Tutorial of the Awesome Dirty Do Checksum Checker V3

How To Fix the “104-Unsupported wireless network device detected. System Halted. Remove device and restart” Error in HP Compaq Notebooks, and run it with your OWN miniPCI Wireless Card?

In V3 of the ADDCC there is a fully implemented Decompresser-Routine. No separate executables are needed anymore. A Source with the principal LOOP and a Pseudo-Code explanation is included. Study it on the last sites of this document.

Knowledge you need:

- the Whitelist
- your new own PCI\VEN ID's
- some locations with ASCII-String to fix the Checksum-Error
- HOW TO Recovering the BIOS when your HP Notebook wont boot after BIOS flash

Software you need:

- Awesome Dirty Do Checksum Checker v3 (ADDCC)
- .NET FX 3.5 for the ADDCC
- Hexeditor: Hxd, or something like this
- Rompaq for flashing your BIOS in real DOS mode (recommended)
  verify the BIOS-File checksum BEFORE flashing

  Alternative:

- a patched HPQFlash for flashing your BIOS in Windows (not recommended) NO checksum verify

  Optional:

- Cabpack for compressing your Rom.bin into Rom.cab
- BootDisk 2 BootStick
- Virtual Floppy Drive (VFD)
- 7zip or something like this

HOW TO Recovering the BIOS

Search in the INTERNET with the Words “Maintenance and Service Guide” , “Recovering the BIOS” and your Notebook Model. Than hopefully you find a HP .pdf with the explanation of the Recovering Procedure for Notebook. Sometimes there is no hint inside, but the Procedure works also, in example nw8440. The Following description is copied from the
Maintenance and Service Guide HP Compaq nc4200 Notebook PC
Document Part Number: 444624-002 April 2007

Recovering the BIOS
✎*The BIOS recovery procedure requires a* ***USB diskette drive*** *and a formatted diskette.*
The BIOS can be recovered if the flash memory is corrupted. Flash memory corruption can occur if the notebook powers down while the BIOS is being updated. When the notebook is turned on, the boot block portion of the flash memory performs an integrity check on the rest of the BIOS image and enters recovery mode if the image is corrupt. BIOS recovery can be forced on a non-functioning notebook by turning on the notebook while pressing and holding the Windows logo key + B on the nonfunctioning notebook keyboard until the caps lock light blinks.
To recover the BIOS:
1. If the nonfunctioning *notebook is docked in an optional docking device, undock the notebook.*
2. Attach the USB diskette drive *directly to a USB port on the nonfunctioning notebook. (USB hubs are not supported for BIOS recovery).*
3. Insert the correct ROMPaq *diskette for the product being updated. The BIOS image file must be located in the root directory of the diskette and must be in contiguous sectors. The easiest way to ensure this is to visit http://www.hp.com, download the Softpaq, and let the Softpaq create the ROMPaq diskette.*
4. Press and hold the Windows logo key + B *on the notebook keyboard (do not use an external keyboard) and turn on the notebook and wait for the caps lock light to start blinking.*
5. Release the Windows logo key + B. *The BIOS recovery procedure takes approximately one minute to read the image from the diskette, and then an additional 15 seconds to program the image into flash memory. The notebook restarts when the BIOS recovery procedure is complete. Do not attempt to turn off the notebook after starting a recovery. If the BIOS recovery procedure stalls, the caps lock light will begin blinking. This situation can arise if the diskette is corrupt or the incorrect ROMPaq is used. If the notebook does not restart after approximately 3 minutes, press and hold the power button, or slide and hold the power switch, for at least 5 seconds to force the notebook to turn itself off. Then repeat the BIOS recovery procedure.*

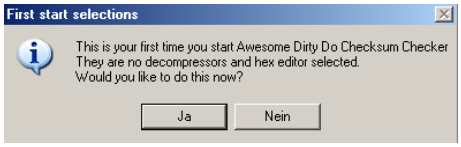
Another way to test the Recovering Procedure is to run it on your laptop without a fail and without any USB drive plugged. Hold down the Keys and start the engine, if the CAPS-LOCK LED blink alternative the Recovering Procedure works principal on your notebook. In my case, the NC4200 case, they wrote “hold the Windows logo key + B” but it works with the four arrow-keys too. A normal USB stick wont work, maybe a special USB stick with a controller that can emulate a Hardware Floppy likes the HP USB FLOPPY DRIVE KEY or some old JETFLASH or some new Netac Minisafe Models. You have to change/modify the USB_DEVICE_DESCRIPTOR.
http://www.gizmodo.de/2008/10/26/usb-stick-kann-auch-floppy-disk.html
http://www.rm.com/Support/TechnicalArticle.asp?cref=TEC134761&nav=0

Tutorial of the Awesome Dirty Do Checksum Checker V3

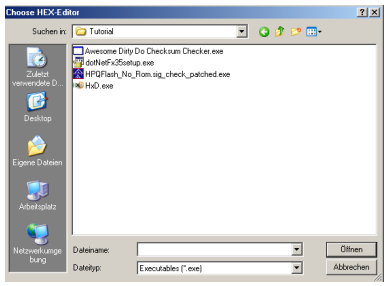
Purchasing the basic BIOS File from the HP Archive:

- extract anyway the original BIOS binary file from the HP archive
  (I prefer the DOS-Floppy-Image and mount it with VFD)
- you will get a file that looks like that
  - 68DTH.BIN, 68TT2.BIN, 68BDD.BIN, 68YAF.BIN, 68YGU.BIN, 68BAS.BIN, ROM.BIN …

