D3D9 Hooking Tutorial

by purple.d1amond

Inhalt

Credits && Thanks

GameDeception.net – Azorbix
MP-hacks.net – Gordon
Sourcehack.de – root

Einleitung

In diesem Tutorial werde ich versuchen zu erklären, wie ihr euch einen Super-1337 D3D Hack für euer Lieblingsgame coded. Dabei versuche ich, das Tutorial einerseits so ausführlich und verständlich wie möglich zu halten, andererseits auch dafür zu sorgen, dass man den Code nicht einfach per Copy&Paste in seinen Compiler reinhacken kann. Am Ende des Tutorials solltet ihr zu so etwas fähig sein (ich weiß, Eigenlob stinkt xD):

Diamondhack - Operation 7 - Menü ist leider etwas ungenau =)

Vorraussetzungen

Also als erstes solltet ihr natürlich C++ programmieren können, am besten ist es wenn ihr schon den ein oder anderen Trainer gecoded habt. Gewisse Kentnisse im Bezug auf DLL-Programmierung wären auch nicht schlecht. Weiterhin:

[+] Visual C++ Compiler. Express Edition (umsonsts):
http://msdn2.microsoft.com/en-us/express/aa975050.aspx
Es kann sein dass ihr zu der Express Edition noch das Platform SDK von Microsoft benötigt um überhaupt irgendwas compilen zu können. Ich persönlich benutz das alte VC++ 6.0

[+] DirectX SDK. http://www.microsoft.com/downloads/details.aspx?FamilyId=5493F76A-6D37-478D-BA17-28B1CCA4865A&displaylang=en
Installieren und in VC++ die Verzeichnisse Include, Bin und Lib einfügen. Extras->Optionen->Projekte und Projektmappen->VC++-Verzeichnisse: unter Librarys added ihr das 'lib' Verzeichnis vom SDK, unter Binarys added ihr das 'bin' Verzeichnis und unter 'Includes' added ihr das 'Include' Verzeichnis.

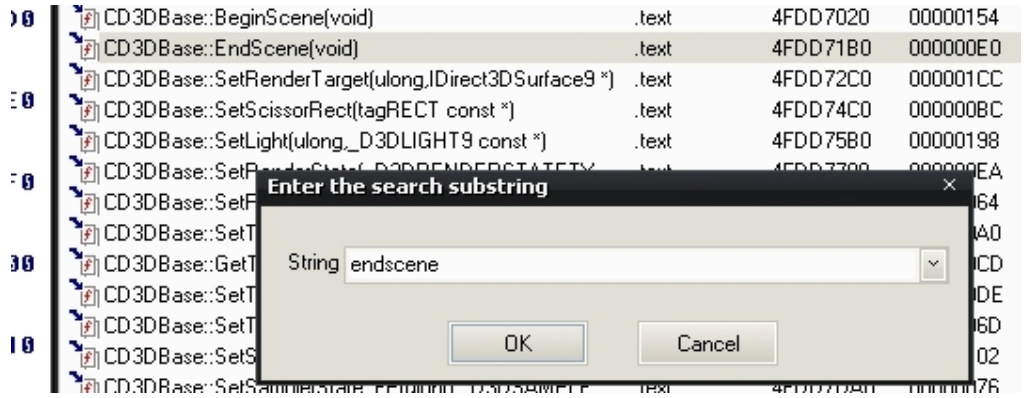
[+] IDA Pro. http://www.brothersoft.com/ida-pro-download-69919.html

[+] Viel Zeit und jede menge kühle Coke xD

VC++ sollte jetzt ohne Probleme eure Projekte compilen können und das SDK richtig eingebunden haben.

Vorbereitungen

So, jetzt können wir endlich anfangen. Als erstes ein bisschen Theorie zu DirectX (gibt auch jede Menge Tutorials im Internet dazu).
Die für uns wichtigste Funktion die D3D9 bereitstellt, ist EndScene. Diese Funktion wird pro Bild das dargestellt wird genau 1x aufgerufen. Und zwar immer am Ende (wie der Name ja sagt).
In dieser Funktion werden wir auch unsere ganze Zeichenarbeit (das Menü) verrichten. Unser Ziel ist es also, diese Funktion zu hooken. Was Hooks sind, lest ihr euch am besten bei Wikipedia durch (sucht nach Hook). Über diese Funktion holen wir uns den D3D DevicePointer, mit dem wir dann Zugriff auf die Zeichenfunktionen von D3D bekommen.
Also, dann schmeißen wir mal IDA Pro an um die Funktion ausfindig zu machen. Dazu öffnet ihr mit IDA die Datei d3d9.dll aus dem Windows/System32 Ordner. Aktiviert beim Anfangsdialog alle Haken um die DLL komplett zu laden. Jetzt müsst ihr ein bisschen warten bis IDA die ganze Library analysiert hat.
Oben in der Mitte seht ihr den Tab Functions. Den aktiviert ihr. Jetzt drückt ihr im Hauptmenü von IDA auf Search->Search und tippt EndScene ein. Ihr solltet hier landen:



