talk is cheap;Later equals never;

nasm : 用栈传递 int13h ah=42h 的 disk address packet 参数分析

逆向ultraIso制作的MBR, 看到 disk address packet 参数是通过栈传递的.

这样搞，确实省空间, 不过容易看错, 也不好理解.

[plain]

```
; =============== S U B R O U T I N E =======================================

; /// @fn fn_read_sectors_into_memory
; /// @brief 读扇区到内存
; /// @param bp 被激活的分区表入口地址
; /// @param cx 要拷贝的扇区数量
; /// @param dx 激活的分区表入口中指定第一个扇区索引的高8位
; /// @param ax 激活的分区表入口中指定第一个扇区索引的低8位
; /// @param bx 扇区读取后，要拷贝到目的内存地址

fn_read_sectors_into_memory:
    push di
    mov di, 5 ; ///< 读取扇区到内存失败时的重试次数

fn_read_sectors_into_memory_retry:
    pusha

;   Format of disk address packet:

;Offset  Size    Description     (Table 00272)
```

```
;10h    QWORD   (EDD-3.0, optional) 64-bit flat address of transfer buffer;
;               used if DWORD at 04h is FFFFh:FFFFh
;08h    QWORD   starting absolute block number
;               (for non-LBA devices, compute as
;               (Cylinder*NumHeads + SelectedHead) * SectorPerTrack +
;               SelectedSector - 1
;04h    DWORD   -> transfer buffer
;02h    WORD    number of blocks to transfer (max 007Fh for Phoenix EDD)
;01h    BYTE    reserved (0);
;00h    BYTE    size of packet (10h or 18h)

    ; /// 压入 disk address packet, 并将栈顶给si, 供 int 13h, ah = 0x42 使用
    ; /// 这么搞确实省空间, 不过很容易填错参数, 分析者也容易看错参数

    xor cx, cx

    ; /// 开始的绝对块号 = 1
    ; /// 此时 cx == 0, dx:ax = 0000,0001
    push cx ;08h    QWORD   starting absolute block number
    push cx
    push dx
    push ax

    ; /// 此时, cx:bx = 0000,7e00
    push cx ;04h    DWORD   -> transfer buffer
    push bx

    ; /// 此时 cx = 0
    inc cx ; ///< 此时, cx = 1, 说明要传送的块数为1
    push cx ;02h    WORD    number of blocks to transfer (max 007Fh for Phoenix EDD)

    push 0010h ;00h    BYTE    size of packet (10h or 18h)
               ;01h    BYTE    reserved (0);

    ; /// DS:SI -> disk address packet
    mov si, sp

    ; /// 根据分区表是否有效, 选择使用int 13h ah = 42h 扩展读扇区, 还是使用int 13h ah = 2 根据CHS来读扇区
    test byte[byte_status], BIT_VALID_PARTITION_SET
    jz label_read_disk_to_memory_by_chs ; ///< 不是TRUE(分区校验无效), 转 int13h, ah = 2 读扇区

    ; /// 分区表有效的处理
    ; /// 使用 int13h, ah = 0x42 读扇区
    mov ah, 0x42
    mov dl, [byte_udisk_sn]     ; DL = drive
    ; /// DS:SI -> disk address packet, 已经由函数入口处压入了 disk address packet, 压入时, 已经填好了参数, 并将 sp => si
    jmp label_read_sectors_into_memory