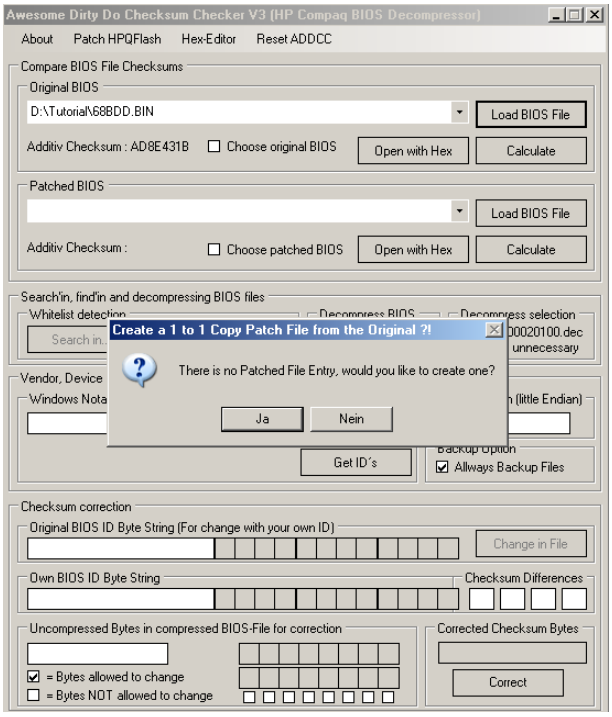
First Start the ADDCC will select a necessary tools



- a Hex-Editor

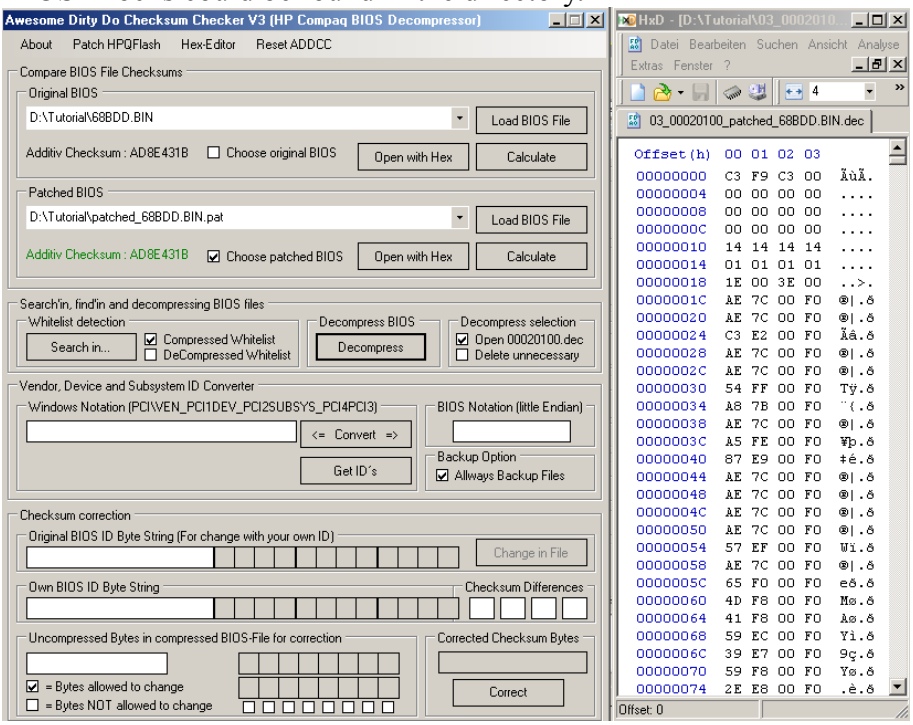


Load the original BIOS file and create automatically a patched copy for your modifications.



Decompress the BIOS

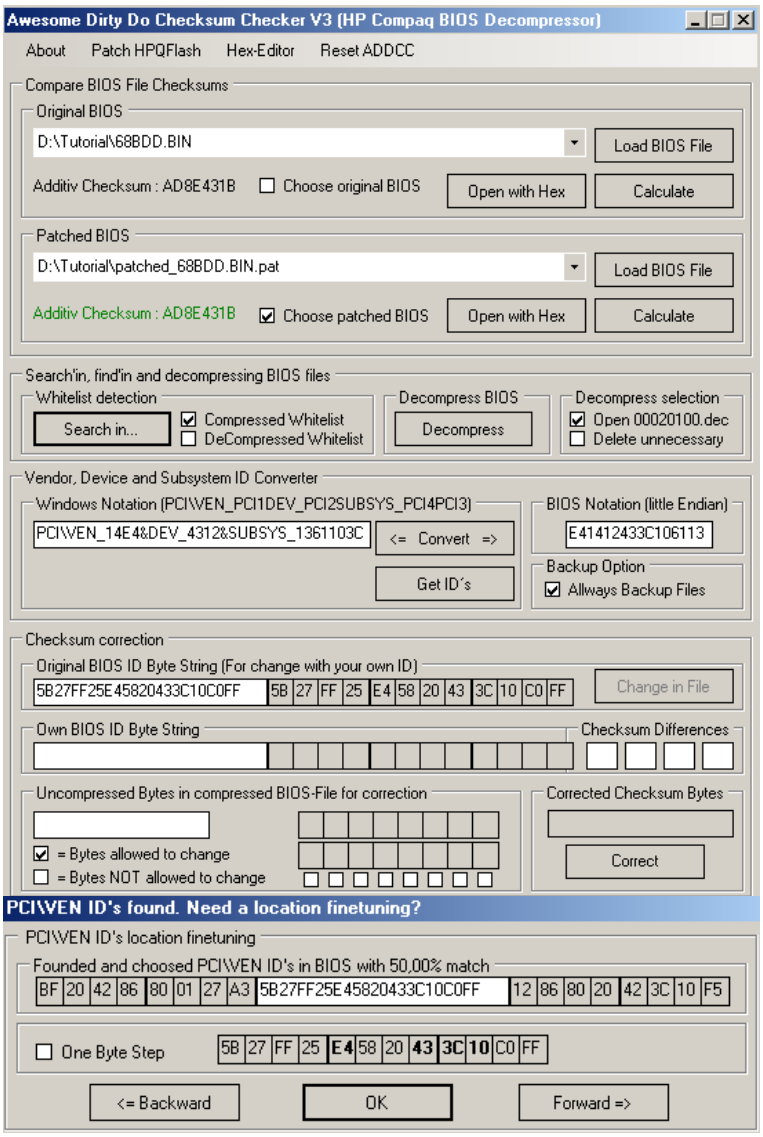
The Decompress-Code will extract both known HP Compaq Headers. It doesn't matter if there some FF's in the BIOS File or not. The decompresses will automatically detect this and give a separate FF Fill bytes Block out. After pressing the Decompress-Button there will be pop up the hex editor with the second POST Bios-Block that includes the decompressed Whitelist. After correct decompression the Search in … DeCompressed Whitelist Function will be enabled and the decompressed BIOS-Blocks could be found in the directory.



