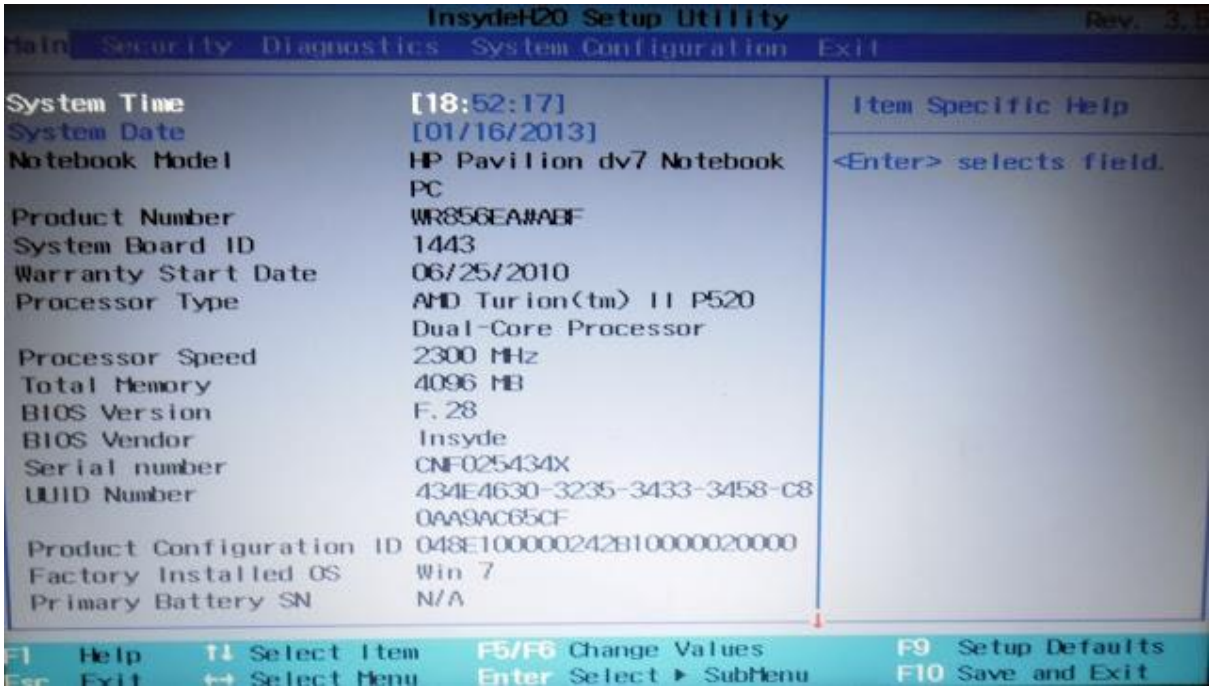
Monday, June 10, 2013

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.

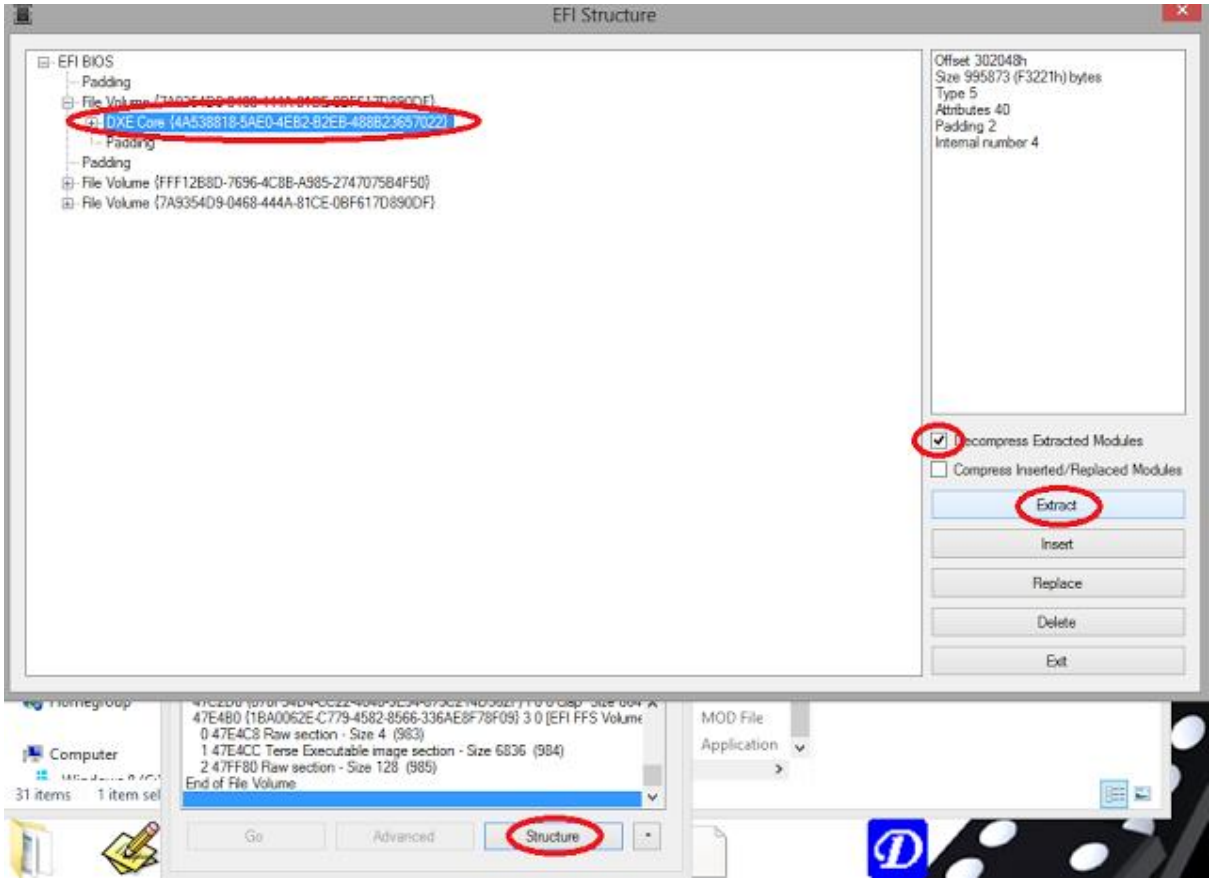


As you can see, it currently has Main, Security, Diagnostics, System Configuration, and Exit tabs. So, I'll show you how to enable the hidden tabs.

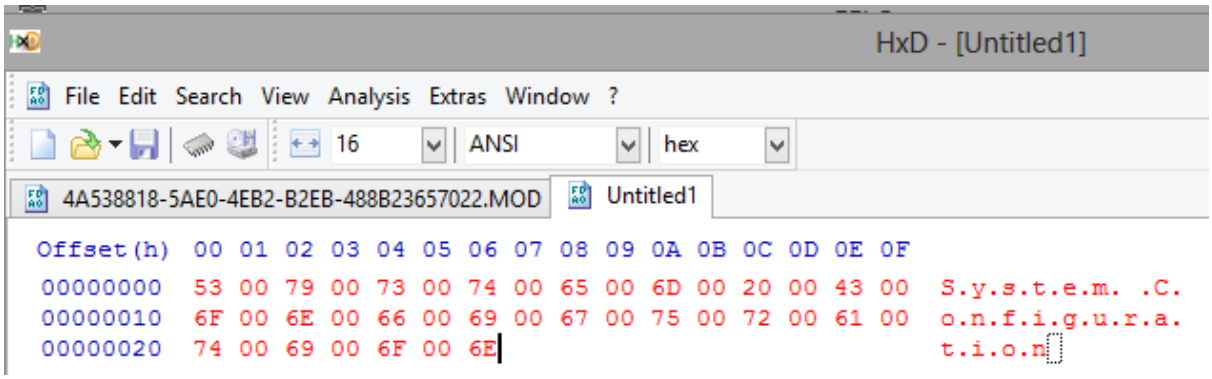
I'll try to keep all my tutorials as generic as possible, but I already know that this will be impossible. Rarely do different BIOS implement these restrictions in the exact same way, so don't expect this to be a sure-fire way to unlock your hidden tabs. As simple as I make these tutorials seem, it still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

