

How to patch the opengl32.dll to create a wallhack for the game Half-Life

This tutorial shows how to patch the opengl32.dll stored in the windows/system directory to create an XQZ wallhack for Half-Life.

The opengl32.dll is a „wrapper“ between an opengl-program and the grafic card driver. Its stored in <windows>system32 directoy on WindowsXP, Windows2000 and WindowsNT and stored in <windows>\system directory on Windows95, Windows98, Windows98 SE and WindowsME.

Half-Life loads the opengl32.dll by calling the Windows API LoadLibraryA with parameter „opengl32.dll“.

```
0013CEE0 01D87B7D CALL to LoadLibraryA from hl.01D87B77
0013CEE4 01DFC4F4 FileName = "opengl32.dll"
0013CEE8 004E2860 hl.004E2860
0013CEEC 01D6617F RETURN to hl.01D6617F from hl.01D87B60
0013CEF0 01DFC4F4 ASCII "opengl32.dll"
0013CEF4 0013CF10
0013CEF8 77D1E010 RETURN to USER32.77D1E010 from USER32.77D1B0C0
```

This bug (which isn't really a bug) is used by many opengl hacks, because of the way Windows is loading libraries. If Half-Life would load the opengl library by using LoadLibraryA(„<systemdir>\opengl32.dll“) none of the opengl hacks - where u must put a hacked opengl32.dll into the Half-Life directory - would work. But if you use LoadLibraryA („opengl32.dll“) like Half-Life does, Windows first looks into the directory where the executable of the program is stored and if Windows find an „opengl32.dll“ library there, it is using this.

The way is this:

hl.exe opengl game (Half-Life)	uses
kernel32.dll LoadLibraryA(„opengl32.dll“)	loads
opengl32.dll in Half-Life directory (wrapper with wallhack etc.)	uses
kernel32.dll LoadLibraryA(„<systemdir>\opengl32.dll“)	loads
opengl32.dll, original	loads
driver (for nvidia nvoglnt.dll, for ATI atioglxx.dll)	

What we want to do is patching the opengl32.dll directly so we dont need a wrapper:

hl.exe opengl game (Half-Life)	uses
kernel32.dll LoadLibraryA(„opengl32.dll“)	loads
opengl32.dll in Half-Life directory (hacked original opengl32.dll)	loads
driver (for nvidia nvoglnt.dll, for ATI atioglxx.dll)	

So the first step is coping the opengl32.dll from the <systemdirectory> to the Half-Life directory where the hl.exe is stored.

After Half-Life is now loading the original opengl32.dll from the Half-Life directory we have a deeper look into the way Half-Life draws something in opengl.

The main function we want to hook is

procedure glBegin(mode: cardinal); stdcall;	(delphi)
void __stdcall glBegin(dword mode)	(c++)

The parameter „mode“ is the way Half-Life wants to draw something. Possible modes are:

GL_POINTS	= 0x0000 ;	//(c++) \$0000 (delphi)
GL_LINE	= 0x0001 ;	//(c++) \$0001 (delphi)
GL_LINE_LOOP	= 0x0002 ;	//(c++) \$0002 (delphi)
GL_LINE_STRIP	= 0x0003 ;	//(c++) \$0003 (delphi)
GL_TRIANGLES	= 0x0004 ;	//(c++) \$0004 (delphi)
GL_TRIANGLE_STRIP	= 0x0005 ;	//(c++) \$0005 (delphi)
GL_TRIANGLE_FAN	= 0x0006 ;	//(c++) \$0006 (delphi)
GL_QUADS	= 0x0007 ;	//(c++) \$0007 (delphi)
GL_QUAD_STRIP	= 0x0008 ;	//(c++) \$0008 (delphi)
GL_POLYGON	= 0x0009 ;	//(c++) \$0009 (delphi)