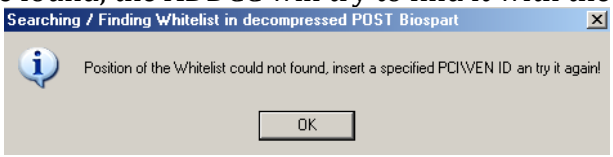
Finding the Whitelist:

After decompression you have two options to find your Whitelist. In some Models the End of the Whitelist can be found with search string “PSQRVW” in the second POST Block. 00021000.dec. The first result is the end of the Whitelist. The second option is the classical, with searching for a PCI\VEN ID. This function is available in both Whitelists.

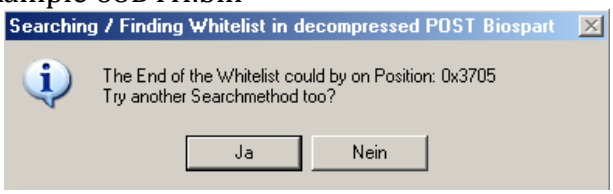
- Insert a PCI\VEN ID from there you believe they could be complete or only a part on the Whitelist
- Choose a file, I prefer the patched BIOS
- after “Convert” the “Find Whitelist” Button will be activate if there is an BIOS Notation entry



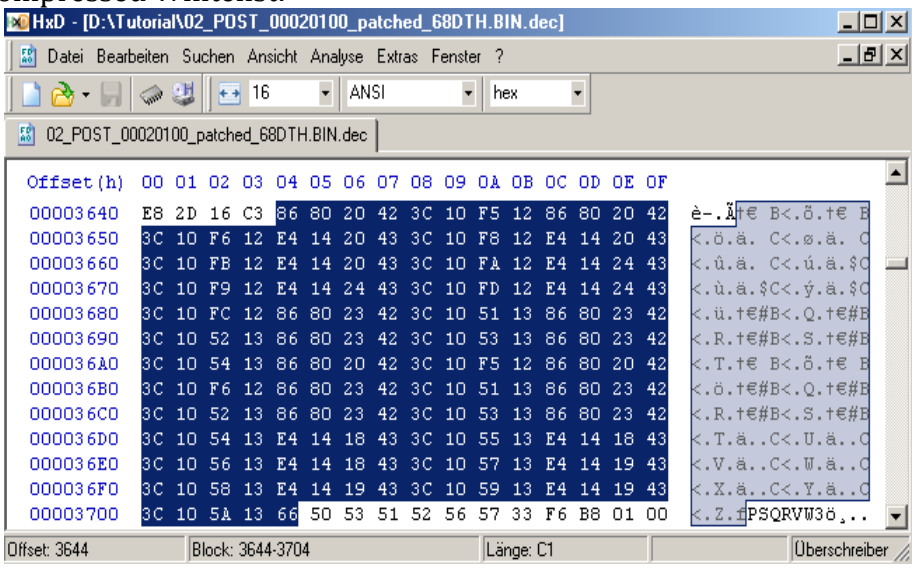
If “PSQRVW” could not be found, the ADDCC will try to find it with the PCI\VEN ID's.



If the “PSQRVW” is found, the ADDCC will give a HEX offset of the end from the Whitelist in the second POST Block. In example 68DTH.bin



In Hex Editor goto Offset 0x3705 and then you will find with the method of “sharp seeing” the complete decompressed Whitelist.

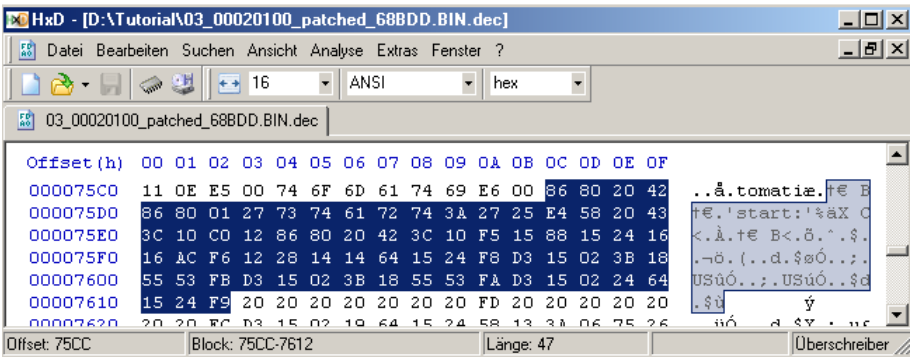


The decompressed files from the BIOS:

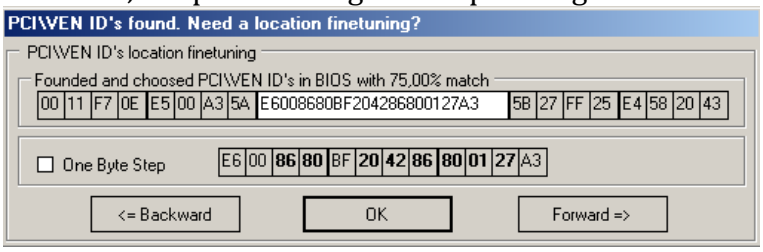
01_00060000_patched_68BDD.BIN.dec
02_00020000_patched_68BDD.BIN.dec
03_00020100_patched_68BDD.BIN.dec
04_000F0000_patched_68BDD.BIN.dec
05_00068000_patched_68BDD.BIN.dec
06_00040000_patched_68BDD.BIN.dec
07_00010100_patched_68BDD.BIN.dec
08_000C0000_patched_68BDD.BIN.dec
09_00018000_patched_68BDD.BIN.dec
10_000E0000_patched_68BDD.BIN.dec
11_00160000_patched_68BDD.BIN.dec
12_00090000_patched_68BDD.BIN.bmp
13_00094000_patched_68BDD.BIN.bmp
14_00096000_patched_68BDD.BIN.bmp
15_00098000_patched_68BDD.BIN.bmp
16_00100000_patched_68BDD.BIN.dec
17_00140000_patched_68BDD.BIN.dec
18_00150000_patched_68BDD.BIN.dec
19_FFF9B88D_ROMBLOCK_patched_68BDD.BIN.bin

