Grub2 Booting Process

Mike Wang (wmike@outlook.com)

May 2nd, 2016

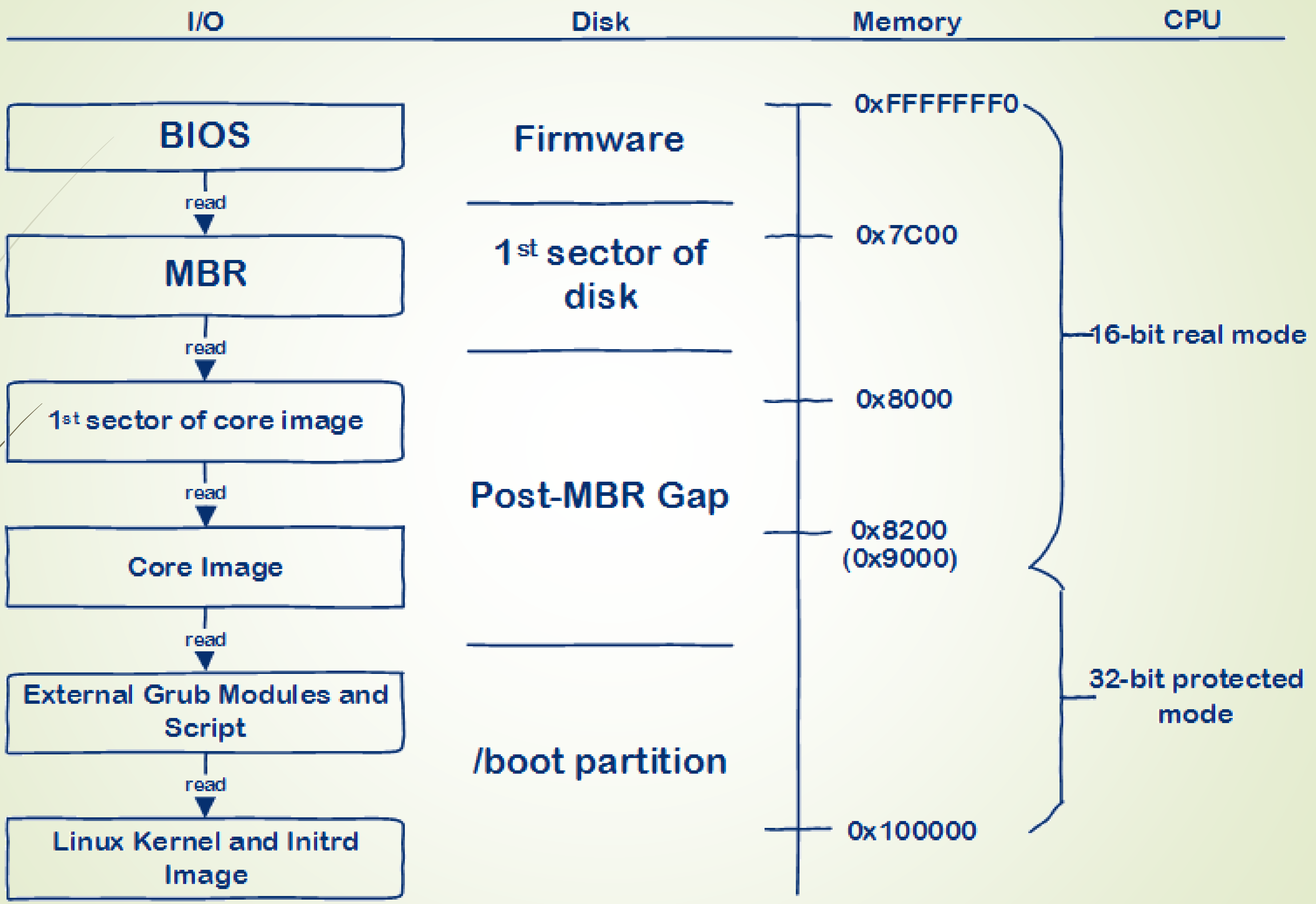
Agenda

- Scope
- Classical booting process (MBR booting)
- Build grub2 image
- UEFI booting
- Booting failure analysis

Scope

- Grub2 supports many target, but we will only focus on:
  - i386-pc
    - X86 CPU, BIOS based, Local disk boot
    - Linux distributors use it for both 32-bit system and 64-bit system
  - x86_64-efi
    - X86 64-bit CPU, UEFI based, Local disk boot
- Not include XEN, not include PXE
- Not include multiboot
- Reference
  - grub source code 2.02 beta3
  - Linux distributions
    - RHEL 7
    - SLES 12
    - Ubuntu 16.04 LTS

Classical Booting Process


I/O
Disk
Memory
CPU
BIOS
read
MBR
read
1st sector of core image
read
Core Image
read
External Grub Modules and Script
read
Linux Kernel and Initrd Image
Firmware
1st sector of disk
Post-MBR Gap
/boot partition
0xFFFFFFF0
0x7C00
0x8000
0x8200
(0x9000)
0x100000
16-bit real mode
32-bit protected mode

BIOS

- Intel x86 CPU always start (in real mode) by running the instructions at 0xFFFFFFF0 (mapping to ROM)
- That's the entry point of BIOS code
- BIOS goes through a list of pre-configured boot devices, until it finds a bootable device
- Bootable device is the one which last two bytes of first sector contains boot signature 0xAA55
- BIOS read the first sector to memory address 0x7C00
- Error message depends on BIOS vendor
  - No operating system
  - Operating system not found
  - Booting failure ...

