Vol. 2, No. 2 (2005)
http://www.CodeBreakers-Journal.com

Advanced Award BIOS v4.51PG Hacking

Mappatutu Salihun Darmawan*

E-Mail: mamanzip@yahoo.com

* Corresponding Author

Received: 07. Mar. 2005, Accepted: 10. Mar. 2005, Published: 20. Mar. 2005

Abstract

This tutorial is intended for people who already done some award bios modification before, and already knows the core component of award bios. In case you haven't done it or haven't know anything yet, you can read somewhere else. I have provided links to bios related website in the front page of this website and also I've made tutorial called Preliminary Bios Modification Guide and Mainboard Bios Components. As the title said, what I'm going to explain here only apply exactly to Award Bios version 4.51PG. However, the principle of this modification can be applied to other bioses as well, provided that you have enough knowledge in assembly language and using disassembler. As usual, I didn't held any responsibility in the damage that may occur if you apply the steps explained here, proceed at your own risk, you have been warned. I'd like to thank to Petr Soucek for his inspiring tutorial, Gigabyte ga586hx bios modification. I'm indebted to him for opening my eyes about what could possibly be done to the award bios file.

Keywords: *Reverse Code Engineering; BIOS*

1. Introduction

OK, let's get down to the business. I'm using Iwill VD133 (the slot 1 version) mainboard as my testbed. This mainboard uses award bios version 4.51PG dated 28 July 2000. In this article I'll explain how to do "bios code injection", i.e. injecting our patch into original.tmp (system bios) file. The area in original.tmp that we are going to inject with code is somewhere around memory test area code which is part of the POST (Power On Self Test). This area is right above the area which handles hdd initialization as described by Petr in his article. We will need these software tools:

1. Modbin. I'm using the version 4.50.80C. Your bios requirement may vary.
2. Cbrom. I'm using the version 2.08. Your bios requirement may vary. Cbrom and modbin are available for download at http://www.biosmods.com.
3. A binary file editor (hex editor). I'm using Hexworkshop version 3.0b.
4. Award Bios Editor version 1.0 or another tool/trick that enables you to extract and replace original.tmp from the motherboard bios with the modified version. Award Bios Editor version 1.0 are available for download at http://awdbedit.sourceforge.net.
5. A disassembler. I'm using ndisasmw.exe which comes with nasmw version 0.98. This is the windows version of the disassembler that comes with netwide assembler package, it's availabe for download at http://nasm.sourceforge.net.
6. An assembler and its linker. I'm using microsoft macro assembler (Masm) version 6.15 and its linker i.e. LINK version 5.60.33, this linker still support emiting 16 bit Intel binary. I'm using it since sometimes nasm emit 16 bit code that is not suitable for my computer, which sometimes end up with a lock up during POST. It's available for download at http://win32asm.cjb.net (in the download section).

Now we are armed with the right tools. What we need to do next are summarised below :

1. Extract original.tmp from the bios file somewhere so that we can edit it separately.
2. Disassemble original.tmp and look for code spot(s) where we can inject our code.
3. Build our own code then inject it into original.tmp and then replace some of original.tmp code so that our injected code will be executed upon booting, or completely replace "unneeded" procedure(s) in original.tmp
4. Replace original.tmp in the mainboard bios file with the modified one.

Do some finishing touches to ensure that everything will be ok.
Details on how these are accomplished will be explained in the next sections, so read on.

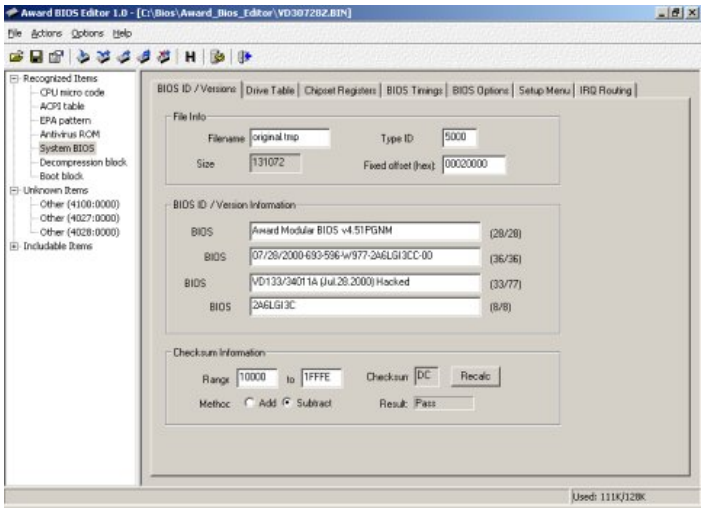
2. Hacking Steps in Detail

WARNING: The explanations in this section remain valid, however, there is stability issue with this approach in my testbed. Read more about this issue here. In conjuction with the issue, I strongly recommend you to read this section first before proceeding, since it contains fundamental tricks that are used throughout this article.

2.1. Extracting original.tmp

This step is very easy if you have the right tool. I accomplish this step by using award bios editor v1.0 program which is created by Mike Tedder a.k.a bpoint. Credit goes to him for providing us with this excellent tool :-). But be warned that it may still have some issue when used with award bios version 6.0PG.

I just do some clicking on the related menus to do this. Here how it's done: select the system bios item in the tree on the left then open up action menu then select Extract File menu and proceed at your will. Be sure to place builtins.dll in the same directory as awdbedit.exe. We need this since upon starting, awdbedit scans the directory in which it's located for plugin. builtins.dll is the main plugin which provided the capability to recognize core (and some extended) award bios components, such as original.tmp, the epa file, etc. Note that the awdbedit that I'm using already been compiled several times and undergone some cosmetic patches, since the version from sourceforge site uses "annoying size" fonts which displays horrible in my small CRT monitor. Below is the screenshot of award bios editor that I'm using:



2.2. Disassembling original.tmp

This step is tricky. It depends a lot in your skill and experience in using assembly language and disassembler, however I'm going to share some tricks that I gained during my journey through patching my bios.

Open up the extracted original.tmp from previous step in your hexeditor, then examine it. Here's a snapshot from Hexworkshop that I'm using:

```
Address  Hexadecimal values                                  Ascii representation
00000000 4273 4704 00E0 00F0 0060 0040 0000 0000 1130 2A30 BsG.....`.@.....0*0
00000014 4330 5C30 080B 0C02 4465 7465 6374 696E 6720 4944 C0\0....Detecting ID
00000028 4520 5072 696D 6172 7920 4D61 7374 6572 2020 2E2E E Primary Master  ..
0000003C 2E20 5B50 7265 7373 2006 4634 0820 746F 2073 6B69 . [Press .F4. to ski
00000050 705D 2000 0813 1320 2020 2020 2020 2020 2020 2020 p] .... 
00000064 2020 2020 2013 1200 536B 6970 2000 4E6F 6E65 2000      ...Skip .None .
00000078 1327 5072 696D 6172 7920 536C 6176 6520 2020 0C16 .'Primary Slave   ..
0000008C 2000 1327 5365 636F 6E64 6172 7920 4D61 7374 6572  ..'Secondary Master
000000A0 0C16 2000 1327 5365 636F 6E64 6172 7920 536C 6176 .. ..'Secondary Slav
000000B4 6520 0C16 2000 080B 466F 756E 6420 4344 524F 4D20 e .. ...Found CDROM
000000C8 3A20 0008 0B46 6F75 6E64 2041 5441 5049 2044 6576 : ...Found ATAPI Dev
000000DC 6963 6520 3A20 001E 060F A866 B8FF FFFF FF66 8986 ice : .....f.....f..
000000F0 E901 6689 8604 0266 33C0 6689 86D7 01C6 86D6 0130 ..f....f3.f........0
00000104 8886 F301 6689 8600 0288 86FF 0188 865A 0168 00F0 ....f..........Z.h..
00000118 0FA9 B840 008E D8B8 0100 8EC0 8D06 59EC 26A3 F000 ...@..........Y.&...
0000012C 8026 B600 FBBE 942F E847 6B0A C074 0580 0EB6 0004 .&...../.Gk..t......
00000140 C606 7500 0080 A6E1 01F0 C606 B500 008A 4612 C0E8 ..u.............F...
00000154 040A C074 3C3C 0F75 3480 7E19 2F75 2E80 BE82 0006 ...t<<.u4.~./u......
00000168 7305 C686 8200 06BB C276 66C1 EE10 808E E101 01E8 s........vf.........
0000017C 4D08 7207 F686 E401 0174 0A80 4E12 F0C6 4619 2FEB M.r......t..N...F./.
00000190 04FE 0675 008A 4612 240F 7503 EB43 903C 0F75 2C80 ...u..F.$.u..C.<.u,.
000001A4 7E1A 2F75 26BB 8A77 8D36 7800 66C1 E610 808E E101 ~./u&..w.6x.f.......
000001B8 02E8 0F08 7207 F686 E401 0274 0A80 4E12 0FC6 461A ....r......t..N...F.
000001CC 2FEB 128A 0E75 00D0 E1B0 01D2 E008 06B5 00FE 0675 /....u.............u
000001E0 0080 7E67 0074 3A80 7E67 2F75 22BB 5278 8D36 8E00 ..~g.t:.~g/u".Rx.6..
000001F4 66C1 E610 808E E101 04E8 CB07 7207 F686 E401 0474 f...........r......t
00000208 06C6 4667 2FEB 128A 0E75 00D0 E1B0 02D2 E008 06B5 ..Fg/....u..........
0000021C 00FE 0675 0080 7E70 0074 3A80 7E70 2F75 22BB 1A79 ...u..~p.t:.~p/u"..y
00000230 8D36 A400 66C1 E610 808E E101 08E8 8B07 7207 F686 .6..f...........r...
00000244 E401 0874 06C6 4670 2FEB 128A 0E75 00D0 E1B0 03D2 ...t..Fp/....u......
00000258 E008 06B5 00FE 0675 0060 33D2 8AC2 2403 0FB6 F081 .......u.`3...$.....
0000026C C6E9 01F6 C202 750B F686 D601 1074 1CB4 10EB 09F6 ......u......t......
00000280 86D6 0120 7411 B420 803A FF74 0A80 3A02 7605 C602 ... t.. .:.t..:.v...
00000294 020A F480 FA03 7404 FEC2 EBC4 8A86 D601 24CF 0AC6 ......t.........$...
000002A8 8886 D601 61E8 FF0D FFB6 E901 FFB6 EB01 FFB6 D701 ....a...............
000002BC FFB6 D901 8B86 0802 8D36 15F0 E863 6DE8 4511 B800 .........6...cm.E...
```

