

talk is cheap;Later equals never;

目录视图

摘要视图

RSS 订阅

资料连接

android-blog-dacainiao

Android Os code online view

Lua5.3ReferenceManual

Lua程序设计

Ralf Brown's Interrupt List

MBR

sourceforge

github search

csdn search

wiki-aes

catch22

wtl yakergong

wikispaces

viksoe.dk

zh-google-styleguide

vckbase

my cnblog

文章搜索

文章分类

- 常用 (13)
- asm (74)
- 编译器代码定式 (1)
- NASM (30)
- C (73)
- c++ (236)
- InstallShield (17)
- Linux (9)

EMUI5.0技术开放日成都站

Swift 问题与解答

免费的知识库，你的知识库

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

2015-10-03 14:49

399人阅读

评论(0)

收藏

举报

分类：

NASM (29)

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]
01. ; ===== S U B R O U T I N E =====
02.
03. ; /// @fn fn_read_sectors_into_memory
04. ; /// @brief 读扇区到内存
05. ; /// @param bp 被激活的分区表入口地址
06. ; /// @param cx 要拷贝的扇区数量
07. ; /// @param dx 激活的分区表入口中指定第一个扇区索引的高8位
08. ; /// @param ax 激活的分区表入口中指定第一个扇区索引的低8位
09. ; /// @param bx 扇区读取后, 要拷贝到目的内存地址
10.
11. fn_read_sectors_into_memory:
12.     push di
13.     mov di, 5 ; ///< 读取扇区到内存失败时的重试次数
14.
15. fn_read_sectors_into_memory_retry:
16.     pusha
17.
18. ; Format of disk address packet:
19.
20. ;Offset Size Description (Table 00272)
```

UI (31)

windows driver (43)

实验 (82)

工具 (32)

数据结构 (58)

编程基本操作整理 (35)

编程资料 (26)

调试 (73)

软件工程 (29)

STL (11)

字符串 (11)

IDE (18)

DLL (13)

硬件 (2)

XML (4)

代码格式化 (2)

服务 (6)

算法 (22)

SlickEdit (2)

计算机维护 (58)

Life (4)

编程资源 (8)

PE (16)

python (1)

IdaPython (6)

WDK FAQ (1)

进程 (7)

逆向 (21)

file (18)

COM (19)

Class (1)

MyDemo (1)

Web browser (17)

```
21. ;10h QWORD (EDD-3.0, optional) 64-bit flat address of transfer buffer;
22. ;
23. ;08h QWORD starting absolute block number
24. ; (for non-LBA devices, compute as
25. ; (Cylinder*NumHeads + SelectedHead) * SectorPerTrack +
26. ; SelectedSector - 1
27. ;04h DWORD -> transfer buffer
28. ;02h WORD number of blocks to transfer (max 007Fh for Phoenix EDD)
29. ;01h BYTE reserved (0);
30. ;00h BYTE size of packet (10h or 18h)
31.
32. ; /// 压入 disk address packet, 并将栈顶给si, 供 int 13h, ah = 0x42 使用
33. ; /// 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
34.
35. xor cx, cx
36.
37. ; /// 开始的绝对块号 = 1
38. ; /// 此时 cx == 0, dx:ax = 0000,0001
39. push cx ;08h QWORD starting absolute block number
40. push cx
41. push dx
42. push ax
43.
44. ; /// 此时, cx:bx = 0000,7e00
45. push cx ;04h DWORD -> transfer buffer
46. push bx
47.
48. ; /// 此时 cx = 0
49. inc cx ; ///< 此时, cx = 1, 说明要传送的块数为1
50. push cx ;02h WORD number of blocks to transfer (max 007Fh for Phoenix EDD)
51.
52. push 0010h ;00h BYTE size of packet (10h or 18h)
53. ;01h BYTE reserved (0);
54.
55. ; /// DS:SI -> disk address packet
56. mov si, sp
57.
58. ; /// 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
59. test byte[byte_status], BIT_VALID_PARTITION_SET
60. jz label_read_disk_to_memory_by_chs ; ///< 不是TRUE(分区校验无效), 转 int13h, ah = 2 读扇区
61.
62. ; /// 分区表有效的处理
63. ; /// 使用 int13h, ah = 0x42 读扇区
64. mov ah, 0x42
65. mov dl, [byte_udisk_sn] ; DL = drive
66. ; /// 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
67. jmp label_read_sectors_into_memory
68.
```

- 宏 (9)
- Socket (12)
- windbg (6)
- 锁 (3)
- GDI (3)
- message (2)
- callback (2)
- DuiLib (36)
- time (2)
- zip (3)
- 编译错误 (4)
- cUrl (3)
- svn (4)
- WTL (4)
- json (1)
- 注册表 (1)
- Design Patterns (3)
- refactor (1)
- 美工资源 (1)
- qt (1)
- osx (4)
- MFC (24)
- 窗体 (4)
- nankai100 (4)
- template (17)
- 运算符重载 (1)
- math (3)
- 调用约定 (1)
- SDK (12)
- 菜单 (0)
- thread (3)
- Android (30)
- lua (4)

