

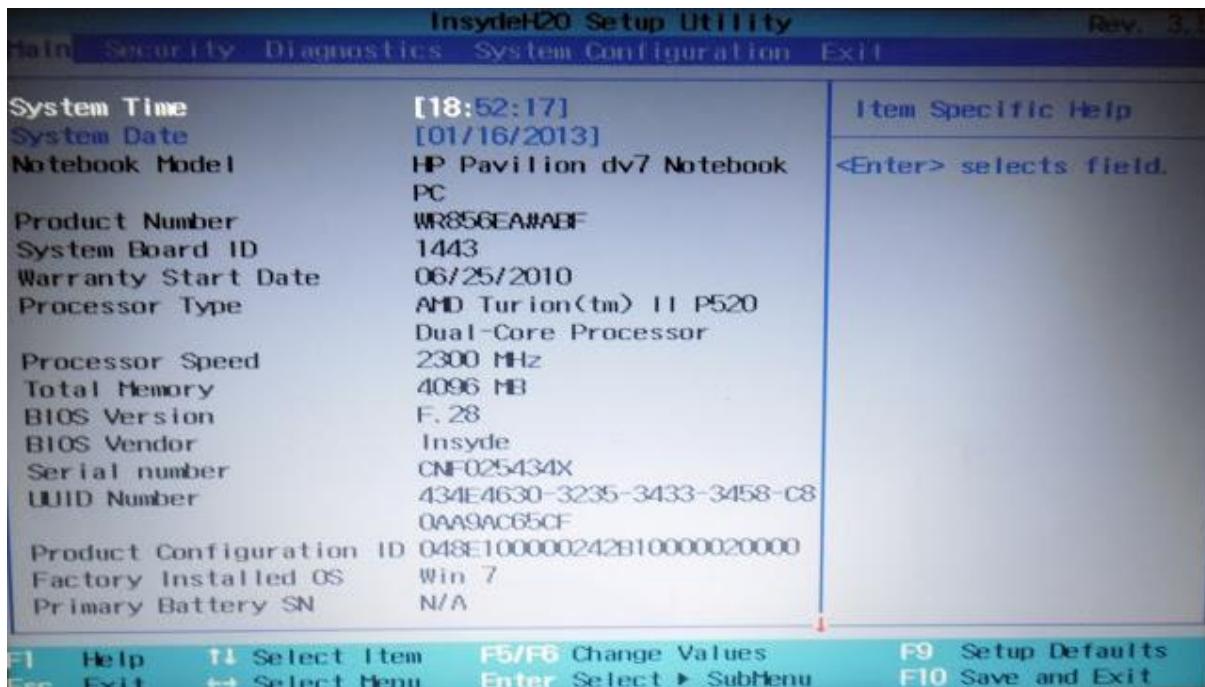
donovan6000's Blog

Monday, June 10, 2013

Insyde Bios Modding: Advanced and Power Tabs

Due to a request I received by [drakonn](#), I'll be covering how to enable the advanced and power tabs in the setup utility. Also special thanks to [Florin9doi](#) for his impressive knowledge of BIOS. I rewrote my splash screen tutorial based on his input.

There's not really much background information I can put here, it's kind of public knowledge that there are hidden tabs in the setup utility. I think this decision is ultimately up to the OEM, so HP decided that we don't need to have access to these hidden tabs. This is most likely because changing some settings can damage your computer, so they're actually looking out for us. So, here's a picture of what my unmodified setup utility looks like. If you'd like to follow along with this tutorial by using the same BIOS that I am, then here's where you can download [it](#).

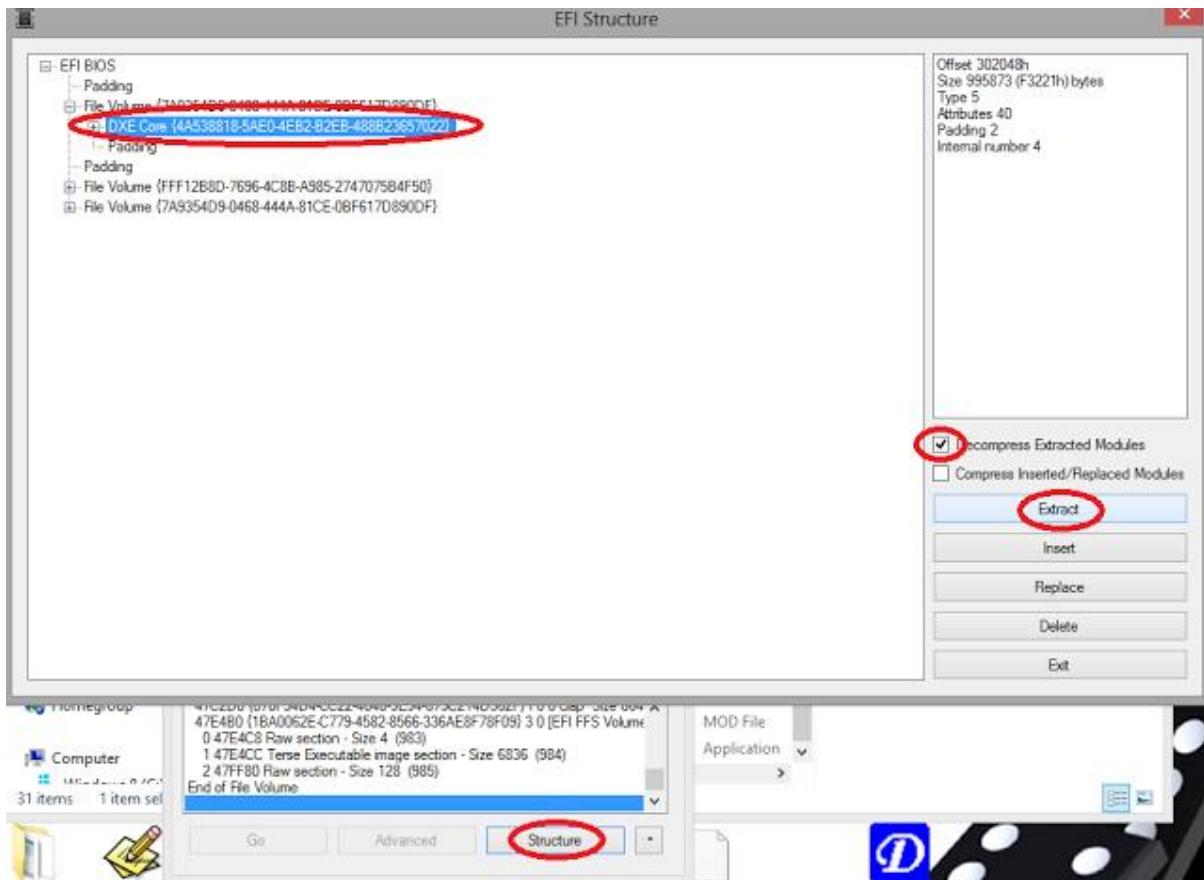


As you can see, it currently has Main, Security, Diagnostics, System Configuration, and Exit tabs. So, I'll show you how to enable the hidden tabs.

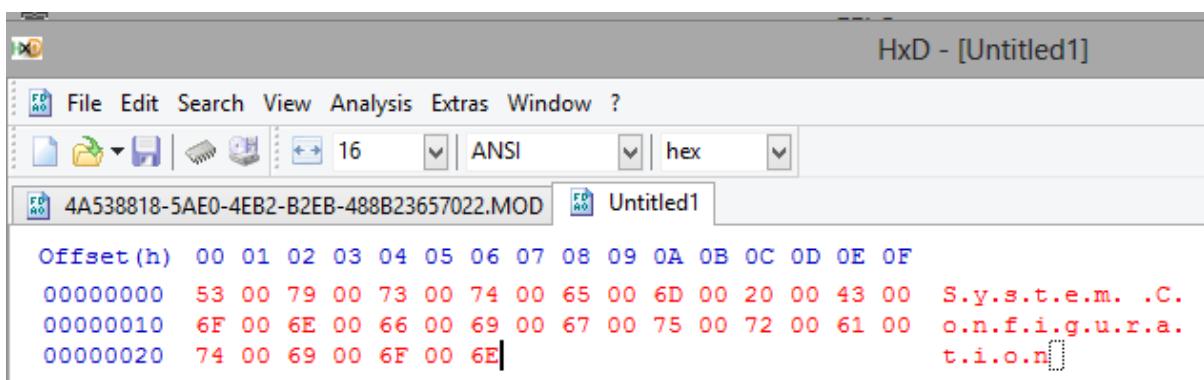
I'll try to keep all my tutorials as generic as possible, but I already know that this will be impossible. Rarely do different BIOS implement these restrictions in the exact same way, so don't expect this to be a sure-fire way to unlock your hidden tabs. As simple as I make these tutorials seem, it still took me several weeks to get each modification working on my own BIOS.

To get started make sure you unpack your BIOS installer so that you have access to the BIOS rom. Then open it with

Andy's tool, go to the structure view, check the Decompress Extracted Modules box, and extract the DXE Core module. The latest version of Andy's tool can be downloaded [here](#).