Whitelist finetuning in compressed BIOS:

- with a complete entry,the first, from decompressed Whitelist is it so much easier to find the Whitelist in the compressed BIOS. In this case by searching manual with “86 80 20” ➔ “8680204286800127”



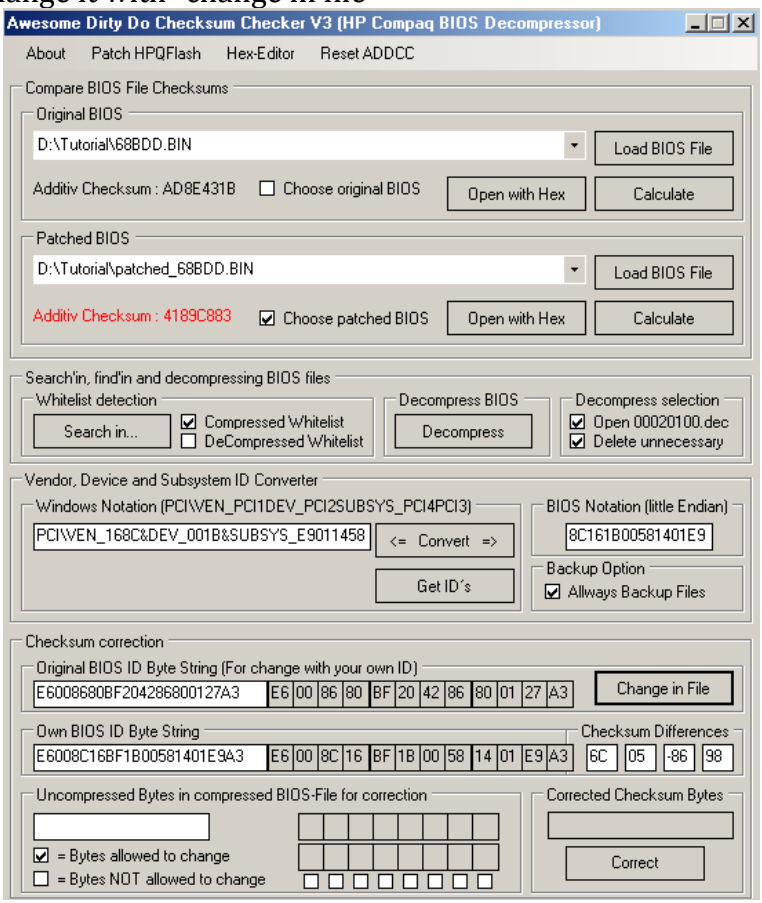
- the position bye shifting is with respect to four byte order for a correct checksum fixing
- the Whitelist is found, the position is good for patching it with another PCI\VEN ID



- this ID is found completely

Change the PCI\VEN ID with your Own new PCI\VEN ID's:

- after OK the Original BIOS ID string is filled with the 12 Byte string from above
- insert your OWN BIOS Byte String ID's end refill it with the redundant bytes and the "BF"
- PCI\VEN_168C&DEV_001B&SUBSYS_E9011458
- 8C161B00581401E9
- a mouse-hover over the "checksum Differences" will calculate the differences caused from your changes
- write and change it with "change in file"



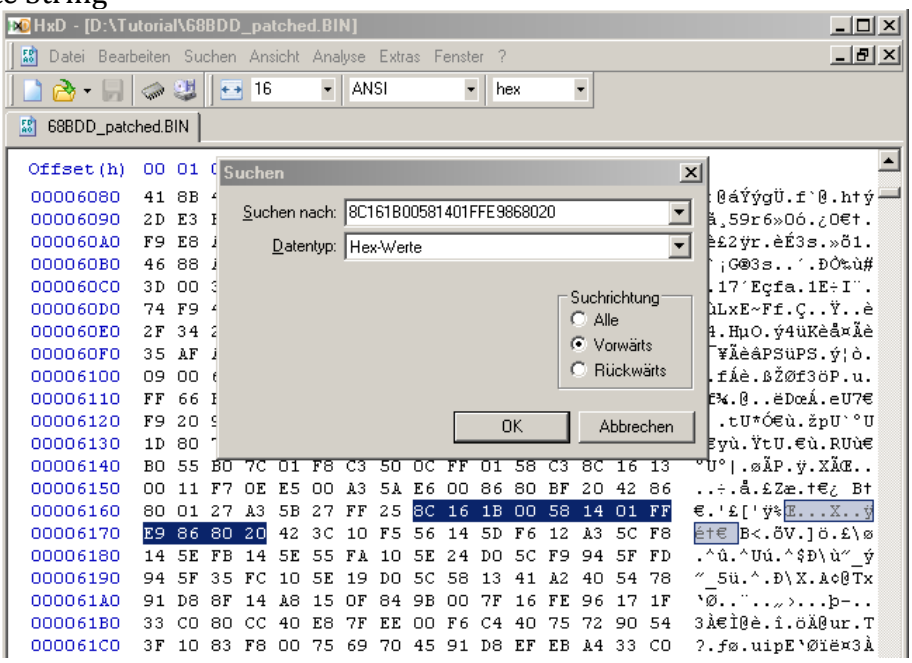
- after the change, the checksum will be wrong, now its time to fix it and check Whitelist for the new ID

Fix the Checksum:

- normally you can use another PCI\VEN ID from the compressed Whitelist to correct the checksum
- in some case, there is only one PCI\VEN Entry that you already used for the change. Than use an Ascii string in your compressed
  00020100.dec that don't need, example “Wake up on LAN”.