EndScene ist also offensichtlich eine Methode der Klasse CD3DBase. Wir sehen hier auch die Adresse an der unsere Funktion steht: 0x4FDD71B0 Das ist erstmal das wichtigste für uns. Jetzt können wir die Funktion mithilfe von Detours hooken. Was Detours genau sind und wozu man sie benutzen kann versucht ihr am besten über Google herauszufinden. Grob funktionieren Detours so: Wenn ich einen Detour (=Umweg) auf eine Funktion anwende, dann wird die Funktion nicht mehr normal ausgeführt, sondern zu mir in das Programm umgeleitet. Dann hab ich vollen Zugriff auf die Funktion.

Jetzt erstmal ne Viertelstunde Kaffeepause, dann geht's ans coden =)

Die DLL Coden

Öffnet jetzt euer VC++ und legt ein neues Dynamic Link Library Projekt an. Stellt sicher, dass ihr ein komplett leeres Projekt bekommt. Dort fügt ihr dann folgenden Code ein:

```
int WINAPI DllMain(HINSTANCE hInst,DWORD reason,LPVOID reserved)
{
	switch(reason)
	{
	case DLL_PROCESS_ATTACH:
		// Hier kommt unser Code rein
		break;
	}
	return true;
}
```

Das ist Standard für jede DLL die in ein Programm injected wird. Sollte jeder von euch schon einmal gesehen haben. Unser Code wird ausgeführt sobald die DLL an unseren Prozess angehängt wird. Compilen können wir das so noch nicht, dazu müssen wir obendrüber ersteinmal die wichtigsten Header mit einbinden.

```
#include <windows.h>
#include <cstdio>
#include <d3d9.h>
#include <d3dx9.h>
```

Jetzt habt ihr auch gleichzeitig die DirectX funktionen includiert. Wenn ihr das SDK richtig in VC++ eingestellt habt sollte das jetzt auch ohne Fehler zu compilen sein. Eventuell müsst ihr folgendes noch obendrüber einfügen:

```
#pragma once
#pragma comment(lib, "d3d9.lib")
#pragma comment(lib, "d3dx9.lib")
```

Das #pragma once sagt dem Compiler dass alles nur einmal eingebunden wird. Ich bekomm sonst oft Fehler... Spätestens jetzt sollte das fehlerfrei compilebar sein. Wenn nicht, guckt euch im Internet um und stellt sicher, dass ihr das SDK richtig in VC++ eingestellt habt.

Jetzt können wir uns den Hook genauer anschauen. Am besten ist es, wenn man das ganze in einem Thread macht, damit man sicher gehen kann dass alles läuft. Also erstellen wir einen neuen Thread:

```
void InitHook()
{
```

```
	HMODULE hModule = NULL;
	while( !hModule )
	{
		hModule = GetModuleHandleA( "d3d9.dll" );	// Handle zur DLL holen
		Sleep( 100 );  // 100ms warten
	}
}
```

So da ist jetzt schon einiges vorweggenommen, aber es sollte klar sein was hier passiert. Eigentlich brauchen wir das Handle nicht mehr, aber so stellen wir sicher, dass die DLL auch wirklich geladen wurde und dass D3D9 überhaupt läuft. An dieser Stelle müssen wir jetzt die DetourFunction einfügen. Wer will kann die von Microsoft benutzen, ich nehm die von GameDeception.

```
void *DetourFunc(BYTE *src, const BYTE *dst, const int len) // credits to gamedeception
{
	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);
}
```

Bevor wir jetzt ans detouren gehen, gibt's noch einiges nachzuholen. Unser Thread muss natürlich auch gestartet werden. Deswegen adden wir jetzt in DLLMain:

```
CreateThread(0, 0, (LPTHREAD_START_ROUTINE) InitHook, 0, 0, 0);
```

Mehr Infos zu Threads gibt's bei Google =)
Jetzt aber zu unserer EndScene Funktion. Als erstes brauchen wir eine Typendefinition, damit der Compiler bei dem Detour später nicht meckert. Die EndScene Funktion schaut folgendermaßen aus:

```
HRESULT __stdcall EndScene(LPDIRECT3DDEVICE9 pDevice)
```