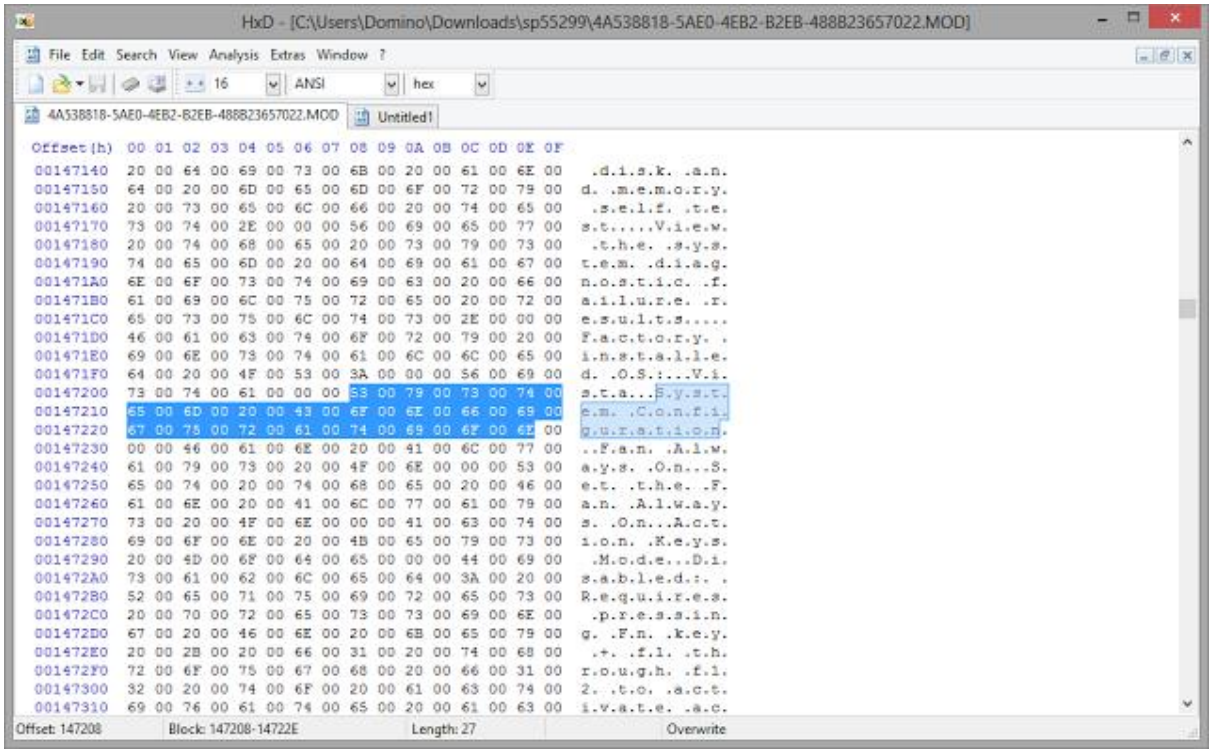
Andy's tool, go to the structure view, check the Decompress Extracted Modules box, and extract the DXE Core module. The latest version of Andy's tool can be downloaded here.



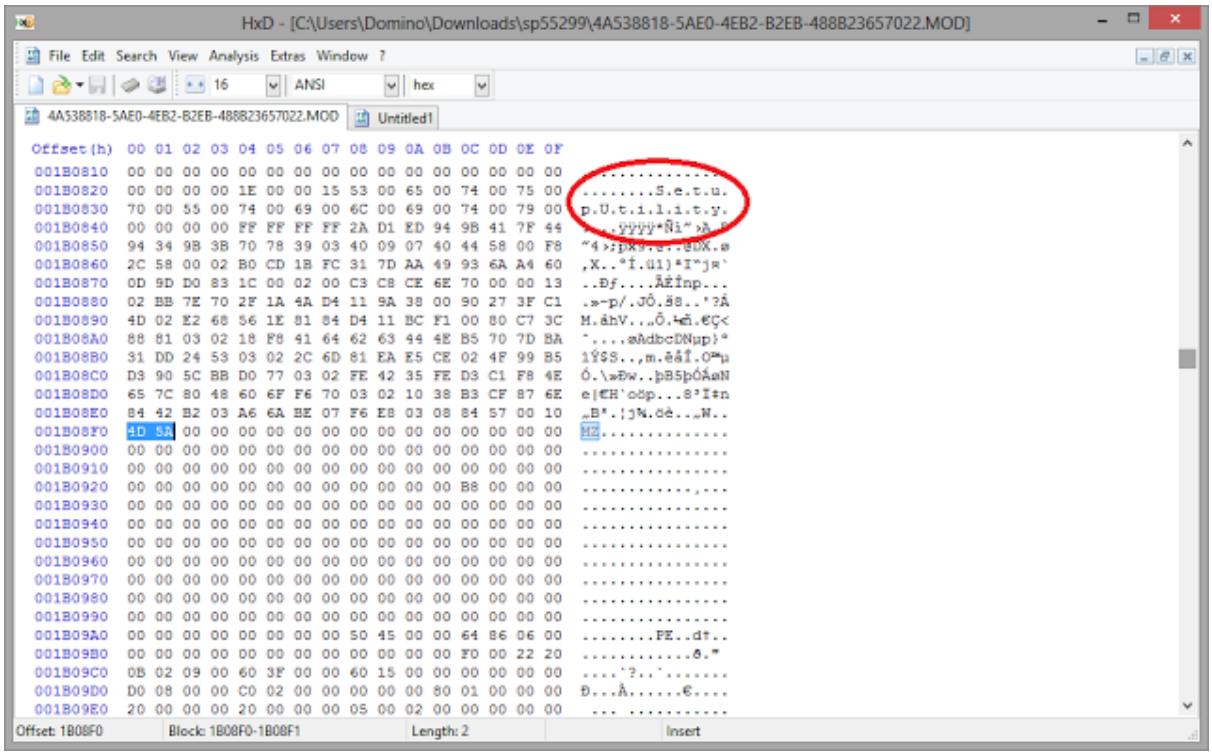
My extracted module is named 4A538818-5AE0-4EB2-B2EB-488B23657022.MOD. Yours might be named something different. So, lets open that module with a hex editor, and search for a familiar string so that we can locate what module contains the setup utility. The hex editor I use is HxD. As a side note, my 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.



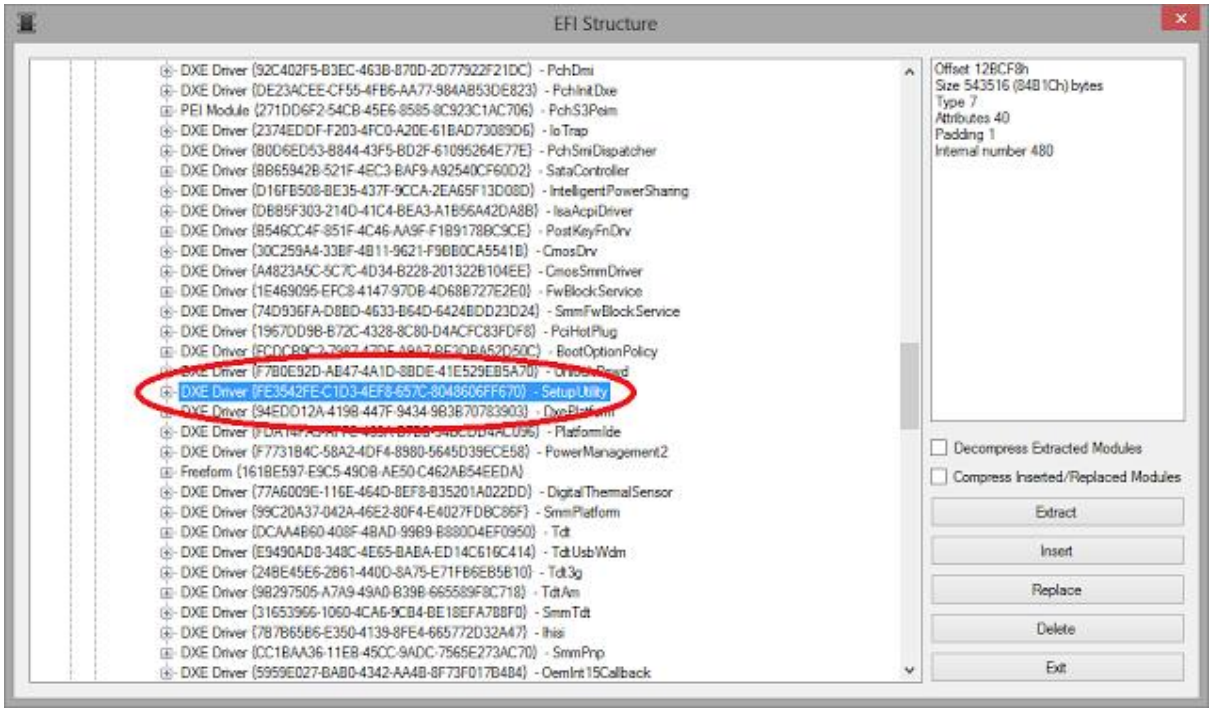
So, lets search for this string in our DXE Core module and see if it exists. Awesome! It found it at offset 0x1A8B8C.