My extracted module is named 4A538818-5AE0-4EB2-B2EB-488B23657022.MOD. Yours might be named something different. So, lets open that module with a hex editor, and search for a familiar string so that we can locate what module contains the setup utility. The hex editor I use is [HxD](#). As a side note, my BIOS uses Unicode strings. This means that after each letter, there's a 00-hex character. This is because each character is actually two bytes long. I think all Insyde BIOS are that way, but I'm not sure. So, here's what I am going to search for, notice how I have blank characters between each letter. The name of one of my tabs is System Configuration, so the module that contains this string should also contain the setup utility.



So, lets search for this string in our DXE Core module and see if it exists. Awesome! It found it at offset 0x1A8B8C.

HxD - [C:\Users\Domino\Downloads\sp55299\4A538818-5AE0-4EB2-B2EB-488B23657022.MOD]

File	Edit	Search	View	Analysis	Extras	Window	?
16	ANSI	hex					
4A538818-5AE0-4EB2-B2EB-488B23657022.MOD Untitled1							
Offset(h)	00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F						
00147140	20 00 64 00 69 00 73 00 6B 00 20 00 61 00 6E 00	. d.i.s.k. .a.n.					
00147150	64 00 20 00 6D 00 65 00 6D 00 6F 00 72 00 79 00	d. .m.e.m.o.r.y.					
00147160	20 00 73 00 65 00 6C 00 66 00 20 00 74 00 65 00	.s.e.l.f. .t.e.					
00147170	73 00 74 00 2E 00 00 00 56 00 69 00 65 00 77 00	s.t....V.i.e.w.					
00147180	20 00 74 00 68 00 65 00 20 00 73 00 79 00 73 00	.t.h.e. .s.y.s.					
00147190	74 00 65 00 6D 00 20 00 64 00 69 00 61 00 67 00	t.e.m. .d.i.a.g.					
001471A0	6E 00 6F 00 73 00 74 00 69 00 63 00 20 00 66 00	n.o.s.t.i.c..f.					
001471B0	61 00 69 00 6C 00 75 00 72 00 65 00 20 00 72 00	a.i.l.u.r.e. .r.					
001471C0	65 00 73 00 75 00 6C 00 74 00 73 00 2E 00 00 00	e.s.u.l.t.s....					
001471D0	46 00 61 00 63 00 74 00 68 00 72 00 79 00 20 00	F.a.c.t.o.r.y. .					
001471E0	69 00 6E 00 73 00 74 00 61 00 6C 00 66 00 65 00	i.n.s.t.a.l.l.e.					
001471F0	64 00 20 00 4F 00 53 00 3A 00 00 00 56 00 69 00	d.O.S...V.i.					
00147200	73 00 74 00 61 00 00 00 58 00 79 00 73 00 74 00	s.t.a...S.Y.M.t.					
00147210	65 00 6D 00 20 00 43 00 6F 00 5E 00 66 00 68 00	s.m. .C.on.f.i.					
00147220	67 00 78 00 72 00 61 00 74 00 89 00 6E 00 62 00	g.u.r.a.t.i.o.n.					
00147230	00 00 46 00 61 00 6E 00 20 00 41 00 6C 00 77 00	.F.a.n. .A.l.w.					
00147240	61 00 79 00 73 00 20 00 4F 00 6E 00 00 53 00	a.y.s. .O.n...S.					
00147250	65 00 74 00 20 00 74 00 68 00 65 00 20 00 46 00	e.t. .t.h.e. .F.					
00147260	61 00 6E 00 20 00 41 00 6C 00 77 00 61 00 79 00	a.m. .A.l.W.a.y.					
00147270	73 00 20 00 4F 00 6E 00 00 41 00 63 00 74 00	s. .O.n...A.c.t.					
00147280	69 00 6F 00 6E 00 20 00 4B 00 65 00 79 00 73 00	i.o.n. .K.e.y.s.					
00147290	20 00 4D 00 6E 00 64 00 65 00 00 44 00 69 00	.M.o.d.e...D.i.					
001472A0	73 00 61 00 62 00 6C 00 65 00 64 00 3A 00 20 00	s.a.b.l.e.d.: -					
001472B0	52 00 65 00 71 00 75 00 69 00 72 00 65 00 73 00	R.e.q.u.i.r.e.s.					
001472C0	20 00 70 00 72 00 65 00 73 00 73 00 69 00 6E 00	.p.r.e.s.s.i.n.					
001472D0	67 00 20 00 46 00 6E 00 20 00 6B 00 65 00 79 00	g. .F.n. .k.e.y.					
001472E0	20 00 2B 00 20 00 66 00 31 00 20 00 74 00 68 00	.+. .f.l. .t.h.					
001472F0	72 00 6F 00 75 00 67 00 68 00 20 00 66 00 31 00	r.o.u.g.h. .f.i.l.					
00147300	32 00 20 00 74 00 6F 00 20 00 61 00 63 00 79 00	2. .t.c. .a.c.t.					
00147310	69 00 76 00 61 00 74 00 65 00 20 00 61 00 63 00	i.v.a.t.e. .a.c.					

Offset: 147208 Block: 147208-14722E Length: 27 Overwrite

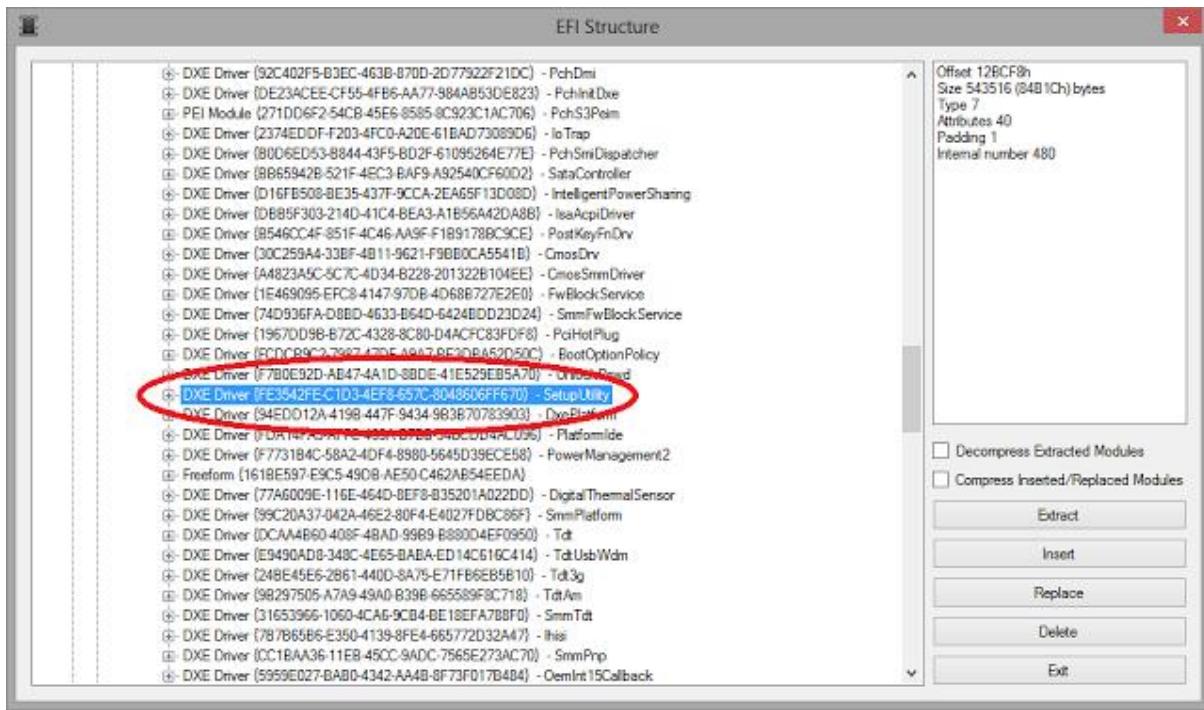
Now we know we're in the correct module. Now search for the hex values 4D 5A. These values are always at the start of a module, and the name of a module is always at the end of a module. So, here's what it finds:

HxD - [C:\Users\Domino\Downloads\sp55299\4A538818-5AE0-4EB2-B2EB-488B23657022.MOD]

