantData,UINT BoolCount) PURE;
STDMETHOD(SetStreamSource)(THIS_ UINT StreamNumber,IDirect3DVertexBuffer9* pStreamData,UINT
OffsetInBytes,UINT Stride) PURE;
STDMETHOD(GetStreamSource)(THIS_ UINT StreamNumber,IDirect3DVertexBuffer9** ppStreamData,UINT*
pOffsetInBytes,UINT* pStride) PURE;
STDMETHOD(SetStreamSourceFreq)(THIS_ UINT StreamNumber,UINT Setting) PURE;
STDMETHOD(GetStreamSourceFreq)(THIS_ UINT StreamNumber,UINT* pSetting) PURE;
STDMETHOD(SetIndices)(THIS_ IDirect3DIndexBuffer9* pIndexData) PURE;
STDMETHOD(GetIndices)(THIS_ IDirect3DIndexBuffer9** ppIndexData) PURE;
STDMETHOD(CreatePixelShader)(THIS_ CONST DWORD* pFunction,IDirect3DPixelShader9** ppShader) PURE;
STDMETHOD(SetPixelShader)(THIS_ IDirect3DPixelShader9* pShader) PURE;
STDMETHOD(GetPixelShader)(THIS_ IDirect3DPixelShader9** ppShader) PURE;
STDMETHOD(SetPixelShaderConstantF)(THIS_ UINT StartRegister,CONST float* pConstantData,UINT
Vector4fCount) PURE;
STDMETHOD(GetPixelShaderConstantF)(THIS_ UINT StartRegister,float* pConstantData,UINT Vector4fCount)
PURE;
STDMETHOD(SetPixelShaderConstantI)(THIS_ UINT StartRegister,CONST int* pConstantData,UINT Vector4iCount)
PURE;
STDMETHOD(GetPixelShaderConstantI)(THIS_ UINT StartRegister,int* pConstantData,UINT Vector4iCount) PURE;
STDMETHOD(SetPixelShaderConstantB)(THIS_ UINT StartRegister,CONST BOOL* pConstantData,UINT BoolCount)
PURE;
STDMETHOD(GetPixelShaderConstantB)(THIS_ UINT StartRegister,BOOL* pConstantData,UINT BoolCount) PURE;
STDMETHOD(DrawRectPatch)(THIS_ UINT Handle,CONST float* pNumSegs,CONST D3DRECTPATCH_INFO*
pRectPatchInfo) PURE;
STDMETHOD(DrawTriPatch)(THIS_ UINT Handle,CONST float* pNumSegs,CONST D3DTRIPATCH_INFO* pTriPatchInfo)
PURE;
```

```
STDMETHOD(DeletePatch)(THIS_ UINT Handle) PURE;
STDMETHOD(CreateQuery)(THIS_ D3DQUERYTYPE Type,IDirect3DQuery9** ppQuery) PURE;

#ifdef D3D_DEBUG_INFO
D3DDEVICE_CREATION_PARAMETERS CreationParameters;
D3DPRESENT_PARAMETERS PresentParameters;
D3DDISPLAYMODE DisplayMode;
D3DCAPS9 Caps;

UINT AvailableTextureMem;
UINT SwapChains;
UINT Textures;
UINT VertexBuffers;
UINT IndexBuffers;
UINT VertexShaders;
UINT PixelShaders;

D3DVIEWPORT9 Viewport;
D3DMATRIX ProjectionMatrix;
D3DMATRIX ViewMatrix;
D3DMATRIX WorldMatrix;
D3DMATRIX TextureMatrices[8];

DWORD FVF;
UINT VertexSize;
DWORD VertexShaderVersion;
DWORD PixelShaderVersion;
BOOL SoftwareVertexProcessing;

D3DMATERIAL9 Material;
D3DLIGHT9 Lights[16];
BOOL LightsEnabled[16];

D3DGAMMARAMP GammaRamp;
RECT ScissorRect;
BOOL DialogBoxMode;
#endif
};
```

위와 같이 vTable은 선언된 순서대로 들어가 있기 때문에 후킹 하기 원하는 함수가 있다면 헤더파일에서 선언된 순서를 알아내고 EndScene 처럼 후킹하면 된다.

4. 끝맺음

아직 실력이 많이 부족하고 글 쓰는 재주까지 없지만 용기 내어 문서를 제작해보았습니다. 해당 문서를 통해 d3d hooking를 악용하는 기술들을 방어하는 기술에 대해 정보 공유를 해보았으면 좋겠습니다. 좀 더 공부하여 d3d hooking(detours hook)방어법에 대한 문서를 가지고 다시 나타나겠습니다.

이 문서에서 분석한 d3d hooking 의 key point는 detours hook, 즉 vTable의 주소를 찾아낸 것과 메모리를 변조하여 프로그램의 흐름을 변경하였습니다. vTable의 주소위치를 은폐하고 메모리 변조에 대해 감지하고 확인되지 않은 dll 의 injection을 차단 할 수 있는 쪽으로 막을 수 있지 않을까 조심스레 생각해봅니다. 한번 연구 해보겠습니다.

문서에서 잘못 된 부분이나 조언을 해주실 분은 제 이메일이나 블로그에 글을 남겨주시면 최대한 성실하게 답변 및 수렴하겠습니다.

끝으로, 이 문서를 악용하는 일에 사용되지 않았으면 좋겠습니다.

감사합니다.

by NullBr4in

블로그 : http://ajlab.tistory.com/
E-Mail : NullBr4in@gmail.com

2012-1-20

· 5. 전체 소스

- Main.cpp