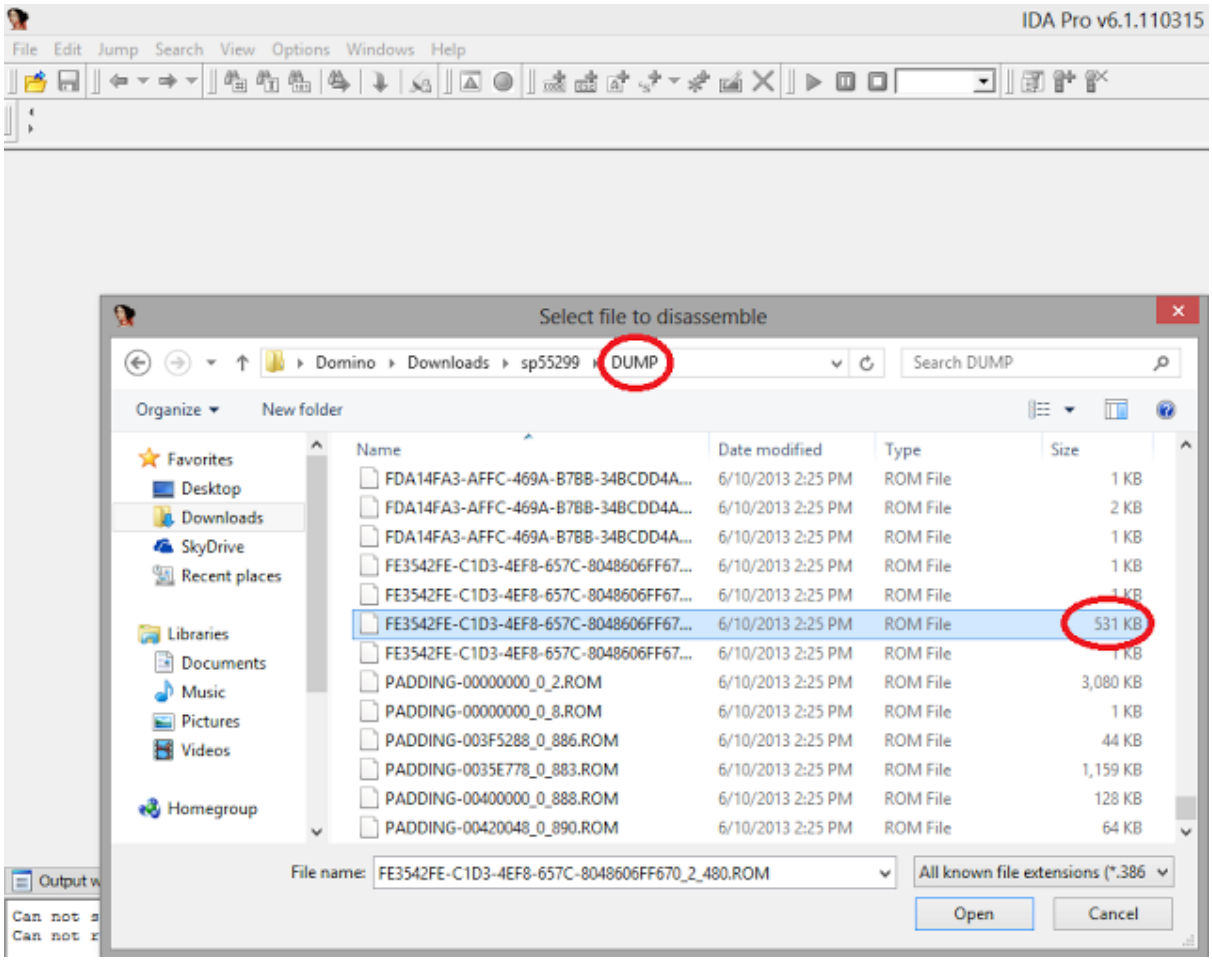
Now we know we're in the correct module. Now search for the hex values 4D 5A. These values are always at the start of a module, and the name of a module is always at the end of a module. So, here's what it finds:



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.

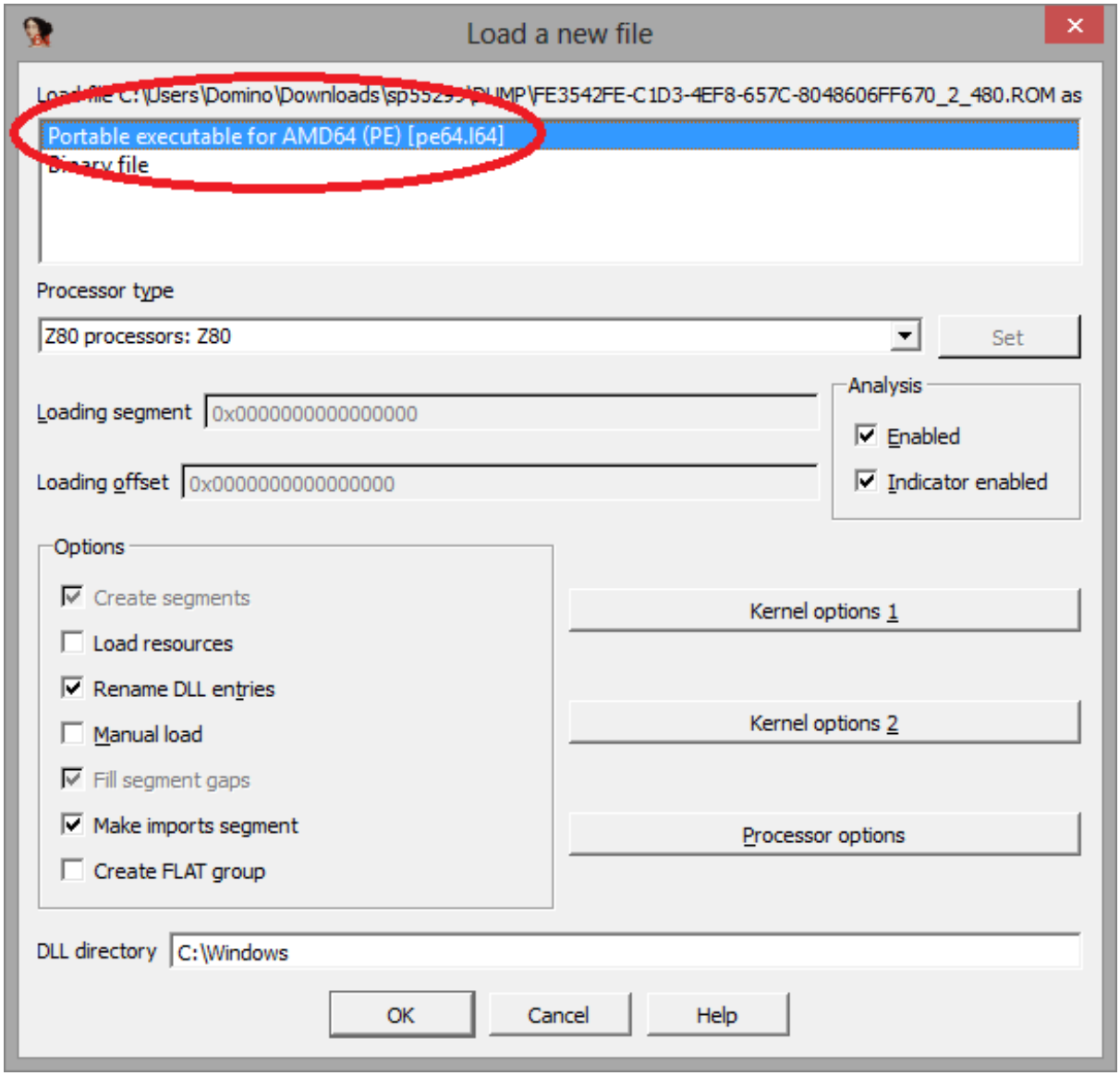


My SetupUtility' GUID is FE3542FE-C1D3-4EF8-657C-8048606FF670. So, lets disassemble this module to get a better understanding of how to mod it. To do this we need to go into the DUMP folder that Andy's tool makes when opening a BIOS file, and open the SetupUtility in there with IDA Pro. Here what I'm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