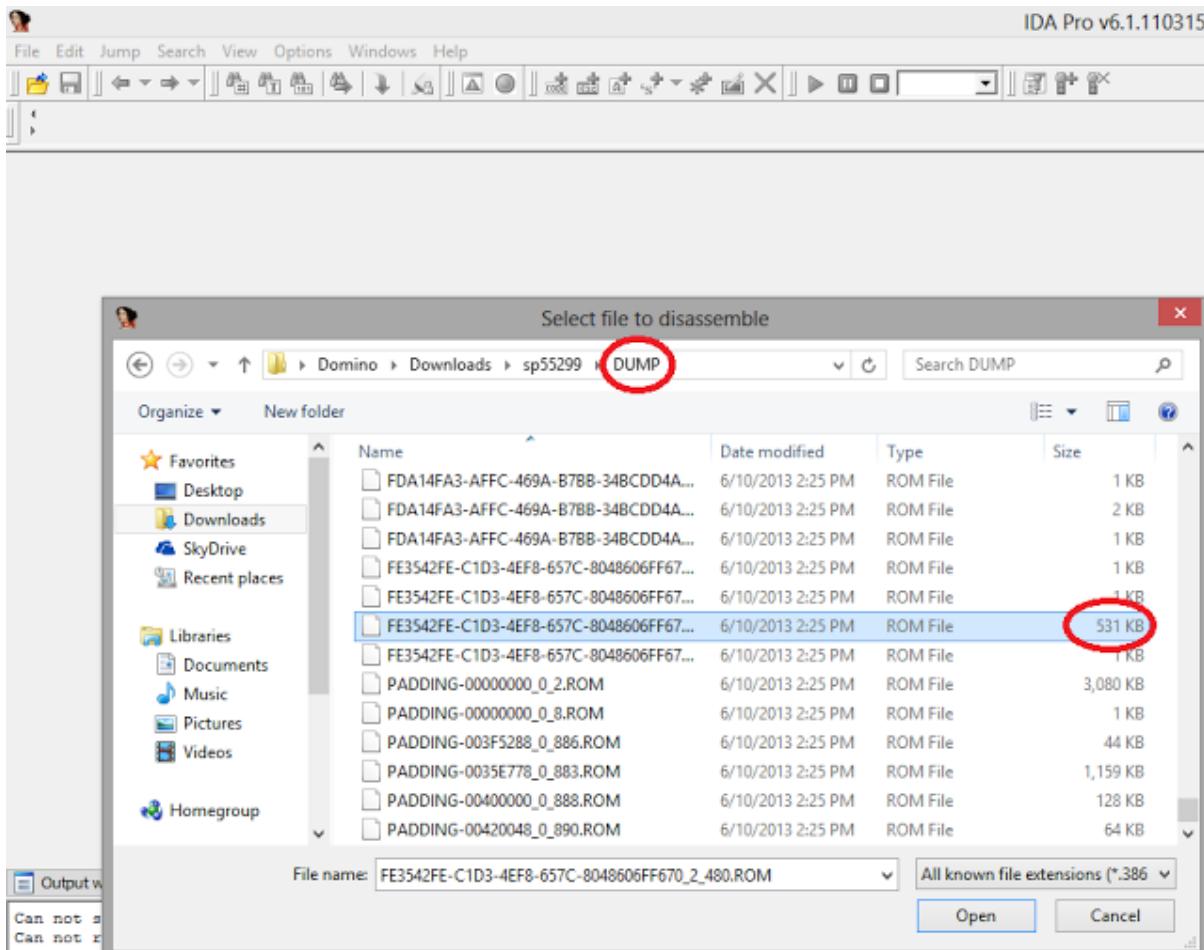
File	Edit	Search	View	Analysis	Extras	Window	?
16	ANSI	hex					
4A538818-5AE0-4EB2-B2EB-488B23657022.MOD Untitled1							
Offset(h)	00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F						
001B0810	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00					
001B0820	00 00 00 00 1E 00 00 15 53 00 65 00 74 00 75 00S.e.t.u.					
001B0830	70 00 55 00 74 00 69 00 6C 00 69 00 74 00 79 00	p.U.t.i.l.i.t.Y.					
001B0840	00 00 00 00 FF FF FF 2A D1 ED 94 9B 41 7F 44	.Y.Y.Y.Y.Ni^>A					
001B0850	94 34 9B 3B 70 78 39 03 40 09 07 40 44 58 00 F8	"4>;px9...;BXD.s					
001B0860	2C 58 00 02 B0 CD 1B FC 31 7D AA 49 93 6A A4 60	,X."i.U1;"I"jR`					
001B0870	0D 9D 00 83 1C 00 02 00 C3 CB CE 6E 70 00 00 13	.l.Bf...;R.EinP...					
001B0880	02 BB 7E 70 2F 1A 4A D4 11 9A 38 00 90 27 3F C1	.s=p/.J0.88..?A					
001B0890	4D 02 E2 68 56 1E 81 84 D4 11 BC F1 00 80 C7 3C	M.ahV...;O.Dh.Eç<					
001B08A0	88 81 03 02 18 F8 41 64 62 63 44 4E B5 70 7D BA@AdbcDNup!*					
001B08B0	31 DD 24 53 03 02 2C 6D 81 EA E5 CE 02 4F 99 B5	1798..,m.ë1.Opu					
001B08C0	D3 90 5C BB D0 77 03 02 FE 42 35 FE D3 C1 F8 4E	Ó.\wBw..pB5pDAnN					
001B08D0	65 7C 80 48 60 6F F6 70 03 02 10 38 B3 CF 87 6E	e €H'odp...?Itn					
001B08E0	B4 42 B2 03 A6 6A BE 07 F6 E8 03 08 84 57 00 10	.B*.j34.đé..W..					
001B08F0	4D 5A 00 00 00 00 00 00 00 00 00 00 00 00 00 00	H2.....					
001B0900	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00					
001B0910	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00					
001B0920	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00					
001B0930	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00					
001B0940	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00					
001B0950	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00					
001B0960	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00					
001B0970	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00					
001B0980	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00					
001B0990	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00					
001B09A0	00 00 00 00 00 00 00 00 50 45 00 00 64 86 06 00PE..dt..					
001B09B0	00 00 00 00 00 00 00 00 00 00 00 00 F0 00 22 20ô"					
001B09C0	DB 02 09 00 60 3F 00 00 60 15 00 00 00 00 00 00'2....					
001B09D0	00 08 00 00 C0 02 00 00 00 00 80 01 00 00 00 00	.B...A....E....					
001B09E0	20 00 00 00 20 00 00 05 00 02 00 00 00 00 00 00					

Offset: 1B08F0 Block: 1B08F0-1B08F1 Length: 2 Insert

I circled the module's name in red. So now we need to remember the GUID of the SetupUtility module. Let's go back to Andy's tool to see what it is.

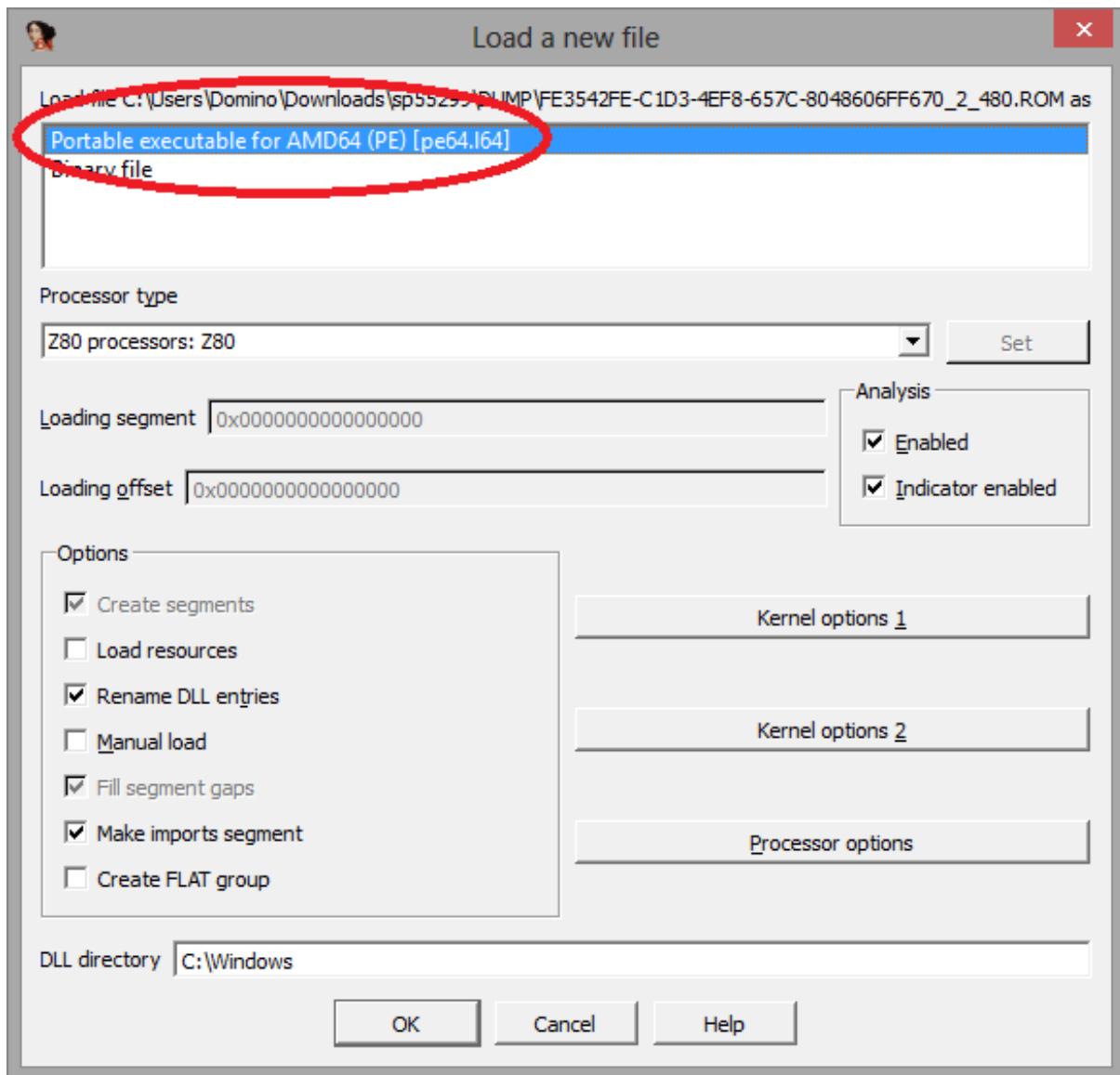


My SetupUtility' GUID is FE3542FE-C1D3-4EF8-657C-8048606FF670. So, lets disassemble this module to get a better understanding of how to mod it. To do this we need to go into the DUMP folder that Andy's tool makes when opening a BIOS file, and open the SetupUtility in there with IDA Pro. Here what I'm saying:

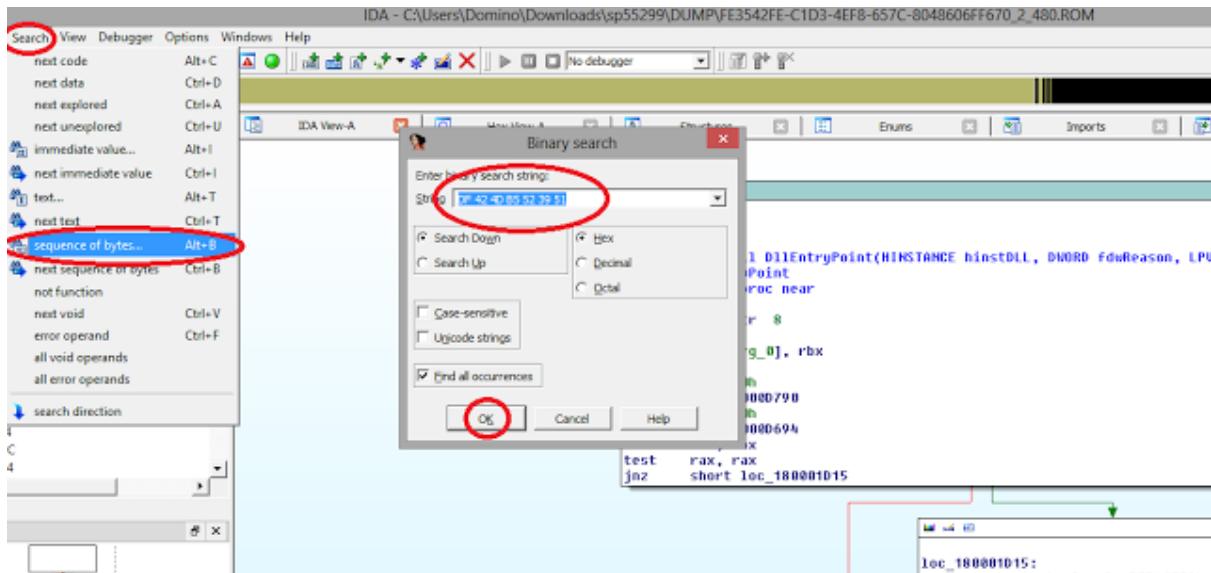


And make sure you open the largest file whose name is similar to your SetupUtility's GUID. Since mine was FE3542FE-C1D3-4EF8-657C-8048606FF670, I'm going to open the 531 kB file which is named similar, FE3542FE-

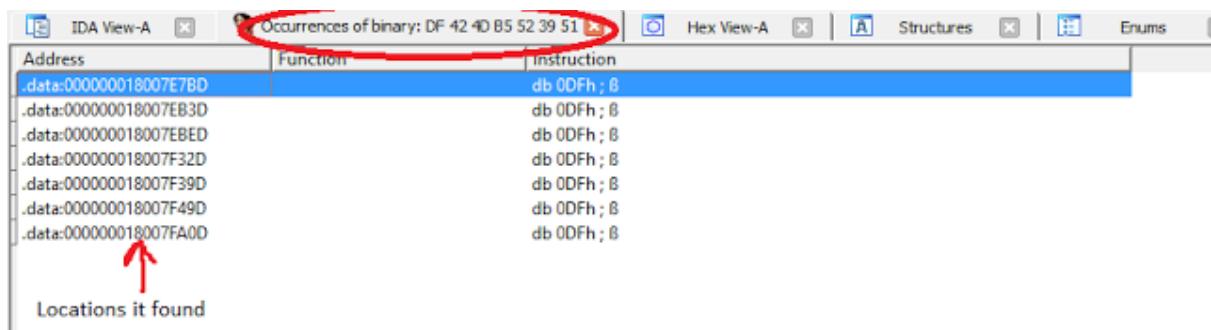
C1D3-4EF8-657C-8048606FF670_2_480.ROM. So, IDA Pro should automatically determine the file type. For me, it's a Portable executable for AMD64.



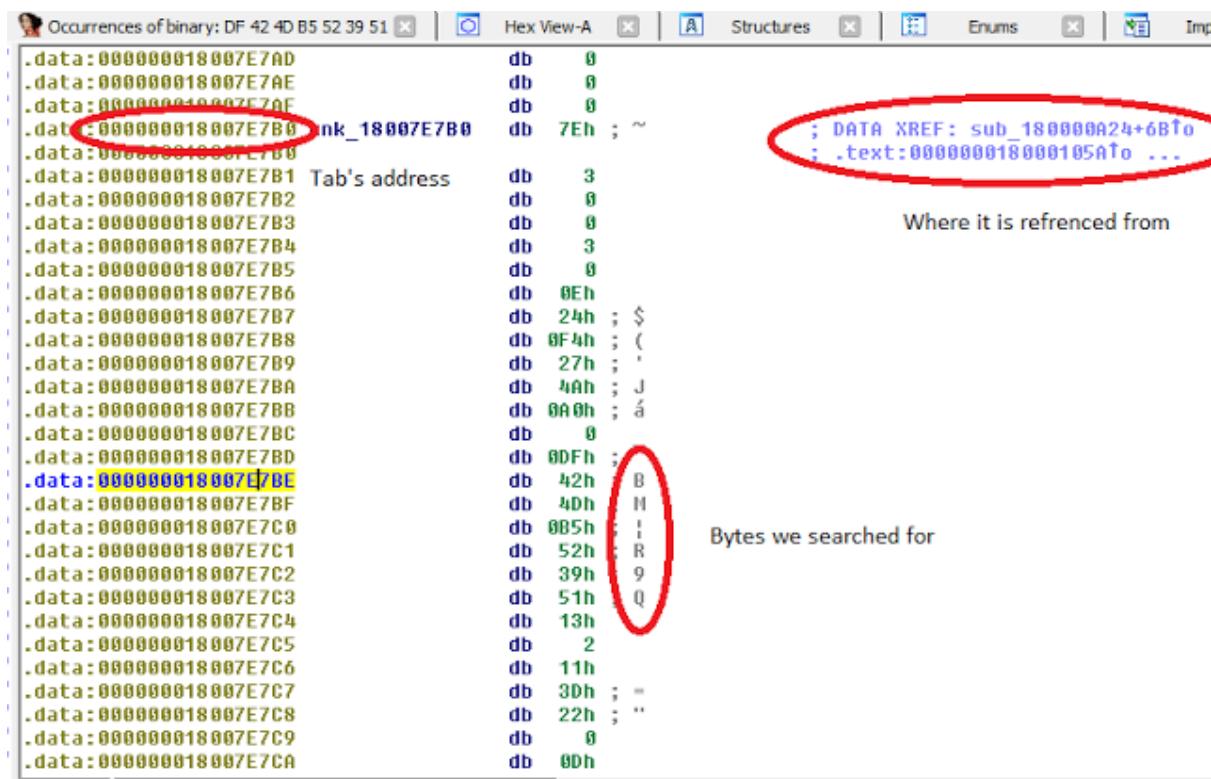
So now that it's disassembled, we have to find out where the tabs are located, then we can see what calls them. I created a program that can dump the internal forms representation used in EFI's human interface infrastructure. This can assist in finding the tab offsets, so you can download it [here](#) if you want to try using it. If you'd rather find them manually, then in IDA Pro go to Search | sequence of bytes. Then enter DF 42 4D B5 52 39 51 and press Ok. These hex values seem to always be in the header of the tabs and are about 13 bytes after the start of the beginning of the tab's offset.



Now this window will come up that shows where these bytes were found. Each one of these locations could potentially lead to one of the tabs.