To successfully disassemble this file we need some common sense and intelligent guesses. I used this guide :

- Use your logical sense to examine the disassembly result. Does it makes sense at all, if not then you are in the wrong direction, you don't skip some parts that suppose to be a data section, not code section, if this is the case then the disassembler will emit a strange looking assembly code. You can use the next guide(s) to help you against this.
- The majority of ascii string are terminated with binary zero, i.e. displayed as 00 in the hexadecimal value above.
- Some ascii string are terminated by '$' character or 24 hexadecimal.
- Some ascii string are simply "discarded" by "unconditionally jumping" through it.
- Due to 1,2, and 3 we have to instruct the disassembler to skip through this ascii string, so that we get the real instruction.
- Some instructions possibly executed in 32 bit mode, such as the memory detection routine. This is logical since AFAIK, to access memory after 1 MByte we need to activate gate A20 and to access several hundred megabytes (as in our daily case) we need 32 bit mode. I even get into speculation that the current award bios (version 4.51PG, version 6.0PG and version 6.0) might already setup data structures needed to operate x86 in 32 bit protected mode such as GDT and temporarily switch into protected mode to accomplish this, and then switch back to 16 bit real mode for compatibility reason. This should be taken into account, hence we have to be very careful and just place our 16 bit code(s) wherever we are sure that the processor operating mode still in 16 bit real mode. After spending some more time disassembling the

POST routine, I found that my suspicion prove to be correct, the memory detection routine have switched the machine into protected mode momentarily!

- The last 64 Kbytes of original.tmp have a checksum that is calculated by substracting every bytes in advance from 10000h until 1FFFFh and it is placed in the last byte.
- The first 64 KBytes of original.tmp seems to contain POST (Power On Self Test) code. Due to this reason, this time we'll examine only the first 64 Kbytes and inject our code there. In my experience, the end of this 64 Kbytes contains lots of padding FFh bytes and 00h bytes (around 4 Kbytes, i.e. from around F000h to FFFFh), we can place our code there. I have one more reason to place the code there, based on my experience too, it seems that this end of the first 64 Kbytes area is free from checksum related issue. Generally the first 64 Kbytes are also free from this checksum issue, i.e. the first 64 KBytes is not subject to checksum calculation as in the case of the last 64 KBytes mentioned above.

Based on the guide above, we disassemble original.tmp using ndisasmw.exe. Since I was bored enough to type such a massive command in the command line, we are going to use a windows batch file as follows :

1. Examine its content in your hexeditor then split them into parts and grow the parts (increase the part size) as needed as you trace through the part's disassembly result. Most of the time this is needed due to branch instruction such as variation of jump and call instructions. In the beginning, original.tmp is big enough to make you discouraged doing a disassembly on it in one shot. Note that I only disassemble the first 64 Kbytes to achieve my modification.
2. Below is an example of the content of a batch file that I'm using :

```
@echo off
ndisasmw.exe -b16 -k 0,0xE3 -k 0x106A,0x3F -k 0x2E76,0xA -k 0x2F2F,0x16
-k 0x328D,0x2B -k 0x43E0,0x360
-k 0x5D5D,0x52 -k 0x7063,0x33 sys_part.bin >> sys_part.txt
```

I dump the result into a text file to ease analyzing it.
Note :

- sys_part.bin is a file that contain part of the original.tmp, it contains much less code.
- the -k xxx,yyy switch is used to skip bytes that I suspect to be data section, not code section.
- the -b16 switch is used to tell the disassembler to treat the binary as 16 bit codes.

3. The snapshot of the disassembly result as follows:

```
Address   Binary Code       Mnemonic
00000000  skipping 0xE3 bytes
000000E3  1E                push ds
000000E4  06                push es
000000E5  0FA8              push gs
000000E7  66B8FFFFFFFF      mov eax,0xffffffff
000000ED  668986E901        mov [bp+0x1e9],eax
000000F2  6689860402        mov [bp+0x204],eax
000000F7  6633C0            xor eax,eax
000000FA  668986D701        mov [bp+0x1d7],eax
000000FF  C686D60130        mov byte [bp+0x1d6],0x30
00000104  8886F301          mov [bp+0x1f3],al
00000108  6689860002        mov [bp+0x200],eax
0000010D  8886FF01          mov [bp+0x1ff],al
00000111  88865A01          mov [bp+0x15a],al
00000115  6800F0            push word 0xf000
00000118  0FA9              pop gs
........
00000D73  E85A03            call 0x10d0
00000D76  268B05            mov ax,[es:di]
00000D79  3DFFFF            cmp ax,0xffff
00000D7C  0F84C702          jz near 0x1047
00000D80  8904              mov [si],ax
00000D82  268B4502          mov ax,[es:di+0x2]
00000D86  268A7506          mov dh,[es:di+0x6]
00000D8A  268A550C          mov dl,[es:di+0xc]
00000D8E  26F6450180        test byte [es:di+0x1],0x80
00000D93  7549              jnz 0xdde
00000D95  26F6456302        test byte [es:di+0x63],0x2
00000D9A  7442              jz 0xdde
00000D9C  2666837D7800      cmp dword [es:di+0x78],byte +0x0
00000DA2  743A              jz 0xdde
00000DA4  2666837D78FF      cmp dword [es:di+0x78],byte -0x1
00000DAA  7432              jz 0xdde
00000DAC  26817D7A0050      cmp word [es:di+0x7a],0x5000
00000DB2  772A              ja 0xdde
00000DB4  26817D7AF003      cmp word [es:di+0x7a],0x3f0
00000DBA  7202              jc 0xdbe
00000DBC  B2FF              mov dl,0xff
00000DBE  52                push dx
00000DBF  8AC6              mov al,dh
00000DC1  F6E2              mul dl
00000DC3  660FB7C8          movzx ecx,ax
00000DC7  26668B4578        mov eax,[es:di+0x78]
00000DCC  6633D2            xor edx,edx
00000DCF  66F7F1            div ecx
00000DD2  663D00000100      cmp eax,0x10000
00000DD8  7203              jc 0xddd
00000DDA  B8FFFF            mov ax,0xffff
00000DDD  5A                pop dx
00000DDE  894402            mov [si+0x2],ax
00000DE1  887404            mov [si+0x4],dh
00000DE4  885410            mov [si+0x10],dl
00000DE7  268A455E          mov al,[es:di+0x5e]
00000DEB  BA4000            mov dx,0x40
00000DEE  8EDA              mov ds,dx
00000DF0  5A                pop dx
........
```

In the upper part, you can see that I skipped some bytes that clearly shown as "readable ascii string" in my hexeditor. This string ends up with a terminating zero (00h), so I'm pretty sure of what I'm doing.

The lower part of the snapshot above shows similar code to what Petr Soucek describe in his bios modification article, i.e. hdd identification routine. This puts me into better position to apply my "code injection".

2.3. Building and Injecting Code into original.tmp

Here's our plan :

1. We are going to search for 3 bytes instruction in the first 64 KByte area to be replaced by a near call instruction into our injected code
2. The injected code will be placed at EFF0h address. We are replacing FFh bytes there as needed.
3. The majority of our code will be generated by a 16 bit capable assembler and linker, and then we will need to "hand tune" it to suit our purpose, since sometimes our assembler and linker simply emit code that isn't suitable to our needs. We are going to name our patch code file sys_patch.asm

Now we have the plan, but before proceeding further we have to take care some cautions. Note that sys_patch.asm are assembled using masm615 with LINK version 5.60.339. We have to take some protective measure to protect ourself against bad code as follows:

- Masm615 still emit some incompatible code, such as:

```
cmp ax,0ffffh ; the code in the original.tmp that I replaced.
```

  Masm615 doesn't emit 3DFFFFh binary for this code (which is the original code in the original.tmp file) . Hence, I have to replace that 3 bytes which Masm615 emits with the suitable code (3DFFFFh) using hexeditor. Things like this have to be taken into account when patching original.tmp.