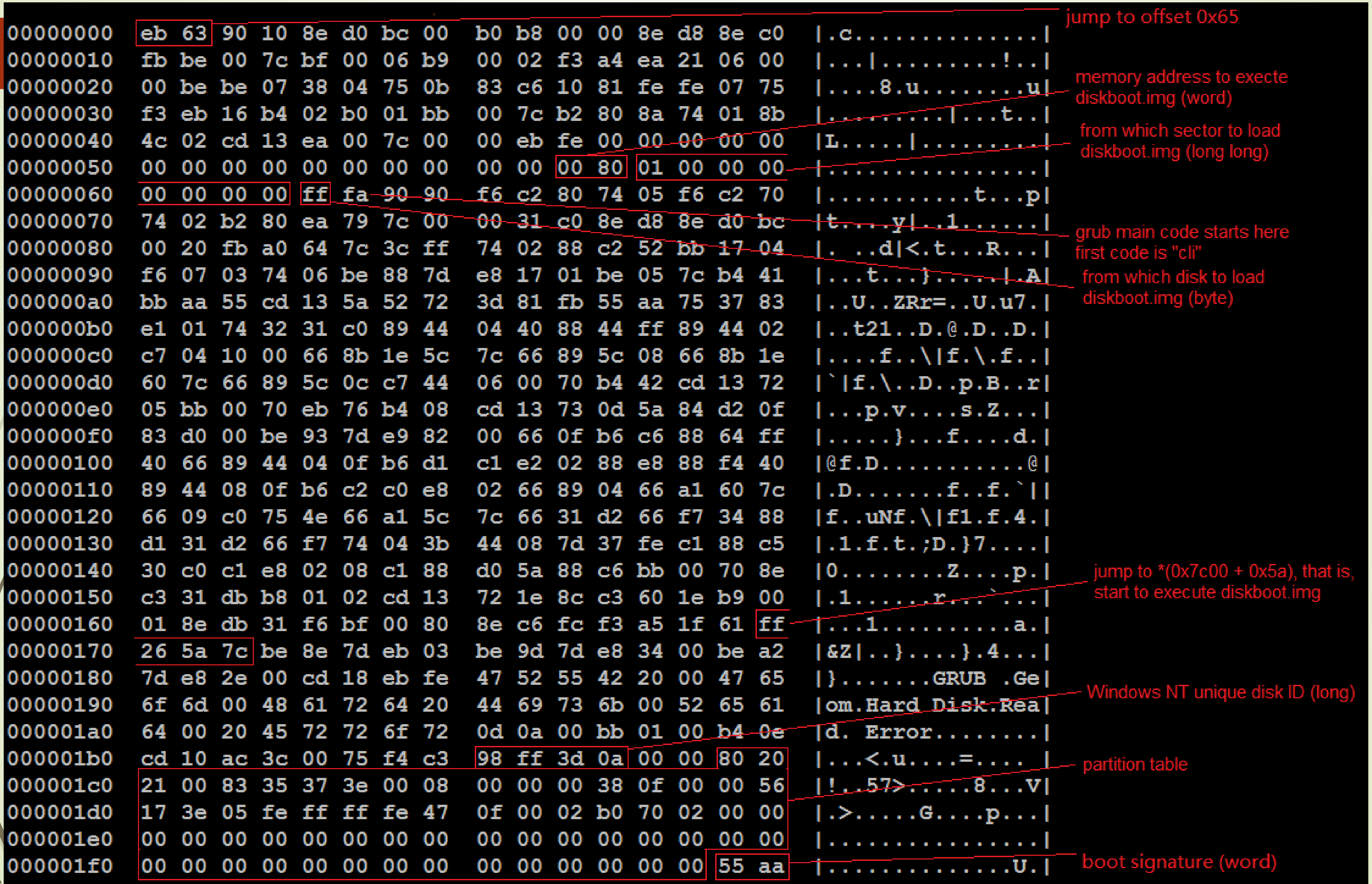
MBR

- Grub image file boot.img
- Code is at offset 0x65
- Load 1st sector of core image (diskboot.img) at memory address 0x8000
  - INT13H is used to read disk (try LBA, and then CHS)
- Contains boot information

offset	Length (bytes)	description
0x5a	2	Memory address to execute 1st sector of core image
0x5c	8	Where to load 1st sector of core image
0x64	1	The disk to load core image. 0xff means current boot disk.
0x1be	64	Partition table

Check MBR: `hexdump –C /dev/sda | more`

MBR – disk layout



```
00000000  eb 63 90 10 8e d0 bc 00  b0 b8 00 00 8e d8 8e c0  |.c..............|
00000010  fb be 00 7c bf 00 06 b9  00 02 f3 a4 ea 21 06 00  |...|.........!..|
00000020  00 be be 07 38 04 75 0b  83 c6 10 81 fe fe 07 75  |....8.u........u|
00000030  f3 eb 16 b4 02 b0 01 bb  00 7c b2 80 8a 74 01 8b  |.........|...t..|
00000040  4c 02 cd 13 ea 00 7c 00  00 eb fe 00 00 00 00 00  |L.....|.........|
00000050  00 00 00 00 00 00 00 00  00 00 00 80 01 00 00 00  |................|
00000060  00 00 00 00 ff fa 90 90  f6 c2 80 74 05 f6 c2 70  |...........t...p|
00000070  74 02 b2 80 ea 79 7c 00  00 31 c0 8e d8 8e d0 bc  |t....y|..1......|
00000080  00 20 fb a0 64 7c 3c ff  74 02 88 c2 52 bb 17 04  |. ..d|<.t...R...|
00000090  f6 07 03 74 06 be 88 7d  e8 17 01 be 05 7c b4 41  |...t...}.....|.A|
000000a0  bb aa 55 cd 13 5a 52 72  3d 81 fb 55 aa 75 37 83  |..U..ZRr=..U.u7.|
000000b0  e1 01 74 32 31 c0 89 44  04 40 88 44 ff 89 44 02  |..t21..D.@.D..D.|
000000c0  c7 04 10 00 66 8b 1e 5c  7c 66 89 5c 08 66 8b 1e  |....f..\|f.\.f..|
000000d0  60 7c 66 89 5c 0c c7 44  06 00 70 b4 42 cd 13 72  |`|f.\..D..p.B..r|
000000e0  05 bb 00 70 eb 76 b4 08  cd 13 73 0d 5a 84 d2 0f  |...p.v....s.Z...|
000000f0  83 d0 00 be 93 7d e9 82  00 66 0f b6 c6 88 64 ff  |.....}...f....d.|
00000100  40 66 89 44 04 0f b6 d1  c1 e2 02 88 e8 88 f4 40  |@f.D...........@|
00000110  89 44 08 0f b6 c2 c0 e8  02 66 89 04 66 a1 60 7c  |.D.......f..f.`||
00000120  66 09 c0 75 4e 66 a1 5c  7c 66 31 d2 66 f7 34 88  |f..uNf.\|f1.f.4.|
00000130  d1 31 d2 66 f7 74 04 3b  44 08 7d 37 fe c1 88 c5  |.1.f.t.;D.}7....|
00000140  30 c0 c1 e8 02 08 c1 88  d0 5a 88 c6 bb 00 70 8e  |0........Z....p.|
00000150  c3 31 db b8 01 02 cd 13  72 1e 8c c3 60 1e b9 00  |.1......r...`...|
00000160  01 8e db 31 f6 bf 00 80  8e c6 fc f3 a5 1f 61 ff  |...1..........a.|
00000170  26 5a 7c be 8e 7d eb 03  be 9d 7d e8 34 00 be a2  |&Z|..}....}.4...|
00000180  7d e8 2e 00 cd 18 eb fe  47 52 55 42 20 00 47 65  |}.......GRUB .Ge|
00000190  6f 6d 00 48 61 72 64 20  44 69 73 6b 00 52 65 61  |om.Hard Disk.Rea|
000001a0  64 00 20 45 72 72 6f 72  0d 0a 00 bb 01 00 b4 0e  |d. Error........|
000001b0  cd 10 ac 3c 00 75 f4 c3  98 ff 3d 0a 00 00 80 20  |...<.u....=.... |
000001c0  21 00 83 35 37 3e 00 08  00 00 00 38 0f 00 00 56  |!..57>.....8...V|
000001d0  17 3e 05 fe ff ff fe 47  0f 00 02 b0 70 02 00 00  |.>.....G....p...|
000001e0  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  00 00 00 00 00 00 55 aa  |..............U.|
```

jump to offset 0x65
memory address to execte diskboot.img (word)
from which sector to load diskboot.img (long long)
grub main code starts here first code is "cli"
from which disk to load diskboot.img (byte)
jump to *(0x7c00 + 0x5a), that is, start to execute diskboot.img
Windows NT unique disk ID (long)
partition table
boot signature (word)

MBR - message

message	type	description
GRUB	information	Print this message at the beginning of grub start (Ubuntu: shown when key “shift” is pressed) (RedHat: not show anything)
Hard Disk Error	error	Not be able to get geometry information from hard disk. CHS only.
Geom Error	error	Invalid value for start sector address. CHS only.
Read Error	error	Failed to read hard disk. CHS only.