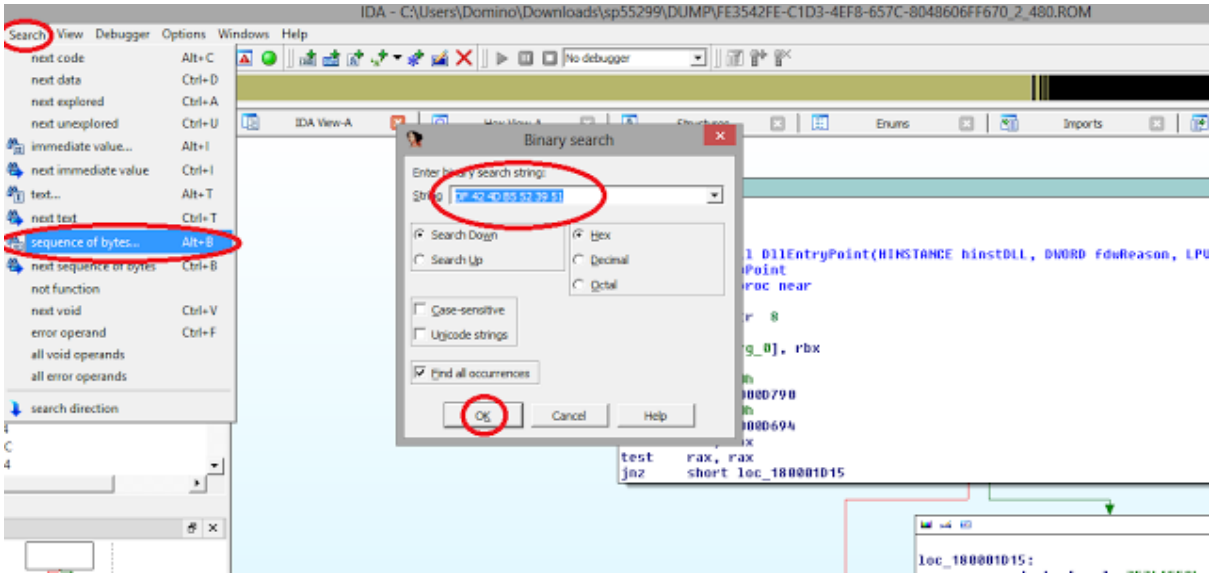


And make sure you open the largest file whose name is similar to your SetupUtility's GUID. Since mine was FE3542FE-C1D3-4EF8-657C-8048606FF670, I'm going to open the 531 kB file which is named similar, FE3542FE-

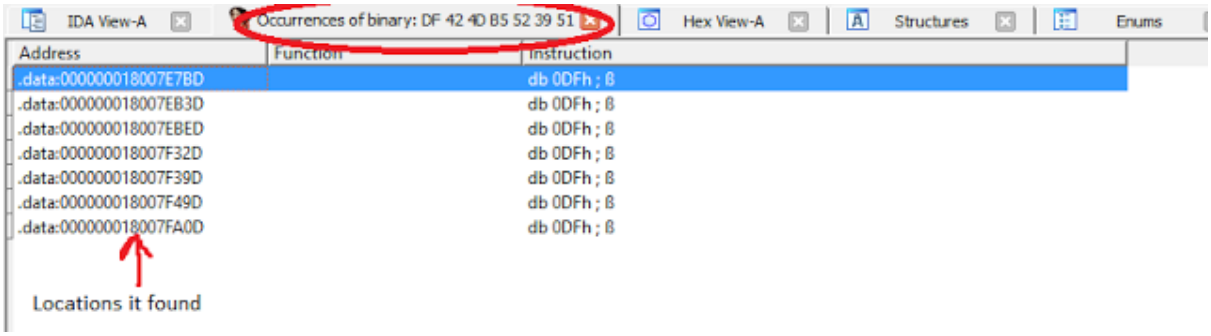
C1D3-4EF8-657C-8048606FF670_2_480.ROM. So, IDA Pro should automatically determine the file type. For me, it's a Portable executable for AMD64.



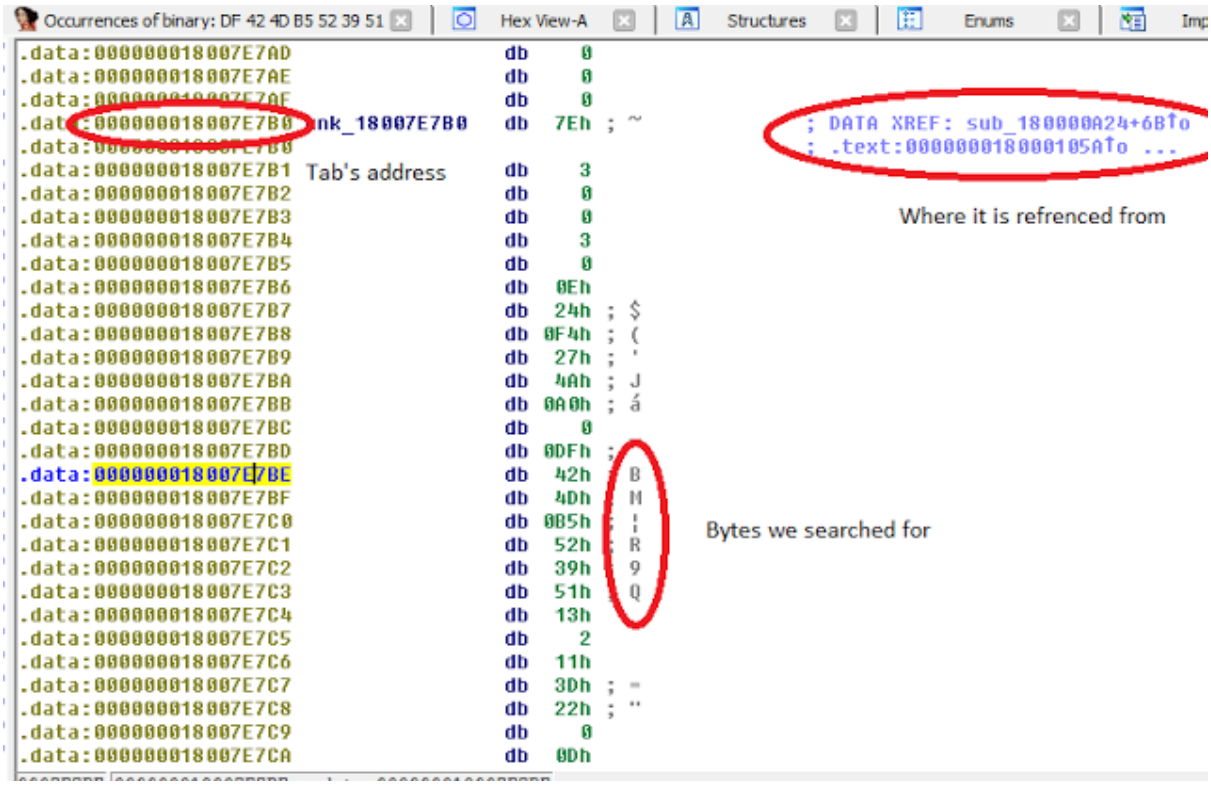
So now that it's disassembled, we have to find out where the tabs are located, then we can see what calls them. I created a program that can dump the internal forms representation used in EFI's human interface infrastructure. This can assist in finding the tab offsets, so you can download it here if 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.



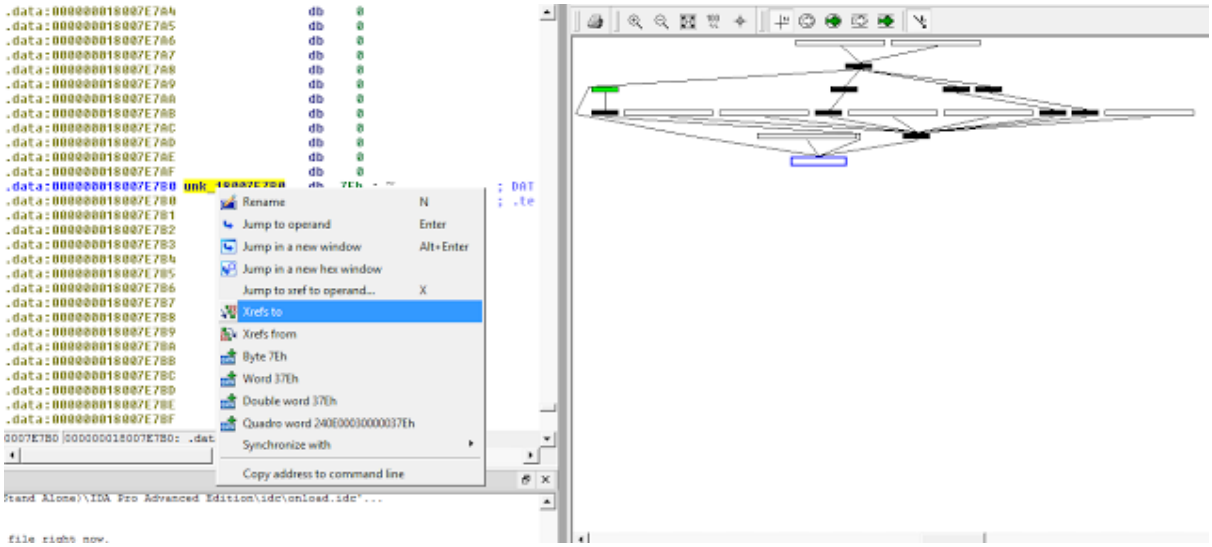
So lets double click on the first one, which takes us here. I said that that byte sequence was in the header, so we need to scroll up about 13 bytes to get to the start of the tab subroutine.



