GRUB 启动分析之 stage1.5

前言

上一篇博文介绍了 GRUB 源码的 stage1.S 会汇编成一段 446 字节的 sourcecode，stage1,grub 会将这个 stage1 放入 MBR 中。我们通过分析，知道这段代码的唯一作用就是将第二个扇区（0 柱面 0 磁道 2 扇区）处的 512 字节加载到内存中去。

一个问题就来了这个 512 个字节是从何而来，这 512 个字节又意欲何为？江湖上风传已久的 stage1.5 是什么东东，stage2 又是干什么的？ 本文将要解释这些内容。

start.S 源码分析

加载到内存的第二个扇区的内容是由 GRUB 源码 stage/start.S 汇编而成，这个汇编文件汇编出来的二进制文件大小也是 512B，一个扇区的大小。stage1 阶段结束后，这段代码就被加载进了内存。那么这段代码是干啥的呢？

我先透个底，整个 start.S 代码的作用是从 LBA 扇区号 2（0 柱面 0 磁道 3 扇区） 开始拷贝若干个扇区到内存。起始扇区号，扇区个数 内存目的地址都在 start.S 中定义了。下面我们看下代码：

```
          lastlist:

          /*
           *  This area is an empty space between the main body of code below
which
           *  grows up (fixed after compilation, but between releases it may
change
           *  in size easily), and the lists of sectors to read, which grows
down
           *  from a fixed top location.
           */

              .word 0
              .word 0

              . = _start + 0x200 - BOOTSEC_LISTSIZE

                 /* fill the first data listing with the default */
```

```
blocklist_default_start:
    .long 2     /* this is the sector start parameter, in logical
                sectors from the start of the disk, sector 0 */
blocklist_default_len:
              /* this is the number of sectors to read */
#ifdef STAGE1_5
    .word 0     /* the command "install" will fill this up */
#else
    .word (STAGE2_SIZE + 511) >> 9
#endif
blocklist_default_seg:
#ifdef STAGE1_5
    .word 0x220
#else
    .word 0x820 /* this is the segment of the starting address
                to load the data into */
#endif

firstlist:  /* this label has to be after the list data!!! */
```

firstlast 是扇区后第一个字节，start.S 对应的二进制文件对应于第二个扇区，也即（512-1024）字节。那么 firstlast 就是第二个扇区后的第一位置，即磁盘的 1024 位置处，第二扇区的 512 字节处。

blocklist_default_seg 占 2 个字节，blocklist_default_len 占两个字节，blocklist_default_start 占四个字节，那么我们可以算出（firstlist-8）的地址就是 blocklist_default_start 的地址。这个位置记录的是起始扇区。我们看到起始扇区是 2,LBA 模式扇区 number 2 对应的是（0 柱面 0 磁道 3 扇区）。至于 blocklist_default_len，注释用有提到，install 的时候，填写这个数字。

下面我们看下我们磁盘的 start.S 对应的二进制文件，我们知道，start.S 对应磁盘 512～1024 部分。

```
dd if=/dev/sda of=mbr_512_1024 bs=512 skip=1 count=1
```

现在我们看看这个 start.S 汇编出来的二进制文件是：

按照我们当前的分析。firstlist 对应上图中的 0x200 位置，那么 blocklist_default_seg 就是最后 2 个字节 0220h。从此处可以看出，对于我们的 start.S 中#ifdef STAGE1_5 这个宏是打开的。另外我们可以从上图中的 loading stage1.5 可以看出。

```
#ifdef STAGE1_5
notification_string:    .string "Loading stage1.5"
```

```
#else
notification_string:    .string "Loading stage2"
#endif
```

```
00000100  00 eb fe 4c 6f 61 64 69  6e 67 20 73 74 61 67 65  |...Loading stage|
00000110  31 2e 35 00 2e 00 0d 0a  00 47 65 6f 6d 00 52 65  |1.5......Geom.Re|
00000120  61 64 00 20 45 72 72 6f  72 00 bb 01 00 b4 0e cd  |ad. Error.......|
00000130  10 46 8a 04 3c 00 75 f2  c3 00 00 00 00 00 00 00  |.F..<.u.........|
00000140  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
000001f0  00 00 00 00 00 00 00 00  02 00 00 00 0e 00 20 02  |.............. .|
00000200
```