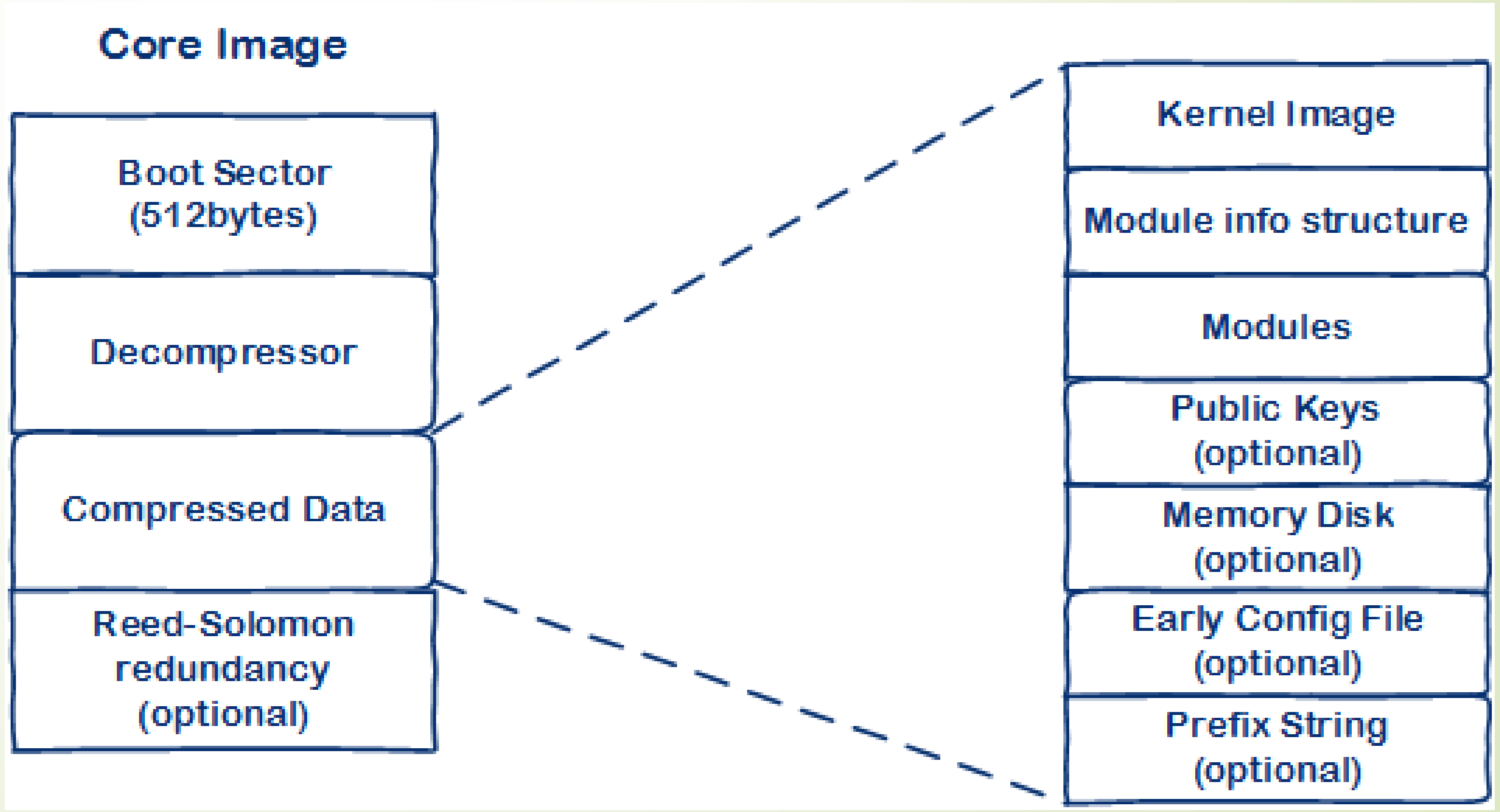
Note 1: INT13 CHS mode may only read first 8GB of disk (or 137GB)

Note 2: INT13 needs buffer must be in one memory segment.

Note 3: It may run into CHS mode because start sector exceeds disk size which causes LBA mode fail.

Core image

- Grub image file core.img



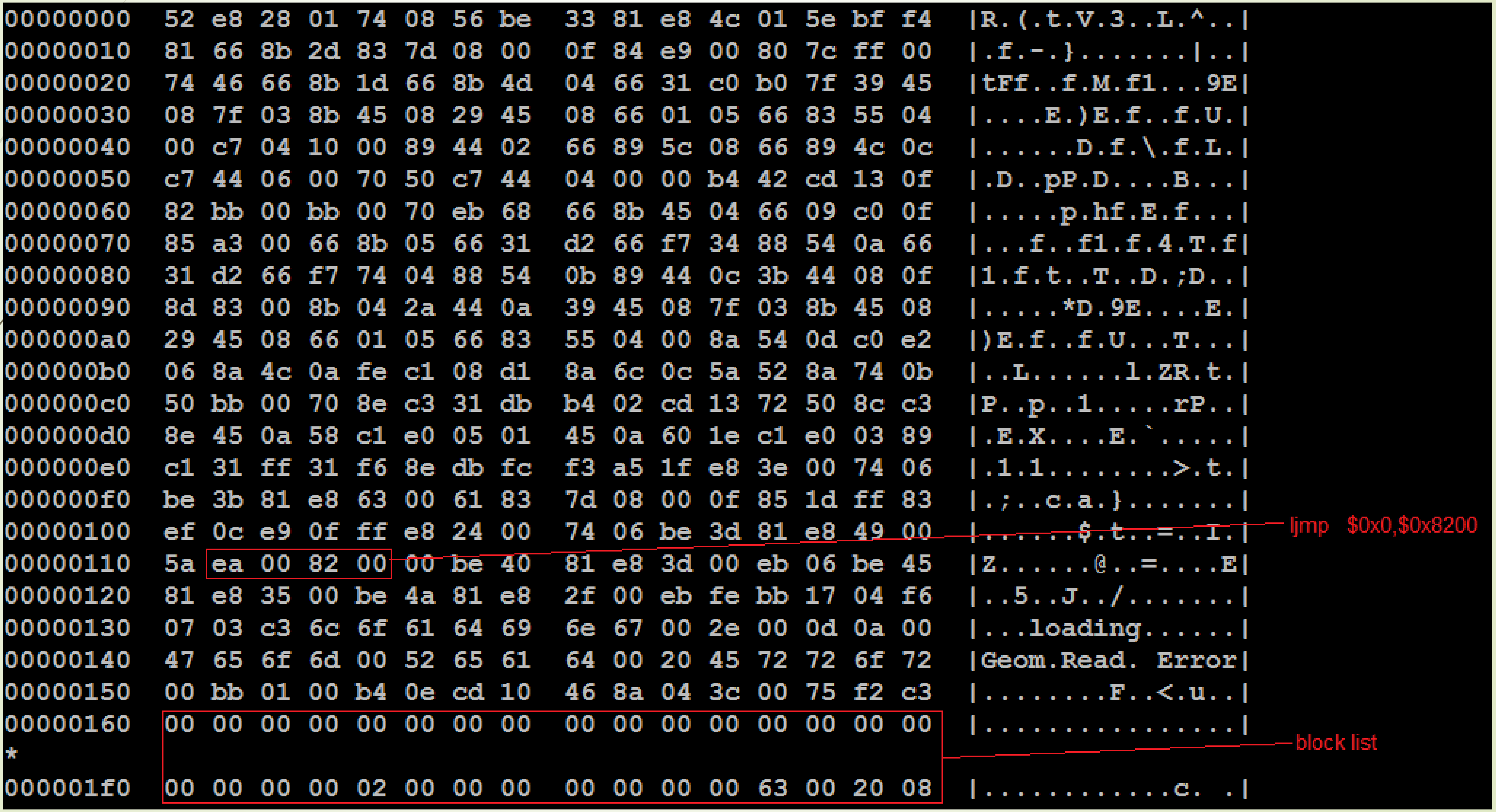
Core image maximum size relies on post-MBR gap size:

- Traditional: 63 sectors
- Today: 2048 sectors

Boot sector of core image

- Grub image file diskboot.img
- Start at memory address 0x8000
- Load core image based on block list to memory address 0x8200
- Grub-bios-setup (grub-install) will update block list for the area which core.img is actually installed
  - Continuous area is not a must