Wir sehen hier auch, dass das Argument vom Typ LPDIRECT3DDEVICE9 ist. Hier können wir also auf den Pointer zum D3D Device zugreifen. Ich nenn ihn jetzt einmal pDevice. Nähere Information zu der Funktion gibt's in der d3d9.h (dort ist sie deklariert) oder im Internet.
Jetzt zurück zu unserem Typedef. Fügt folgendes in euer Projekt ein:

```
typedef HRESULT(__stdcall* EndScene_t)(LPDIRECT3DDEVICE9);
EndScene_t pEndScene;
```

Wozu wir das brauchen wird gleich klar. Typedefs sind C-Standart, ihr solltet wissen was diese zwei Zeilen bewirken (Wenn nicht: wie immer -> Google).

Jetzt können wir die Funktion Detouren. Allgemein schaut das immer so aus:

```
PointerOriginalFunktion = (TYPECAST) DetourFunc ( OffsetOriginalFunktion, UnsereFunktion, Länge );
```

Das ganze wird klar wenn ich das auf EndScene anwende. Bei der Microsoft Detour variante brauchen wir keine Länge anzugeben. Hier verwenden wir einfach immer 5, das ist das Minimum. Auf unsere Funktion angewandt:

```
pEndScene = ( EndScene_t )DetourFunc((PBYTE)  0x4FDD71B0,(PBYTE)hkEndScene, 5);
```

Die Zeile fügt ihr in unseren Thread ein. Vergleicht sie dann mit dem Typedef von gerade eben, dann versteht ihr das ganze gleich besser. Allerdings taucht hier jetzt hkEndScene auf, und die ist ja noch nirgendwo deklariert. Also holen wir das nach. Fügt am Anfang die gehookte Funktion ein:

```
HRESULT __stdcall hkEndScene(LPDIRECT3DDEVICE9 pDevice)
{
	return pEndScene(pDevice);
}
```

Das ist jetzt die Funktion in der wir Zeichnen können. Wie ihr seht wird zur Originalfunktion returned. Aber das mit den Detours ist ja längst klar ;D
An dieser Stelle sind wir eigentlich fertig, unser Ziel den D3D DevicePointer zu bekommen haben wir hiermit erreicht. Jetzt wollen wir aber gleich etwas Zeichnen um unseren Hook zu testen.

Rechtecke && Schrift Zeichnen

An dieser Stelle paste ich einfach mal meine Funktion zum Zeichnen von Rechtecken:

```
void 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); // bei Google gibt's näheres
}
```

Jetzt könnt ihr in der gehookten EndScene Funktion z.B. folgende Zeile einfügen:

```
DrawRect ( pDevice, 10, 10, 200, 200, txtPink);
```

txtPink ist bei euch noch nicht deklariert. Also holt das nach:

```
const D3DCOLOR txtPink = D3DCOLOR_ARGB(255, 255, 0, 255); // Alpha, Rot, Grün, Blau
```

Wenn ihr die fertige DLL jetzt z.B. in Bioshock injected, seht ihr hoffentlich links oben ein großes Pinkes Rechteck. Das ist der Beweis das euer Hook funktioniert ;D
Als Injector könnt ihr z.B. den OSInject von KN4CK3R nehmen.

Jetzt zur Schrift. Als erstes Erstellen wir einen FontPointer:

```
ID3DXFont *pFont;
```

Dann muss unsere Schrift initialisiert werden. Dazu rufen wir folgende Funktion auf:

```
D3DXCreateFont(pDevice, 14, 0, FW_NORMAL, 1, 0, DEFAULT_CHARSET,  OUT_DEFAULT_PRECIS,
ANTIALIASED_QUALITY, DEFAULT_PITCH | FF_DONTCARE, "Arial", &g_pFont );
```

Lest euch mal das hier durch: http://www.drunkenhyena.com/cgi-bin/view_cpp_article.pl?chapter=3;article=17
Da gibt's genauere Informationen. Wir callen diese Funktion in unserer gehookten EndScene, da wir ja nur da auf den DevicePointer zugreifen können. Aber Vorsicht: Die Funktion muss nur 1x aufgerufen werden. Wäre unsinnig die Schrift jedes mal neu zu initialisieren. Also sorgt dafür, dass die Funktion nur 1x gecalled wird.