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

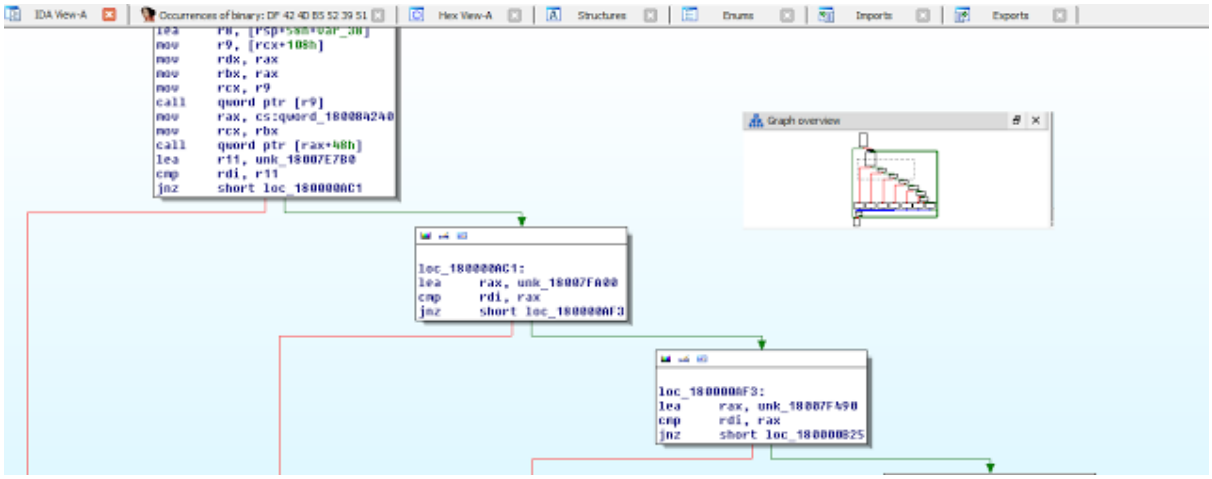
Let’s go back to the first tab and see where it's being referenced from. Right click on the location and select Xrefs to. This will display the connections between this offset and other functions. You can zoom in to get a better view. Here's mine:



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 initialling 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 unk_18007E7B0    db  7Eh ; ~             ; DATA XREF: sub_18000A24+6B↑o
.data:000000018007E7B0                                          ; .text:000000018000105A↑o ...
.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:000000018007E7B8                  db 0F4h ; (
.data:000000018007E7B9                  db   27h ; '
.data:000000018007E7BA                  db   4Ah ; J
.data:000000018007E7BB                  db 0A0h ; á
.data:000000018007E7BC                  db    0
.data:000000018007E7BD                  db 0DFh ; _
.data:000000018007E7BE                  db   42h ; B
.data:000000018007E7BF                  db   4Dh ; M
.data:000000018007E7C0                  db 0B5h ; ¦
.data:000000018007E7C1                  db   52h ; R
.data:000000018007E7C2                  db   39h ; 9
.data:000000018007E7C3                  db   51h ; Q
```

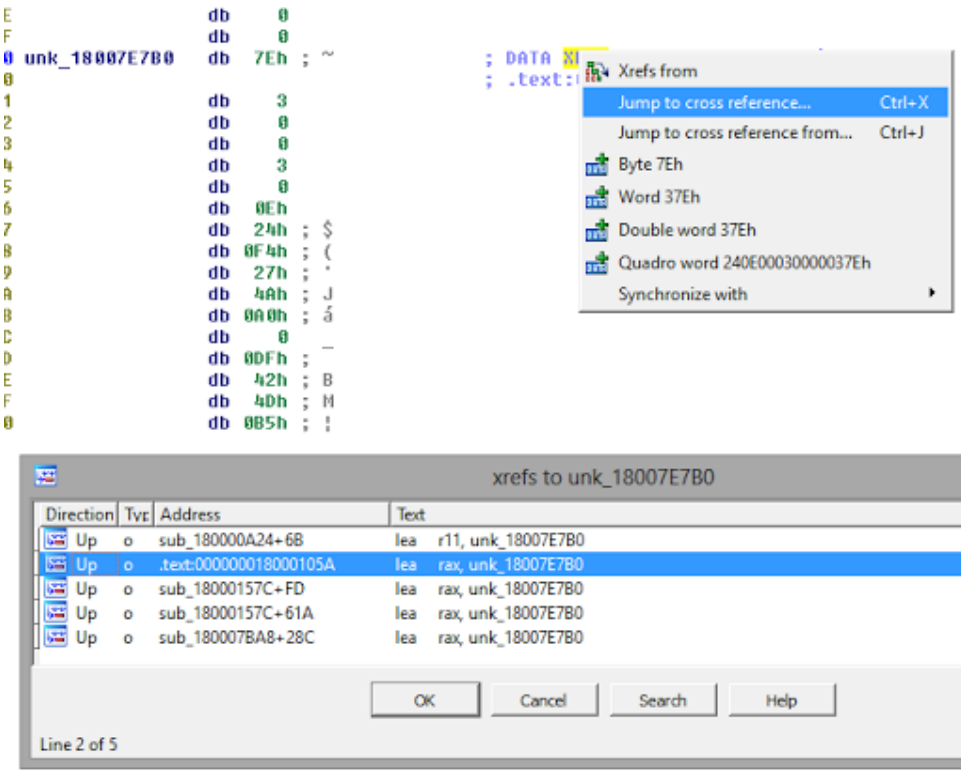
Here's the calling location:



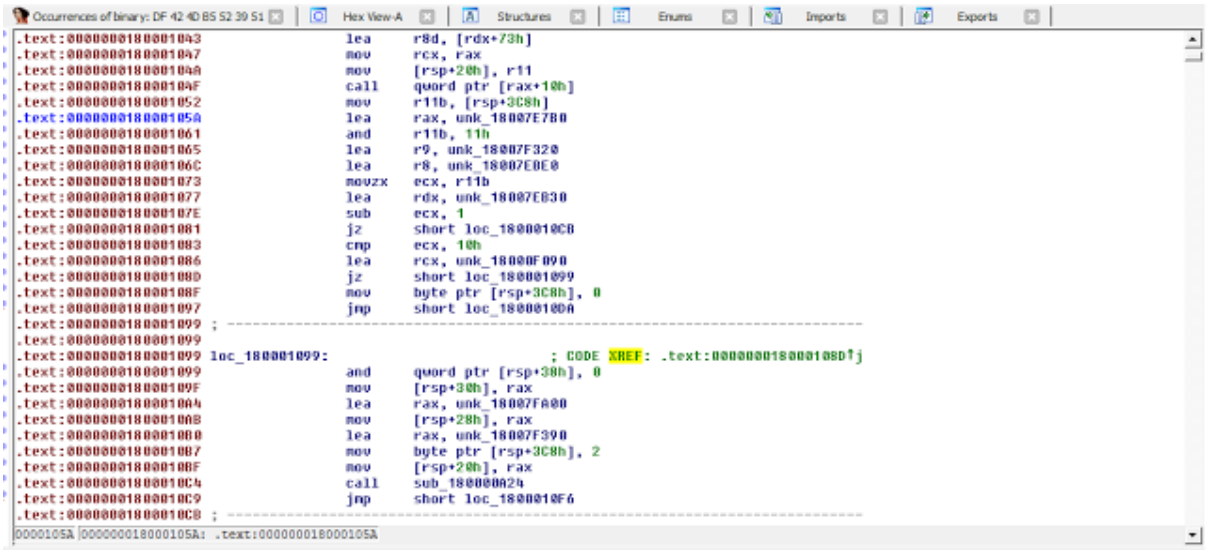
You might be able to determine in IDA Pro's Graph overview that this subroutine is most likely what switches between the tabs when you press left and right. It does reference all the tab offsets, but this is just to determine which one it's currently selecting. This function is not the one we're looking 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 unk_18007E7B0    db  7Eh ; ~             ; DATA XREF: sub_18000A24+6B↑o
.data:000000018007E7B0                                          ; .text:000000018000105A↑o ...
.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:000000018007E7B8                  db 0F4h ; (
.data:000000018007E7B9                  db   27h ; '
.data:000000018007E7BA                  db   4Ah ; J
.data:000000018007E7BB                  db 0A0h ; á
.data:000000018007E7BC                  db    0
.data:000000018007E7BD                  db 0DFh ; _
.data:000000018007E7BE                  db   42h ; B
.data:000000018007E7BF                  db   4Dh ; M
.data:000000018007E7C0                  db 0B5h ; ¦
.data:000000018007E7C1                  db   52h ; R
.data:000000018007E7C2                  db   39h ; 9
.data:000000018007E7C3                  db   51h ; Q
```

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.