```

```
69.     ; /// 否则顺序执行下列代码
70. label_read_disk_to_memory_by_chs:
71.     ; /// ax 激活的分区表入口中指定第一个扇区索引的低8位
72.     div word [word_int13h_8_rc_cs]
73.     inc dx
74.     mov cx, dx
75.     xor dx, dx
76.     div word [word_int13h_8_rc_status_code]
77.     mov ch, al          ; CH = track
78.     shr ax, 2
79.     and al, 0C0h
80.     or cl, al          ; CL = sector
81.     mov dh, dl          ; DH = head
82.     mov ax, 201h
83.
84. label_read_sectors_into_memory:
85.     int 13h             ; DISK - READ SECTORS INTO MEMORY
86.
87.     lea sp, [si+10h] ; ///< 堆栈平衡
88.     popa
89.
90.     jnb fn_read_sectors_into_memory_end
91.     dec di
92.     jz fn_read_sectors_into_memory_ok
93.     pusha
94.     xor ah, ah
95.     mov dl, [byte_udisk_sn]
96.     int 13h             ; DISK - RESET DISK SYSTEM
97.                         ; DL = drive (if bit 7 is set both hard disks and floppy disks reset)
98.     popa
99.     jmp fn_read_sectors_into_memory_retry
100.
101. fn_read_sectors_into_memory_ok:
102.     stc
103.
104. fn_read_sectors_into_memory_end:
105.     pop di
106.     ret
```

事先已经将1#扇区填充成了特别的内容, 这样int 13h ah=42h执行完后，就可以通过查看0x7e00的内存，观察，是否已经拷贝成功.

```
[plain]

01.  Offset       0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
02.
```

IDA (3)
packer (2)
Crypt (2)

文章存档

2016年12月 (11)
2016年11月 (18)
2016年10月 (24)
2016年09月 (35)
2016年08月 (17)

展开

```
000000200   20 00 00 00 00 00 00 00  00 00 00 00 00 00 00 20
000000210   21 00 00 00 00 00 00 00  00 00 00 00 00 00 00 21   !               !
000000220   22 00 00 00 00 00 00 00  00 00 00 00 00 00 00 22   "               "
000000230   23 00 00 00 00 00 00 00  00 00 00 00 00 00 00 23   #               #
000000240   24 00 00 00 00 00 00 00  00 00 00 00 00 00 00 24   $               $
```