Boot sector of core image – disk layout



```
00000000  52 e8 28 01 74 08 56 be  33 81 e8 4c 01 5e bf f4  |R.(.t.V.3..L.^..|
00000010  81 66 8b 2d 83 7d 08 00  0f 84 e9 00 80 7c ff 00  |.f.-.}.......|..|
00000020  74 46 66 8b 1d 66 8b 4d  04 66 31 c0 b0 7f 39 45  |tFf..f.M.f1...9E|
00000030  08 7f 03 8b 45 08 29 45  08 66 01 05 66 83 55 04  |....E.)E.f..f.U.|
00000040  00 c7 04 10 00 89 44 02  66 89 5c 08 66 89 4c 0c  |......D.f.\.f.L.|
00000050  c7 44 06 00 70 50 c7 44  04 00 00 b4 42 cd 13 0f  |.D..pP.D....B...|
00000060  82 bb 00 bb 00 70 eb 68  66 8b 45 04 66 09 c0 0f  |.....p.hf.E.f...|
00000070  85 a3 00 66 8b 05 66 31  d2 66 f7 34 88 54 0a 66  |...f..f1.f.4.T.f|
00000080  31 d2 66 f7 74 04 88 54  0b 89 44 0c 3b 44 08 0f  |1.f.t..T..D.;D..|
00000090  8d 83 00 8b 04 2a 44 0a  39 45 08 7f 03 8b 45 08  |.....*D.9E....E.|
000000a0  29 45 08 66 01 05 66 83  55 04 00 8a 54 0d c0 e2  |)E.f..f.U...T...|
000000b0  06 8a 4c 0a fe c1 08 d1  8a 6c 0c 5a 52 8a 74 0b  |..L......l.ZR.t.|
000000c0  50 bb 00 70 8e c3 31 db  b4 02 cd 13 72 50 8c c3  |P..p..1.....rP..|
000000d0  8e 45 0a 58 c1 e0 05 01  45 0a 60 1e c1 e0 03 89  |.E.X....E.`.....|
000000e0  c1 31 ff 31 f6 8e db fc  f3 a5 1f e8 3e 00 74 06  |.1.1........>.t.|
000000f0  be 3b 81 e8 63 00 61 83  7d 08 00 0f 85 1d ff 83  |.;..c.a.}.......|
00000100  ef 0c e9 0f ff e8 24 00  74 06 be 3d 81 e8 49 00  |......$.t..=..I.|
00000110  5a ea 00 82 00 00 be 40  81 e8 3d 00 eb 06 be 45  |Z......@..=....E|
00000120  81 e8 35 00 be 4a 81 e8  2f 00 eb fe bb 17 04 f6  |..5..J../.......|
00000130  07 03 c3 6c 6f 61 64 69  6e 67 00 2e 00 0d 0a 00  |...loading......|
00000140  47 65 6f 6d 00 52 65 61  64 00 20 45 72 72 6f 72  |Geom.Read. Error|
00000150  00 bb 01 00 b4 0e cd 10  46 8a 04 3c 00 75 f2 c3  |........F..<.u..|
00000160  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
000001f0  00 00 00 00 02 00 00 00  00 00 00 00 63 00 20 08  |............c. .|
```

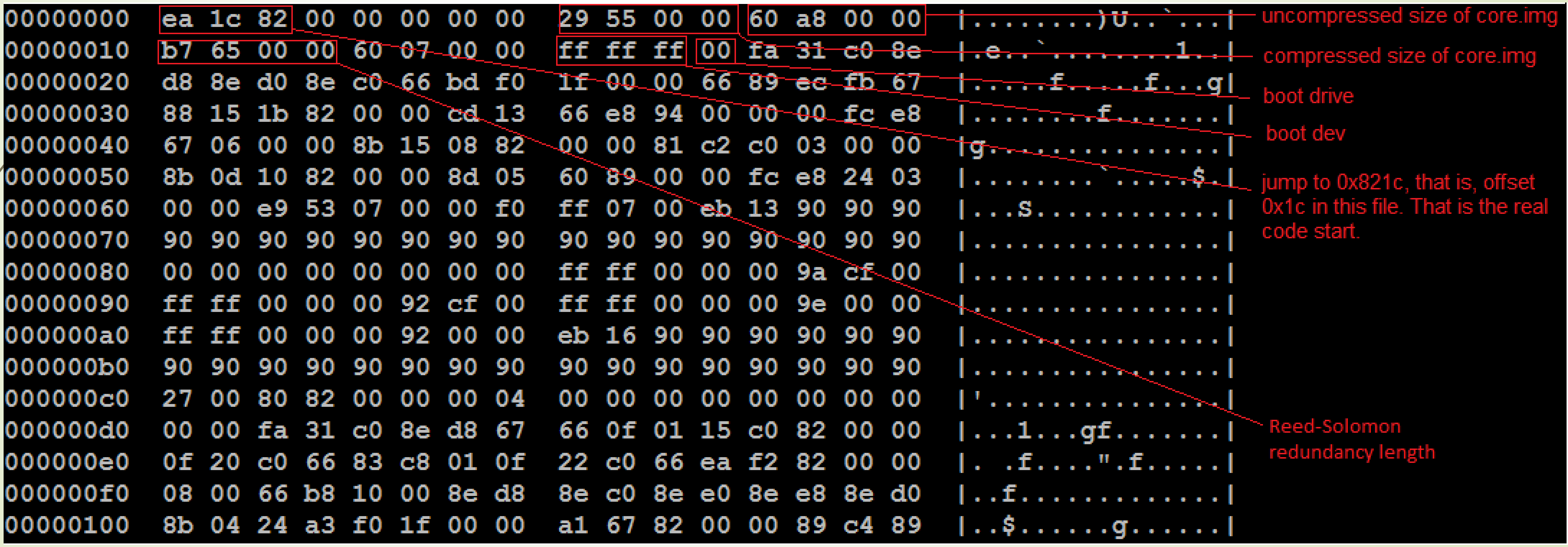
Boot sector of core image - message

message	type	description
Loading…	information	Print "loading" at the beginning of the code print one dot after load one block print new line after load all blocks (Ubuntu: shown when key “shift” is pressed) (RedHat: not print "Loading")
Geom Error	error	The value of block list is invalid. CHS only.
Read Error	error	Failed to read hard disk

Decompressor of Core Image

- Grub image file: lzma_decompress.img
- Start at memory address 0x8200
- Not really only for decompressing
  - Transition to 32-bit protected mode
  - Error correction using Reed-Solomon
  - Decompress
- Decompress data to 0x100000, and then copy kernel.img code(not include modules and its later) to 0x9000