Jetzt hier zur eigentlich DrawFont Funktion:

```
void DrawFont (int X, int Y, D3DCOLOR Color, char *format, ...)
{
	char buffer[256];
	va_list args;			// deswegen: #include <cstdio>
	va_start (args, format);
	vsprintf (buffer,format, args);
		RECT FontRect = { X, Y, X + 120, Y + 16 };
		g_pFont->DrawText( NULL, buffer, -1, &FontRect, DT_NOCLIP , Color );		// Zeichnen
	va_end (args);
}
```

Mit va_lists solltet ihr euch auskennen und somit die Funktion verstehen können. In EndScene:

```
DrawFont ( 300, 50, txtPink, "I <3 purple" );
```

Compilen, Injecten und staunen (hoffentlich xD). In CSS verursacht die Font Probleme, da braucht ihrs gar nicht erst zu versuchen.

Wie man ein Menü programmiert erklär ich nicht, wer soweit gekommen ist schafft das von alleine. Hier jetzt noch kurz warum ich va_list mit eingefügt hab:

```
DrawFont ( 300, 50, txtPink, "Wallhack %s", charWallhack );
```

Damit könnt ihr euch die Werte von euren Variablen ausgeben lassen. Praktisch fürs Menü.

Memoryhacks aus der DLL heraus

Dazu erzähl ich jetzt nicht viel, nur: Wenn wir schon im Prozess injected sind brauchen wir kein WriteProcessMemory mehr. Jetzt nutzen wir folgende Funktion:

```
void WriteMem(DWORD dwOffset, DWORD dwValue, int len)
{
	unsigned long Protection;
		VirtualProtect((void*)dwOffset, 1, PAGE_READWRITE, &Protection);
		memcpy((void*)dwOffset, (const void*)dwValue, len);
		VirtualProtect((void*)dwOffset, 1, Protection, 0);
}
```

Informiert euch mit Google darüber was hier passiert.

Chams

So, und gleich geht's weiter mit Chams. Zuerst müsst ihr diese Funktion von GameDeception adden:

```
HRESULT GenerateTexture(IDirect3DDevice9 *pD3Ddev, IDirect3DTexture9 **ppD3Dtex, DWORD
colour32)
{
	if( FAILED(pD3Ddev->CreateTexture(8, 8, 1, 0, D3DFMT_A4R4G4B4, D3DPOOL_MANAGED,
ppD3Dtex, NULL)) )
		return E_FAIL;

	WORD colour16 =	((WORD)((colour32>>28)&0xF)<<12)
			|(WORD)(((colour32>>20)&0xF)<<8)
			|(WORD)(((colour32>>12)&0xF)<<4)
			|(WORD)(((colour32>>4)&0xF)<<0);

	D3DLOCKED_RECT d3dlr;
	(*ppD3Dtex)->LockRect(0, &d3dlr, 0, 0);
	WORD *pDst16 = (WORD*)d3dlr.pBits;

	for(int xy=0; xy < 8*8; xy++)
		*pDst16++ = colour16;

	(*ppD3Dtex)->UnlockRect(0);

	return S_OK;
}
```

Damit können wir die Texturen für unsere Chams erzeugen. Also gleich mal ein Beispiel. Added das:

```
LPDIRECT3DTEXTURE9 texPink;
```

Jetzt können wir eine Textur erzeugen. Dazu rufen wir die Funktion folgendermaßen auf:

```
GenerateTexture(pDevice, &texPink,txtPink);
```

Wie die Font darf auch diese Funktion nur 1x aufgerufen werden. Stellt das sicher.
Jetzt haben wir eine einfarbige Pinke Textur. Um die jetzt auf unseren Player zu bekommen ist noch einiges an Arbeit nötig. Bis jetzt haben wir nämlich nur EndScene gehookt, das reicht uns jetzt nicht mehr. Werft jetzt wieder IDA an und hookt folgende Funktionen:

```
HRESULT __stdcall SetStreamSource(LPDIRECT3DDEVICE9 pDevice,UINT
StreamNumber,IDirect3DVertexBuffer9* pStreamData,UINT OffsetInBytes,UINT Stride);
HRESULT __stdcall DrawIndexedPrimitive(LPDIRECT3DDEVICE9 pDevice,D3DPRIMITIVETYPE Type,INT
BaseVertexIndex,UINT MinVertexIndex,UINT NumVertices,UINT startIndex,UINT primCount);
```

Versucht es analog zu EndScene herzubringen. Dann erstellen wir eine neue Variable:

```
UINT myStride;
```

Jetzt wechseln wir zu unserer gehookten SetStreamSource Funktion und entnehmen uns hier den benötigten Parameter:

```
if (Stride != NULL ) myStride = Stride;
```