bochs调试过程记录:

```
[plain]

Next at t=0
(0) [0x0000fffffff0] f000:fff0 (unk. ctxt): jmpf 0xf000:e05b          ; ea5be000f0
<bochs:1> pb 0x7c00
<bochs:2> c
(0) Breakpoint 1, 0x0000000000007c00 in ?? ()
Next at t=156816100
(0) [0x000000007c00] 0000:7c00 (unk. ctxt): cli                       ; fa
<bochs:3> s
...
<bochs:53> s
Next at t=156825905
(0) [0x000000007ce0] 0000:7ce0 (unk. ctxt): pusha                     ; 60
<bochs:54> s
Next at t=156825906
(0) [0x000000007ce1] 0000:7ce1 (unk. ctxt): xor cx, cx                ; 31c9
<bochs:55> s
Next at t=156825907
(0) [0x000000007ce3] 0000:7ce3 (unk. ctxt): push cx                   ; 51
<bochs:56> s
Next at t=156825908
(0) [0x000000007ce4] 0000:7ce4 (unk. ctxt): push cx                   ; 51
<bochs:57> s
Next at t=156825909
(0) [0x000000007ce5] 0000:7ce5 (unk. ctxt): push dx                   ; 52
<bochs:58> s
Next at t=156825910
(0) [0x000000007ce6] 0000:7ce6 (unk. ctxt): push ax                   ; 50
<bochs:59> s
Next at t=156825911
(0) [0x000000007ce7] 0000:7ce7 (unk. ctxt): push cx                   ; 51
<bochs:60> s
Next at t=156825912
(0) [0x000000007ce8] 0000:7ce8 (unk. ctxt): push bx                   ; 53
<bochs:61> s
Next at t=156825913
```

```
(0) [0x000000007ce9] 0000:7ce9 (unk. ctxt): inc cx                    ; 41
<bochs:62> s
Next at t=156825914
(0) [0x000000007cea] 0000:7cea (unk. ctxt): push cx                   ; 51
<bochs:63> s
Next at t=156825915
(0) [0x000000007ceb] 0000:7ceb (unk. ctxt): push 0x0010               ; 6a10
<bochs:64> s
Next at t=156825916
(0) [0x000000007ced] 0000:7ced (unk. ctxt): mov si, sp                ; 89e6
<bochs:65> s
Next at t=156825917
(0) [0x000000007cef] 0000:7cef (unk. ctxt): test byte ptr ds:0x7d33, 0x01 ; f606337d01
<bochs:66> r
rax: 00000000_00000001 rcx: 00000000_00090001
rdx: 00000000_00000000 rbx: 00000000_00007e00
rsp: 00000000_00007bdc rbp: 00000000_00007dbe
rsi: 00000000_000e7bdc rdi: 00000000_00000005
r8 : 00000000_00000000 r9 : 00000000_00000000
r10: 00000000_00000000 r11: 00000000_00000000
r12: 00000000_00000000 r13: 00000000_00000000
r14: 00000000_00000000 r15: 00000000_00000000
rip: 00000000_00007cef
eflags 0x00000202: id vip vif ac vm rf nt IOPL=0 of df IF tf sf zf af pf cf
<bochs:67> xp /1bx 0x7d33
[bochs]:
0x0000000000007d33 <bogus+       0>:  0x01
<bochs:68> s
Next at t=156825918
(0) [0x000000007cf4] 0000:7cf4 (unk. ctxt): jz .+8 (0x00007cfe)       ; 7408
<bochs:69> s
Next at t=156825919
(0) [0x000000007cf6] 0000:7cf6 (unk. ctxt): mov ah, 0x42              ; b442
<bochs:70> s
Next at t=156825920
(0) [0x000000007cf8] 0000:7cf8 (unk. ctxt): mov dl, byte ptr ds:0x7d34 ; 8a16347d
<bochs:71> s
Next at t=156825921
(0) [0x000000007cfc] 0000:7cfc (unk. ctxt): jmp .+27 (0x00007d19)     ; eb1b
<bochs:72> r
rax: 00000000_00004201 rcx: 00000000_00090001
rdx: 00000000_00000080 rbx: 00000000_00007e00
rsp: 00000000_00007bdc rbp: 00000000_00007dbe
rsi: 00000000_000e7bdc rdi: 00000000_00000005
r8 : 00000000_00000000 r9 : 00000000_00000000
r10: 00000000_00000000 r11: 00000000_00000000
r12: 00000000_00000000 r13: 00000000_00000000
r14: 00000000_00000000 r15: 00000000_00000000
```

```
rip: 00000000_00007cfc
eflags 0x00000202: id vip vif ac vm rf nt IOPL=0 of df IF tf sf zf af pf cf
<bochs:73> u
00007cfc: (                    ): jmp .+27                 ; eb1b
<bochs:74> s
Next at t=156825922
(0) [0x000000007d19] 0000:7d19 (unk. ctxt): int 0x13                 ; cd13
<bochs:75> u 0x7d19 0x7d29
00007d19: (                    ): int 0x13                 ; cd13
00007d1b: (                    ): lea sp, word ptr ds:[si+16] ; 8d6410
00007d1e: (                    ): popa                     ; 61
00007d1f: (                    ): jnb .+16                 ; 7310
00007d21: (                    ): dec di                   ; 4f
00007d22: (                    ): jz .+12                  ; 740c
00007d24: (                    ): pusha                    ; 60
00007d25: (                    ): xor ah, ah               ; 30e4
00007d27: (                    ): mov dl, byte ptr ds:0x7d34 ; 8a16347d
<bochs:77> pb 0x7d1b
<bochs:78> r
rax: 00000000_00004201 rcx: 00000000_00090001
rdx: 00000000_00000080 rbx: 00000000_00007e00
rsp: 00000000_00007bdc rbp: 00000000_00007dbe
rsi: 00000000_000e7bdc rdi: 00000000_00000005
r8 : 00000000_00000000 r9 : 00000000_00000000
r10: 00000000_00000000 r11: 00000000_00000000
r12: 00000000_00000000 r13: 00000000_00000000
r14: 00000000_00000000 r15: 00000000_00000000
rip: 00000000_00007d19
eflags 0x00000202: id vip vif ac vm rf nt IOPL=0 of df IF tf sf zf af pf cf
<bochs:79> c
(0) Breakpoint 5, 0x0000000000007d1b in ?? ()
Next at t=156827497
(0) [0x000000007d1b] 0000:7d1b (unk. ctxt): lea sp, word ptr ds:[si+16] ; 8d6410
<bochs:80> xp /16bx si
[bochs]:
0x0000000000007bdc <bogus+       0>:  0x10    0x00    0x01    0x00    0x00    0x7e    0x00    0x00
0x0000000000007be4 <bogus+       8>:  0x01    0x00    0x00    0x00    0x00    0x00    0x00    0x00
<bochs:81> r
rax: 00000000_00000001 rcx: 00000000_00090001
rdx: 00000000_00000080 rbx: 00000000_00007e00
rsp: 00000000_00007bdc rbp: 00000000_00007dbe
rsi: 00000000_000e7bdc rdi: 00000000_00000005
r8 : 00000000_00000000 r9 : 00000000_00000000
r10: 00000000_00000000 r11: 00000000_00000000
r12: 00000000_00000000 r13: 00000000_00000000
r14: 00000000_00000000 r15: 00000000_00000000
rip: 00000000_00007d1b
eflags 0x00000202: id vip vif ac vm rf nt IOPL=0 of df IF tf sf zf af pf cf
```

```
<bochs:82> xp /32bx 0x7e00
[bochs]:
0x0000000000007e00 <bogus+       0>:  0x20    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x0000000000007e08 <bogus+       8>:  0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x20
0x0000000000007e10 <bogus+      16>:  0x21    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x0000000000007e18 <bogus+      24>:  0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x21
<bochs:83> q
(0).[156827497] [0x000000007d1b] 0000:7d1b (unk. ctxt): lea sp, word ptr ds:[si+16] ; 8d6410
```