从我们打印出来的 2 进制文件可以看出，起始 LBA 扇区号 blocklist_default_start 是 2,blocklist_default_len 是 0x0e。

现在我们闲言少叙，继续分析代码：

```
movw    $ABS(firstlist - BOOTSEC_LISTSIZE), %di

/* save the sector number of the second sector in %ebp */
movl    (%di), %ebp

   /* this is the loop for reading the secondary boot-loader in */
bootloop:

/* check the number of sectors to read */
cmpw    $0, 4(%di)

/* if zero, go to the start function */
je  bootit
```

就如刚才分析的，将 blocklist_default_start 对应的地址存到 di 寄存器中，同时将 blocklist_default_start 位置存储的值 2 存储到 ebp 寄存器。 blocklist_default_start+4 = blocklist_default_len 的地址。那么 cmpw 这个指令比较的是需要 copy 的扇区个数，对于我们的例子是比较 0x0e 和 0,既然不相等，那么不跳转，继续执行：

```
setup_sectors:
   /* check if we use LBA or CHS */
   cmpb    $0, -1(%si)

   /* jump to chs_mode if zero */
   je  chs_mode
```

```
lba_mode:
   /* load logical sector start */
   movl    (%di), %ebx

   /* the maximum is limited to 0x7f because of Phoenix EDD */
   xorl    %eax, %eax
   movb    $0x7f, %al

   /* how many do we really want to read? */
   cmpw    %ax, 4(%di) /* compare against total number of sectors */
   /* which is greater? */
   jg  1f

   /* if less than, set to total */
   movw    4(%di), %ax
```

这段代码的含义是，先判断走那个分支，对于我们是 LBA mode，然后比较需要 copy 的扇区个数是否高于 0x7f，如果高于 7fh 个，要分批次 copy。对于我们而言，不需要因为我们 4（%di）处存放的值是 0xe。我们将 0xe 存储到 ax 寄存器

```
   1:
   /* subtract from total */
   subw    %ax, 4(%di)

   /* add into logical sector start */
   addl    %eax, (%di)

   /* set up disk address packet */

   /* the size and the reserved byte */
   movw    $0x0010, (%si)

   /* the number of sectors */
   movw    %ax, 2(%si)

   /* the absolute address (low 32 bits) */
   movl    %ebx, 8(%si)

   /* the segment of buffer address */
   movw    $BUFFERSEG, 6(%si)

   /* save %ax from destruction! */
   pushw   %ax

   /* zero %eax */
```

```
        xorl    %eax, %eax

        /* the offset of buffer address */
        movw    %ax, 4(%si)

        /* the absolute address (high 32 bits) */
        movl    %eax, 12(%si)
```

这部分工作是做 INT 13H（FUNCTION 0x42）的准备工作。我们在 GRUB 分析之 stage1 中做过一次分析了。我们再次分析下：

INT 13h AH=42h: Extended Read Sectors From Drive [edit]

Parameters:

Registers	
AH	42h = function number for extended read
DL	drive index (e.g. 1st HDD = 80h)
DS:SI	segment:offset pointer to the DAP, see below

DAP : Disk Address Packet		
offset range	size	description
00h	1 byte	size of DAP = 16 = 10h
01h	1 byte	unused, should be zero
02h..03h	2 bytes	number of sectors to be read, (some Phoenix BIOSes are limited to a maximum of 127 sectors)
04h..07h	4 bytes	segment:offset pointer to the memory buffer to which sectors will be transferred (note that x86 is little-endian: if declaring the segment and offset separately, the offset must be declared before the segment)
08h..0Fh	8 bytes	absolute number of the start of the sectors to be read (1st sector of drive has number 0)