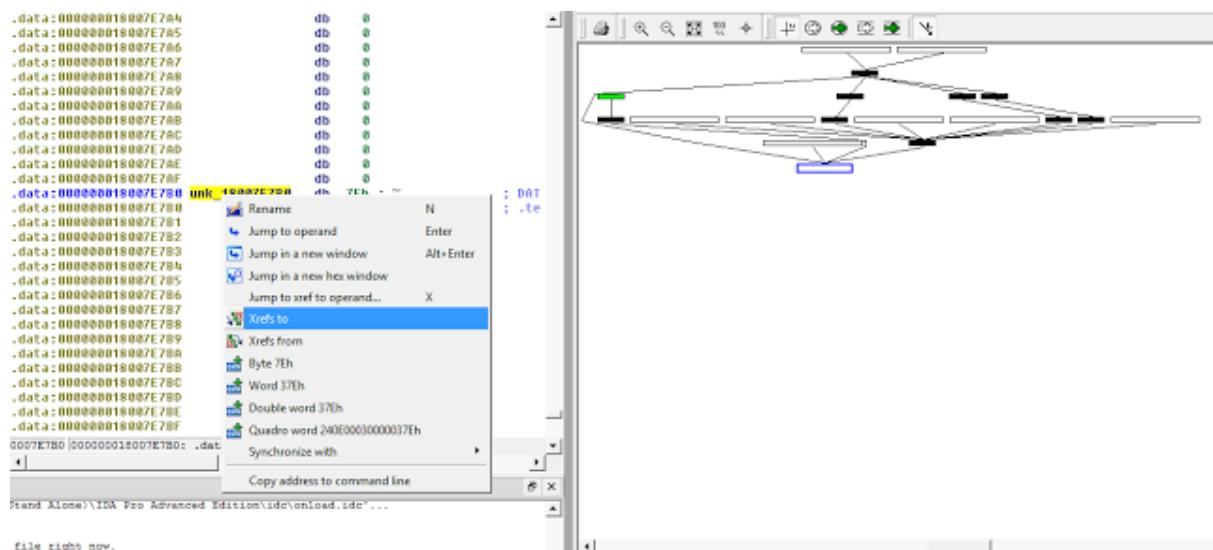
So lets double click on the first one, which takes us here. I said that that byte sequence was in the header, so we need to scroll up about 13 bytes to get to the start of the tab subroutine.



Just go to each one of the places where those bytes sequence occurred to find out the offsets we're looking for. Make sure you write them down. Here's all mine.

Tabs:	Offsets:
1	0x18007E7B0
2	0x18007EB30
3	0x18007EBE0
4	0x18007F320
5	0x18007F390
6	0x18007F490
7	0x18007FA00

Let's go back to the first tab and see where it's being referenced from. Right click on the location and select Xrefs to. This will display the connections between this offset and other functions. You can zoom in to get a better view. Here's mine:



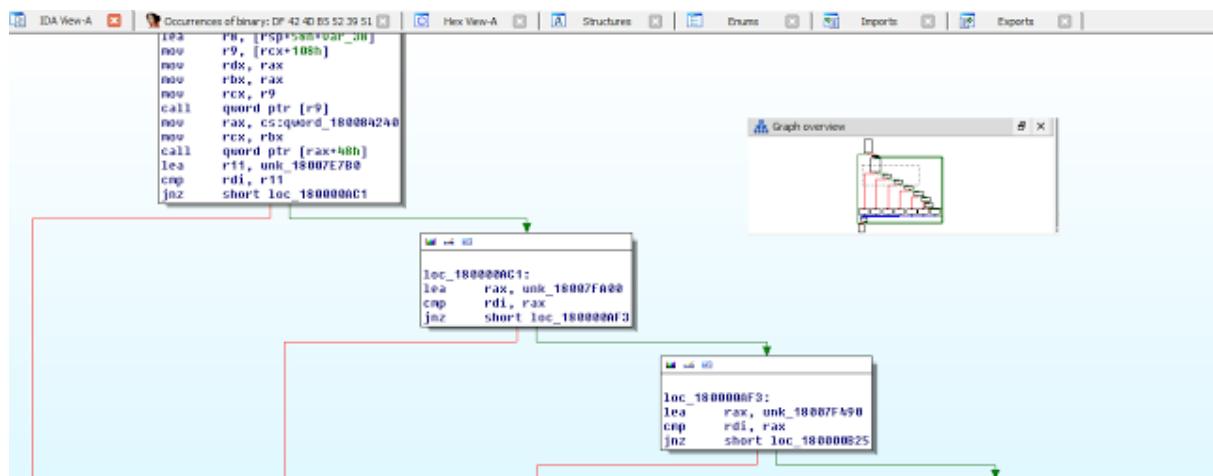
At one of these locations the setup utility is determining which tabs to show. This calling function will probably be closer to the start of the module's code, because that's where it is initialising everything. So, lets double click on the first calling location and see if it looks suspicious.

```

.data:000000018007E7AE db 0
.data:000000018007E7AF db 0
.data:000000018007E7B0 db 7Eh : ~ ; DATA XREF: sub_1800010A24+6B↑o
.data:000000018007E7B0 unk_18007E7B0 db 0
.data:000000018007E7B1 db 3
.data:000000018007E7B2 db 0
.data:000000018007E7B3 db 0
.data:000000018007E7B4 db 3
.data:000000018007E7B5 db 0
.data:000000018007E7B6 db 0Eh
.data:000000018007E7B7 db 24h : $ ; DATA XREF: sub_1800010A24+6C↑o
.data:000000018007E7B8 db 0F4h : { ; DATA XREF: sub_1800010A24+6D↑o
.data:000000018007E7B9 db 27h : . ; DATA XREF: sub_1800010A24+6E↑o
.data:000000018007E7BA db 4Ah : J ; DATA XREF: sub_1800010A24+6F↑o
.data:000000018007E7BB db 0A8h : ā ; DATA XREF: sub_1800010A24+70↑o
.data:000000018007E7BC db 0
.data:000000018007E7BD db 0DFh : — ; DATA XREF: sub_1800010A24+71↑o
.data:000000018007E7BE db 42h : B ; DATA XREF: sub_1800010A24+72↑o
.data:000000018007E7BF db 40h : M ; DATA XREF: sub_1800010A24+73↑o
.data:000000018007E7C0 db 0B5h : i ; DATA XREF: sub_1800010A24+74↑o
.data:000000018007E7C1 db 52h : R ; DATA XREF: sub_1800010A24+75↑o
.data:000000018007E7C2 db 39h : 9 ; DATA XREF: sub_1800010A24+76↑o
.data:000000018007E7C3 db 51h : 0 ; DATA XREF: sub_1800010A24+77↑o

```

Here's the calling location:



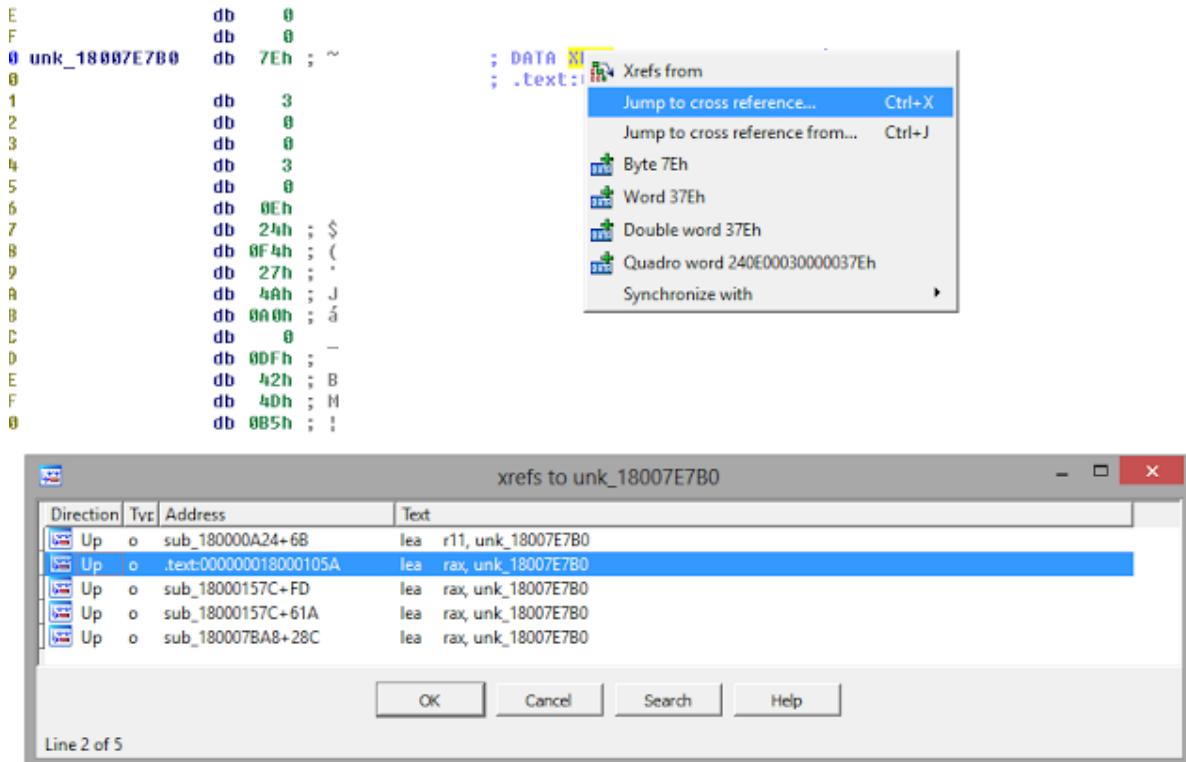
You might be able to determine in IDA Pro's Graph overview that this subroutine is most likely what switches between the tabs when you press left and right. It does reference all the tab offsets, but this is just to determine which one it's currently selecting. This function is not the one we're looking for. If you want to make sure of this, you can modify some conditional jumps, but you will probably brick your computer this way. I should make a tutorial on how to recover from a brick. Let's check out the next calling function.

```

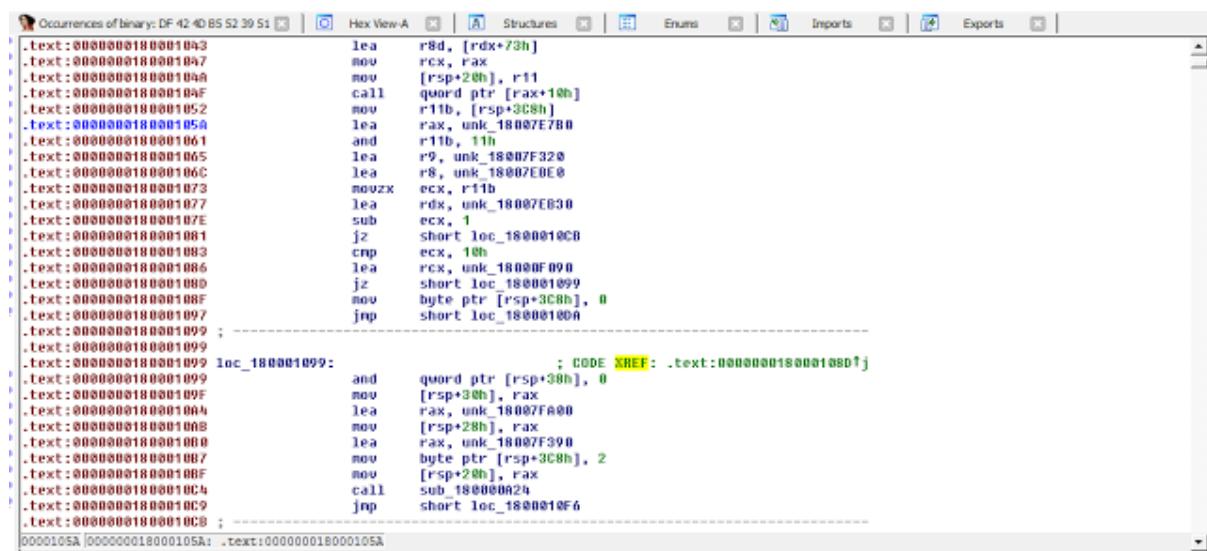
.data:000000018007E7AE db 0
.data:000000018007E7AF db 0
.data:000000018007E7B0 db 7Eh : ~ ; DATA XREF: sub_1800010A24+6B↑o
.data:000000018007E7B0 unk_18007E7B0 db 0
.data:000000018007E7B1 db 3
.data:000000018007E7B2 db 0
.data:000000018007E7B3 db 0
.data:000000018007E7B4 db 3
.data:000000018007E7B5 db 0
.data:000000018007E7B6 db 0Eh
.data:000000018007E7B7 db 24h : $ ; DATA XREF: sub_1800010A24+6C↑o
.data:000000018007E7B8 db 0F4h : { ; DATA XREF: sub_1800010A24+6D↑o
.data:000000018007E7B9 db 27h : . ; DATA XREF: sub_1800010A24+6E↑o
.data:000000018007E7BA db 4Ah : J ; DATA XREF: sub_1800010A24+6F↑o
.data:000000018007E7BB db 0A8h : ā ; DATA XREF: sub_1800010A24+70↑o
.data:000000018007E7BC db 0
.data:000000018007E7BD db 0DFh : — ; DATA XREF: sub_1800010A24+71↑o
.data:000000018007E7BE db 42h : B ; DATA XREF: sub_1800010A24+72↑o
.data:000000018007E7BF db 40h : M ; DATA XREF: sub_1800010A24+73↑o
.data:000000018007E7C0 db 0B5h : i ; DATA XREF: sub_1800010A24+74↑o
.data:000000018007E7C1 db 52h : R ; DATA XREF: sub_1800010A24+75↑o
.data:000000018007E7C2 db 39h : 9 ; DATA XREF: sub_1800010A24+76↑o
.data:000000018007E7C3 db 51h : 0 ; DATA XREF: sub_1800010A24+77↑o

```

If you ever have trouble selecting the different calling functions in IDA Pro you can right click on the DATA XREF and select Jump to cross reference. Then just double click on the address to jump to that location.



So, here's what the second calling function looks like. Don't be surprised that it's not in a flow chart view. IDA Pro isn't perfect, so sometimes it can't produce this style for all functions. As a side note, you can press the space bar to swap back and forth between the flow chart view and the assembly view. Since I know how this tutorial is going to end, I'm going to tell you that this is the function that decides what tabs are available in my BIOS. In yours, you might have to go through several more of the calling functions before you find the one you're looking for.



Since we can't see the bigger picture of this subroutine easily, we'll have to look through it. The main things you want to search for are conditional jumps that avoid one of the tab offsets. So once again, here's the ones I'm searching for:

Tabs:	Offsets:
1	0x18007E7B0
2	0x18007EB30
3	0x18007EBE0
4	0x18007F320
5	0x18007F390
6	0x18007F490
7	0x18007FA00

Back to the second calling function. Wow! Almost immediately I notice almost all of my tab locations being referenced. There are also two conditional jumps:

```

    .text:000000180001043 lea r8d, [rdx+78h]
    .text:000000180001047 mov rcx, rax
    .text:000000180001048 mov [rsp+28h], r11
    .text:00000018000104F call qword ptr [rax+10h]
    .text:000000180001052 mov r11b, [rsp+28h+1]
    .text:00000018000105A lea rax, unk_18007E7B0
    .text:000000180001061 and r11b, r11
    .text:000000180001065 lea r9, unk_18007F320
    .text:00000018000106C lea r8, unk_18007EBE0
    .text:000000180001073 movzx ecx, r9
    .text:000000180001077 lea rdx, unk_18007EB30
    .text:00000018000107E sub ecx, 1
    .text:000000180001081 jz short loc_1800010CB
    .text:000000180001083 cmp ecx, 10
    .text:000000180001086 lea rcs, unk_18000F99
    .text:00000018000108D jz short loc_180001099
    .text:00000018000108F mov byte ptr [rsp+3C8h], 0
    .text:000000180001097 jnp short loc_1800010A0
    .text:000000180001099 ; ...

    .text:000000180001099 loc_180001099: ; CODE XREF: .text:0000001800010801j
    .text:000000180001099 and qword ptr [rsp+38h], 0
    .text:00000018000109F mov [rsp+38h], rax
    .text:0000001800010A4 lea rax, unk_18007FA00
    .text:0000001800010B8 mov [rsp+28h], rax
    .text:0000001800010B0 lea rax, unk_18007F390
    .text:0000001800010B7 mov byte ptr [rsp+3C8h], 2
    .text:0000001800010BF mov [rsp+28h], rax
    .text:0000001800010C4 call sub_180000A24
    .text:0000001800010C9 jnp short loc_1800010F6
    .text:0000001800010CB ; ...

0000105A 00000018000105A: .text:00000018000105A

```

Let me know if you guys think I add too many pictures. I want these tutorials to be thorough, but I'm feeling like this is almost to slow. Let me know what you think. So, lets see where these conditional jumps go to. I just scrolled down a little. So, it seems like the first one it is jumping over two of the tab locations. These could be the two hidden tabs. The second conditional jump is going directly to the two tab locations. And the third unconditional jump at the end is also going to bypass the two tabs.