- Beware of registers and instructions you are using. I strongly recommend to save flags and every register your code uses before executing your injected code and restore them back prior to returning to the real original.tmp code. If you fail to do so, there's a big chance you'll screw up your system. You've been warned.
- We have to avoid intersegment call as entry point into our injected code, since this tend to put us in trouble. That's why we decided to put our injected code in the first 64 KByte area, so that we stick to intrasegment near call as our entry point.
- If you replace the original code in original.tmp with 3 bytes of call instruction to your injected code, be sure to calculate the relative distance between the call and your injected code VERY CAREFULLY, near call instruction distance are measured as follows:

  relative_call_distance = **addr_of_1st_injected_code** - **addr_after_call_instruction**

For example, the near call instruction located at D79h and your injected code begins at EFF0h then the **relative_call_distance** will be :

relative_call_distance = **EFF0**h - **D7C**h
relative_call_distance = **E274**h

, since the call instruction itself uses 3 bytes. Hence you have to replace the original 3 bytes of code with a near call instruction i.e. **E8 74 E2**, the distance bytes are reversed since intel x86 uses little endian scheme.

- Don't forget to include the replaced code (the 3 bytes of code which you replace with a call instruction) in your injected code prior to returning. Generally your injected code will looks like:

```
... (main injected code here)
....
the code that you replaced here
ret
```

- Don't forget that we are targeting 16 bits real mode environment. I'm using the following command to assemble the file:

```
ml /AT sys_patch.asm /Fe sys_patch.bin /link /TINY
```

Note: You don't need to pay attention to the linker warning, it has no effect to us since we are not building a real *.com file, we are targeting 16 bit pure binary code. The command above will end up buiding sys_patch.bin as the resulting binary file that we are going to use.

Then we proceed to search for the code to replace. Petr Soucek in his article points out about the hdd initialization routine, we are going to place our code just above that code. The disassembly result of my bios around that code as follows:

```
Address   Binary Code       Mnemonic
......
00000D76  268B05            mov ax,[es:di]
00000D79  3DFFFF            cmp ax,0xffff -- this is where we're going to place the
call to our code
00000D7C  0F84C702          jz near 0x1047
00000D80  8904              mov [si],ax
00000D82  268B4502          mov ax,[es:di+0x2]
00000D86  268A7506          mov dh,[es:di+0x6]
00000D8A  268A550C          mov dl,[es:di+0xc]
00000D8E  26F6450180        test byte [es:di+0x1],0x80
00000D93  7549              jnz 0xdde
00000D95  26F6456302        test byte [es:di+0x63],0x2
00000D9A  7442              jz 0xdde
00000D9C  2666837D7800      cmp dword [es:di+0x78],byte +0x0
00000DA2  743A              jz 0xdde
00000DA4  2666837D78FF      cmp dword [es:di+0x78],byte -0x1
00000DAA  7432              jz 0xdde
00000DAC  26817D7A0050      cmp word [es:di+0x7a],0x5000
00000DB2  772A              ja 0xdde
00000DB4  26817D7AF003      cmp word [es:di+0x7a],0x3f0
00000DBA  7202              jc 0xdbe
00000DBC  B2FF              mov dl,0xff
00000DBE  52                push dx
00000DBF  8AC6              mov al,dh
00000DC1  F6E2              mul dl
```

```
00000DC3  660FB7C8          movzx ecx,ax
00000DC7  26668B4578        mov eax,[es:di+0x78]
00000DCC  6633D2            xor edx,edx
00000DCF  66F7F1            div ecx
00000DD2  663D00000100      cmp eax,0x10000
00000DD8  7203              jc 0xddd
00000DDA  B8FFFF            mov ax,0xffff
00000DDD  5A                pop dx
........
```

Then proceed to look where we can place our injected code. In my case, I found the suitable area beginning at **EFF0**h. The following is the hex dump of my original.tmp around that area :

```
Address  Hexadecimal values                                Ascii representation
........
0000EFC4 68CF EF68 8854 EA88 6100 E0C3 C300 0000 0000 0000 h..h.T..a...........
0000EFD8 0000 0000 0000 0000 C300 0000 0000 0000 0000 0000 ....................
0000EFEC 0000 0000 FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF ....................
0000F000 FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF ....................
0000F014 FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF ....................
0000F028 FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF ....................
0000F03C FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF ....................
0000F050 FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF ....................
0000F064 FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF ....................
0000F078 FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF ....................
0000F08C FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF ....................
0000F0A0 FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF ....................
........
```

As you can see, these are just padding bytes. The **FF**h bytes continues until **FFFF**h address. So, I'll just place my code beginning at **EFF0**h. Now we proceed to build our injected code. Here's the listing of my code:

```
; ------------------- sys_patch.asm ---------------------------------------------
.486p

CSEG    SEGMENT PARA PUBLIC USE16 'CODE'
        ASSUME CS:CSEG
        ORG 0

; equates, have been tested and works fine

        in_port    equ 0cf8h
        out_port   equ 0cfch

        ioq_mask   equ 00000080h
        ioq_reg    equ 80000050h
        bank_mask  equ 20000844h
        bank_reg   equ 80000068h
        tlb_mask   equ 00000008h
        tlb_reg    equ 8000006ch
        dram_mask  equ 00020202h
        dram_reg   equ 80000064h

INIT    PROC    NEAR

; save all register that will be affected by our code
        push eax
        push ebx
        push edx
        pushfd
```

```
; patch the ioq reg
        mov eax,ioq_reg
        mov ebx,ioq_mask
        call PATCH_PCI

; patch the DRAM controller, i.e. the interleaving part
        mov eax,dram_reg
        mov ebx,dram_mask
        call PATCH_PCI

; patch bank active page ctl reg
        mov eax,bank_reg
        mov ebx,bank_mask
        call PATCH_PCI

; Activate Fast TLB lookup
        mov eax,tlb_reg
        mov ebx,tlb_mask
        call PATCH_PCI

; restore register contents and call the replaced instruction prior to return
        popfd
        pop edx
        pop ebx
        pop eax
        cmp ax,0ffffh ; This is the instruction that we replace in the POST code
        ret    ; return and proceed to the next initialization code

INIT    ENDP

PATCH_PCI PROC NEAR
        ; The register address is passed via EAX register
        ; The mask value is passed via EBX register
        mov dx,0cf8h ; fetch the input port addr of PCI cfg space
        out dx,eax
        mov dx,0cfch
        in  eax,dx
        or  eax,ebx  ; mask the regs value (activate certain bits)
        out dx,eax
        ret          ; return to initialization code above
PATCH_PCI       ENDP

CSEG    ENDS
                END

; -------------------------- END OF sys_patch.asm --------------------------------
```

There are some things that we have to note here. **First**, every branching instruction is a near branching instruction which uses relative address. **Second**, we are telling the assembler to emit 16 bits code. **Third**,we preserve all the register status during our injected code execution, I've been bitten by forgetting to preserve dx register during my code injection experiment, I learn a lot from it :-). The resulting binary from the code above as follows:

```
Address  Hexadecimal values                                Ascii representation
00000000 6650 6653 6652 669C 66B8 5000 0080 66BB 8000 0000 fPfSfRf.f.P...f.....
00000014 E839 0066 B864 0000 8066 BB02 0202 00E8 2A00 66B8 .9.f.d...f......*.f.
00000028 6800 0080 66BB 4408 0020 E81B 0066 B86C 0000 8066 h...f.D.. ...f.l...f
0000003C BB08 0000 00E8 0C00 669D 665A 665B 6658 83F8 FFC3 ........f.fZf[fX....
00000050 BAF8 0C66 EFBA FC0C 66ED 660B C366 EFC3           ...f....f.f..f..
```

The disassembly of this binary using " ndisasmw.exe -b16 sys_patch.com > sys_patch.txt " as follows:

```
Address   Binary Code       Mnemonic
00000000  6650              push eax
00000002  6653              push ebx
00000004  6652              push edx
00000006  669C              pushfd
00000008  66B850000080      mov eax,0x80000050
0000000E  66BB80000000      mov ebx,0x80
00000014  E83900            call 0x50
00000017  66B864000080      mov eax,0x80000064
0000001D  66BB02020200      mov ebx,0x20202
00000023  E82A00            call 0x50
00000026  66B868000080      mov eax,0x80000068
0000002C  66BB44080020      mov ebx,0x20000844
00000032  E81B00            call 0x50
00000035  66B86C000080      mov eax,0x8000006c
0000003B  66BB08000000      mov ebx,0x8
00000041  E80C00            call 0x50
00000044  669D              popfd
00000046  665A              pop edx
00000048  665B              pop ebx
0000004A  6658              pop eax
0000004C  83F8FF            cmp ax,byte -0x1  --- This isn't what expected :-(
0000004F  C3                ret
00000050  BAF80C            mov dx,0xcf8
00000053  66EF              out dx,eax
00000055  BAFC0C            mov dx,0xcfc
00000058  66ED              in eax,dx
0000005A  660BC3            or eax,ebx
0000005D  66EF              out dx,eax
0000005F  C3                ret
```