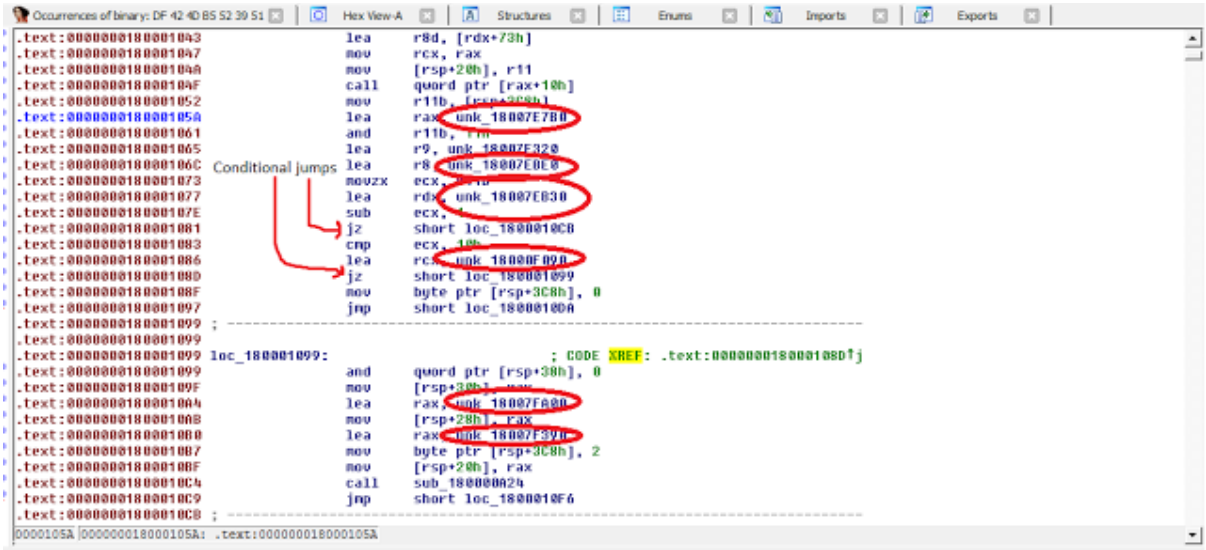
So, here's what the second calling function looks like. Don't be surprised that it's not in a flow chart view. IDA Pro isn't perfect, so sometimes it can't produce this style for all function. As a side note, you can press the space bar to swap back and forth between the flow chart view and the assembly 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 function before you find the one you're looking for.



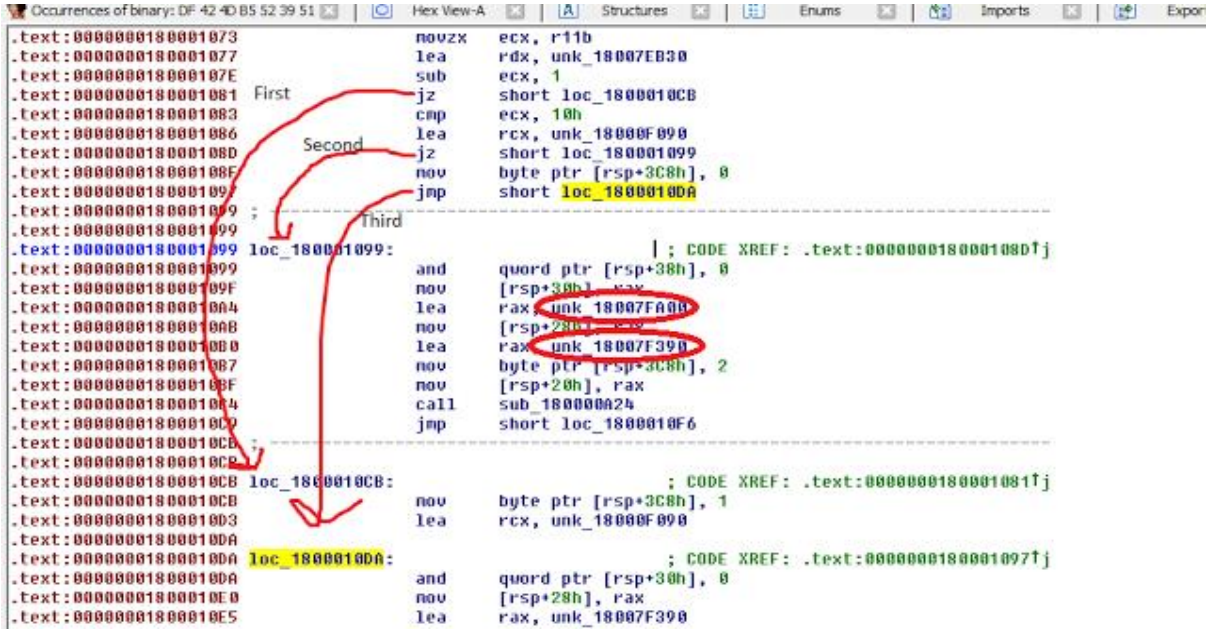
Since we can't see the bigger picture of this subroutine easily, we'll have to look through it. The main things you want to search for are conditional jumps that avoid one of the tab offsets. So once again, here's the ones I'm searching for:

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

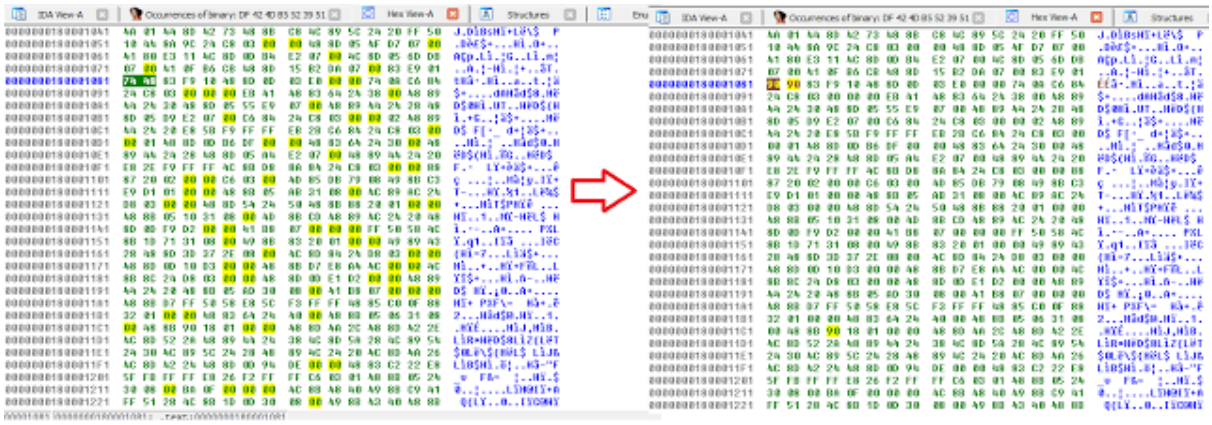
Back to the second calling function. Wow! Almost immediately I notice almost all of my tab locations being referenced. There are also two conditional jumps:



Let me know if you guys think I add too many pictures. I want these tutorials to be through, 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.