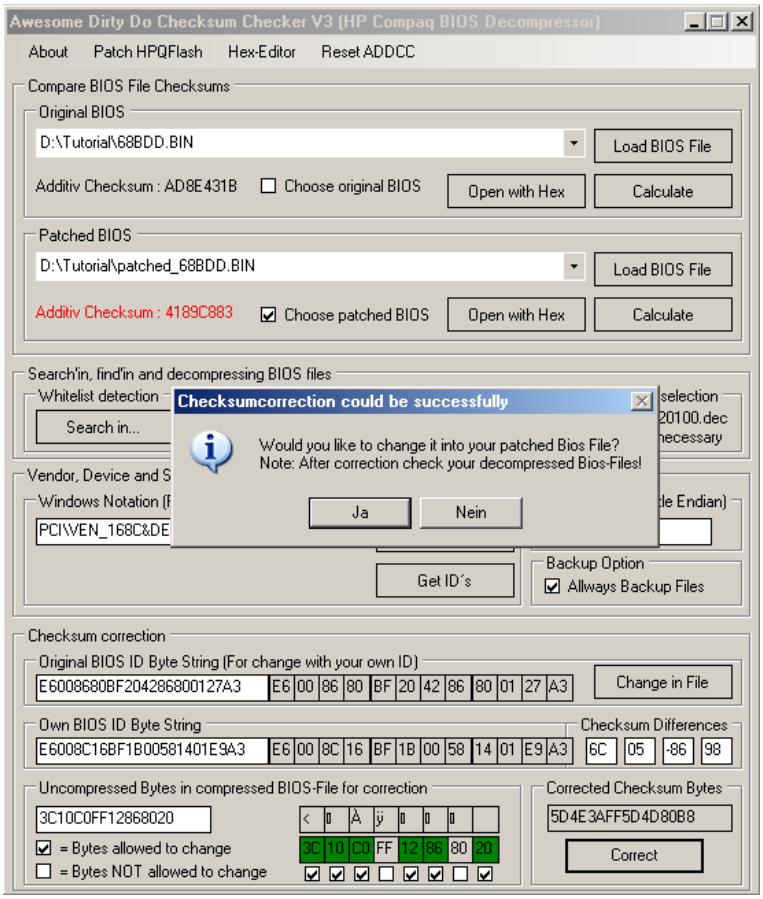
Fix it with another PCI\VEN ID:

- Open the “patched_68BDD.BIN.pat” with Hex-Editor and search for your BIOS ID Byte String



- you can use so many bytes after your own BIOS ID's
- in this case to fix the checksum is very easy

- use the bytes 3C10C0FF12868020 to correct it with the checksum-differences -6C 05 -86 98



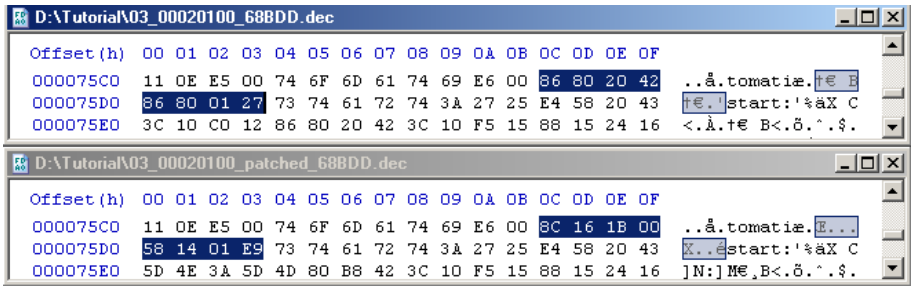
- yes for changing it in your patched BIOS-File and the job is done!

- yes for changing the last different in your patched BIOS-File
- you can see from the step before that there only the first byte is not fixed. After this step, all bytes will be fixed

Tutorial of the Awesome Dirty Do Checksum Checker V3

Verify that the decompressed Whitelist is correct:

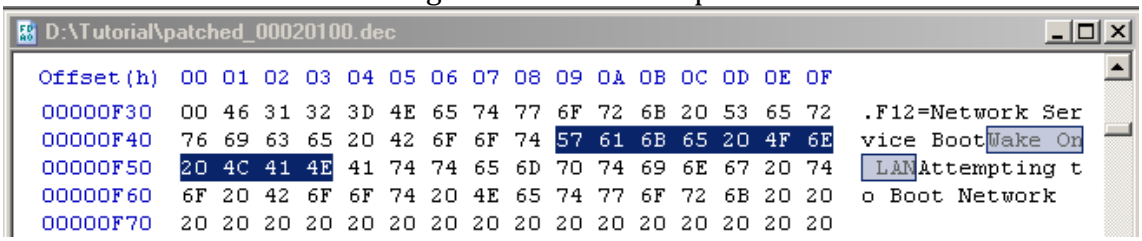
- Open with Hex the original and the patched “00020100.dec”
- compare the both Whitelists



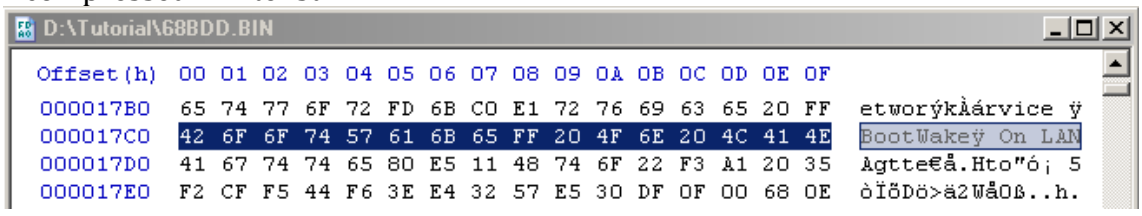
- on the first look its seems to be good, they are only differences on the Whitelist.
- The Positions of the changed PCE\VEN ID's
  are exactly the same
- The Two-File-Compare-Function from the Hex Editor will confirm the result

Fix it with an Ascii-String:

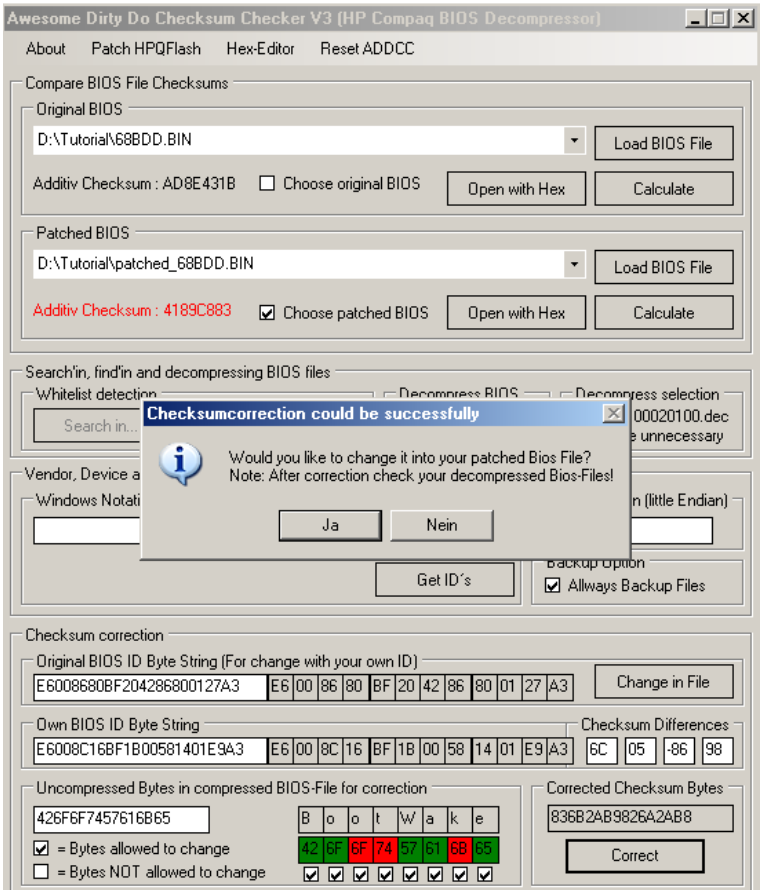
- Search and find an Ascii-String with in the decompressed “00020100.dec”



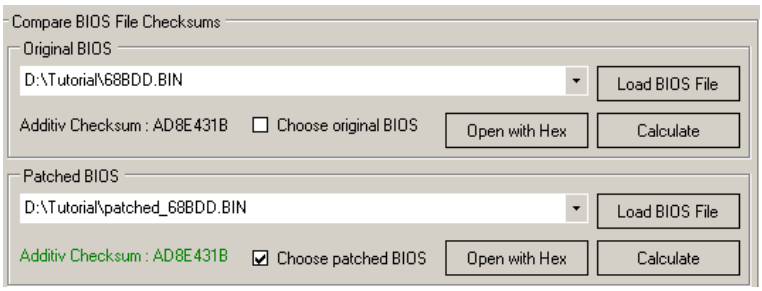
- Well done, there is an Ascii String in it, so lets find them in the compressed Whitelist



- In this case too, we need two steps to correct the checksum
  - 1. use the bytes 426F6F7457616B65 to correct it with the modified checksum-differences -8B 3E C4 43

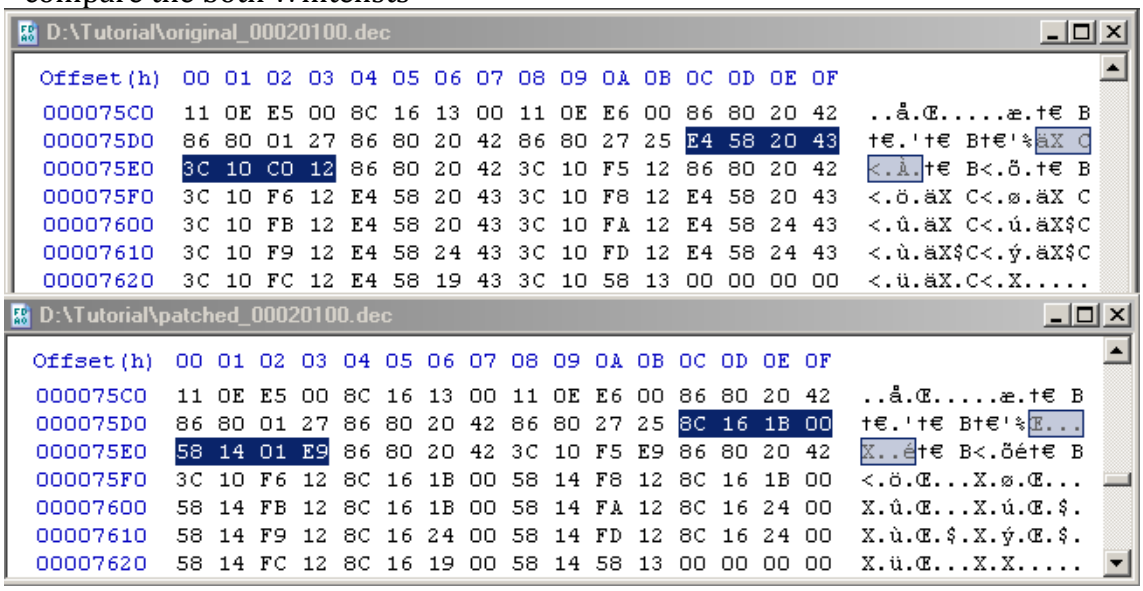


– yes for changing it in your patched BIOS-File

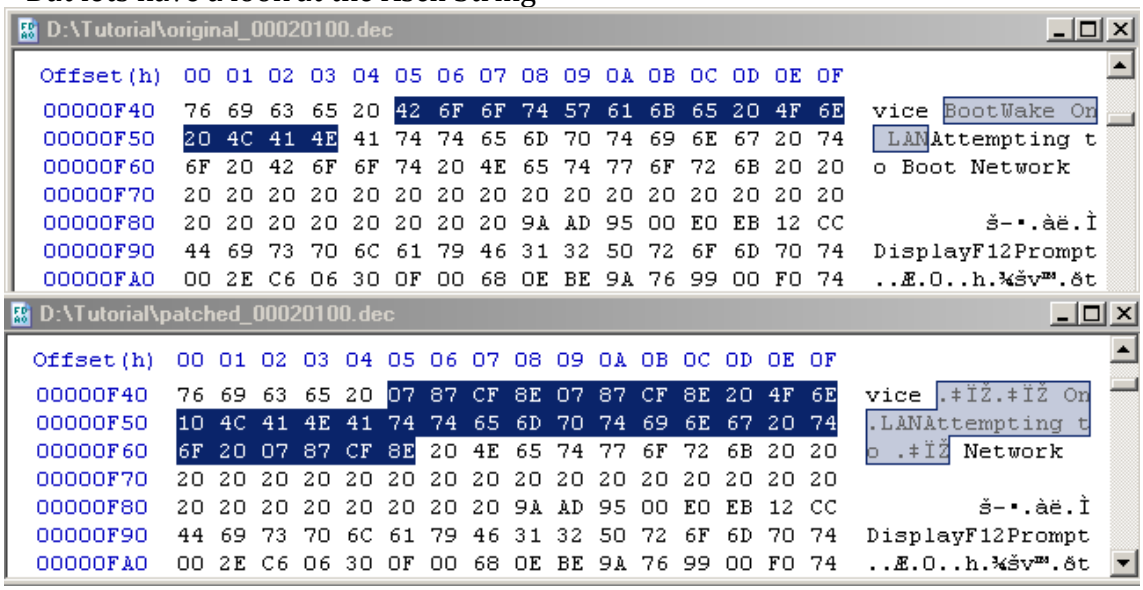


Verify that the decompressed Whitelist is correct:

- Open with Hex the original and the patched “00020100.dec” compare the both Whitelists



- on the first look its seems to be good, they are only differences on the Whitelist.
- The Positions of the changed PCE\VEN ID's are exactly the same
- But lets have a look at the Ascii String



- It looks like only “Boot” and “Wake” are changed

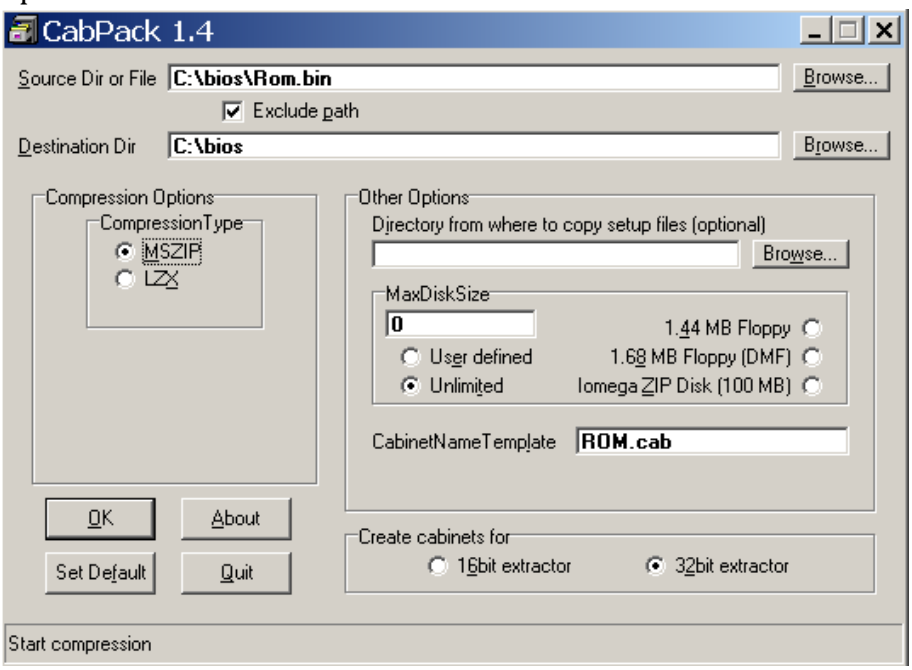
- The Two-File-Compare-Function from the Hex Editor will show, that all “Boot” String's in the “00020100.dec” are changed to “.‡ÏŽ”
- so the BIOS will boot, but some “Boot” Strings will be written in Wingdings...

Flashing in DOS:

Rename the 68BDD_Patched.BIN into 68BDD.BIN and flash it with the original rompaq.exe. Reportedly some rompaq.exe is only flashing the bios file, if the checksum is fixed. It checks the checksum BEFORE flashing and abort if the checksum is invalid. sp34752.exe ist a Biosfile from NX9420.

Flashing in Windows:

Patch the HPQFlash.exe with the ADDCC PatchButton. Rename the 68BDD_Patched.BIN into ROM.BIN. Delete the Rom.sig, its not needed anymore. Pack the ROM.BIN in ROM.CAB. In example with Cabpack.



The Source Code of the Decompresser Routine:

The Header Struct:

The HP Compaq Bios's have two known Header Struct's. (Dez. 2009) The Table below shows the differences.

Header ID 1	Body		
4 Bytes	3 x 4 Bytes, each 4 Byte in little endian		
ID and Control	unpacked_size	packed_size	target_address
Header ID 1 ("00 10 01 00" or "00 10 00 00")			
unknown_00	Headersize	more_blocks	unknown_03
Must be 0x00	Must be 0x10 (16 Dec)	0x01 True, 0x00 False	Must be 0x00

Headersize: 0x10 in Hex, 16 in Decimal. Means the byte count is 16, and the first Instruction Byte of the compressed data is the 17.

more_blocks: If this Byte is “True” (0x01) there is a next part with a header to decompress. If its “False” (0x00), this is the LAST part to decompress. After this part there is no decompressed data, maybe “normal” uncompressed bytes. Like the Bootblock or some other ROM Information.

Header ID 2	Body			Extension
4 Bytes	3 x 4 Bytes, each 4 Byte in little endian			4 Byes (Ascii)
ID and Control	unpacked_size	packed_size	target_address	Part Description
Header ID 2 ("01 00 14 01" or "01 00 14 00")				
unknown_00	unknown_01	Headersize	more_blocks	
Must be 0x01	Must be 0x00	Must be 0x14 (20 Dec)	0x01 True, 0x00 False	

Headersize: 0x14 in Hex, 20 in Decimal. Means the byte count is 20, and the first Instruction Byte of the compressed data is the 21. The 4 Bytes on 17,18,19 and 20 are Ascii strings. Like “POST”, “MESS”, or “HPLO”.

more_blocks: If this Byte is “True” (0x01) there is a next part with a header to decompress. If its “False” (0x00) at the first time, it means there two more parts to decompress left. Than it follow one part with “True” (0x01) and than it follow the really last header with “False” (0x00). After this part there is no decompressed data, maybe “normal” uncompressed bytes. Like the Bootblock or some other ROM Information.