Looking at the resulting binary and its disassembly result, we know that we need to replace the code that is not supposed to be like that by using hex editor. But we have to ensure that the amount of bytes we are replacing is the same with the assembled version, if not, then all the branching instruction will be screwed up. Fortunately we have 3 bytes to be replaced with also 3 bytes instruction, so this shouldn't be a problem. Now, we replace the **83 F8 FF** bytes with **3D FF FF** bytes which is the "correct instruction" based on what our previous bios code does. The resulting hex dump and disassembly as follows:

Hexdump:

```
Address  Hexadecimal values                              Ascii representation
00000000 6650 6653 6652 669C 66B8 5000 0080 66BB 8000 0000 fPfSfRf.f.P...f.....
00000014 E839 0066 B864 0000 8066 BB02 0202 00E8 2A00 66B8 .9.f.d...f......*.f.
00000028 6800 0080 66BB 4408 0020 E81B 0066 B86C 0000 8066 h...f.D.. ...f.l...f
0000003C BB08 0000 00E8 0C00 669D 665A 665B 6658 3DFF FFC3 ........f.fZf[fX=...
00000050 BAF8 0C66 EFBA FC0C 66ED 660B C366 EFC3           ...f....f.f..f..
```

Disassembly:

```
Address   Binary Code       Mnemonic
00000000  6650              push eax
00000002  6653              push ebx
00000004  6652              push edx
00000006  669C              pushfd
00000008  66B850000080      mov eax,0x80000050
0000000E  66BB80000000      mov ebx,0x80
00000014  E83900            call 0x50
00000017  66B864000080      mov eax,0x80000064
0000001D  66BB02020200      mov ebx,0x20202
00000023  E82A00            call 0x50
00000026  66B868000080      mov eax,0x80000068
0000002C  66BB44080020      mov ebx,0x20000844
00000032  E81B00            call 0x50
00000035  66B86C000080      mov eax,0x8000006c
0000003B  66BB08000000      mov ebx,0x8
00000041  E80C00            call 0x50
00000044  669D              popfd
00000046  665A              pop edx
00000048  665B              pop ebx
0000004A  6658              pop eax
0000004C  3DFFFF            cmp ax,0xffff    --- This is what we expected :-)
0000004F  C3                ret
00000050  BAF80C            mov dx,0xcf8
00000053  66EF              out dx,eax
00000055  BAFC0C            mov dx,0xcfc
00000058  66ED              in eax,dx
0000005A  660BC3            or eax,ebx
0000005D  66EF              out dx,eax
0000005F  C3                ret
```

Then we proceed to inject our code into the area that we mentioned above. It's very easy, just paste our binary into that area using a hex editor. But before that, be sure to remove the same amount of **FF**h bytes in that area as the size of our binary, in my case, this is **60**h bytes. Here's the resulting hex dump:

```
Address  Hexadecimal values                                Ascii representation
........
0000EFB0 00B4 20B0 10E8 00DF B002 E81A E466 5B07 1FF8 C30E .. .........f[.....
0000EFC4 68CF EF68 8854 EA88 6100 E0C3 C300 0000 0000 0000 h..h.T..a...........
0000EFD8 0000 0000 0000 0000 C300 0000 0000 0000 0000 0000 ....................
0000EFEC 0000 0000 6650 6653 6652 669C 66B8 5000 0080 66BB ....fPfSfRf.f.P...f.
0000F000 8000 0000 E839 0066 B864 0000 8066 BB02 0202 00E8 .....9.f.d...f......
0000F014 2A00 66B8 6800 0080 66BB 4408 0020 E81B 0066 B86C *.f.h...f.D.. ...f.l
0000F028 0000 8066 BB08 0000 00E8 0C00 669D 665A 665B 6658 ...f........f.fZf[fX
0000F03C 3DFF FFC3 BAF8 0C66 EFBA FC0C 66ED 660B C366 EFC3 =......f....f.f..f..
0000F050 FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF ....................
0000F064 FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF ....................
........
```

Now, as our last step in this section is replacing the original code with a call into our injected code. First, we have to calculate the intrasegment distance as follows :

distance = **EFF0**h - **D7C**h
distance = **E274**h

Second, we make the binary for the call instruction. That would be: E8 74 E2 , the distance value is reversed, since intel x86 uses little endian scheme. Note that call instruction distance is calculated relative to the next instruction after the call instruction itself, that's why we are using **D7C**h above, **not D79**h, keep this in mind.

Third, replace the code in address **D79**h with our code, i.e. E8 74 E2 using our hex editor. The resulting disassembly after replacing that code will be:

```
Address   Binary Code       Mnemonic
........
00000D73  E85A03            call 0x10d0
00000D76  268B05            mov ax,[es:di]
00000D79  E874E2            call 0xeff0
00000D7C  0F84C702          jz near 0x1047
00000D80  8904              mov [si],ax
00000D82  268B4502          mov ax,[es:di+0x2]
00000D86  268A7506          mov dh,[es:di+0x6]
00000D8A  268A550C          mov dl,[es:di+0xc]
00000D8E  26F6450180        test byte [es:di+0x1],0x80
00000D93  7549              jnz 0xdde
........
```

Well, now we have done what we want to original.tmp, this completed our steps in this section.

2.4. Replacing the Old original.tmp

For me, this is just a trivial task. I just open up award bios editor and then select the system bios item in the left, then choose replace file from menu to replace the original.tmp with my modified version of original.tmp. Eventhough this step seems to be very easy, there is a catch however. Read the next section for explanation in this issue.

2.5. Finishing Touches

Award bios editor seems still have some issue with award bios v4.51PG checksum. In my case, if I use different name than original.tmp as the name for the modified original.tmp file, award bios editor didn't complain,so does modbin and cbrom. But in fact, there are still wrong checksum in the resulting bios file (after its orignal.tmp replaced with the modified one). This is what I experienced : after successfully flashing the bios into my mainboard, the board boot, but it complains about wrong bios checksum and only booted into the award boot block bios. This issue can be fixed by opening and modifying our modded bios in modbin, and after that saving our changes into the corresponding bios file. I accomplish this by changing the bios name string and then saving my changes into the bios file using modbin v4.50.80C. After flashing the bios into my motherboard everything works as expected. Beside what I've said, I strongly recommend you to try running cbrom and modbin in your resulting modified bios file to ensure that it is a valid bios file before flashing it into your mainboard.

3. A More Radical original.tmp Hacking

WARNING : The explanations in this section remain valid, however there is stability issue with this approach in my testbed. Read more about it here.

The modification that we are going to do in this part is similar to what described above, in *Hacking Steps in Detail*. The difference is in the part of the original bios code (original.tmp) that we are going to replace with a call into our code. This hack also will incorporate some complexities of the call instruction itself. I will only highlights the differences here, I'm not going to repeat what I've explained above.

We proceed through the steps mentioned above until we arrive at *Disassembling original.tmp*. In this step, the code spot where we are going to place our call into the injected routine is around **2A0**h. I'm not so sure what this area of code doing, but one thing for sure, this is still part of the POST code area. Below is the disassembly of this part in my bios :

```
Address   Binary Code       Mnemonic
........
000002AC  61                popa
000002AD  E8FF0D            call 0x10af
000002B0  FFB6E901          push word [bp+0x1e9]
000002B4  FFB6EB01          push word [bp+0x1eb]
000002B8  FFB6D701          push word [bp+0x1d7]
000002BC  FFB6D901          push word [bp+0x1d9]
000002C0  8B860802          mov ax,[bp+0x208]
000002C4  8D3615F0          lea si,[0xf015]
000002C8  E8636D            call 0x702e
000002CB  E84511            call 0x1413
000002CE  B800ED            mov ax,0xed00
000002D1  26A3C801          mov [es:0x1c8],ax
000002D5  26C706CA0100F0    mov word [es:0x1ca],0xf000
000002DC  B824ED            mov ax,0xed24
000002DF  26A3CC01          mov [es:0x1cc],ax
000002E3  26C706CE0100F0    mov word [es:0x1ce],0xf000
000002EA  E8FC0F            call 0x12e9   -- insert a call to our code here
000002ED  C606C60000        mov byte [0xc6],0x0
000002F2  F686E4010F        test byte [bp+0x1e4],0xf
000002F7  7441              jz 0x33a
000002F9  8A86E401          mov al,[bp+0x1e4]
000002FD  32E4              xor ah,ah
000002FF  8A0EEA00          mov cl,[0xea]
00000303  8AE9              mov ch,cl
00000305  80E103            and cl,0x3
........
```