Results:

CF	Set On Error, Clear If No Error
AH	Return Code

```
si[0]  -----10h
si[1]  -----00h
s[2 3] -----扇区个数 0xe
s·[6 7]-----拷贝到的内存位置 $BUFFERSEG=0x7000
s[08h..0Fh]----LBA 起始扇区号 对于我们是 0x2
```

继续分析，就是到了执行 INT 13H

```
        movb    $0x42, %ah
        int $0x13

        jc  read_error

        movw    $BUFFERSEG, %bx
        jmp copy_buffer
```

这多说了，就是拷贝扇区内容到$BUFFERSEG 内存处。

最后到了 copy_buffer,作用和上一篇一样。movsb 指令将 ds:si（0x7000：0x0000）处连续的 %bx 字节内容传输到 es:di 指定的内存地址（0×2200:0×0000）。其中，rep 指令的含义就是重复执行后一句指令，每执行一次。cx 减 1，直至 cx 为 0。好了，从头慢慢来，首先将 6（%di）上的 0x220 赋给%es，就是目的段地址了；然后取得 ax 的值，ax 的值是本次读出的扇区数，左移 5 位并且把值赋给 6（%si），这就是下一次循环执行这个程序时新的目的段地址了（对于我们只有 0xege 扇区自然是不需要了）。然后保存 ds 的值，接着再将 ax 左移 4 位，相当于一共左移了 9 位，也就是将 ax 的值乘以 512，然后将这个值赋给 cx 寄存器，正是要传送的字节数。

由于我们的根文件系统是 ext3 文件系统，我们发现在/boot/grub/下除了 stage1,还有 e2fs_stage1_5 这个文件，这个文件的作用是识别 ext3 文件系统的。我们知道 GRUB 开始没有 OS，也没有文件系统的概念。那么 GRUB 是从何时开始有文件系统的功能的呢。这就是 stage1.5 干的事情，stage1.5 过后，GRUB 就能识别文件系统了，就能在磁盘上识别加载文件了。怎么做到的？start.S 加载的磁盘上的那些扇区的内容，就是文件系统的代码。把这 0xe 也就是 14 个扇区的内容加载到内存后，就具备了操作启动设备上面文件的功能了。

但是文件系统千千万，我们不可能把所有文件系统的功能文件放在磁盘的扇区里面，那怎么办呢？grub 执行 setup 的时候，能够识别启动设备的文件系统，比如我们，是 ext3 文件系统，所以只需要将 ext3 部分的 e2fs_stage1_5 放入扇区。

如何证明？

事实上，e2fs_stage1_5 的前 512 个字节的内容是 start.S 对应的二进制文件，后面的部分是操作文件系统部分。我们证明下： 准备活动：

```
    root@manu:~/code/c/classical/grub/grub_test# dd if=mbr_1024_
of=mbr_1024_8096 bs=1 count=7072
    记录了 7072+0 的读入
    记录了 7072+0 的写出
    7072 字节(7.1 kB)已复制，0.0478542 秒，148 kB/秒
    root@manu:~/code/c/classical/grub/grub_test# dd if=e2fs_stage1_5
of=e2fs_stage1_5_512_7584 bs=1 skip=512 count=7072
    记录了 7072+0 的读入
    记录了 7072+0 的写出
    7072 字节(7.1 kB)已复制，0.0239908 秒，295 kB/秒
```

```
root@manu:~/code/c/classical/grub/grub_test# dd if=e2fs_stage1_5 of=e2fs_stage1_5_0_512 bs=1 count=512
记录了512+0 的读入
记录了512+0 的写出
512字节(512 B)已复制，0.0269245 秒，19.0 kB/秒
```