Occurrences of binary: DF 42 4D B5 52 39 51

Hex View-A | [A] Structures | Enums | Imports | Export

```
.text:0000000180001073 novzx ecx, r11b
.text:0000000180001077 lea rdx, unk_18007EB30
.text:000000018000107E sub ecx, 1
.text:0000000180001081 First jz short loc_1800010CB
.text:0000000180001083 cmp ecx, 10h
.text:0000000180001086 lea rcx, unk_18000F898
.text:000000018000108D jz short loc_180001099
.text:000000018000108F nov byte ptr [rsp+3C8h], 0
.text:0000000180001091 jmp short loc_1800010DA
.text:0000000180001093 ; CODE XREF: .text:000000018000108Dj
.text:0000000180001099 loc_180001099: and quord ptr [rsp+38h], 0
.text:0000000180001099 nov [rsp+38h], rax
.text:000000018000109F lea rax, unk_18007FA00
.text:00000001800010A4 nov [rsp+28h], rax
.text:00000001800010A8 lea rax, unk_18007F390
.text:00000001800010B0 nov byte ptr [rsp+3C8h], 2
.text:00000001800010B7 nov [rsp+28h], rax
.text:00000001800010BF nov call sub_180000A24
.text:00000001800010C4 jmp short loc_1800010F6
.text:00000001800010C9 ; CODE XREF: .text:0000000180001081j
.text:00000001800010CB loc_1800010CB: nov byte ptr [rsp+3C8h], 1
.text:00000001800010CB lea rcx, unk_18000F898
.text:00000001800010D3 ; CODE XREF: .text:0000000180001097j
.text:00000001800010DA loc_1800010DA: and quord ptr [rsp+38h], 0
.text:00000001800010DA nov [rsp+38h], rax
.text:00000001800010E8 lea rax, unk_18007F390
.text:00000001800010E5
```

So, to make sure that those two tabs get referenced, we have to change the two conditional jumps. By changing the first one from a jump if zero (JZ) to nothing, and by changing the second JZ to a JMP, we can accomplish our objective. To view the hex values for the first jump, select it and go to IDA Pro's hex view by clicking on the hex view tab. As you can see it's 74 48. Since we want to remove it, lets change them to no operations (NOP 90). Here's what we're actually changing:

And the second conditional jump's hex values are 74 0A. The first byte is the type of jump and the second is where it's going to jump to. This is a short jump, and the hex value for an unconditional short jump is EB. So here's what we're actually changing:

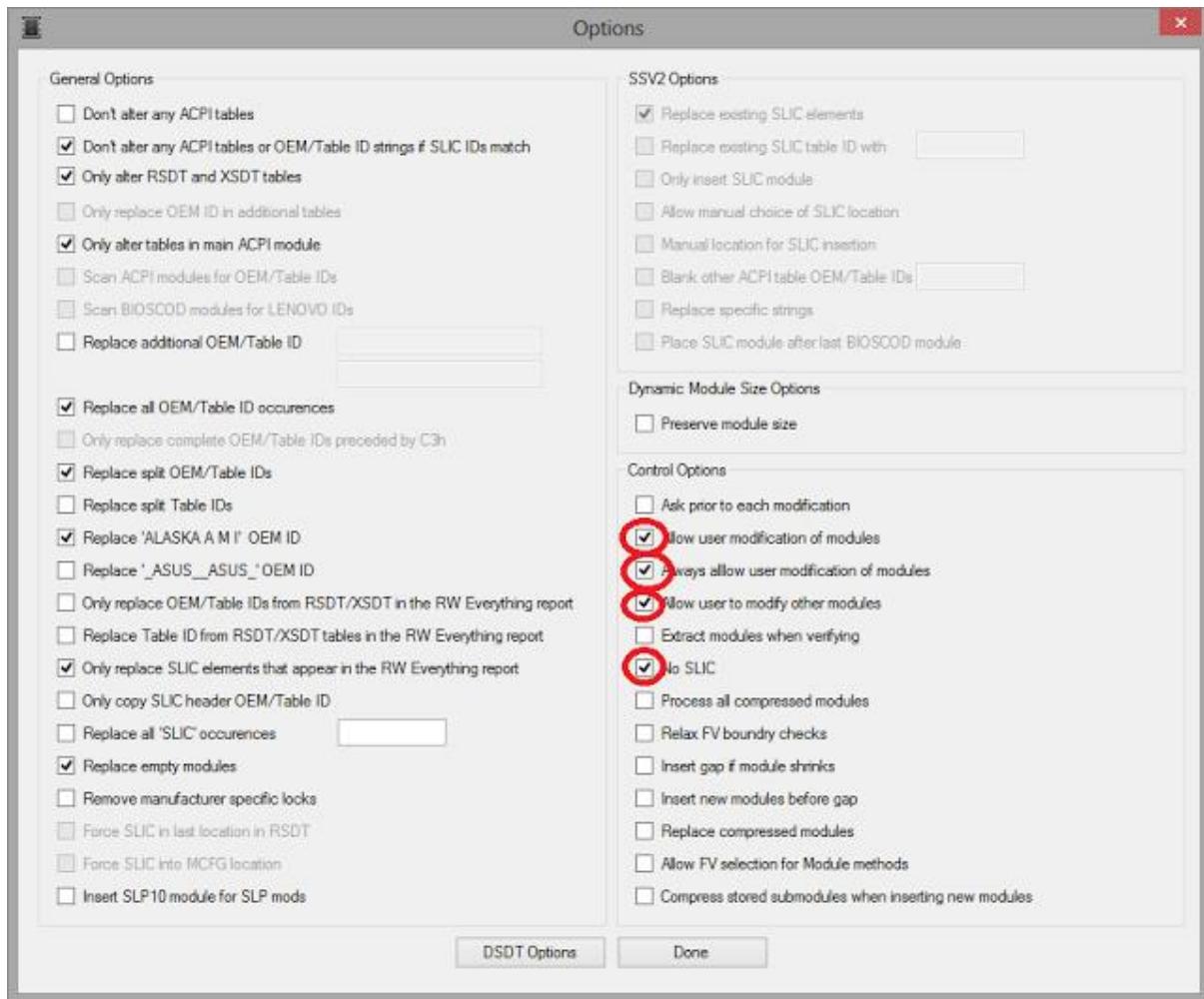
So, here's what the resulting changes look like:

```

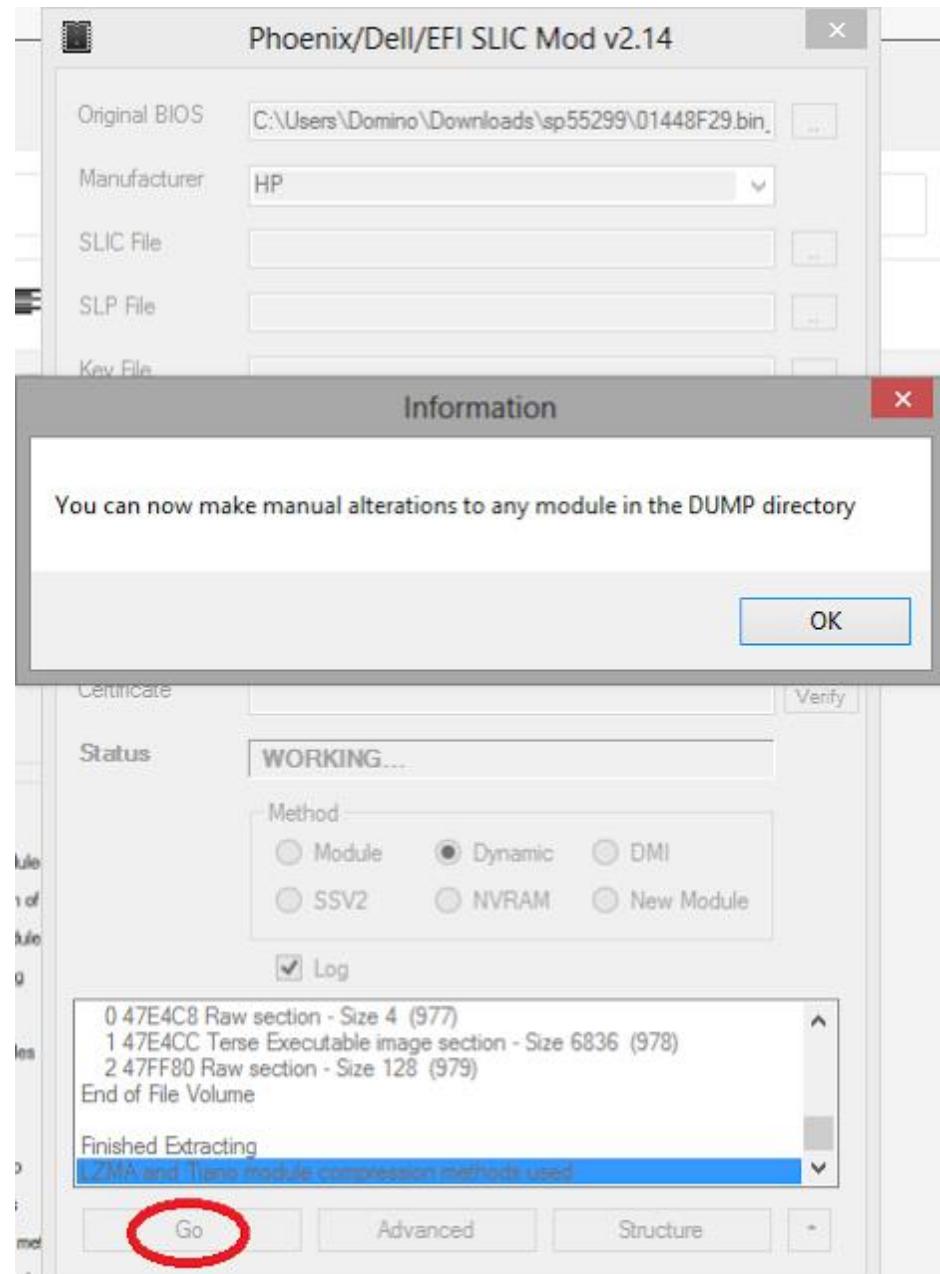
Occurrences of binary: DF 42 4D B5 52 39 51 | Hex View-A | Structures | Enums | Imports
.text:0000000180001061    and    r11b, 11h
.text:0000000180001065    lea    r9, unk_18007F320
.text:000000018000106C    lea    r8, unk_18007EBE0
.text:0000000180001073    movzx  ecx, r11b
.text:0000000180001077    lea    rdx, unk_18007EB30
.text:0000000180001081    sub    ecx, 1
.nop
.nop
.text:0000000180001082    nop
.text:0000000180001083    cmp    rcx, unk_18000F090
.text:0000000180001086    jnp    short loc_180001099
.text:000000018000108D    nov
.jnp    byte ptr [rsp+3C8h], 0
.short loc_1800010DA
.text:000000018000108F ; -----
.text:000000018000108F    nov
.jnp    short loc_180001099
.text:0000000180001097 ; -----
.text:0000000180001099 ; -----
.text:0000000180001099 loc_180001099: ; CODE XREF: .text:000000018000108D↑j
.and   qword ptr [rsp+38h], 0
.mov  [rsp+30h], rax
.lea   rax, unk_18007FA00
.text:00000001800010A4    mov    [rsp+28h], rax
.text:00000001800010B0    lea    rax, unk_18007F390
.text:00000001800010B7    mov    byte ptr [rsp+20h], 2
.text:00000001800010BF    mov    [rsp+20h], rax
.text:00000001800010C4    call   sub_180000A24
.text:00000001800010C9    jmp    short loc_1800010F6
.text:00000001800010CB ; -----
.text:00000001800010CB    mov    byte ptr [rsp+3C8h], 1
.text:00000001800010D3    lea    rcx, unk_18000F090
.text:00000001800010DA ; -----
.text:00000001800010DA loc_1800010DA: ; CODE XREF: .text:0000000180001097↑j
0000108D|000000018000108D: .text:000000018000108D