If the mode in glBegin is GL_TRIANGLESTRIP (5) or GL_TRIANGLE_FAN (6) Half-Life wants to draw a model, so we disable the depth test, that everything which is now drawn with glVertex3f is directly on the screen.

```

delphi:
procedure glBegin(mode: cardinal) stdcall;
begin
    if (mode = GL_TRIANGLE_STRIP) or (mode = GL_TRIANGLE_FAN) then
        glDisable(GL_DEPTH_TEST) else glEnable(GL_DEPTH_TEST);
    oldglBegin(mode);
end;

c++:
void __stdcall glBegin(dword dwmode) {
    if ((dwmode == GL_TRIANGLE_STRIP) || (dwmode == GL_TRIANGLE_FAN))
        glDisable(GL_DEPTH_TEST) else glEnable(GL_DEPTH_TEST);
    oldglBegin(dwmode);
}

GL_DEPTH_TEST                = 0x0B71;                //(c++) $0B71 (delphi)

```

glBegin should be our function which is called instead of the original glBegin.
oldglBegin is the original glBegin function.
glEnable and glDisable are opengl functions to activate and deactivate something.
GL_DEPTH_TEST is the constant for the depth test, you can enable the depth test by
calling glEnable(GL_DEPTH_TEST) and disabling it by calling glDisable(GL_DEPTH_TEST)

If we compile it in Delphi and setting up a breakpoint we get the following result:

As you can see we have the assembler code of our glBegin function.

- But we have now 2 problems.
- 1.) we have two distance calls, glBegin and glEnd
 - 2.) oldglBegin is a pointer call, we must change it to a distance call

So we can use inline assembler for creating our own glBegin function. I have added some optimizations.

Unit1.pas.28: begin		
0044C8C0 55	push ebp	
0044C8C1 8BEC	mov ebp,esp	
0044C8C3 53	push ebx	
0044C8C4 8B5D08	mov ebx,[ebp+\$08]	
Unit1.pas.29: if (mode = GL_TRIANGLE_STRIP) or (mode = GL_TRIANGLE_FAN) then		
0044C8C7 83FB05	cmp ebx,\$05	
0044C8CA 7405	jz +\$05	
0044C8CC 83FB06	cmp ebx,\$06	
0044C8CF 750C	jnz +\$0c	
Unit1.pas.30: glDisable(GL_DEPTH_TEST) else glEnable(GL_DEPTH_TEST);		
0044C8D1 68710B0000	push \$00000b71	
0044C8D6 E815FEFFFF	call glDisable	
0044C8DB E80A	jmp +\$0a	
0044C8DD 68710B0000	push \$00000b71	
0044C8E2 E811FEFFFF	call glEnable	
Unit1.pas.31: oldglBegin(mode);		
0044C8E7 53	push ebx	
0044C8E8 FF15DCFB4400	call dword ptr [oldglBegin]	
Unit1.pas.32: end;		
0044C8EE 5B	pop ebx	
0044C8EF 5D	pop ebp	
Unit1.pas.37: MOV EAX,DWORD PTR SS:[ESP+\$4]		
0044C878 368B442404	mov eax,[esp+\$04]	
Unit1.pas.38: CMP EAX,\$5		
0044C87D 83F805	cmp eax,\$05	
Unit1.pas.39: JE @ismodel		
0044C880 7405	jz +\$05	
Unit1.pas.40: CMP EAX,\$6		
0044C882 83F806	cmp eax,\$06	
Unit1.pas.41: JNZ @isnotmodel		
0044C885 750C	jnz +\$0c	
Unit1.pas.43: PUSH \$0B71		
0044C887 68710B0000	push \$00000b71	
Unit1.pas.44: CALL oldglDisable		
0044C88C E8E3FF7FFF	call oldglDisable	
Unit1.pas.45: JMP @end		
0044C891 E80A	jmp +\$0a	
Unit1.pas.47: PUSH \$0B71		
0044C893 68710B0000	push \$00000b71	
Unit1.pas.48: CALL oldglEnable		
0044C898 E8D3FF7FFF	call oldglEnable	
Unit1.pas.50: PUSH DWORD PTR SS:[ESP+\$4]		
0044C89D 36FF742404	push dword ptr [esp+\$04]	
Unit1.pas.51: CALL oldglBegin		
0044C8A2 E8C5FF7FFF	call oldglBegin	
Unit1.pas.52: RET 4		
0044C8A7 C20400	ret \$0004	