看下 e2fs_stage1_5 前 512 字节：

```
root@manu:~/code/c/classical/grub/grub_test# hexdump -C e2fs_stage1_5_0_512
00000000  52 56 be 03 21 e8 2a 01  5e bf f8 21 66 8b 2d 83  |RV..!.*.^..!f.-.|
00000010  7d 04 00 0f 84 ca 00 80  7c ff 00 74 3e 66 8b 1d  |}.......|..t>f..|
00000020  66 31 c0 b0 7f 39 45 04  7f 03 8b 45 04 29 45 04  |f1...9E....E.)E.|
00000030  66 01 05 c7 04 10 00 89  44 02 66 89 5c 08 c7 44  |f.......D.f.\..D|
00000040  06 00 70 50 66 31 c0 89  44 04 66 89 44 0c b4 42  |..pPf1..D.f.D..B|
00000050  cd 13 0f 82 9f 00 bb 00  70 eb 56 66 8b 05 66 31  |........p.Vf..f1|
00000060  d2 66 f7 34 88 54 0a 66  31 d2 66 f7 74 04 88 54  |.f.4.T.f1.f.t..T|
00000070  0b 89 44 0c 3b 44 08 7d  74 8b 04 2a 44 0a 39 45  |..D.;D.}t..*D.9E|
00000080  04 7f 03 8b 45 04 29 45  04 66 01 05 8a 54 0d c0  |....E.)E.f...T..|
00000090  e2 06 8a 4c 0a fe c1 08  d1 8a 6c 0c 5a 52 8a 74  |...L......l.ZR.t|
000000a0  0b 50 bb 00 70 8e c3 31  db b4 02 cd 13 72 46 8c  |.P..p..1.....rF.|
000000b0  c3 8e 45 06 58 c1 e0 05  01 45 06 60 1e c1 e0 04  |..E.X....E.`....|
000000c0  89 c1 31 ff 31 f6 8e db  fc f3 a4 1f be 14 21 e8  |..1.1.........!.|
000000d0  60 00 61 83 7d 04 00 0f  85 3c ff 83 ef 08 e9 2e  |`.a.}....<......|
000000e0  ff be 16 21 e8 4b 00 5a  ea 00 22 00 00 be 19 21  |...!.K.Z.."....!|
000000f0  e8 3f 00 eb 06 be 1e 21  e8 37 00 be 23 21 e8 31  |.?.....!.7..#!.1|
00000100  00 eb fe 4c 6f 61 64 69  6e 67 20 73 74 61 67 65  |...Loading stage|
00000110  31 2e 35 00 2e 00 0d 0a  00 47 65 6f 6d 00 52 65  |1.5......Geom.Re|
00000120  61 64 00 20 45 72 72 6f  72 00 bb 01 00 b4 0e cd  |ad. Error.......|
00000130  10 46 8a 04 3c 00 75 f2  c3 00 00 00 00 00 00 00  |.F..<.u.........|
00000140  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
000001f0  00 00 00 00 00 00 00 00  02 00 00 00 00 00 20 02  |.............. .|
00000200
```