Decompressor of Core Image – disk layout



```
00000000  ea 1c 82 00 00 00 00 00  29 55 00 00 60 a8 00 00  |........)U..`...|
00000010  b7 65 00 00 60 07 00 00  ff ff ff 00 fa 31 c0 8e  |.e..`........1..|
00000020  d8 8e d0 8e c0 66 bd f0  1f 00 00 66 89 ec fb 67  |.....f.....f...g|
00000030  88 15 1b 82 00 00 cd 13  66 e8 94 00 00 00 fc e8  |........f.......|
00000040  67 06 00 00 8b 15 08 82  00 00 81 c2 c0 03 00 00  |g...............|
00000050  8b 0d 10 82 00 00 8d 05  60 89 00 00 fc e8 24 03  |........`.....$.|
00000060  00 00 e9 53 07 00 00 f0  ff 07 00 eb 13 90 90 90  |...S............|
00000070  90 90 90 90 90 90 90 90  90 90 90 90 90 90 90 90  |................|
00000080  00 00 00 00 00 00 00 00  ff ff 00 00 00 9a cf 00  |................|
00000090  ff ff 00 00 00 92 cf 00  ff ff 00 00 00 9e 00 00  |................|
000000a0  ff ff 00 00 00 92 00 00  eb 16 90 90 90 90 90 90  |................|
000000b0  90 90 90 90 90 90 90 90  90 90 90 90 90 90 90 90  |................|
000000c0  27 00 80 82 00 00 00 04  00 00 00 00 00 00 00 00  |'...............|
000000d0  00 00 fa 31 c0 8e d8 67  66 0f 01 15 c0 82 00 00  |...1...gf.......|
000000e0  0f 20 c0 66 83 c8 01 0f  22 c0 66 ea f2 82 00 00  |. .f....".f.....|
000000f0  08 00 66 b8 10 00 8e d8  8e c0 8e e0 8e e8 8e d0  |..f.............|
00000100  8b 04 24 a3 f0 1f 00 00  a1 67 82 00 00 89 c4 89  |..$......g......|
```

uncompressed size of core.img
compressed size of core.img
boot drive
boot dev
jump to 0x821c, that is, offset 0x1c in this file. That is the real code start.
Reed-Solomon redundancy length

Note: Post MBR gap is 2048 sectors.

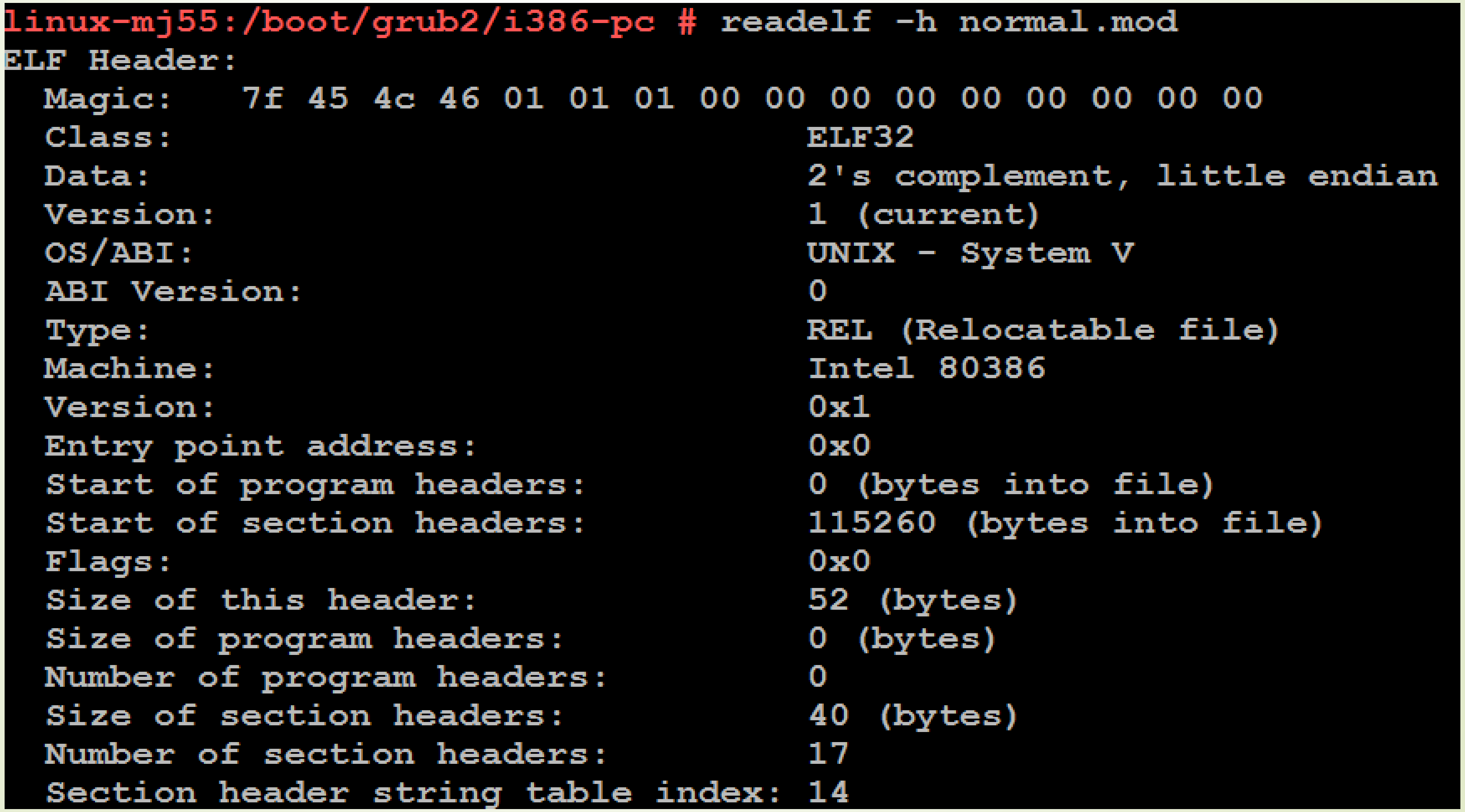
Core Image – kernel.img

- Load embedded modules
- Set “prefix” and “root”
  - They are used when load external modules
  - “prefix” can be derived from boot drive, if it’s not set explicitly
  - “root” is the device part of “prefix”
- Register core commands – commands which do not rely on external modules
  - Only four: set, unset, ls, insmod
- Parse early configure file
- Load “normal” module (and its dependencies) and run it
- When command “normal” is executed:
  - Load grub external script file grub.cfg
  - Show menu interface
- If something goes wrong, run into “rescue” mode

Grub2 modules

- Folder /boot/grub2/<target_name>/
  - For example, /boot/grub2/i386-pc
  - External modules are not recommended for UEFI boot
- Much like Linux kernel module
  - ELF file format
  - There is "init" and "fini" function
  - Check license type during loading
  - Register commands in "init" function
- Contains dependency information
  - So its dependent modules can be loaded automatically
- Important files
  - moddep.lst: dependency relationship, used by grub-install, grub-mkimage
  - command.lst: command-module relationship. So you can use a command without need "insmod"
  - modinfo.sh: grub compile information, include version number

Grub2 modules – ELF header



```
linux-mj55:/boot/grub2/i386-pc # readelf -h normal.mod
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF32
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              REL (Relocatable file)
  Machine:                           Intel 80386
  Version:                           0x1
  Entry point address:               0x0
  Start of program headers:          0 (bytes into file)
  Start of section headers:          115260 (bytes into file)
  Flags:                             0x0
  Size of this header:               52 (bytes)
  Size of program headers:           0 (bytes)
  Number of program headers:         0
  Size of section headers:           40 (bytes)
  Number of section headers:         17
  Section header string table index: 14
```

Core Image – boot Linux kernel

- Linux kernel file contains three parts
  - Boot sector
  - Setup code (16-bit real mode)
  - Protected mode code (vmlinux)
- Use command "linux" to load Linux kernel
  - 32-bit boot protocol is used
  - Grub will prepare environment for Linux kernel, and load protected mode code
  - Jump to Linux kernel protected mode code when boots
- Use command "linux16" to load Linux kernel
  - Load both setup code and protected mode code
  - Jump to setup code (means go back to real mode) when boots
  - Kernel setup code will prepare environment, and then run protected mode code
- For 64-bit kernel, it's kernel's code enters into long mode

SUSE is different

- MBR code is from syslinux
  - Relocate itself to 0x0000:0x0600
  - Scan active partition to load its first sector to 0x0000:0x7c00
  - Jump to 0x7c00 to continue execute
- The 1st sector of active partition (Volume Boot Record, VBR)
  - It's grub's MBR code
- Error message

Multiple active partitions.\r\n	When multiple partitions are in active state
Operating system load error.\r\n	Failed to read disk
Missing operating system.\r\n	Not found an active partition to boot

Build Grub2 Image

Create and install grub

- Grub-mkimage
  - Create core image based on input
  - Compress algorithm
    - For i386-pc-*, LZMA implemented by grub
    - For x86_64-efi, no compress is used
- Grub-bios-setup
  - Install MBR
    - Avoid overwrite BPB area (used by Windows NT)
    - If it's hard disk, replace offset 0x66 with two 0x90 (workaround for buggy BIOS firmware)
    - Avoid overwriting Windows NT magic code and partition table
  - Install core image
    - Update block list in boot sector
    - Update MBR for where boot sector can be accessed
    - Update Reed-Solomon redundancy field if it's used
- Grub-install
  - Higher level command includes functions of grub-mkimage and grub-bios-setup
  - Decides which modules should be embedded based on configuration of current machine

Store core image in file system?