Here is the new inline assembler code:

The function begins at 0x044C8BC with the assembler instruction
mov eax, [esp+\$04]

the opcode for this specific instruction is
8B 44 24 04

This is what every hexeditor shows if you open an EXE file. Now we must search unused data in the opengl32.dll to overwrite it with our function. The easiest way is using one of the wgl functions because they are big enough to replace them with our wallhack code. The normal functions like glBegin or glDisable are only 2 or 6 assembler instructions.

glBegin is for example (on NT based operation systems)

```

__asm {
mov eax, dword ptr fs:[18h]
jmp dword ptr [eax+7cch]
}

```

5F0D28F0	64:A1 18000000	MOV EAX,DWORD PTR FS:[18]
5F0D28F6	FFA0 CC070000	JMP DWORD PTR DS:[EAX+7CC]

The first instruction writes 4 bytes from the thread data block address +18 into EAX.

On NT based systems (winXP, win2k and winNT, not winXPsp2) FS is everytime 0x7FFDE000. The address standing on 0x7FFDE018 is 0x7FFDE000.

=>

```

mov eax, dword ptr fs:[18h]
is the same like
mov eax, dword ptr fs:[0x18] //18h = hex, for delphi & c++, 0x18 for c++ and $18 for delphi
is the same like
mov eax, 0x7ffde000
add eax, 0x18
mov eax, [eax]
is the same like
mov eax, 0x7ffde018
mov eax, [eax]
is the same like
mov eax, 0x7ffde000

```

The second instruction jumps into the driver. The address where glBegin is stored in the driver stands on EAX+7cch. As we know EAX is everytime 0x7FFDE000 so the pointer standing at 0x7FFDE7CC is our driver glBegin.

```

mov eax, 0x7ffde000
jmp dword ptr [eax+7cch]
is the same like
mov eax, 0x7ffde000
jmp dword ptr [eax+0x7cc]
is the same like
jmp dword ptr [0x7ffde7cc]

```

The glBegin function in opengl32.dll is nothing else than a jump to the driver glBegin address!

```

glBegin export in opengl32.dll

```

5F0D28F0	64:A1 18000000	MOV EAX,DWORD PTR FS:[18]	
5F0D28F6	FFA0 CC070000	JMP DWORD PTR DS:[EAX+7CC]	at ioglxx.692C5130

Address standing on 0x7ffde7cc:

We must swap the bytes ->

30 51 2C 69 => 0x 692C5130 = driver glBegin

Address	Hex dump
7FFDE7CC	30 51 2C 69

692C5130	53	PUSH EBX
692C5131	55	PUSH EBP
692C5132	56	PUSH ESI
692C5133	57	PUSH EDI
692C5134	64:A1 F00B0000	MOV EAX,DWORD PTR FS:[BF0]
692C513A	8B6C24 14	MOV EBP,DWORD PTR SS:[ESP+14]
692C513E	8BF0	MOV ESI,EAX
692C5140	8B86 4C660000	MOV EAX,DWORD PTR DS:[ESI+664C]
692C5146	8B3CA8	MOV EDI,DWORD PTR DS:[EAX+EBP*4]
692C5149	8B86 048B0400	MOV EAX,DWORD PTR DS:[ESI+48B04]
692C514F	33DB	XOR EBX,EBX
692C5151	3BC3	CMP EAX,EBX
692C5153	74 07	JE SHORT at ioglxx.692C515C
692C5155	C686 018B0400 0	MOV BYTE PTR DS:[ESI+48B01],1
692C515C	399E C4000000	CMP DWORD PTR DS:[ESI+C4],EBX
692C5162	74 11	JE SHORT at ioglxx.692C5175
692C5164	B9 02050000	MOV ECX,502
692C5169	E8 72730600	CALL at ioglxx.6932C4E0
692C516E	5F	POP EDI
692C516F	5E	POP ESI
692C5170	5D	POP EBP
692C5171	5B	POP EBX
692C5172	C2 0400	RETN 4
692C5175	389E_D9C50000	CMP_BYTE PTR DS:[ESI+C5D9],BL