在看下磁盘的第二个扇区，也就是 start.S 对应的二进制文件：

```
root@manu:~/code/c/classical/grub/grub_test# hexdump -C mbr_512_1024
00000000  52 56 be 03 21 e8 2a 01  5e bf f8 21 66 8b 2d 83  |RV..!.*.^..!f.-.|
00000010  7d 04 00 0f 84 ca 00 80  7c ff 00 74 3e 66 8b 1d  |}.......|..t>f..|
00000020  66 31 c0 b0 7f 39 45 04  7f 03 8b 45 04 29 45 04  |f1...9E....E.)E.|
00000030  66 01 05 c7 04 10 00 89  44 02 66 89 5c 08 c7 44  |f.......D.f.\..D|
00000040  06 00 70 50 66 31 c0 89  44 04 66 89 44 0c b4 42  |..pPf1..D.f.D..B|
00000050  cd 13 0f 82 9f 00 bb 00  70 eb 56 66 8b 05 66 31  |........p.Vf..f1|
00000060  d2 66 f7 34 88 54 0a 66  31 d2 66 f7 74 04 88 54  |.f.4.T.f1.f.t..T|
00000070  0b 89 44 0c 3b 44 08 7d  74 8b 04 2a 44 0a 39 45  |..D.;D.}t..*D.9E|
00000080  04 7f 03 8b 45 04 29 45  04 66 01 05 8a 54 0d c0  |....E.)E.f...T..|
00000090  e2 06 8a 4c 0a fe c1 08  d1 8a 6c 0c 5a 52 8a 74  |...L......l.ZR.t|
000000a0  0b 50 bb 00 70 8e c3 31  db b4 02 cd 13 72 46 8c  |.P..p..1.....rF.|
000000b0  c3 8e 45 06 58 c1 e0 05  01 45 06 60 1e c1 e0 04  |..E.X....E.`....|
000000c0  89 c1 31 ff 31 f6 8e db  fc f3 a4 1f be 14 21 e8  |..1.1.........!.|
000000d0  60 00 61 83 7d 04 00 0f  85 3c ff 83 ef 08 e9 2e  |`.a.}....<......|
000000e0  ff be 16 21 e8 4b 00 5a  ea 00 22 00 00 be 19 21  |...!.K.Z.."....!|
000000f0  e8 3f 00 eb 06 be 1e 21  e8 37 00 be 23 21 e8 31  |.?.....!.7..#!.1|
00000100  00 eb fe 4c 6f 61 64 69  6e 67 20 73 74 61 67 65  |...Loading stage|
00000110  31 2e 35 00 2e 00 0d 0a  00 47 65 6f 6d 00 52 65  |1.5......Geom.Re|
00000120  61 64 00 20 45 72 72 6f  72 00 bb 01 00 b4 0e cd  |ad. Error.......|
00000130  10 46 8a 04 3c 00 75 f2  c3 00 00 00 00 00 00 00  |.F..<.u.........|
00000140  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
000001f0  00 00 00 00 00 00 00 00  02 00 00 00 0e 00 20 02  |.............. .|
00000200
```

所有的部分都是相同的，只有最后一行的 0x1fc 位置不同，那一个位置我们已经讨论过了，blocklist_default_len，就是需要拷贝扇区的个数。e2fs_stage1_5 里面是 0,磁盘第二个扇区是 0xe，就像代码中注释中提到的/ *the command "install" will fill this up* /，这个值是 install 的时候填充的。