The compressed data Struct:

The decompressed data struct is really simple. At the first time its starts with instruction byte, followed by minimum eight bytes data and than followed the next instruction byte. Generally every Instruction Byte hast EIGHT operations. In depending on the instruction byte there are two bytes for the Wordbook pointer and the Wordbook-Loop-Counter. During the decompression procedure it will produce a Wordbook. Every Byte to wrote in the target-file will be written at the end of the wordbook. Every byte that is read from any pointer in the Wordbook, will be written too, at the end of the wordbook. And so the wordbook rise continuously till unpacked size is reached. The tables below will show some example.

Instruction Byte	Compressed data	Instruction Byte	Compressed data	Repeat...

Example 1. The FF's are the instruction bytes. Between the data.
FF 12 01 2C 01 75 01 98 01 FF AA 01 D1 01 68 02 7F 02 FF C2 02 05 03 3D 03 74 03 FF

FF	12 01 2C 01 75 01 98 01	FF	AA 01 D1 01 68 02 7F 02	FF	C2 02 05 03 3D 03 74 03	FF	

Example 2. The FF's and FA are the instruction bytes. Between the data.
FF 20 64 69 73 6B 20 6F 72 FA 63 12 65 51 11 0D 0A 72 65 70 FF

FF	20 64 69 73 6B 20 6F 72	FA	63 12	65	51 11	0D 0A 72 65 70	FF

Instruction bytes:

How to interpret the instruction byte?! That's easy. FF in Hex is in Binary: 1111 1111. MSB … LSB. That means is it a “one”, copy the next Byte in the Wordbook. Is it a “zero” take the next two bytes REVERSE and build a Wordbook read Pointer and a Loop-Counter. Copy the Loop-Counter bytes at the end of the Wordbook. The Loop-Counter has an Offset from three. In both headers. That means, is there a “zero” and it starts a Wordbook operation, it copies minimum 3 bytes from the Wordbook at the end of the wordbook.

FF	(MSB) 1111 1111 (LSB)
FA	(MSB) 1111 1010 (LSB)

FF		FA		Reverse	Wordbook Pointer	Loop-Counter
(LSB)1	20	(LSB)0	6312 =>	1263	126	(3+3)=6
1	64	1	65			
1	69	0	5111=>	1151	115	(1+3)=4
1	73	1	0D			
1	6B	1	0A			
1	20	1	72			
1	6F	1	65			
(MSB)1	72	(MSB)1	70			

Wordbook (XX for unknown bytes from the three unknown Nibbles Wordbook Pointer)
20 64 69 73 6B 20 6F 72 XX XX XX XX XX XX 65 XX XX XX XX 0D 0A 72 65 70 _

Wordbook Pointer:

The building and interpreting of the Wordbook Pointer is the only really difference between the two header and the two decompression routines.

```
WordBook_byte_read_ptr = "&H" & Mid(WordBook_instruction_String, 1, 3)
WordBook_Loop_Counter = ("&H" & Mid(WordBook_instruction_String, 4, 1)) + 3
```

For Header ID 1 there is an fix offset to add with the three Nibble's. Add the value 0x1012 to the three Nibbles and subtract 0x1000 until the Wordbook Read Pointer is smaller than the Wordbook Write Pointer.

```
 ' for Header 1
If header_ID = 1 Then
   WordBook_byte_read_ptr += "&H" & "1012"
   Do Until WordBook_byte_read_ptr <= WordBook_Write_Counter
      WordBook_byte_read_ptr -= "&H" & "1000"
   Loop
End If
```

For Header ID 2 the Wordbook Read Pointer is equal the Wordbook Write Pointer subtract the three Nibbles.

```
 ' for Header 2
If header_ID = 2 Then
   WordBook_byte_read_ptr = WordBook_Write_Counter - WordBook_byte_read_ptr
End If
```

unpacked_size, packed_size, target_address:

These names are self-explanatory.

The whole decompresser Routine for both Headers:

The following Source Code is the same Loop that I' am use in the new ADDCC V3. Two things I have removed. Before, the declaration of the variables. After it, the write into a file procedures.

```
Do Until unpacked_size_Counter = 0
   If instruction_Bits = String.Empty Then
      instruction_Bits = Convert.ToString(complete_bios(source_byte_read_ptr) + 256, 2)
      instruction_Bits = Mid(instruction_Bits, 2)
      source_byte_read_ptr += 1
      packed_size_Counter -= 1
   End If
   For i = instruction_Bits.Length - 1 To 0 Step -1
      If instruction_Bits(i) = "0" Then
         ' copy from WordBook
         WordBook_instruction_String = String.Format("{0:X2}", complete_bios(source_byte_read_ptr + 1)) & _
                                       String.Format("{0:X2}", complete_bios(source_byte_read_ptr))
         WordBook_byte_read_ptr = "&H" & Mid(WordBook_instruction_String, 1, 3)
         WordBook_Loop_Counter = ("&H" & Mid(WordBook_instruction_String, 4, 1)) + 3
         ' for Header 1
         If header_ID = 1 Then
            WordBook_byte_read_ptr += "&H" & "1012"
            Do Until WordBook_byte_read_ptr <= WordBook_Write_Counter
               WordBook_byte_read_ptr -= "&H" & "1000"
            Loop
         End If
         ' for Header 2
         If header_ID = 2 Then
            WordBook_byte_read_ptr = WordBook_Write_Counter - WordBook_byte_read_ptr
         End If
         source_byte_read_ptr += 2 ' two bytes read, two times INC
      End If

      For k = 1 To WordBook_Loop_Counter
         If WordBook_Loop_Counter >= 3 Then
            ' Read one byte from WordBook-array, INC WordBook_byte_read_ptr
            destination_write_byte = WordBook(WordBook_byte_read_ptr)
            WordBook_byte_read_ptr += 1
         Else
            ' Read one byte from original complete_bios-array, INC WordBook_byte_read_ptr
            destination_write_byte = complete_bios(source_byte_read_ptr)
            source_byte_read_ptr += 1
         End If
         ' write the same byte byte in WordBook-array, INC WordBook_Write_ptr
         WordBook(WordBook_Write_Counter) = destination_write_byte
         WordBook_Write_Counter += 1
         unpacked_size_Counter -= 1
         If unpacked_size_Counter = 0 Or WordBook_Write_Counter = unpacked_size Then
            Exit Do
         End If
      Next k
      WordBook_Loop_Counter = 1
   Next i
   instruction_Bits = String.Empty
Loop
```