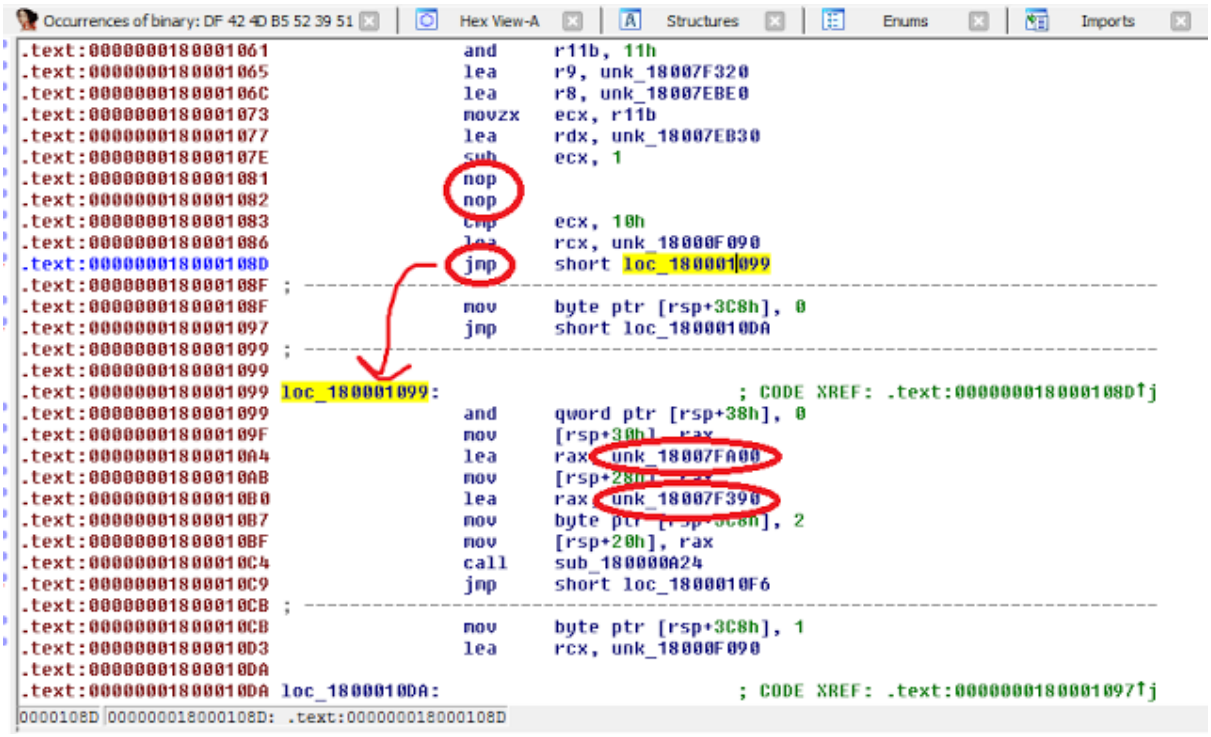


So, to make sure that those two tabs get referenced, we have to change the two conditional jumps. By changing the first one from a jump if zero (JZ) to nothing, and by changing the second JZ to a JMP, we can accomplish our objective. To view the hex values for the first jump, select it and go to IDA 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:

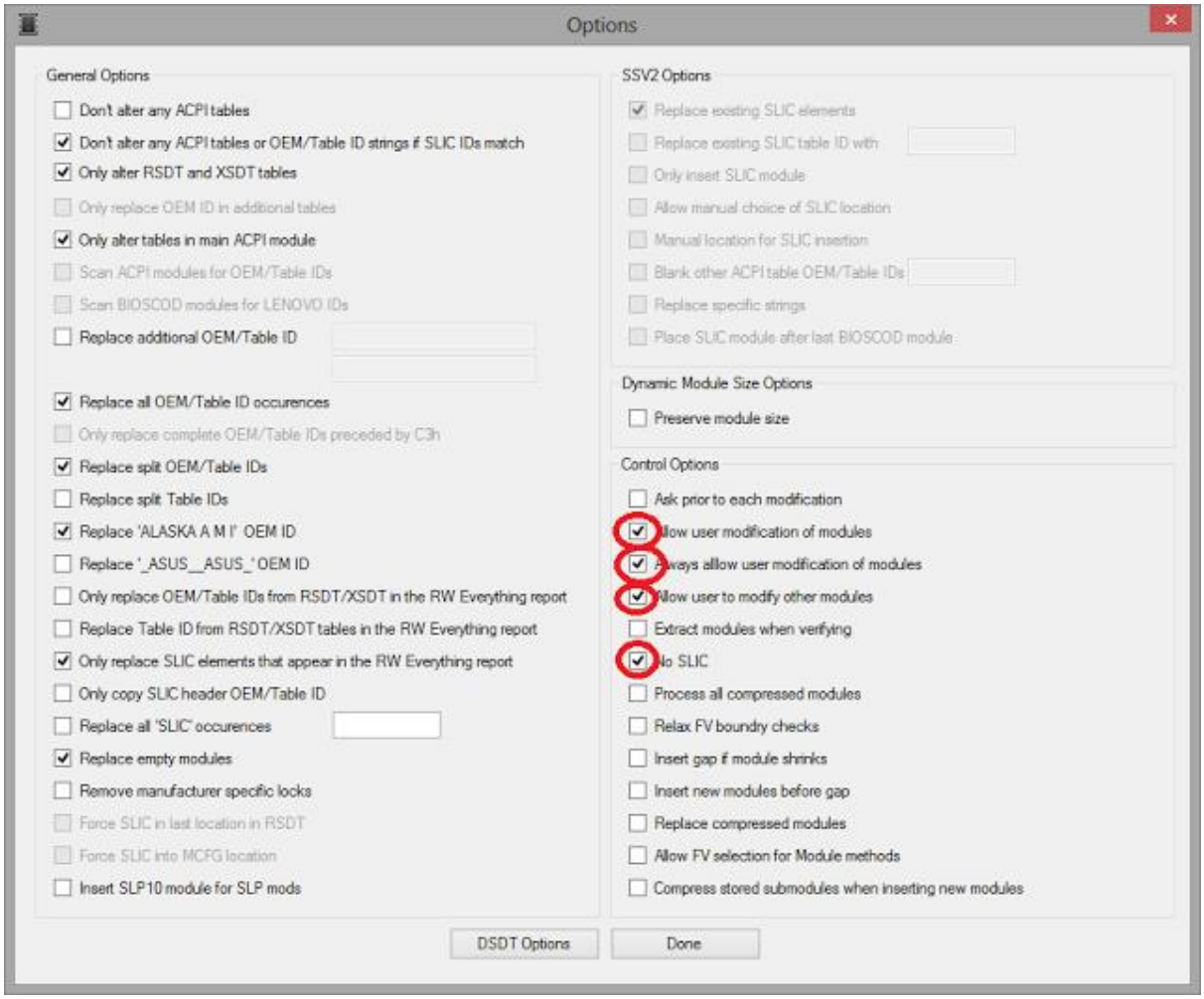


And the second conditional jump's hex values are 74 0A. The first byte is the type of jump and the second is where it's going to jump to. This is a short jump, and the hex value for an unconditional short jump is EB. So here's what we're actually changing:

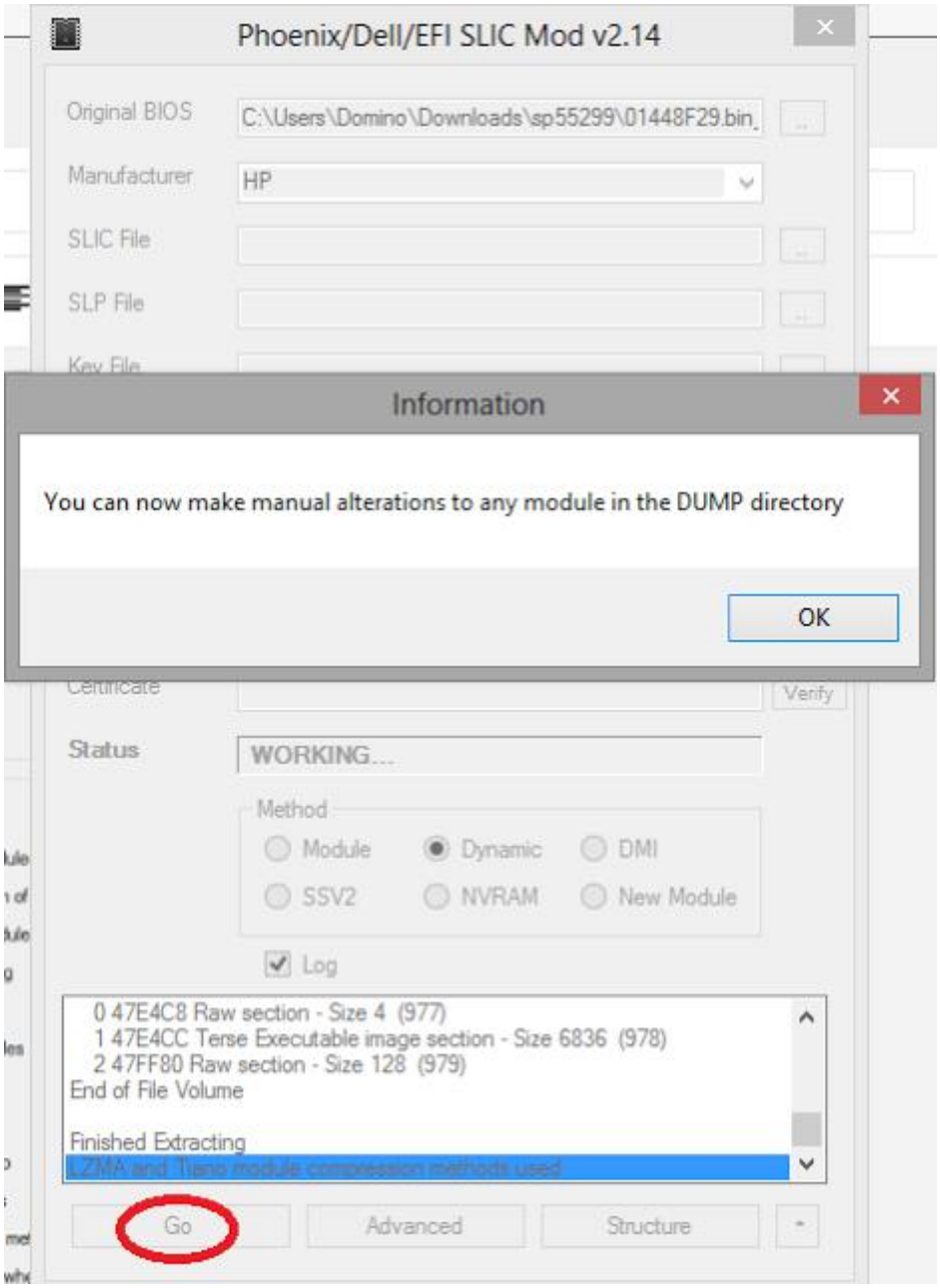
So, here's what the resulting changes look like:



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.

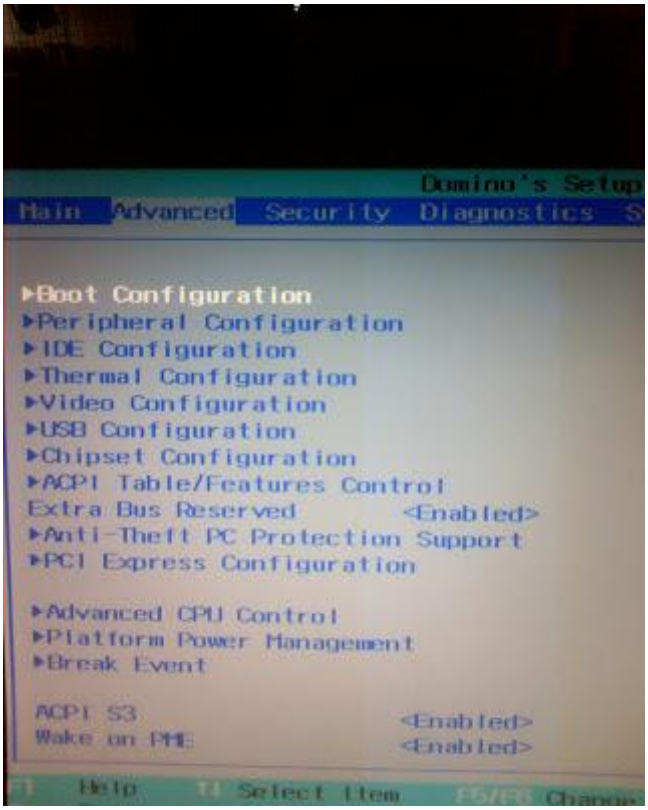


Press Done to get back to the main screen of Andy's tool. Then press the Go button. When this message comes up, don't press Ok yet.

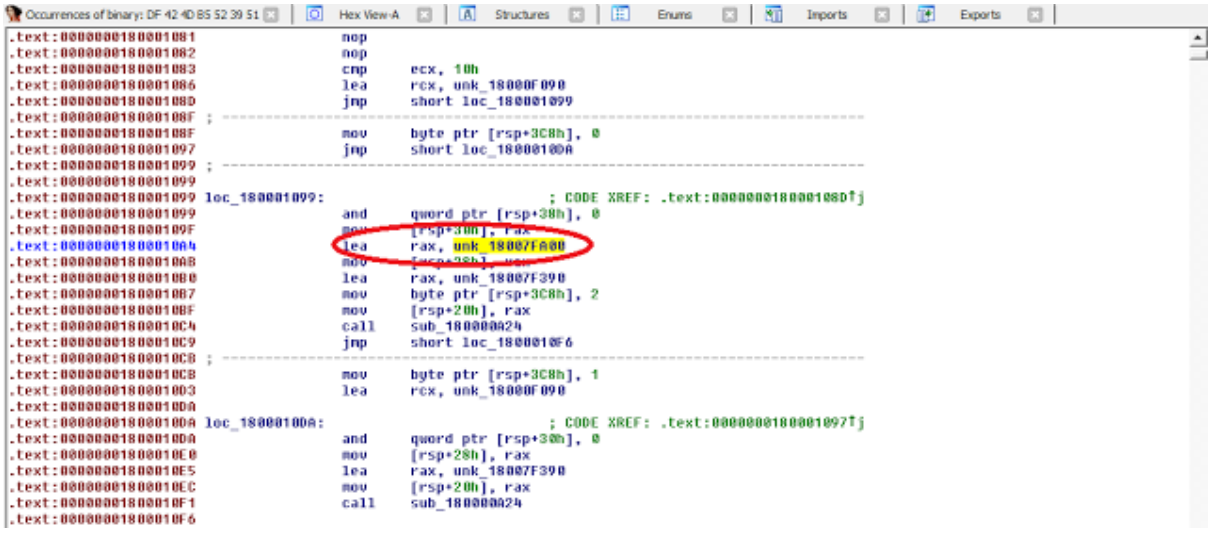


We need to modify the setup utility module first. So, open the same file we disassembled with a hex editor and apply the changes based on what the DIF file says.

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 you 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:

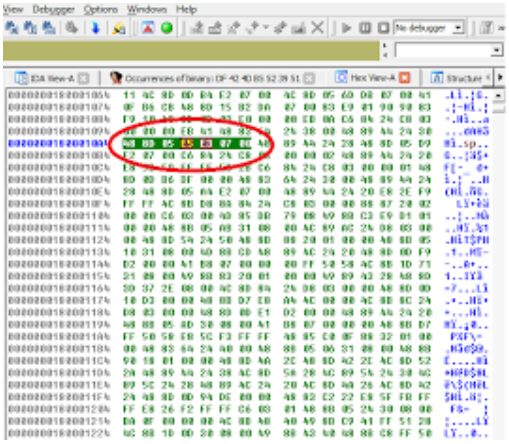


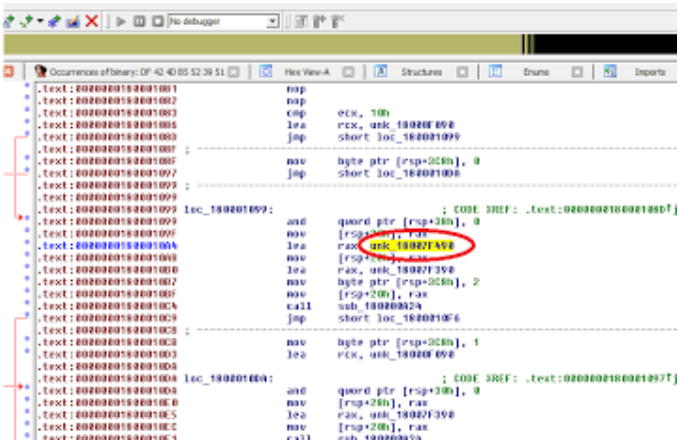
NO WAY!! An advanced tab! That's weird??? Why didn't it unlock two tabs? Shouldn't there be seven tabs now? As it turns out, I haven't found a way to enable all seven tabs at once in my BIOS. But I do have a way of replacing an existing tab with this hidden seventh tab. Lets go back to the disassembled 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.



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 your 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.