Some opengl hacks are changing the address standing on 0x7ffde7cc to their own glBegin adress (Pharlaps opengl hack, OGC SE (Lite), l0rY 1-10, dom1n1ks H1H etc.) This method is very easy and powerfull, because as I know its VAC undetected since the first VAC release. Cheating-Death has many false detects because of the reason that

some drivers are changing the address like the cheats does it. Also some drivers (nvidia forceware) are changing the address to memory which is not in the driver. So Cheating-Death detects some older drivers and lots of nvidia quadro graphic cards as a cheat.

But for our cheat its not all important.
We need the glBegin export address:

```
5F0D2980 .text Export glBegin
5F0D28F0 .text Export glBegin
5F0D2FA4 .text Export glBegin
5F0D29C0 .text Export glBegin
```

which points to:

```
5F0D28F0 64:A1 18000000 MOV EAX,DWORD PTR FS:[18h]
5F0D28F6 FFA0 CC070000 JMP DWORD PTR DS:[EAX+7CC]
```

Also we know that the opengl32.dll changes EAX, so we can use this register in our wallhack function.

The assembler instruction

```
64:A1 18000000 MOV EAX; DWORD PTR FS:[18h]
```

is the same like

```
A1 18000000 MOV EAX; DWORD PTR FS:[18h]
```

0x64 is a prefix but we dont need it. Our wallhack code has 2 prefixes, too.

```
36:8B4424 04 MOV EAX, [ESP+0x04]
```

is the same like

```
8B4424 04 MOV EAX, [ESP+0x04]
```

and

```
36:FF7424 04 PUSH [ESP+0x04]
```

is the same like

```
FF7424 04 PUSH [ESP+0x04]
```

As I said that we need memory which we can overwrite with our wallhack code. Unused wgl functions are the best functions to overwrite, because the normal gl functions (like glBegin) are only 2 assembler instructions (or some more ex. glShadeModel or on win9x operation systems)

Such a function we can us to overwrite is wglUseFontOutlinesW. Its not used by Half-Life and it has some more instructions.

```
5F0DE70C .text Export wglUseFontOutlinesA
5F0DE742 .text Export wglUseFontOutlinesW
5F0D1224 .text Import USER32.WindowFromDC
5F0DE742 8BFF MOV EDI,EDI
5F0DE744 55 PUSH EBP
5F0DE745 8BEC MOV EBP,ESP
5F0DE747 D945 1C FLD DWORD PTR SS:[EBP+1C]
5F0DE74A 6A 01 PUSH 1
5F0DE74C FF75 24 PUSH DWORD PTR SS:[EBP+24]
5F0DE74F FF75 20 PUSH DWORD PTR SS:[EBP+20]
5F0DE752 51 PUSH ECX
5F0DE753 51 PUSH ECX
5F0DE754 D95C24 04 FSTP DWORD PTR SS:[ESP+4]
5F0DE758 D945 18 FLD DWORD PTR SS:[EBP+18]
5F0DE75B D91C24 FSTP DWORD PTR SS:[ESP]
5F0DE75E FF75 14 PUSH DWORD PTR SS:[EBP+14]
5F0DE761 FF75 10 PUSH DWORD PTR SS:[EBP+10]
5F0DE764 FF75 0C PUSH DWORD PTR SS:[EBP+C]
5F0DE767 FF75 08 PUSH DWORD PTR SS:[EBP+8]
5F0DE76A E8 75FDFFFF CALL opengl_1.5F0DE4E4
5F0DE76F 5D POP EBP
5F0DE770 C2 2000 RETN 20
```