在看下 e2fs_stage1_5 文件的 512 字节到最后的内容：

```
root@manu:~/code/c/classical/grub/grub_test# hexdump -C e2fs_stage1_5_512_7584
00000000  ea 70 22 00 00 00 03 02  ff ff ff 00 00 00 00 00  |.p".............|
00000010  02 00 30 2e 39 37 00 ff  ff ff ff 2f 62 6f 6f 74  |..0.97...../boot|
00000020  2f 67 72 75 62 2f 73 74  61 67 65 32 00 00 00 00  |/grub/stage2....|
00000030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000070  fa 31 c0 8e d8 8e d0 8e  c0 67 66 89 2d fc 24 00  |.1.......gf.-.$.|
00000080  00 66 bd f0 1f 00 00 66  89 ec fb 67 88 15 f8 24  |.f.....f...g...$|
00000090  00 00 cd 13 66 e8 61 00  00 00 bf a0 3d 00 00 b9  |....f.a.....=...|
000000a0  7c 3e 00 00 29 f9 30 c0  fc f3 aa e8 98 02 00 00  ||>..).0.........|
000000b0  e8 8c 00 00 00 f4 e9 fa  ff ff ff 8b 44 24 08 a3  |............D$..|
000000c0  00 25 00 00 89 c3 66 8b  44 24 04 66 a3 04 25 00  |.%....f.D$.f..%.|
000000d0  00 c1 e0 04 01 c3 a1 78  3e 00 00 89 83 08 00 00  |.......x>.......|
000000e0  00 8a 15 90 3d 00 00 8b  4c 24 0c e8 51 00 00 00  |....=...L$..Q...|
000000f0  66 89 cd 67 66 ff 2d 00  25 00 00 fa 67 66 0f 01  |f..gf.-.%...gf..|
00000100  15 40 25 00 00 0f 20 c0  66 83 c8 01 0f 22 c0 66  |.@%... .f....".f|
00000110  ea 17 23 00 00 08 00 66  b8 10 00 8e d8 8e c0 8e  |..#....f........|
00000120  e0 8e e8 8e d0 8b 04 24  a3 f0 1f 00 00 a1 f4 24  |.......$.......$|
```

```
00001aa0  ff 75 08 e8 7a ff ff ff  eb 39 89 f0 31 d2 f7 73  |.u..z....9..1..s|
00001ab0  08 89 d1 31 d2 f7 73 04  89 45 e0 89 d6 8b 55 e0  |...1..s..E....U.|
00001ac0  3b 13 b8 00 01 00 00 73  1d 53 57 ff 75 18 8d 41  |;......s.SW.u..A|
00001ad0  01 50 8b 45 08 56 52 ff  75 0c 83 c0 02 50 e8 ec  |.P.E.VR.u....P..|
00001ae0  e6 ff ff 83 c4 20 8d 65  f4 5b 5e 5f 5d c3 45 72  |..... .e.[^_].Er|
00001af0  72 6f 72 20 25 75 0a 00  65 78 74 32 66 73 00 0a  |ror %u..ext2fs..|
00001b00  0a 47 52 55 42 20 6c 6f  61 64 69 6e 67 2c 20 70  |.GRUB loading, p|
00001b10  6c 65 61 73 65 20 77 61  69 74 2e 2e 2e 0a 00 69  |lease wait.....i|
00001b20  6e 74 65 72 6e 61 6c 20  65 72 72 6f 72 3a 20 74  |nternal error: t|
00001b30  68 65 20 73 65 63 6f 6e  64 20 73 65 63 74 6f 72  |he second sector|
00001b40  20 6f 66 20 53 74 61 67  65 20 32 20 69 73 20 75  | of Stage 2 is u|
00001b50  6e 6b 6e 6f 77 6e 2e 00  00 00 00 00 00 00 00 00  |nknown..........|
00001b60  f8 3c 00 00 96 34 00 00  21 35 00 00 f3 35 00 00  |.<...4..!5...5..|
00001b70  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00001b90  ff 00 00 00 01 00 00 00  ff ff ff ff ff ff ff ff  |................|
00001ba0
```