- Not support cases
  - Software RAID/LVM
  - Certain file systems (e.g. BtrFS, ZFS)
  - Cross disk install(/boot being on one disk but MBR on another)
- Show warning to user that this is not reliable and is discouraged
- Only continue if user has specified "--force" option
- Get block list of file core.img and store it in boot sector
- Modify MBR to point to boot sector

Core Image – which modules should be embedded?

- Include:
  - Access disk
  - Parse partition
  - Parse volume manager(if applied)
  - Parse file system
  - And the dependencies of above
- For example (for target i386-pc, BIOS based, local disk boot):
  - Biosdisk (INT13H is used)
  - Part_msdos
  - ext2

Example prefix

- For simple volume
  - (hd0,1)/boot/grub
- For LVM
  - (lvm/lvm_group_name-lvm_logical_boot_partition_name)/boot/grub
  - (lv/system-root)/boot/grub
- For Software RAID
  - (md/md0)/boot/grub
  - (mduuid/123456789abcdef0123456789abcdef0)/boot/grub
- Or a better way
  - search.fs_uuid 01234567-89ab-cdef-0123-456789abcdef root
  - set prefix=($root)/boot/grub

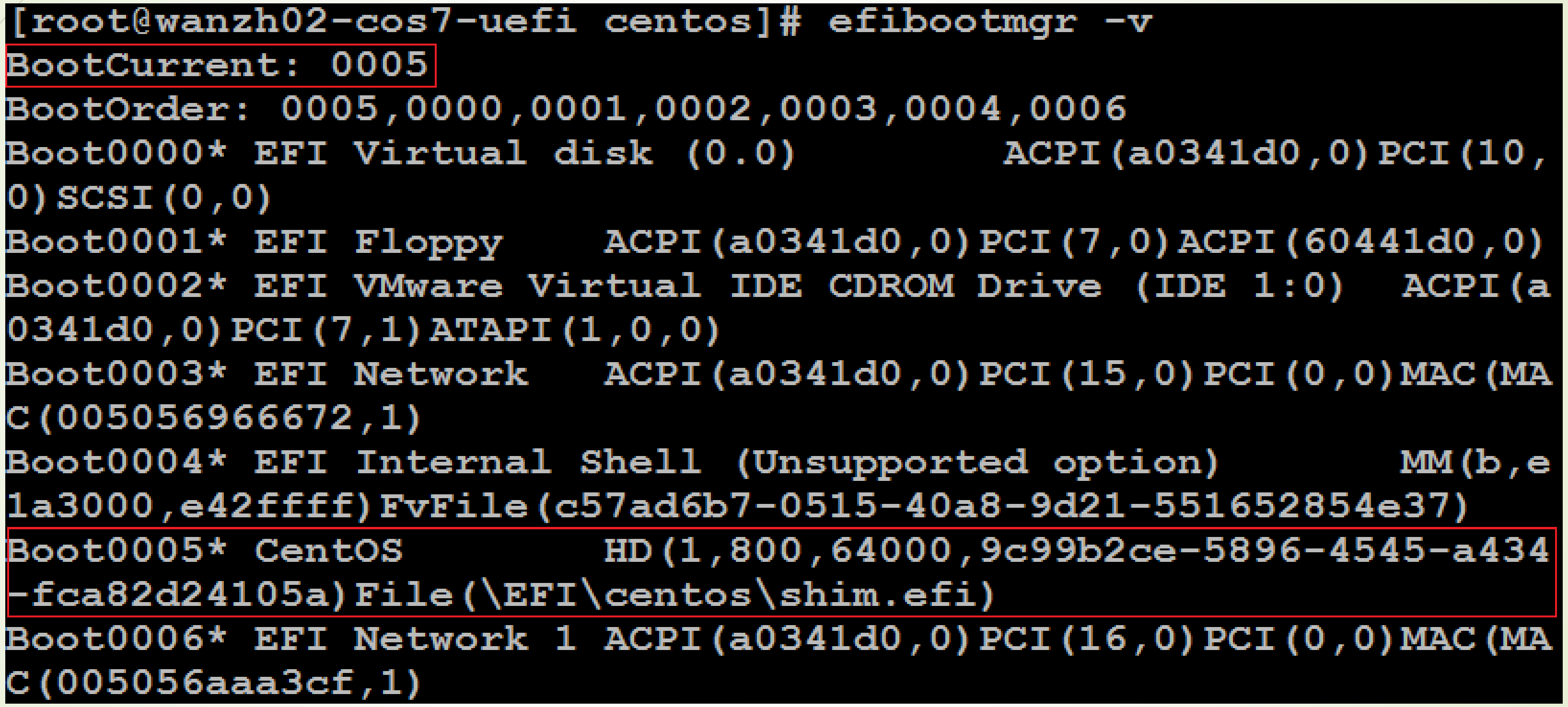
UEFI Booting

For target “x86_64-efi”

UEFI boot

- Firmware enables protected mode
- Firmware understands partition table and FAT file system
- Try to find GPT partition with GUID "**C12A7328-F81F-11D2-BA4B-00A0C93EC93B**" or MBR type **0xEF**.
  - The partition should contains FAT file system
- Try to find boot loader from
  - The default file: /EFI/BOOT/BOOTX64.EFI
- But it can be changed using command "efibootmgr"
- Two stages booting is used because of Secure Boot requirement
- Different vendor uses different boot loader
  - Red Hat, SUSE 12.x: grub2
  - SUSE 11.x: elilo

UEFI boot – boot entry



```
[root@wanzh02-cos7-uefi centos]# efibootmgr -v
BootCurrent: 0005
BootOrder: 0005,0000,0001,0002,0003,0004,0006
Boot0000* EFI Virtual disk (0.0)        ACPI(a0341d0,0)PCI(10,
0)SCSI(0,0)
Boot0001* EFI Floppy    ACPI(a0341d0,0)PCI(7,0)ACPI(60441d0,0)
Boot0002* EFI VMware Virtual IDE CDROM Drive (IDE 1:0)  ACPI(a
0341d0,0)PCI(7,1)ATAPI(1,0,0)
Boot0003* EFI Network   ACPI(a0341d0,0)PCI(15,0)PCI(0,0)MAC(MA
C(005056966672,1)
Boot0004* EFI Internal Shell (Unsupported option)       MM(b,e
1a3000,e42ffff)FvFile(c57ad6b7-0515-40a8-9d21-551652854e37)
Boot0005* CentOS        HD(1,800,64000,9c99b2ce-5896-4545-a434
-fca82d24105a)File(\EFI\centos\shim.efi)
Boot0006* EFI Network 1 ACPI(a0341d0,0)PCI(16,0)PCI(0,0)MAC(MA
C(005056aaa3cf,1)
```

Use “blkid” to get PartUUID for your partitions.