Now we take a hexeditor and search for this assembler instructions:

```
0000DAAC FFC7 45F0 0100 0000 837D FC00 7412 FF75 FC6A 00FF 15C0 110D 5F50 FF15 C811 0D5F 8B75 2085
0000DAD0 F674 08FF 7508 E8A9 F1FF FF83 7DF0 005F 5E75 176A 08FF 15CC 110D 5F8B 45F8 2B45 1450 FF75
0000DAF4 14E8 924E FFFF 8B45 F05B C9C2 2400 FF45 FBEB ADCC CCCC CCCC 8BFF 558B ECD9 451C 6A00 FF75
0000DB18 24FF 7520 5151 D95C 2404 D945 18D9 1C24 FF75 14FF 7510 FF75 0CFF 7508 E8AB FDFD FF5D C220
0000DB3C 00CC CCCC CCCC 8BFF 558B ECD9 451C 6A01 FF75 24FF 7520 5151 D95C 2404 D945 18D9 1C24 FF75
0000DB60 14FF 7510 FF75 0CFF 7508 E875 FDFD FF5D C220 00CC CCCC CCCC 8BFF 558B EC83 EC10 538B 5D08
0000DB84 830B 0480 B95D 0500 0000 5657 0F85 8300 0000 D981 9000 0000 8D81 C419 0000 D808 8B00 BF00
```



We found the right data. The red marked data is looking like our wglUseFontOutlinesW function but it is not. There is only one byte difference -> the call is 0xE8ABFDFF instead of 0xE875FDFFF. If you want to create a hack for other programs be sure you change the correct bytes. If u would find more that 1 result you can do a second check. The last 2 bytes of the address in the hexeditor must be the same than in the debugger:

The debugger shows the wglUseFontOutlinesW function at adress 0x5F0DE742 the hexeditor shows it at address 0x0000DB42. The other function (red marked) starts at address 0x0000DB0C so its not our function.

Now we overwrite the data with our wallhack code (we dont need the two prefixes):

```
00CC CCCC CCCC 8B44 2404 83F8 0574 0583 F806 750C 6871 0B00 00E8 E3FF FFFF EBOA 6871 0B00
00E8 D3FF FFFF FF74 2404 E8C5 FFFF FFC2 0400 2000 CCCC CCCC 8BFF 558B EC83 EC10 538B 5D08
```

In the debugger we can now see our replaced code, if we look at the export function wglUseFontOutlinesW:
Now we must change the distance calls that they are pointing to glEnable glDisable and glBegin.

```
5F0DE742 8B4424 04 MOV EAX,DWORD PTR SS:[ESP+4]
5F0DE746 83F8 05 CMP EAX,5
5F0DE749 v74 05 JE SHORT opengl32.5F0DE750
5F0DE74B 83F8 06 CMP EAX,6
5F0DE74E v75 0C JNZ SHORT opengl32.5F0DE75C
5F0DE750 68 710B0000 PUSH 0B71
5F0DE755 E8 E3FFFFFF CALL opengl32.5F0DE73D
5F0DE75A vEB 0A JMP SHORT opengl32.5F0DE766
5F0DE75C 68 710B0000 PUSH 0B71
5F0DE761 E8 D3FFFFFF CALL opengl32.5F0DE739
5F0DE766 FF7424 04 PUSH DWORD PTR SS:[ESP+4]
5F0DE76A E8 C5FFFFFF CALL opengl32.5F0DE734
5F0DE76F C2 0400 RETN 4
5F0DE772 2000 AND BYTE PTR DS:[EAX],AL
5F0DE774 CC INT3
```

Here are the 3 offsets:

```
glBegin: 0x5F0D28F0
5F0D2F98 .text Export glVertex
5F0D28F0 .text Export glBegin
5F0D2FA4 .text Export glBindTexture

glEnable: 0x5F0D2E54
5F0D2E54 .text Export glEnable
5F0D2FEC .text Export glEnableClientState

glDisable: 0x5F0D2E48
5F0D2E48 .text Export glDisable
5F0D2F8C .text Export glDisableClientState
```

Its easy to calculate the new distance. We must subtract from the function we want to call our call adress and 5 bytes for size of the call instruction:

0x5F0D2E48 (glDisable) - 0x5F0DE755 (our call) - 0x05 (size of call instruction) = 0xFFFF46EE
0x5F0D2E54 (glEnable) - 0x5F0DE761 (our call) - 0x05 (size of call instruction) = 0xFFFF46EE
0x5F0D28F0 (glBegin) - 0x5F0DE76A (our call) - 0x05 (size of call instruction) = 0xFFFF4181

We must swap the bytes (ex. 0xFFFF46EE => 0xEE46FFFF) and change them in the hexeditor:

```
00CC CCCC CCCC 8B44 2404 83F8 0574 0583 F806 750C 6871 0B00 00E8 EE46 FFFF EBOA 6871 0B00
00E8 EE46 FFFF FF74 2404 E881 41FF FFC2 0400 2000 CCCC CCCC 8BFF 558B EC83 EC10 538B 5D08
```

Now the debugger should show this:

```
5F0DE742 8B4424 04 MOV EAX,DWORD PTR SS:[ESP+4]
5F0DE746 83F8 05 CMP EAX,5
5F0DE749 v74 05 JE SHORT opengl32.5F0DE750
5F0DE74B 83F8 06 CMP EAX,6
5F0DE74E v75 0C JNZ SHORT opengl32.5F0DE75C
5F0DE750 68 710B0000 PUSH 0B71
5F0DE755 E8 EE46FFFF CALL opengl32.glDisable
5F0DE75A vEB 0A JMP SHORT opengl32.5F0DE766
5F0DE75C 68 710B0000 PUSH 0B71
5F0DE761 E8 EE46FFFF CALL opengl32.glEnable
5F0DE766 FF7424 04 PUSH DWORD PTR SS:[ESP+4]
5F0DE76A E8 8141FFFF CALL opengl32.glBegin
5F0DE76F C2 0400 RETN 4
5F0DE772 2000 AND BYTE PTR DS:[EAX],AL
5F0DE774 CC INT3
```

We have successful patched the opengl32.dll with our wallhack function. But its never called... We have to change the export table address of glBegin that it points to our new wallhack function.

We know that the export address of glBegin is: 0x5F0D28F0. An entry of the export table is stored in 4 bytes. The first 2 bytes are fixed from windows with the relocation table, because its not sure where windows loads the library into the target process. To get the address how it is stored in the opengl32.dll that we can change it with a hexeditor we must subtract from the address in memory the base address of the opengl32.dll.

```
50450000 00097000 504532DA COMCTL32 5.82 (xpsp_sp2; C:\WINDOWS\system32\COMCTL32.dll
5F0D0000 000CC000 5F0D3322 opengl32 5.1.2600.2100 ( C:\Spiele\Half-Life\opengl32.dll
621F0000 00010000 ..... mcicda 5.1.2600.0 (xpc C:\WINDOWS\system32\mcicda.dll ...
```

0x5F0D28F0 (glBegin address in memory) – 0x5F0D0000 (base address of opengl32.dll) = 0x000028F0
0x5F0DE742 (wglUseFontOutlinesW address in memory) - 0x5F0D0000 (base address of opengl32.dll) = 0x0000E742

so we must search for F0 28 00 00 in the hexeditor

```
000A20B4 | 30A8 0100 DBA5 0100 11A7 0100 8C45 0000 FC47 0000 F051 0000 982F 0000 F028 0000 A42F 0000  
000A20B8 | E022 0000 2048 0000 D028 0000 F438 0000 2421 0000 F021 0000 0021 0000 2022 0000 0021 0000
```

and replace it with 42 E7 00 00.

```
000A20B4 | 30A8 0100 DBA5 0100 11A7 0100 8C45 0000 FC47 0000 F051 0000 982F 0000 42E7 0000 A42F 0000  
000A20B8 | E022 0000 2048 0000 D028 0000 F438 0000 2421 0000 F021 0000 0021 0000 2022 0000 0021 0000
```

Now we can test our patched opengl32.dll, it should look like this:



I know that nobody wants to create a full hack with this method, but if u try it you can learn much as I did.

- uall

tutorial for [Online Game Cheats], [Endorphine Coding Crew] and [Game-Deception]
Thanks to everyone who supports me :)