The majority of the code that we are going to inject is still the same. Only one different, we are substituting the **`cmp ax,0ffffh`** code with a call instruction like what is highlighted in the disassembly result above, but we need some trick to achieve that. The listing as follows :

```
; ------------------- sys_patch.asm ------------------------------------------------
.486p

CSEG    SEGMENT PARA PUBLIC USE16 'CODE'
        ASSUME CS:CSEG
        ORG 0

; equates, have been tested and works fine

        in_port   equ 0cf8h
        out_port  equ 0cfch

        ioq_mask  equ 00000080h
        ioq_reg   equ 80000050h
        bank_mask equ 20000844h
        bank_reg  equ 80000068h
        tlb_mask  equ 00000008h
        tlb_reg   equ 8000006ch
        dram_mask equ 00020202h
        dram_reg  equ 80000064h

INIT    PROC    NEAR

; save all register that will be affected by our code
        push eax
        push ebx
        push edx
        pushfd

; patch the ioq reg
        mov eax,ioq_reg
        mov ebx,ioq_mask
        call PATCH_PCI

; patch the DRAM controller, i.e. the interleaving part
        mov eax,dram_reg
        mov ebx,dram_mask
        call PATCH_PCI

; patch bank active page ctl reg
        mov eax,bank_reg
        mov ebx,bank_mask
        call PATCH_PCI

; Activate Fast TLB lookup
        mov eax,tlb_reg
        mov ebx,tlb_mask
        call PATCH_PCI

; restore register contents and call the replaced instruction prior to return
        popfd
        pop edx
        pop ebx
        pop eax
        call PATCH_PCI ; !!!WARNING!!!Replace the target of this call with the
default from the real bios
        ret            ; return to instruction after the code that we replace in
system bios

INIT    ENDP

PATCH_PCI PROC NEAR
        ; The register address is passed via EAX register
        ; The mask value is passed via EBX register
        mov     dx,0cf8h  ; fetch the input port addr of PCI cfg space
        out     dx,eax
```

```
        mov     dx,0cfch
        in      eax,dx
        or      eax,ebx   ; mask the regs value (activate certain bits)
        out     dx,eax
        ret               ; return to initialization code above
PATCH_PCI       ENDP

CSEG    ENDS
                END

; -------------------------- END OF sys_patch.asm --------------------------------
```

as you can see from the code highlighted above, we are placing a "dummy call" there so that the assembler can assemble our code. If we don't do so, it will complain and say that we are calling non-existent routine. We are going to replace the target of that call instruction manually by using hex editor. The resulting binary and its disassembly as follows :

Hex dump :

```
Address  Hexadecimal values                                Ascii representation
00000000 6650 6653 6652 669C 66B8 5000 0080 66BB 8000 0000 fPfSfRf.f.P...f.....
00000014 E839 0066 B864 0000 8066 BB02 0202 00E8 2A00 66B8 .9.f.d...f......*.f.
00000028 6800 0080 66BB 4408 0020 E81B 0066 B86C 0000 8066 h...f.D.. ...f.l...f
0000003C BB08 0000 00E8 0C00 669D 665A 665B 6658 E801 00C3 ........f.fZf[fX....
00000050 BAF8 0C66 EFBA FC0C 66ED 660B C366 EFC3           ...f....f.f..f..
```

Disassembly result :

```
Address   Binary Code       Mnemonic
00000000  6650              push eax
00000002  6653              push ebx
00000004  6652              push edx
00000006  669C              pushfd
00000008  66B850000080      mov eax,0x80000050
0000000E  66BB80000000      mov ebx,0x80
00000014  E83900            call 0x50
00000017  66B864000080      mov eax,0x80000064
0000001D  66BB02020200      mov ebx,0x20202
00000023  E82A00            call 0x50
00000026  66B868000080      mov eax,0x80000068
0000002C  66BB44080020      mov ebx,0x20000844
00000032  E81B00            call 0x50
00000035  66B86C000080      mov eax,0x8000006c
0000003B  66BB08000000      mov ebx,0x8
00000041  E80C00            call 0x50
00000044  669D              popfd
00000046  665A              pop edx
00000048  665B              pop ebx
0000004A  6658              pop eax
0000004C  E80100            call 0x50 --- this is where we need to patch manually
0000004F  C3                ret
00000050  BAF80C            mov dx,0xcf8
00000053  66EF              out dx,eax
00000055  BAFC0C            mov dx,0xcfc
00000058  66ED              in eax,dx
0000005A  660BC3            or eax,ebx
0000005D  66EF              out dx,eax
0000005F  C3                ret
```

Now, we only need to insert our code into **EFF0**h and then do some "address fixups". This is tricky step . I accomplish it as follows :

1. First, insert our code into original.tmp. We do this to ease the process of calculating the relative addresses that we are going to do next. The hex dump is below :

```
Address  Hexadecimal values                                  Ascii representation
........
0000EFD8 0000 0000 0000 0000 C300 0000 0000 0000 0000 0000 ....................
0000EFEC 0000 0000 6650 6653 6652 669C 66B8 5000 0080 66BB ....fPfSfRf.f.P...f.
0000F000 8000 0000 E839 0066 B864 0000 8066 BB02 0202 00E8 .....9.f.d...f......
0000F014 2A00 66B8 6800 0080 66BB 4408 0020 E81B 0066 B86C *.f.h...f.D.. ...f.l
0000F028 0000 8066 BB08 0000 00E8 0C00 669D 665A 665B 6658 ...f........f.fZf[fX
0000F03C E801 00C3 BAF8 0C66 EFBA FC0C 66ED 660B C366 EFC3 .......f....f.f..f..
0000F050 FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF ....................
........
```

2. Second, calculate all the "fixup" addresses for the call instructions by using the formula in the *Hacking Steps in Detail* above. We got the following results :
   - Relative address for entry point into our code at **2EA**h :
     relative_addr = **addr_of_our_code - addr_after_the_call_at_2EA**
     relative_addr = **EFF0**h - **2ED**h
     relative_addr = **ED03**h
   - Relative address for the call instruction that we replace at **2EA**h. This call is placed inside our injected code as highlighted in hex dump above. The "address fixup" calculation as follows :
     relative_addr = **addr_of_old_call_target - addr_after_call_at_our_code**
     relative_addr = **12E9**h - **FO3F**h
     relative_addr = **- DD56**h
     relative_addr = **10000**h - **DD56**h
     relative_addr = **22AA**h
     The call instruction in intel x86 is a signed call instruction, that's why we use the above formula to calculate the "backward" call instruction in our code. This is working as expected , I've tested it and it's good :-).
3. The last step is to encode the instruction we need and insert them into the appropriate places. The first call instruction encoded into **E8 03 ED** h, the second call instruction encoded into **E8 AA 22** h. Place the first call beginning at **2EA**h and the second at **F03C**h. The resulting hex dump is below :

```
Address  Hexadecimal values                                  Ascii representation
........
000002D0 ED26 A3C8 0126 C706 CA01 00F0 B824 ED26 A3CC 0126 .&...&.......$.&...&
000002E4 C706 CE01 00F0 E803 EDC6 06C6 0000 F686 E401 0F74 ...................t
000002F8 418A 86E4 0132 E48A 0EEA 008A E980 E103 F6C5 0474 A....2.............t
0000030C 133A CC74 15F6 C530 740A C0ED 0480 E503 3AEC 7406 .:.t...0t.......:.t.
........
0000EFD8 0000 0000 0000 0000 C300 0000 0000 0000 0000 0000 ....................
0000EFEC 0000 0000 6650 6653 6652 669C 66B8 5000 0080 66BB ....fPfSfRf.f.P...f.
0000F000 8000 0000 E839 0066 B864 0000 8066 BB02 0202 00E8 .....9.f.d...f......
0000F014 2A00 66B8 6800 0080 66BB 4408 0020 E81B 0066 B86C *.f.h...f.D.. ...f.l
0000F028 0000 8066 BB08 0000 00E8 0C00 669D 665A 665B 6658 ...f........f.fZf[fX
0000F03C E8AA 22C3 BAF8 0C66 EFBA FC0C 66ED 660B C366 EFC3 .."....f....f.f..f..
0000F050 0000 0000 FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF ....................
........
```

Note that I've changed the last 8 bytes of **FF**h bytes with **00**h bytes. This is just for fun :-). That's it, we've done with this third step.

The next steps proceed exactly the same as what described above in *Hacking Steps in Detail*. After applying those steps, we are done and ready to flash the modified bios.

4. My Latest Modification and Possible Downside of Previous Methods

My recent experiment with various "code insertion point" in the original.tmp of my computer's bios reveals that previous modifications that I explained sometimes will result in a freezing system during boot. The possibility of this freezing event is around 10 percent in my system. I haven't found any exact explanation on this, but I think this might be related to instruction timing issue or something else that I haven't know yet. Due to this reason, I'm looking for a better method to achieve my goal. I came across the idea of completely replacing an "unneeded" code in my mainboard bios with my own code. I found that the "EPA procedure" is the most suitable for the time being. This procedure is **only** responsible for displaying EPA logo in award bios. Hence, if we replace it, the only effect is no EPA logo displayed (if you choose not to rewrite this functionality in the modified bios code, like me). It's located between **1F0C**h and **2009**h in my bios's first 64 KByte code (**E000**h segment), it may be a bit different in your bios. It can be found easily by searching for byte sequence : **808EE10110F6461430**h , this byte sequence is the first two instruction of the "EPA procedure". You also can found this procedure by searching for the "AWBM" string (not including the quote).

Note that I have to switch to an older version bios for my mainboard to accomplish this, since my mainboard's latest bios incorporated a different method which doesn't execute the "legacy EPA procedure" at all . I know this, since I've replaced the legacy code that happens to be still included in that bios's original.tmp, but nothing happen, so I conclude it must be in different part of the bios and might be in different bios component. This method displayed full size customized image during boot, not only full sized logo but also customized POST indicators on top of the full sized image. Due to this, the target bios that I'm using is an older one, dated 24th February 2000 and uses the "legacy EPA procedure". Things like this have to be taken into account in case your bios also structured like my latest mainboard bios, i.e. using a "customized EPA procedure" which is not located in the "legacy EPA procedure code spot".