You can also get some information from /sys/firmware/efi/

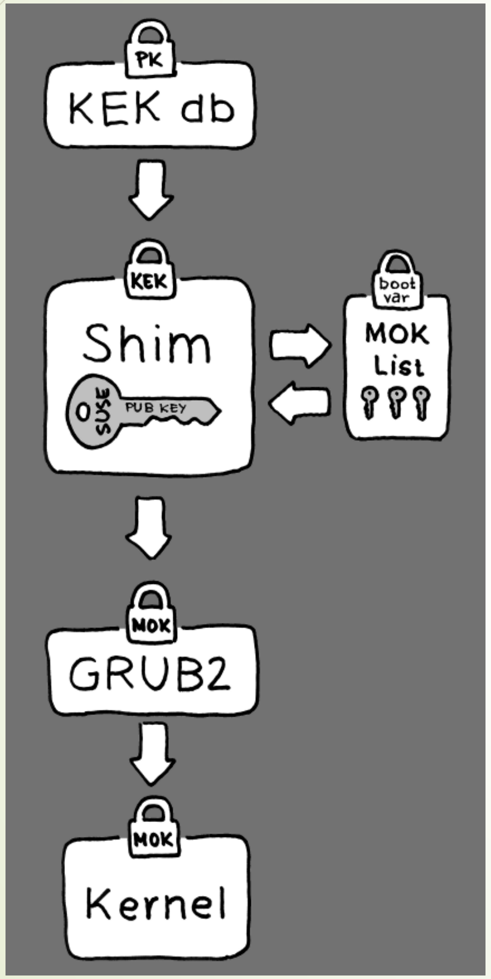
UEFI Boot – boot files

- Shim.efi is signed by Microsoft's KEK, and can be verify by the certificate in UEFI firmware. It contains OS vendor's public key
  - Try to load 2nd stage boot loader (grubx64.efi) by using function provided by UEFI firmware
  - If it's failed, it will verify signature of 2nd stage boot loader by itself, and then load and run it
- grubx64.efi (name cannot be changed because it's hardcoded in shim.efi) contains:
  - 1KB PE header
  - Signature
  - Core image data without compressing
    - Additional module "linuxefi" (not in upstream)
    - Think about what modules should be embedded?
      - Answer is: all (because grub2 modules has no signature, and cannot be verified)
  - Call back to shim to verify linux kernel signature

UEFI Boot – boot files

- MokManager.efi
  - Machine Owner Key Manager
  - Add trust to secure boot
    - Can add as root key
  - Command “mokutil” can be used to register key
  - The key import request is recognized by shim.efi, and it will call MokManager.efi
  - Need reboot and a physical user’s interaction
- Use command “pesign -S –I <file>” to view signature
  - You will find there is bug in CentOS 7.1 where shim.efi is signed by Red Hat Inc.

Secure Boot



- Platform Key
  - Control key update
  - Installed in firmware by hardware vendor
- Key Exchanged Key
  - Used to verify boot loader
  - Microsoft's key is generally installed by default
- Machine Owner Key
  - No need trust relationship with PK or KEK

Ref: https://www.suse.com/documentation/sles-12/book_sle_admin/data/sec_uefi_secboot.html

Booting Failure Analysis

Debug MBR/Boot Sector

- Find clue from error message
- Consult source code
- Reverse opcode to ASM
  - Try to skip data area, otherwise you may get wrong result sometimes

```
objdump -D -b binary -mi386 -Maddr16,data16 mbr.img
```

- Modify opcode for debugging purpose

```
echo -ne "\\x90\\x90\\x90" | dd of=/dev/sda bs=1 seek=6 conv=notrunc
```

ASM code	Opcode	Description
nop	0x90	No operation
jmp N	0xeb 0x(N-2)	Jump offset N to current address
ljmp <segment> <ADDR>	0xea 0x(ADDR_LOW) 0x(ADDR_HIGH) 0x(SEG_LOW) 0x(SEG_HIGH)	Long jump to specified address
call <ADDR>	0xe8 0x(OFFSET_LOW) 0x(OFFSET_HIGH)	Call a function at ADDR. OFFSET=(ADD - CURRENT_ADDR) - 3
ret	0xc3	Return from function call

Debug MBR/Boot Sector – qemu + gdb

- Don’t enable KVM
- Skip interrupt call (e.g. INT13H, break at its next instruction)
- ASM may not correct
- Use “nexti” or “stepi”
- If you use partial image file, be care of its size is valid for MBR code
- Qemu’s BIOS supports LBA mode (is it the same as your source machine?)

Qemu:

```
qemu-system-x86_64 -m 16M -drive file=/dev/sdb,format=raw,if=scsi,readonly -
gdb tcp::1234    -nographic   -vnc :0
```

Gdb:

```
set architecture i8086
target remote localhost:1234
break *0x7c00
layout asm
x /16bx 0x7df0              #print memory content
set *0x7df0 = 10            #modify memory content
info registers eax
```

Figure out where core image is

- Where is the first sector of core image?
  - In MBR (mentioned previously)
- Get partition boundary

```
parted /dev/sda "unit s print"
parted /dev/sda "unit chs print"
sfdisk -u S -l /dev/sda
```

- File system boundary

File System	Super Block Offset (to device)	Magic Number Offset (to super block)	Magic Number Content
Ext4	0x400 (1KB)	0x38	0xEF53
XFS	0x00	0x00	0x58465342("XFSB")
BTRFS	0x10000 (64KB)	0x28	"_BHRfS_M"
ReiserFS	0x10000 (64KB)	0x34	"ReIsEr2Fs"