```
#include "d3dhooks.h"
#include "common.h"

BOOL WINAPI DllMain(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpReserved )
{
	switch( fdwReason )
	{
	case DLL_PROCESS_ATTACH:
		{
			DisableThreadLibraryCalls(hinstDLL);
			StartD3DHooks();
				return true
			break
		}
	case DLL_PROCESS_DETACH:
		{
			MessageBox(NULL,L"detach dll!", L"ok", MB_OK);
			break
		}
	}
	return TRUE;
}
```

- d3dhooks.h

```
#include "common.h"

class DXGH {
public:
        static HRESULT WINAPI h_EndScene(LPDIRECT3DDEVICE9 pDevice);
        void DrawRect (LPDIRECT3DDEVICE9 Device_t, int X, int Y, int L, int H, D3DCOLOR color);

};

int StartD3DHooks();

typedef HRESULT(WINAPI *EndScene_t)(LPDIRECT3DDEVICE9 pDevice);
extern DXGH   DXGameHook;
```

- d3dhooks.cpp

```
#include "common.h"
#include "d3dhooks.h"

#define ENDSCENE 42

DXGH DXGameHook;

typedef HRESULT(__stdcall* EndScene_t)(LPDIRECT3DDEVICE9);

EndScene_t org_EndScene;

const D3DCOLOR txtPink = D3DCOLOR_ARGB(255, 255, 0, 255);


void *DetourFunc(BYTE *src, const BYTE *dst, const int len)
{
	BYTE *jmp = (BYTE*)malloc(len+5);
	DWORD dwback;
	VirtualProtect(src, len, PAGE_READWRITE, &dwback);
	memcpy(jmp, src, len); jmp += len;
	jmp[0] = 0xE9;
	*(DWORD*)(jmp+1) = (DWORD)(src+len - jmp) - 5;
	src[0] = 0xE9;
	*(DWORD*)(src+1) = (DWORD)(dst - src) - 5;
	VirtualProtect(src, len, dwback, &dwback);
	return (jmp-len);
}

bool bDataCompare(const BYTE* pData, const BYTE* bMask, const char* szMask)
{
	for(;*szMask;++szMask,++pData,++bMask)
		if(*szMask=='x' && *pData!=*bMask )
			return false
	return (*szMask) == NULL;
}

DWORD FindPattern(DWORD dwAddress,DWORD dwLen,BYTE *bMask,char * szMask)
{
	for(DWORD i=0; i < dwLen; i++)
		if( bDataCompare( (BYTE*)( dwAddress+i ),bMask,szMask) )
			return (DWORD)(dwAddress+i);
```

```
        return 0;
}

void DXGH::DrawRect (LPDIRECT3DDEVICE9 Device_t, int X, int Y, int L, int H, D3DCOLOR color)
{
        D3DRECT rect = {X, Y, X+L, Y+H};

        Device_t->Clear(1, &rect, D3DCLEAR_TARGET, color, 0, 0);
}

HRESULT WINAPI DXGH::h_EndScene(LPDIRECT3DDEVICE9 pDevice)
{
        DXGameHook.DrawRect(pDevice, 10, 10, 200, 200, txtPink);

        return org_EndScene(pDevice);
}

LPDIRECT3D9   g_pD3D = NULL;
LPDIRECT3DDEVICE9 g_pd3dDevice = NULL;


int StartD3DHooks()
{

        DWORD D3DPattern,*vTable, DXBase=NULL;
        DXBase = (DWORD)LoadLibraryA("d3d9.dll");

        while(!DXBase);
        {
                D3DPattern                =                FindPattern(DXBase,                0x128000,
(PBYTE)"₩xC7₩x06₩x00₩x00₩x00₩x00₩x89₩x86₩x00₩x00₩x00₩x00₩x89₩x86", "xx????xx????xx");
        }

        if(D3DPattern)
        {
                memcpy(&vTable,(void *)(D3DPattern+2),4);
                org_EndScene                                                                =
(EndScene_t)DetourFunc((PBYTE)vTable[ENDSCENE],(PBYTE)DXGameHook.h_EndScene,5);

        }
        return 0;
}
```