The steps that we'll use here is similar to the steps in *Hacking Steps in Detail*, except that we are not only replacing one line of code but the entire procedure and also we are not placing any "injected code" in the last 4 KByte of the first 64 Kbyte of the bios code (original.tmp).

Now, the details. The hex dump around that procedure in my bios as follows :

```
Address  Hexadecimal values                                Ascii representation

00001F00 C706 1A04 1E00 1E00 9DF8 C390 808E E101 ................
00001F10 10F6 4614 3074 01C3 061E 60BE 0060 BF00 ..F.0t....`..`..
00001F20 3AB9 0030 E85A 4EC6 86ED 0155 B812 00CD :..0.ZN....U....
00001F30 10FC B800 008E C0BF 0070 B820 07B9 D007 .........p. ....
00001F40 F3AB BE0C 0AE8 B64C 0AC0 7416 BF04 00E8 .......L..t.....
00001F50 774E B800 408E D866 813E 0000 4157 424D wN..@..f.>..AWBM
00001F60 7422 C686 ED01 00BF 0C00 E85C 4EB8 0040 t".........\N..@
00001F70 8ED8 6681 3E00 0041 5742 4D0F 856B 00C6 ..f.>..AWBM..k..
00001F80 86ED 01AA B801 12B3 36CD 1033 D280 BEED ........6..3....
00001F90 01AA 7503 BA10 0033 C980 BEED 01AA 7503 ..u....3......u.
00001FA0 B9E0 01E8 2F01 80BE ED01 AA74 1403 0E04 ....../......t....
00001FB0 0081 F980 0272 EC03 1606 0081 FAD0 0172 .....r.........r
00001FC0 D6E8 9F00 B800 12B3 36CD 1080 BEED 01AA ........6.......
00001FD0 7418 C686 ED01 AABE 8942 E823 0F74 03BE t........B.#.t..
00001FE0 C442 E846 02C6 86ED 0155 80A6 E101 EFBE .B.F.....U......
00001FF0 003A BF00 60B9 0030 E886 4DB8 0000 8ED8 .:..`..0..M.....
00002000 C606 9004 0061 1F07 C380 BEED 0155 7452 .....a.......UtR
```

The disassembly of the above EPA display procedure as follows :

```
Address   Binary Code       Mnemonic
........
00001F0C  EPA Procedure :
00001F0C  808EE10110        or byte [bp+0x1e1],0x10
00001F11  F6461430          test byte [bp+0x14],0x30
00001F15  7401              jz 0x1f18
00001F17  C3                ret
00001F18  06                push es
00001F19  1E                push ds
00001F1A  60                pusha
00001F1B  BE0060            mov si,0x6000
00001F1E  BF003A            mov di,0x3a00
00001F21  B90030            mov cx,0x3000
00001F24  E85A4E            call 0x6d81
00001F27  C686ED0155        mov byte [bp+0x1ed],0x55
00001F2C  B81200            mov ax,0x12
00001F2F  CD10              int 0x10
00001F31  FC                cld
00001F32  B80000            mov ax,0x0
00001F35  8EC0              mov es,ax
00001F37  BF0070            mov di,0x7000
00001F3A  B82007            mov ax,0x720
00001F3D  B9D007            mov cx,0x7d0
00001F40  F3AB              rep stosw
00001F42  BE0C0A            mov si,0xa0c
00001F45  E8B64C            call 0x6bfe
00001F48  0AC0              or al,al
00001F4A  7416              jz 0x1f62
00001F4C  BF0400            mov di,0x4
00001F4F  E8774E            call 0x6dc9
00001F52  B80040            mov ax,0x4000
00001F55  8ED8              mov ds,ax
00001F57  66813E0000415742  cmp dword [0x0],0x4d425741 --> cmp dword [0x0],"AWBM"
          -4D
00001F60  7422              jz 0x1f84
00001F62  C686ED0100        mov byte [bp+0x1ed],0x0
00001F67  BF0C00            mov di,0xc
00001F6A  E85C4E            call 0x6dc9
```

```
00001F6D  B80040            mov ax,0x4000
00001F70  8ED8              mov ds,ax
00001F72  66813E0000415742  cmp dword [0x0],0x4d425741      --> cmp dword
[0x0],"AWBM"
         -4D
00001F7B  0F856B00          jnz near 0x1fea
00001F7F  C686ED01AA        mov byte [bp+0x1ed],0xaa
00001F84  B80112            mov ax,0x1201
00001F87  B336              mov bl,0x36
00001F89  CD10              int 0x10
00001F8B  33D2              xor dx,dx
00001F8D  80BEED01AA        cmp byte [bp+0x1ed],0xaa
00001F92  7503              jnz 0x1f97
00001F94  BA1000            mov dx,0x10
00001F97  33C9              xor cx,cx
00001F99  80BEED01AA        cmp byte [bp+0x1ed],0xaa
00001F9E  7503              jnz 0x1fa3
00001FA0  B9E001            mov cx,0x1e0
00001FA3  E82F01            call 0x20d5
00001FA6  80BEED01AA        cmp byte [bp+0x1ed],0xaa
00001FAB  7414              jz 0x1fc1
00001FAD  030E0400          add cx,[0x4]
00001FB1  81F98002          cmp cx,0x280
00001FB5  72EC              jc 0x1fa3
00001FB7  03160600          add dx,[0x6]
00001FBB  81FAD001          cmp dx,0x1d0
00001FBF  72D6              jc 0x1f97
00001FC1  E89F00            call 0x2063
00001FC4  B80012            mov ax,0x1200
00001FC7  B336              mov bl,0x36
00001FC9  CD10              int 0x10
00001FCB  80BEED01AA        cmp byte [bp+0x1ed],0xaa
00001FD0  7418              jz 0x1fea
00001FD2  C686ED01AA        mov byte [bp+0x1ed],0xaa
00001FD7  BE8942            mov si,0x4289
00001FDA  E8230F            call 0x2f00
00001FDD  7403              jz 0x1fe2
00001FDF  BEC442            mov si,0x42c4
00001FE2  E84602            call 0x222b
00001FE5  C686ED0155        mov byte [bp+0x1ed],0x55
00001FEA  80A6E101EF        and byte [bp+0x1e1],0xef
00001FEF  BE003A            mov si,0x3a00
00001FF2  BF0060            mov di,0x6000
00001FF5  B90030            mov cx,0x3000
00001FF8  E8864D            call 0x6d81
00001FFB  B80000            mov ax,0x0
00001FFE  8ED8              mov ds,ax
00002000  C606900400        mov byte [0x490],0x0
00002005  61                popa
00002006  1F                pop ds
00002007  07                pop es
00002008  C3                ret
00002008  End of EPA Procedure
........
```

The listing of my code as follows :

```
; ------------------- patch.asm ----------------------------------------------------
.486p

; Macro definition

PATCH_PCI  macro  reg_addr,mask

    mov eax,reg_addr ; fetch the address of the regs to be patched
    mov dx,in_port   ; fetch the input port addr of PCI cfg space
    out dx,eax
    mov dx,out_port
    in  eax,dx
    or  eax,mask     ; mask the regs value (activate certn. bits)
    out dx,eax
           endm


CSEG  SEGMENT PARA PUBLIC USE16 'CODE'
      ASSUME CS:CSEG
      ORG 0
     ; ORG 100h ; only for testing in dos

; equates, have been tested & works

         in_port    equ 0cf8h
         out_port   equ 0cfch

         ioq_mask   equ 00000080h
         ioq_reg    equ 80000050h
         bank_mask  equ 20000844h
         bank_reg   equ 80000068h
         tlb_mask   equ 00000008h
         tlb_reg    equ 8000006ch
         dram_mask  equ 00020202h
         dram_reg   equ 80000064h


INIT  PROC  NEAR

; save current state
    pushfd
    pushad

; disable interrupt during execution of our routine
    cli

; patch everything as needed
    PATCH_PCI ioq_reg, ioq_mask   ; patch the ioq reg
    PATCH_PCI dram_reg, dram_mask ; patch the DRAM controller
                                  ; i.e. the interleaving part
    PATCH_PCI bank_reg, bank_mask ; patch bank active page ctl reg
    PATCH_PCI tlb_reg, tlb_mask   ; activate Fast TLB lookup

; set video mode
    sti         ; enable interrupt for our video mode setup routine below
    mov ax,12h  ; set video mode to 640x480 pixel, 16 color
    int 10h

; restore last state prior to our routine
    popad
    popfd  ; implicitly restores the Interrupt flag state

; return to main bios routine (original.tmp)
    retn
```

```
INIT  ENDP

    db 8Fh dup (90h) ; fill the remaining "code space" with nop instruction

CSEG  ENDS
      END
; -------------------------- END OF patch.asm -----------------------------------
```