- File blocks mapping

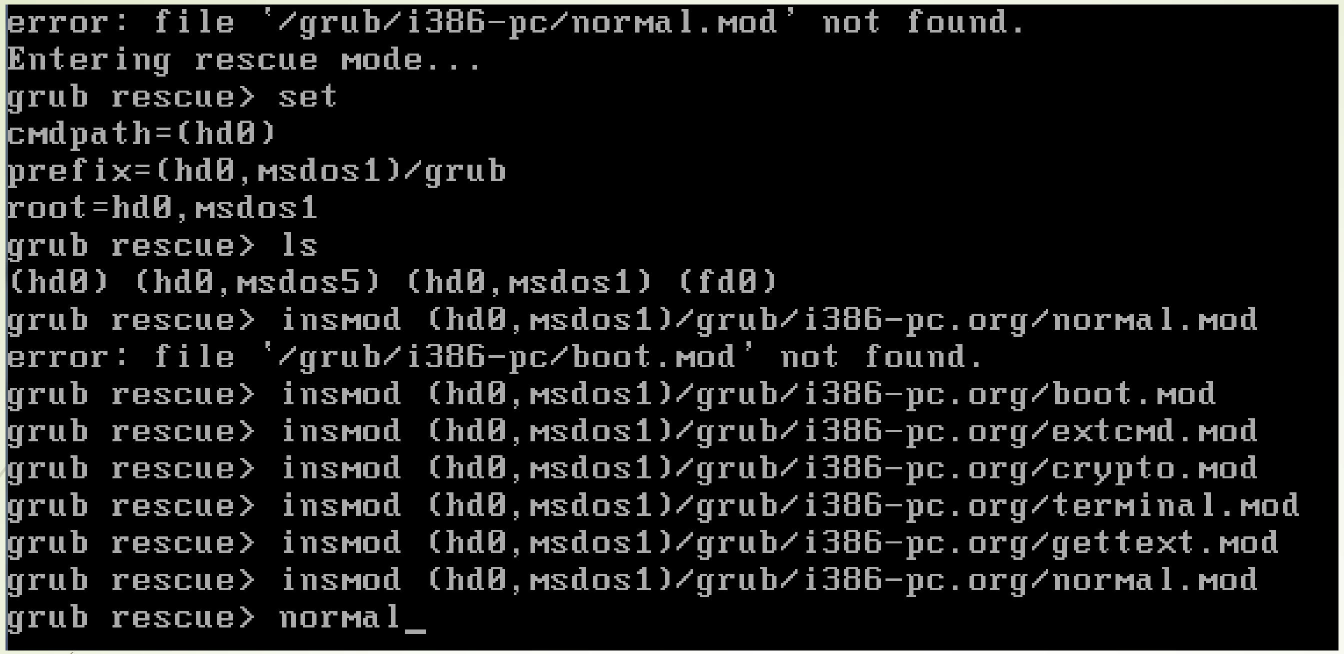
```
filefrag -v /file
```

Debug in grub environment

- Show debug message

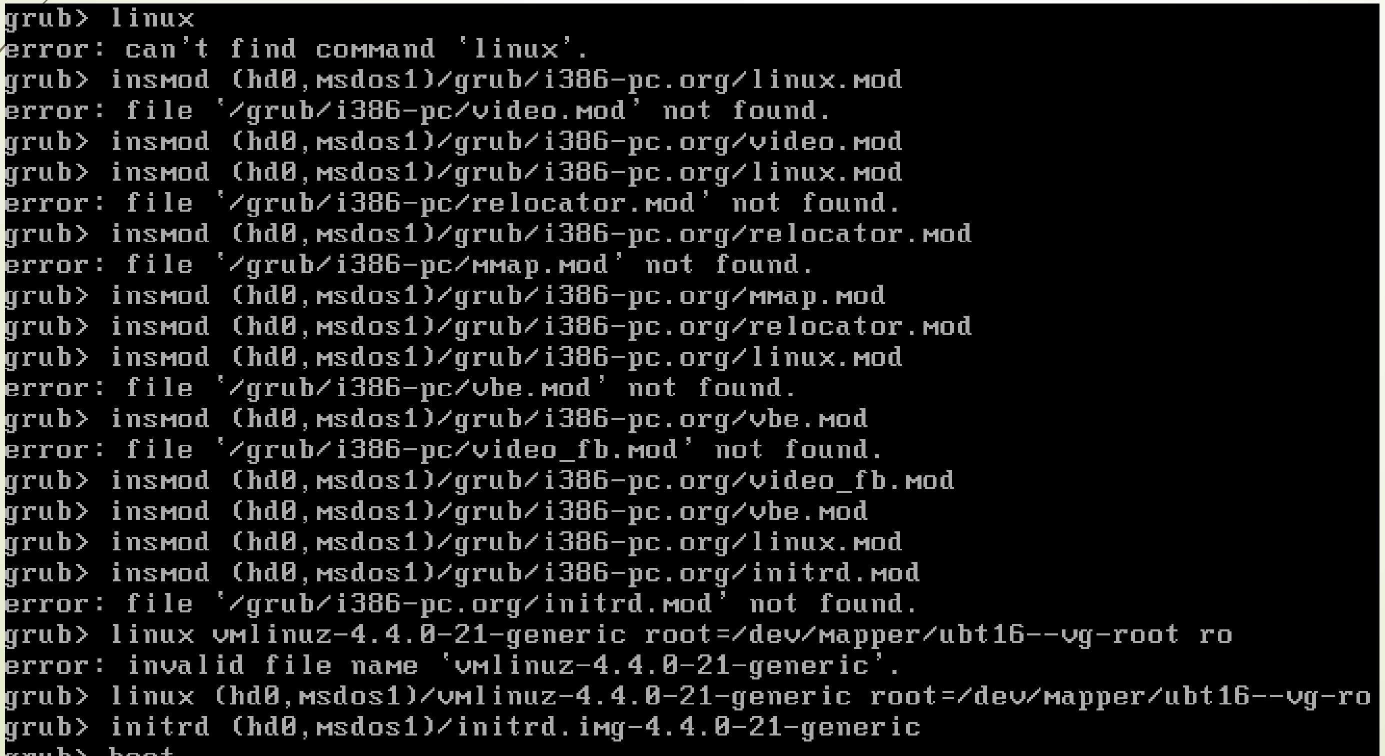
set pager=1	pause output after each screenful and wait for keyboard input
set debug=all	Candidate value can also be specific module, e.g. disk,linux,partition,modules

- Test case: recover from grub boot failure
  - Ubuntu 16.04 LTS
  - /boot is a separated volume
  - Rename folder i386-pc to i386-pc.org.



```
error: file '/grub/i386-pc/normal.mod' not found.
Entering rescue mode...
grub rescue> set
cmdpath=(hd0)
prefix=(hd0,msdos1)/grub
root=hd0,msdos1
grub rescue> ls
(hd0) (hd0,msdos5) (hd0,msdos1) (fd0)
grub rescue> insmod (hd0,msdos1)/grub/i386-pc.org/normal.mod
error: file '/grub/i386-pc/boot.mod' not found.
grub rescue> insmod (hd0,msdos1)/grub/i386-pc.org/boot.mod
grub rescue> insmod (hd0,msdos1)/grub/i386-pc.org/extcmd.mod
grub rescue> insmod (hd0,msdos1)/grub/i386-pc.org/crypto.mod
grub rescue> insmod (hd0,msdos1)/grub/i386-pc.org/terminal.mod
grub rescue> insmod (hd0,msdos1)/grub/i386-pc.org/gettext.mod
grub rescue> insmod (hd0,msdos1)/grub/i386-pc.org/normal.mod
grub rescue> normal_
```

Grub menu will show up. But we cannot use the menu to boot linux kernel because many modules are missing. Press "c" to enter command line mode.



```
grub> linux
error: can't find command 'linux'.
grub> insmod (hd0,msdos1)/grub/i386-pc.org/linux.mod
error: file '/grub/i386-pc/video.mod' not found.
grub> insmod (hd0,msdos1)/grub/i386-pc.org/video.mod
grub> insmod (hd0,msdos1)/grub/i386-pc.org/linux.mod
error: file '/grub/i386-pc/relocator.mod' not found.
grub> insmod (hd0,msdos1)/grub/i386-pc.org/relocator.mod
error: file '/grub/i386-pc/mmap.mod' not found.
grub> insmod (hd0,msdos1)/grub/i386-pc.org/mmap.mod
grub> insmod (hd0,msdos1)/grub/i386-pc.org/relocator.mod
grub> insmod (hd0,msdos1)/grub/i386-pc.org/linux.mod
error: file '/grub/i386-pc/vbe.mod' not found.
grub> insmod (hd0,msdos1)/grub/i386-pc.org/vbe.mod
error: file '/grub/i386-pc/video_fb.mod' not found.
grub> insmod (hd0,msdos1)/grub/i386-pc.org/video_fb.mod
grub> insmod (hd0,msdos1)/grub/i386-pc.org/vbe.mod
grub> insmod (hd0,msdos1)/grub/i386-pc.org/linux.mod
grub> insmod (hd0,msdos1)/grub/i386-pc.org/initrd.mod
error: file '/grub/i386-pc.org/initrd.mod' not found.
grub> linux vmlinuz-4.4.0-21-generic root=/dev/mapper/ubt16--vg-root ro
error: invalid file name 'vmlinuz-4.4.0-21-generic'.
grub> linux (hd0,msdos1)/vmlinuz-4.4.0-21-generic root=/dev/mapper/ubt16--vg-ro
grub> initrd (hd0,msdos1)/initrd.img-4.4.0-21-generic
grub> boot
```

Linux kernel is booted now.

Thank You