Jetzt können wir den Wert aus Stride auch in den anderen Funktionen nutzen.
Wechselt jetzt zu DrawIndexedPrimitive. Fügt folgenden Code ein:

```
if (myStride == 20) {
```

```
        pDevice->SetRenderState(D3DRS_ZENABLE,false);
        pDevice->SetRenderState(D3DRS_FILLMODE,D3DFILL_SOLID);
        pDevice->SetTexture( 0, texGreen);
        pDevice->DrawIndexedPrimitive(Type,BaseVertexIndex, MinVertexIndex, NumVertices, startIndex,
primCount);
        pDevice->SetRenderState(D3DRS_ZENABLE,true);
        pDevice->SetRenderState(D3DRS_FILLMODE,D3DFILL_SOLID);
        pDevice->SetTexture( 0, texRed);
}
```

Jetzt zur Erklärung:
Strides definieren in unserem Game immer eine bestimmte Gruppen von Objekten die gezeichnet werden. Zum Beispiel die Wände oder eben die Player. Die Werte variieren in jedem Game. In Warrock haben zum Beispiel die Player die Stridenummer 44. In Operation 7 haben die Player 20. Wie findet ihr nun die passende Nummer? Ganz einfach, ihr führt eine Variable ein die sich mit Tastendruck ändern lässt und färbt dann einfach immer diesen bestimmten Stride. Dann loggt ihr euch die Nummer an der eure Player bunt wurden und schon habt ihrs.

Der restliche Code:
Die erste Zeile ist der eigentliche Wallhack. Damit werden alle Stride == 20 Objekte im Vordergrund gezeichnet.
Die zweite Zeile bewirkt, dass unsre Player komplett mit unserer Textur ausgfüllt werden. Die dritte Zeile färbt die Player dann in Grün.
In der vierten Zeile wird das alles angewandt. Jetzt wären wir eigentlich schon fertig, aber wir wollen ja zweifarbige Chams. Grün, wenn sich die Player *hinter* der Wand befinden, und rot wenn sie im *Vordergrund* sind. Deswegen folgen jetzt noch einmal dieselben drei Zeilen wie vorher, mit dem Unterschied, dass in der 5. Zeile der RenderState auf *true* gesetzt wird. Das stellt sicher, dass nur die Player im Vordergrund gezeichnet werden.

Noch mehr Funktionen

Zum Beispiel: Fog disablen

```
pDevice->SetRenderState(D3DRS_FOGENABLE, FALSE); // in DrawIndexedPrimitive
```

Weiterhin für die Tastatureingaben:

```
if (GetAsyncKeyState(VK_INSERT)&1) { bla; }
```

Und noch mehr: da es eher schlecht ist, die ganzen Funktionen mit statischen Offsets zu hooken, hier zwei Methoden die das ganze dynamisch verarbeiten. Hier die Methode von SwTTy für die Adressen der d3d9.dll:

```
HMODULE hModule = NULL;
    while(!hModule)
    {
        hModule = GetModuleHandleA("d3d9.dll");
        Sleep(100);
    }
        dwBeginScene = (DWORD)hModule + 0x87010;
        dwEndScene = (DWORD)hModule + 0x871A0;
        dwReset = (DWORD)hModule + 0x636B0;
        dwDrawIndexedPrimitive = (DWORD)hModule + 0x88830;
        dwSetViewPort = (DWORD)hModule + 0x82F70;
```

Und dann eine noch bessere Methode von Gordon. Dazu brauchen wir erste diese zwei Funktionen von GameDeception:

```
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 dwFindPattern(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;
}
```

Und dann holt man sich das ganze per VMT:

```
HMODULE hModule = NULL;
      while(!hModule)
      {
            hModule = GetModuleHandleA("d3d9.dll");
            Sleep(100);
      }

                  DWORD* VTableStart = 0;
                  DWORD FoundByGordon = dwFindPattern((DWORD)hModule, 0x128000,
(PBYTE)"\xC7\x06\x00\x00\x00\x00\x89\x86\x00\x00\x00\x00\x89\x86", "xx????xx????xx");
                  memcpy(&VTableStart, (void*)(FoundByGordon+2), 4);

dwDrawIndexedPrimitive = (DWORD)VTableStart[82];   // für mehr: blick in die d3d9.h werfen!
dwEndScene = (DWORD)VTableStart[42];
```

Grobe Infos gibt's auf Wikipedia:
http://de.wikipedia.org/wiki/Tabelle_virtueller_Methoden

So, jetzt sind wir eigentlich schon am Schluss angekommen. Ich hoffe ihr konntet hiermit etwas anfangen und euch ist das Thema D3D jetzt ein bisschen klarer. Es gäb natürlich noch viel mehr zu sagen, aber jetzt seid ihr gewappnet und könnt euch den Beispielen auf GameDeception widmen ;D

Cya im cheat-forum.eu
purple.d1amond – www.diamondhack.de