The resulting binary of the above code as follows :

```
Address  Hexadecimal values                                Ascii representation

00000000 669C 6660 FA66 B850 0000 80BA F80C 66EF f.f`.f.P......f.
00000010 BAFC 0C66 ED66 0D80 0000 0066 EF66 B864 ...f.f.....f.f.d
00000020 0000 80BA F80C 66EF BAFC 0C66 ED66 0D02 ......f....f.f..
00000030 0202 0066 EF66 B868 0000 80BA F80C 66EF ...f.f.h......f.
00000040 BAFC 0C66 ED66 0D44 0800 2066 EF66 B86C ...f.f.D.. f.f.l
00000050 0000 80BA F80C 66EF BAFC 0C66 ED66 83C8 ......f....f.f..
00000060 0866 EFFB B812 00CD 1066 6166 9DC3 9090 .f.......faf....
00000070 9090 9090 9090 9090 9090 9090 9090 9090 ................
00000080 9090 9090 9090 9090 9090 9090 9090 9090 ................
00000090 9090 9090 9090 9090 9090 9090 9090 9090 ................
000000A0 9090 9090 9090 9090 9090 9090 9090 9090 ................
000000B0 9090 9090 9090 9090 9090 9090 9090 9090 ................
000000C0 9090 9090 9090 9090 9090 9090 9090 9090 ................
000000D0 9090 9090 9090 9090 9090 9090 9090 9090 ................
000000E0 9090 9090 9090 9090 9090 9090 9090 9090 ................
000000F0 9090 9090 9090 9090 9090 9090 90        ............
```

The disassembly of the code around that area after manual modification using a hex editor as follows :

```
Address   Binary Code       Mnemonic
........
00001F0C  New EPA procedure :
00001F0C  669C              pushfd
00001F0E  6660              pushad
00001F10  FA                cli
00001F11  66B850000080      mov eax,0x80000050
00001F17  BAF80C            mov dx,0xcf8
00001F1A  66EF              out dx,eax
00001F1C  BAFC0C            mov dx,0xcfc
00001F1F  66ED              in eax,dx
00001F21  660D80000000      or eax,0x80
00001F27  66EF              out dx,eax
00001F29  66B864000080      mov eax,0x80000064
00001F2F  BAF80C            mov dx,0xcf8
00001F32  66EF              out dx,eax
00001F34  BAFC0C            mov dx,0xcfc
00001F37  66ED              in eax,dx
00001F39  660D02020200      or eax,0x20202
00001F3F  66EF              out dx,eax
00001F41  66B868000080      mov eax,0x80000068
00001F47  BAF80C            mov dx,0xcf8
00001F4A  66EF              out dx,eax
00001F4C  BAFC0C            mov dx,0xcfc
00001F4F  66ED              in eax,dx
00001F51  660D44080020      or eax,0x20000844
00001F57  66EF              out dx,eax
00001F59  66B86C000080      mov eax,0x8000006c
00001F5F  BAF80C            mov dx,0xcf8
00001F62  66EF              out dx,eax
00001F64  BAFC0C            mov dx,0xcfc
```

```
00001F67  66ED              in eax,dx
00001F69  6683C808          or eax,byte +0x8
00001F6D  66EF              out dx,eax
00001F6F  FB                sti
00001F70  B81200            mov ax,0x12
00001F73  CD10              int 0x10
00001F75  6661              popad
00001F77  669D              popfd
00001F79  C3                ret
00001F7A  90                nop
........
00002008  90                nop
00002008  End of New EPA procedure
........
```

The rest is just exactly the same as the previous modification explained in *Hacking Steps in Detail* and in the *A More Radical original.tmp Hacking*. That's it and we're done :-).

-- Update --

Finally I managed to mod my mainboard's latest bios that uses "custom EPA procedure" as mentioned above. In that bios binary, I found a custom BIOS component that responsible for the "custom EPA procedure", however it still contain the "legacy EPA procedure" in it's original.tmp. Below is output from unmodified version of that BIOS :

```
CBROM V2.08 (C)Award Software 2000 All Rights Reserved.

              ********  vd30728.bin BIOS component ********

 No. Item-Name         Original-Size   Compressed-Size Original-File-Name
==============================================================================
  0. System BIOS       20000h(128.00K) 15532h(85.30K)  original.tmp
  1. XGROUP CODE       057C0h(21.94K)  03AADh(14.67K)  awardext.rom
  2. CPU micro code    0A000h(40.00K)  05D03h(23.25K)  CPUCODE.BIN
  3. ACPI table        021A6h(8.41K)   00E21h(3.53K)   ACPITBL.BIN
  4. EPA LOGO          02D3Ch(11.31K)  00382h(0.88K)   iwillbmp.bmp
  5. NNOPROM           00FECh(3.98K)   00A5Fh(2.59K)   nnoprom.bin
  6. VRS ROM           02280h(8.62K)   014BBh(5.18K)   anti_vir.bin
  7. ROS ROM           14380h(80.88K)  0F670h(61.61K)  E:\2A6LGI3C\AAA\ROSUPD.BIN

  Total compress code space  = 35532h(213.30K)
  Total compressed code size = 3140Fh(197.01K)
  Remain compress code space = 04123h(16.28K)

                         ** Micro Code Information **
Update ID  CPUID  |  Update ID  CPUID  |  Update ID  CPUID  |  Update ID  CPUID
-----------------+-------------------+--------------------+-----------------
PPGA   11   0681|  PPGA   10   0683|  PPGA   08   0686|  PPGA   03   0665
SLOT1  13   0630|  SLOT1  20   0632|  SLOT1  34   0633|  SLOT1  35   0634
SLOT1  40   0651|  SLOT1  2A   0652|  SLOT1  10   0653|  SLOT1  0A   0660
SLOT1  03   0671|  SLOT1  10   0672|  SLOT1  0E   0673|  SLOT1  14   0680
SLOT1  0D   0681|  SLOT1  0C   0683|  SLOT1  07   0686|
```

As you can see, there's a component called ROS ROM. I found that this component responsible for the custom EPA procedure. Extracting this bios component can be done by excuting :

```
cbrom208 vd30728.bin /ROS Extract
```

I don't have enough time disassembling and dissecting this component, hence I only extract it and looking at it's binary contents using hexeditor. A quick read over this binary confirm my suspicion.

Then I just release this component from that bios and then "patch" the "legacy EPA procedure" in its original.tmp. Releasing this component is achieved by invoking :

```
cbrom208 vd30728.bin /ROS Release
```

To ensure that my goal to patch my BIOS will be achieved, I decided to flash the mainboard bios that has no more ROS ROM and see what happens. As predicted, the bios "fallback" to executing "legacy EPA procedure". However, we have 2 jobs to accomplish. **First**, to replace the EPA procedure and **second**, to ensure that the old "custom EPA procedure" is not called anymore, since if it's called, there's possibility our "modified EPA procedure" will not be called, you'll see the reason very soon. The "legacy EPA procedure" in my bios located between **1F1C**h and **200B**h. The disassembly as follows :

```
Address   Binary Code       Mnemonic
........
00001F1C   --- EPA Procedure ---
00001F1C  808EE10110        or byte [bp+0x1e1],0x10
00001F21  F6461430          test byte [bp+0x14],0x30
00001F25  7401              jz 0x1f28
00001F27  C3                ret
00001F28  06                push es
00001F29  1E                push ds
00001F2A  60                pusha
00001F2B  BE0060            mov si,0x6000
00001F2E  BF003A            mov di,0x3a00
00001F31  B90030            mov cx,0x3000
00001F34  E8CA4E            call 0x6e01
00001F37  C686ED0155        mov byte [bp+0x1ed],0x55
00001F3C  B81200            mov ax,0x12
00001F3F  CD10              int 0x10
00001F41  FC                cld
00001F42  B80000            mov ax,0x0
00001F45  8EC0              mov es,ax
00001F47  BF0070            mov di,0x7000
00001F4A  B82007            mov ax,0x720
00001F4D  B9D007            mov cx,0x7d0
00001F50  F3AB              rep stosw
00001F52  BE0C0A            mov si,0xa0c
00001F55  E8264D            call 0x6c7e
00001F58  0AC0              or al,al
00001F5A  7416              jz 0x1f72
00001F5C  BF0400            mov di,0x4
00001F5F  E8E74E            call 0x6e49
00001F62  B80040            mov ax,0x4000
00001F65  8ED8              mov ds,ax
00001F67  66813E0000415742  cmp dword [0x0],0x4d425741      -- cmp dword
[0x0],"AWBM"
          -4D
00001F70  7422              jz 0x1f94
00001F72  C686ED0100        mov byte [bp+0x1ed],0x0
00001F77  BF0C00            mov di,0xc
00001F7A  E8CC4E            call 0x6e49
00001F7D  B80040            mov ax,0x4000
00001F80  8ED8              mov ds,ax
00001F82  66813E0000415742  cmp dword [0x0],0x4d425741      -- cmp dword
[0x0],"AWBM"
          -4D
00001F8B  0F855D00          jnz near 0x1fec
00001F8F  C686ED01AA        mov byte [bp+0x1ed],0xaa
00001F94  B80112            mov ax,0x1201
00001F97  B336              mov bl,0x36
00001F99  CD10              int 0x10
00001F9B  33D2              xor dx,dx
```

```
00001F9D  80BEED01AA        cmp byte [bp+0x1ed],0xaa
00001FA2  7503              jnz 0x1fa7
00001FA4  BA1000            mov dx,0x10
00001FA7  33C9              xor cx,cx
00001FA9  80BEED01AA        cmp byte [bp+0x1ed],0xaa
00001FAE  7503              jnz 0x1fb3
00001FB0  B9E001            mov cx,0x1e0
00001FB3  E81A01            call 0x20d0
00001FB6  80BEED01AA        cmp byte [bp+0x1ed],0xaa
00001FBB  7414              jz 0x1fd1
00001FBD  030E0400          add cx,[0x4]
00001FC1  81F98002          cmp cx,0x280
00001FC5  72EC              jc 0x1fb3
00001FC7  03160600          add dx,[0x6]
00001FCB  81FAD001          cmp dx,0x1d0
00001FCF  72D6              jc 0x1fa7
00001FD1  E89100            call 0x2065
00001FD4  B80012            mov ax,0x1200
00001FD7  B336              mov bl,0x36
00001FD9  CD10              int 0x10
00001FDB  80BEED01AA        cmp byte [bp+0x1ed],0xaa
00001FE0  740A              jz 0x1fec
00001FE2  C686ED01AA        mov byte [bp+0x1ed],0xaa
00001FE7  C686ED0155        mov byte [bp+0x1ed],0x55
00001FEC  80A6E101EF        and byte [bp+0x1e1],0xef
00001FF1  BE003A            mov si,0x3a00
00001FF4  BF0060            mov di,0x6000
00001FF7  B90030            mov cx,0x3000
00001FFA  E8044E            call 0x6e01
00001FFD  B80000            mov ax,0x0
00002000  8ED8              mov ds,ax
00002002  C606900400        mov byte [0x490],0x0
00002007  61                popa
00002008  1F                pop ds
00002009  07                pop es
0000200A  C3                ret
0000200A  -- End of EPA Procedure --
........
```