```

Now the program always jumps to 0x180001099 which references those two tabs. So, lets try this out. Produce a DIF file in IDA Pro by going to File | Produce file | Create DIF file. I recommend you don't save it in the DUMP folder because it will most likely be deleted by Andy's tool at some point. A DIF file contains the offsets and changes that we made in IDA Pro. IDA Pro can't physically edit a file, so we have to use the information in the DIF file and a hex editor to apply the changes. You can close IDA Pro now. Before actually applying the changes with a hex editor, go back to Andy's tool and press the Advanced button. We want to enable the ability to make modifications to the modules. So, these are the settings I changed. I also checked No SLIC because otherwise we would have to select a SLIC table in order to repack our changes. I'm fine with my BIOS current SLIC table.



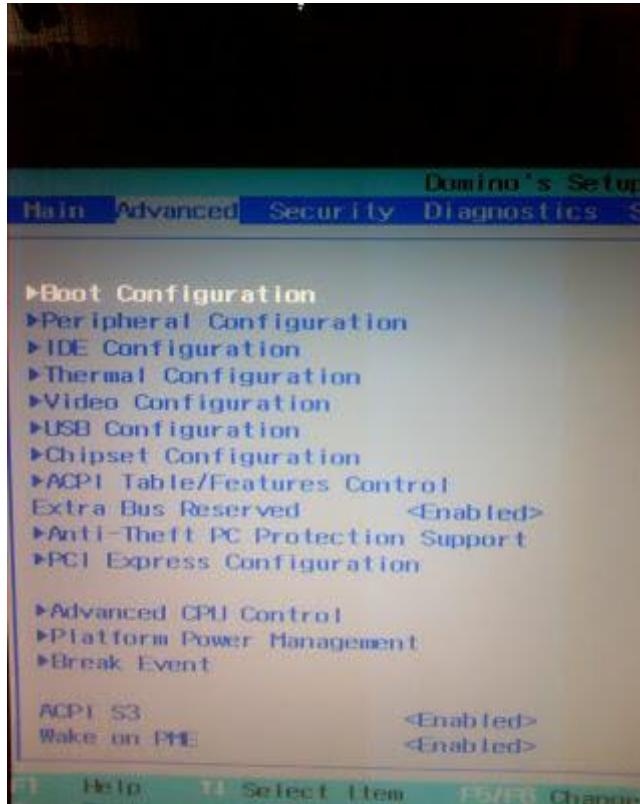
Press Done to get back to the main screen of Andy's tool. Then press the Go button. When this message comes up, don't press Ok yet.



We need to modify the setup utility module first. So, open the same file we disassembled with a hex editor and apply the changes based on what the DIF file says.

This difference file has been created by IDA Pro
PE39542PE-C1D3-4E7B-657C-0048406FFC30_2_480.DROM
00001031: 7E 90
00001032: 48 90
00001033: 7E 9E
  These are the new values.
Go to these offsets

Save the file. Now you can press Ok on the message from Andy's tool, and it should repack your BIOS with your modified SetupUtility module. Let's try it out. Rename Andy's tool's outputted file, mine's named 01448F29_SLIC.bin, to what the original rom was called, mine's 01448F29.bin. This'll replace the original rom with the modified one. Now run InsydeFlash.exe. Press Start, wait for it to initialize, then press Ok. It will now flash your computer with your modified BIOS then restart. Upon startup, press the key that corresponds to your setup utility, mine's F10, to view your changes. Here's mine:



NO WAY!! An advanced tab! That's weird??? Why didn't it unlock two tabs? Shouldn't there be seven tabs now? As it turns out, I haven't found a way to enable all seven tabs at once in my BIOS. But I do have a way of replacing an existing tab with this hidden seventh tab. Lets go back to the disassembled code where we changed the jump locations. Now let's change one of the referenced tabs to the seventh tab. My hidden tab is at address 0x18007F490 (I know this because it's the only one not referenced in the disassembled function we edited), so let's change the line of code "lea rax, 0x18007FA00" to reference this tab.

```
Occurrences of binary: DF 42 4D B5 52 39 51 | Hex View A | Structures | Enums | Imports | Exports | 
.text : 000000180001081      nop
.text : 000000180001082      nop
.text : 000000180001083      cmp    ecx, 10h
.text : 000000180001086      lea    rcx, unk_18000F090
.text : 00000018000108D      jnp    short loc_180001099
.text : 00000018000108F ; -----
.text : 00000018000108F      mov    byte ptr [rsp+3C8h], 0
.text : 000000180001097      jnp    short loc_18000109A
.text : 000000180001099 ; -----
.text : 000000180001099      lea    rax, 0x18007FA00 ; CODE XREF: .text:000000180001080+j
.text : 000000180001099      and   qword ptr [rsp+3Bh], 0
.text : 00000018000109F      mov   [rsp+3Bh], rax
.text : 0000001800010A4      lea   rax, unk_18007F390
.text : 0000001800010A8      mov   [rsp+2Bh], rax
.text : 0000001800010B0      lea   rax, unk_18007F390
.text : 0000001800010B7      mov   byte ptr [rsp+3C8h], 2
.text : 0000001800010BF      mov   [rsp+2Bh], rax
.text : 0000001800010C4      call  sub_180000A24
.text : 0000001800010C9      jnp   short loc_1800010F6
.text : 0000001800010C8 ; -----
.text : 0000001800010CB      mov   byte ptr [rsp+3C8h], 1
.text : 0000001800010C8      lea   rcx, unk_18000F090
.text : 0000001800010D0      ; CODE XREF: .text:000000180001097+j
.text : 0000001800010D0      and   qword ptr [rsp+3Bh], 0
.text : 0000001800010D6      mov   [rsp+3Bh], rax
.text : 0000001800010E5      lea   rax, unk_18007F390
.text : 0000001800010EC      mov   [rsp+2Bh], rax
.text : 0000001800010F1      call  sub_180000A24
.text : 0000001800010F6
```

The hex values for this line are 48 8D 05 55 E9 07 00. The first three bytes are the load affective address into rax part,

and the last four bytes are the offset of the address. This is a relative address based off the current instructions address. And it's stored in little endian. So, if you good with math you can determine the new values with a calculator, or you can just change some values and see if they'll work right by seeing what IDA Pro displays. Here's what mine looked like when I was finished:

Now do the same procedure as before with the DIF file, hex editor, and Andy's tool to produce a newly modified BIOS. Now flash it, and one of the tabs should be replaced with a different one.



HMMMM.... that power tab doesn't have anything new in it that the advanced tab didn't already have. I guess there is something new about interrupts under one of the settings, which I'll never change anyway. I hope your power tab isn't as lame as mine. So, I'm just revert it back to having the advanced tab instead.

This method of replacing one tab with another is probably the easiest way of unlocking one of the hidden tabs. The only downside to it is that you'll have to give up one of the other tabs.

I hope you enjoyed this tutorial. I know it was a long one, but it was worth reading because it did cover some pretty good fundamentals of reverse engineering. IDA Pro makes this process much easier since it can quickly show what references what at any time. You can still do this same process with any other disassembler, but it probably won't be as easy.