看下磁盘从第三个扇区开始，共 7072 字节的内容：

```
root@manu:~/code/c/classical/grub/grub_test# hexdump -C mbr_1024_8096
00000000  ea 70 22 00 00 00 03 02  ff ff ff 00 00 00 00 00  |.p".............|
00000010  02 00 30 2e 39 37 00 ff  ff 00 ff 2f 67 72 75 62  |..0.97...../grub|
00000020  2f 73 74 61 67 65 32 20  2f 67 72 75 62 2f 67 72  |/stage2 /grub/gr|
00000030  75 62 2e 63 6f 6e 66 00  00 00 00 00 00 00 00 00  |ub.conf.........|
00000040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000070  fa 31 c0 8e d8 8e d0 8e  c0 67 66 89 2d fc 24 00  |.1.......gf.-.$.|
00000080  00 66 bd f0 1f 00 00 66  89 ec fb 67 88 15 f8 24  |.f.....f...g...$|
00000090  00 00 cd 13 66 e8 61 00  00 00 bf a0 3d 00 00 b9  |....f.a.....=...|
000000a0  7c 3e 00 00 29 f9 30 c0  fc f3 aa e8 98 02 00 00  ||>..).0.........|
000000b0  e8 8c 00 00 00 f4 e9 fa  ff ff ff 8b 44 24 08 a3  |............D$..|
000000c0  00 25 00 00 89 c3 66 8b  44 24 04 66 a3 04 25 00  |.%....f.D$.f..%.|
000000d0  00 c1 e0 04 01 c3 a1 78  3e 00 00 89 83 08 00 00  |.......x>.......|
000000e0  00 8a 15 90 3d 00 00 8b  4c 24 0c e8 51 00 00 00  |....=...L$..Q...|
000000f0  66 89 cd 67 66 ff 2d 00  25 00 00 fa 67 66 0f 01  |f..gf.-.%...gf..|
00000100  15 40 25 00 00 0f 20 c0  66 83 c8 01 0f 22 c0 66  |.@%... .f....".f|
00000110  ea 17 23 00 00 08 00 66  b8 10 00 8e d8 8e c0 8e  |..#....f........|
00000120  e0 8e e8 8e d0 8b 04 24  a3 f0 1f 00 00 a1 f4 24  |.......$.......$|
```

```
00001aa0  ff 75 08 e8 7a ff ff ff  eb 39 89 f0 31 d2 f7 73  |.u..z....9..1..s|
00001ab0  08 89 d1 31 d2 f7 73 04  89 45 e0 89 d6 8b 55 e0  |...1..s..E....U.|
00001ac0  3b 13 b8 00 01 00 00 73  1d 53 57 ff 75 18 8d 41  |;......s.SW.u..A|
00001ad0  01 50 8b 45 08 56 52 ff  75 0c 83 c0 02 50 e8 ec  |.P.E.VR.u....P..|
00001ae0  e6 ff ff 83 c4 20 8d 65  f4 5b 5e 5f 5d c3 45 72  |..... .e.[^_].Er|
00001af0  72 6f 72 20 25 75 0a 00  65 78 74 32 66 73 00 0a  |ror %u..ext2fs..|
00001b00  0a 47 52 55 42 20 6c 6f  61 64 69 6e 67 2c 20 70  |.GRUB loading, p|
00001b10  6c 65 61 73 65 20 77 61  69 74 2e 2e 2e 0a 00 69  |lease wait.....i|
00001b20  6e 74 65 72 6e 61 6c 20  65 72 72 6f 72 3a 20 74  |nternal error: t|
00001b30  68 65 20 73 65 63 6f 6e  64 20 73 65 63 74 6f 72  |he second sector|
00001b40  20 6f 66 20 53 74 61 67  65 20 32 20 69 73 20 75  | of Stage 2 is u|
00001b50  6e 6b 6e 6f 77 6e 2e 00  00 00 00 00 00 00 00 00  |nknown..........|
00001b60  f8 3c 00 00 96 34 00 00  21 35 00 00 f3 35 00 00  |.<...4..!5...5..|
00001b70  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00001b90  ff 00 00 00 01 00 00 00  ff ff ff ff ff ff ff ff  |................|
00001ba0
```

比较一下可以得知，大部分内容是一样的，除了头部的/boot/grub 那些路径不同。

总结

通过这两篇文章我们知道了一下我们获取的信息：

1. MBR code 部分和/boot/grub/stage1 部分一样，这部分二进制文件是有 grub 源代码中

的/stage1/stage1.S 汇编出的。所谓的 stage1,作用只有一个，就是将磁盘第二个扇区的内容加载的到内存

2. 第二个扇区的内容和/boot/grub/e2fs_stage1_5 文件的前 512 字节一样，这部分内容是有 grub 源码/stage2/start.S 汇编出的，而这个 start.S 的作用就是加载磁盘的第三个扇区到第 N 个扇区到内存，N 取几，取决与文件系统的支撑代码的大小。
3. 文件系统支撑代码到内存之后，我们在也不需要直接调用 INT 13 加载扇区内容，我们有了文件系统，我们可以直接操作文件了。那么/boot/grub/stage2 这样的比较大的文件可以直接操作了。

   接下来学要搞清楚的就是 GRUB stage2 阶段提供的命令如 grub-install ,setup,还有 grub shell 相关的内容。这个只能留待后面研究了。