通过和事先在1#扇区填入的内容比较，可以看到：已经将1#扇区的内容读入了0x7e00开始的512字节空间.

如果还有可用的代码空间，可以考虑将 DS:SI -> disk address packet 指向数据区定义的 disk address packet 结构, 可以提高清晰度.

毕竟用栈传递参数时的参数填写，不是很直观.

逆向的这个MBR编译完, 还有48字节的空间, 可以考虑使用预先定义的结构体来传递disk address packet参数.

看UltraIso制作的MBR, 里面有动态替换代码的实现，更像是手工patch第三方的MBR后产生的成果.

- 上一篇 nasm : 使用带参数的宏实现逻辑控制
- 下一篇 nasm : UltraIso制作的MBR的逆向整理

我的同类文章

NASM (29)

- nasm : 做自己的MBR 从定制的扇区索... 2015-10-06 阅读 316
- (ZT) CHS conversion 2015-10-04 阅读 135
- nasm : 使用带参数的宏实现逻辑控制 2015-10-03 阅读 308
- nasm : test instruction - test 2015-10-01 阅读 124
- nasm : dynamically modify bin nasm... 2015-09-30 阅读 153
- (ZT)算术移位和逻辑移位的区别 2015-10-04 阅读 156
- nasm : UltraIso制作的MBR的逆向整理 2015-10-03 阅读 222
- (ZT)在int13h中使用es:bx的例子 2015-10-02 阅读 292
- nasm : test jmp instruction 2015-09-30 阅读 198
- (ZT)PC BIOS Code and Data Layout 2015-09-29 阅读 148

更多文章

猜你在找

C语言系列之 数据结构栈的运用
div+css视频教程
DIV+CSS布局
C语言指针与汇编内存地址
ArcGIS for javascript 项目实…
int 13h 参数大全
int 13H参数详解
INT21H的0Ah号功能实现原理
汇编原理3分析ah4ah时的int 21h
BIOS中断大全INT 14H INT 1AH

查看评论

暂无评论

您还没有登录,请[登录]或[注册]

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场