I replace this "legacy EPA procedure" with my code. The listing of the code as follows :

```
; ------------------- patch.asm ----------------------------------------------------
.486p

; Macro definition

PATCH_PCI  macro  reg_addr,mask

    mov eax,reg_addr ; fetch the address of the regs to be patched
    mov dx,in_port   ; fetch the input port addr of PCI cfg space
    out dx,eax
    mov dx,out_port
    in  eax,dx
    or  eax,mask     ; mask the regs value (activate certn. bits)
    out dx,eax
           endm

CSEG  SEGMENT PARA PUBLIC USE16 'CODE'
      ASSUME CS:CSEG
      ORG 0

; equates, have been tested & works
```

```
    in_port    equ 0cf8h
    out_port   equ 0cfch

    ioq_mask   equ 00000080h
    ioq_reg    equ 80000050h
    bank_mask  equ 20000844h
    bank_reg   equ 80000068h
    tlb_mask   equ 00000008h
    tlb_reg    equ 8000006ch
    dram_mask  equ 00020202h
    dram_reg   equ 80000064h


INIT  PROC  NEAR

; save current state
    pushfd
    pushad

; disable interrupt during execution of our routine
    cli

; patch everything as needed
    PATCH_PCI ioq_reg, ioq_mask   ; patch the ioq reg
    PATCH_PCI dram_reg, dram_mask ; patch the DRAM controller
                                  ; i.e. the interleaving part
    PATCH_PCI bank_reg, bank_mask ; patch bank active page ctl reg
    PATCH_PCI tlb_reg, tlb_mask   ; activate Fast TLB lookup

; set video mode
    sti  ; enable interrupt for our video mode setup routine below

    ; set video mode to 640x480 pixel, 16 color and clear video buffer (assumed to
be VGA Bios)
    mov ax,12h
    int 10h

; restore last state prior to our routine
    popad
    popfd  ; implicitly restores the Interrupt flag state

; return to main bios routine (original.tmp)
    retn
INIT  ENDP

    ORG 0EFh ; zero fill the remaining "code space"

CSEG  ENDS
      END
; -------------------------- END OF patch.asm ----------------------------------
```

to assemble this code I use the following command :

```
ml /AT patch.asm /Fe patch.bin /link /TINY
```

I use hexeditor to replace the "legacy EPA procedure" in my mainboard's original.tmp with the assembled code. Now, to ensure the old "custom EPA procedure" won't be called, we look for a call to the epa procedure in the disassembly of the first 64 KB original.tmp code (segment **E000**h). I found it at address **1E56**h in my mainboard's latest bios. The disassembly as follows :

```
Address   Binary Code       Mnemonic
........
00001E3B  6800E0            push word 0xe000
00001E3E  684C1E            push word 0x1e4c
00001E41  6831EC            push word 0xec31
00001E44  68864F            push word 0x4f86
00001E47  EA30EC00F0        jmp 0xf000:0xec30
00001E4C  B800F0            mov ax,0xf000
00001E4F  8ED8              mov ds,ax
00001E51  E88C11            call 0x2fe0  -- this is the call to "custom EPA
procedure"
00001E54  7303              jnc 0x1e59  --  this jump instruction have to be
eliminated
00001E56  E8C300            call 0x1f1c  --  this is the call to "legacy EPA
procedure"
00001E59  E8AF01            call 0x200b
00001E5C  BA0018            mov dx,0x1800
........
```

I came across the above commented code after comparing my mainboard's latest bios (which has "custom EPA procedure") and its older bios which has no "custom EPA procedure". The disassembly of the bios without "custom EPA procedure" as follows :

```
Address   Binary Code       Mnemonic
........
00001E37  6800E0            push word 0xe000
00001E3A  68481E            push word 0x1e48
00001E3D  6831EC            push word 0xec31
00001E40  68F54E            push word 0x4ef5
00001E43  EA30EC00F0        jmp 0xf000:0xec30
00001E48  B800F0            mov ax,0xf000
00001E4B  8ED8              mov ds,ax
00001E4D  E8BC00            call 0x1f0c -- this is call to "legacy EPA procedure"
00001E50  E8B601            call 0x2009
00001E53  BA0018            mov dx,0x1800
........
```

so, to achieve our second goal, just replace the unneeded instruction with a nop (**90**h) instruction by using hexeditor. The disassembly of the patched code as follows :

```
Address   Binary Code       Mnemonic
........
00001E3B  6800E0            push word 0xe000
00001E3E  684C1E            push word 0x1e4c
00001E41  6831EC            push word 0xec31
00001E44  68864F            push word 0x4f86
00001E47  EA30EC00F0        jmp 0xf000:0xec30
00001E4C  B800F0            mov ax,0xf000
00001E4F  8ED8              mov ds,ax
00001E51  90                nop
00001E52  90                nop
00001E53  90                nop
00001E54  90                nop
00001E55  90                nop
00001E56  E8C300            call 0x1f1c - call to "legacy EPA procedure" (now our
modded EPA procedure)
00001E59  E8AF01            call 0x200b
00001E5C  BA0018            mov dx,0x1800
........
```

The rest is just exactly the same as the previous modification explained in *Hacking Steps in Detail* and in the *A More Radical original.tmp Hacking*. I suspect that every mainboard with a custom EPA procedure might conform to the structure that I explained here, i.e. having a "custom EPA procedure" and a "legacy EPA procedure". Due to this reason we could possibly mod any Award BIOS by using the method I explained here, it's your job to confirm this :-).

WARNING: After reading the last method described above, we might be tempted to replace the "custom EPA procedure" with call to our own "injected" procedure somewhere in the first 64 KByte (**E000**h segment) and left the "legacy EPA Procedure" to handle the "EPA display procedure". However, further experiment in my system reveals that the so called "legacy EPA procedure" above (which is in my mainboard's latest bios) is an incomplete "EPA procedure". I've try it and found that even the "legacy EPA procedure" is "partially embedded" in the "custom EPA procedure", hence the result is complete lock up during boot. We have to take this into account when doing this kind of modification. **The conclusion is:** if you want to mod a custom EPA procedure, you have to completely replace it to be safe or trace it thoroughly before modifying it, if you just want to modify it without completely replacing it. I prefer to replace them all with my own code and "save some clockcycle" by not doing the eyecandy that appear ugly to me :-) since VGA card related routine tend to be slow.

---

5. Closing Note

Some of the reader might ask, why I only replace the "exactly 3 bytes of code" instruction with a call into my code in the first two modification method above? The reason is plain and clear, to minimize the effect of the code that we inserted into the bios. I can mess up with 4 bytes instruction and place a nop instruction (**90**h) after the call into my code (which occupy 3 bytes), but I refuse to do so, since I've been experiencing a lockup with that approach in my machine. Logically, it shouldn't happen, but this is a peculiarity that needs more investigation.

If you want a bulletproof original.tmp hacking method, then I suggest the last method. If it's not possible at all in your bios, you can stick to the basic principle, i.e. look for "cosmetic procedure" and replace it with your own code. This is the safest method that I can suggest. If you are an experienced hacker, then you might find a better method. What I write here is only a humble solution to my problem. I'm not an experienced hacker in "direct hardware programming", I'm a newbie in this field :-(.

My main reason "screwing up" with original.tmp is the "isa option rom trick", that I previously employed sometimes not working with my "LAN card in disguise" (which is actually a SCSI controller) if the isa option rom is assembled by using assembler other than masm 6.11 and masm 6.15. The code that masm emit is somehow weird, I can't decode it, it seems to be undocumented opcode :-(. You can read about my "LAN card in disguise" stuff here.

That's all, I hope this would be useful for you. I have no testbed yet available for award version 6.0PG bioses :-(. However, I've just began dissecting and disassembling them. If you have found something wrong in this article or have developed new bios hacking trick based on what I explain here, please let me know.