```
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

```
03. 000000200 20 00 00 00 00 00 00 00 00 00 00 00 00 00 20
04. 000000210 21 00 00 00 00 00 00 00 00 00 00 00 00 21 !
05. 000000220 22 00 00 00 00 00 00 00 00 00 00 00 00 22 "
06. 000000230 23 00 00 00 00 00 00 00 00 00 00 00 00 23 #
07. 000000240 24 00 00 00 00 00 00 00 00 00 00 00 00 24 $
```

bochs调试过程记录:

```
[plain]
01. Next at t=0
02. (0) [0x0000ffffff0] f000:fff0 (unk. ctxt): jmpf 0xf000:e05b ; ea5be00f0
03. <bochs:1> pb 0x7c00
04. <bochs:2> c
05. (0) Breakpoint 1, 0x0000000000007c00 in ?? ()
06. Next at t=156816100
07. (0) [0x000000007c00] 0000:7c00 (unk. ctxt): cli ; fa
08. <bochs:3> s
09. ...
10. <bochs:53> s
11. Next at t=156825905
12. (0) [0x000000007ce0] 0000:7ce0 (unk. ctxt): pusha ; 60
13. <bochs:54> s
14. Next at t=156825906
15. (0) [0x000000007ce1] 0000:7ce1 (unk. ctxt): xor cx, cx ; 31c9
16. <bochs:55> s
17. Next at t=156825907
18. (0) [0x000000007ce3] 0000:7ce3 (unk. ctxt): push cx ; 51
19. <bochs:56> s
20. Next at t=156825908
21. (0) [0x000000007ce4] 0000:7ce4 (unk. ctxt): push cx ; 51
22. <bochs:57> s
23. Next at t=156825909
24. (0) [0x000000007ce5] 0000:7ce5 (unk. ctxt): push dx ; 52
25. <bochs:58> s
26. Next at t=156825910
27. (0) [0x000000007ce6] 0000:7ce6 (unk. ctxt): push ax ; 50
28. <bochs:59> s
29. Next at t=156825911
30. (0) [0x000000007ce7] 0000:7ce7 (unk. ctxt): push cx ; 51
31. <bochs:60> s
32. Next at t=156825912
33. (0) [0x000000007ce8] 0000:7ce8 (unk. ctxt): push bx ; 53
34. <bochs:61> s
35. Next at t=156825913
```

```

36. (0) [0x000000007ce9] 0000:7ce9 (unk. ctxt): inc cx ; 41
37. <bochs:62> s
38. Next at t=156825914
39. (0) [0x000000007cea] 0000:7cea (unk. ctxt): push cx ; 51
40. <bochs:63> s
41. Next at t=156825915
42. (0) [0x000000007ceb] 0000:7ceb (unk. ctxt): push 0x0010 ; 6a10
43. <bochs:64> s
44. Next at t=156825916
45. (0) [0x000000007ced] 0000:7ced (unk. ctxt): mov si, sp ; 89e6
46. <bochs:65> s
47. Next at t=156825917
48. (0) [0x000000007cef] 0000:7cef (unk. ctxt): test byte ptr ds:0x7d33, 0x01 ; f606337d01
49. <bochs:66> r
50. rax: 00000000_00000001 rcx: 00000000_00090001
51. rdx: 00000000_00000000 rbx: 00000000_00007e00
52. rsp: 00000000_00007bdc rbp: 00000000_00007dbe
53. rsi: 00000000_000e7bdc rdi: 00000000_00000005
54. r8 : 00000000_00000000 r9 : 00000000_00000000
55. r10: 00000000_00000000 r11: 00000000_00000000
56. r12: 00000000_00000000 r13: 00000000_00000000
57. r14: 00000000_00000000 r15: 00000000_00000000
58. rip: 00000000_00007cef
59. eflags 0x00000202: id vip vif ac vm rf nt IOPL=0 of df IF tf sf zf af pf cf
60. <bochs:67> xp /1bx 0x7d33
61. [bochs]:
62. 0x0000000000007d33 <bogus+ 0>: 0x01
63. <bochs:68> s
64. Next at t=156825918
65. (0) [0x000000007cf4] 0000:7cf4 (unk. ctxt): jz .+8 (0x00007cfe) ; 7408
66. <bochs:69> s
67. Next at t=156825919
68. (0) [0x000000007cf6] 0000:7cf6 (unk. ctxt): mov ah, 0x42 ; b442
69. <bochs:70> s
70. Next at t=156825920
71. (0) [0x000000007cf8] 0000:7cf8 (unk. ctxt): mov dl, byte ptr ds:0x7d34 ; 8a16347d
72. <bochs:71> s
73. Next at t=156825921
74. (0) [0x000000007cfc] 0000:7cfc (unk. ctxt): jmp .+27 (0x00007d19) ; eb1b
75. <bochs:72> r
76. rax: 00000000_00004201 rcx: 00000000_00090001
77. rdx: 00000000_00000080 rbx: 00000000_00007e00
78. rsp: 00000000_00007bdc rbp: 00000000_00007dbe
79. rsi: 00000000_000e7bdc rdi: 00000000_00000005
80. r8 : 00000000_00000000 r9 : 00000000_00000000
81. r10: 00000000_00000000 r11: 00000000_00000000
82. r12: 00000000_00000000 r13: 00000000_00000000
83. r14: 00000000_00000000 r15: 00000000_00000000

```

```

84. rip: 00000000_00007cfc
85. eflags 0x00000202: id vip vif ac vm rf nt IOPL=0 of df IF tf sf zf af pf cf
86. <bochs:73> u
87. 00007cfc: ( ) : jmp .+27 ; eb1b
88. <bochs:74> s
89. Next at t=156825922
90. (0) [0x000000007d19] 0000:7d19 (unk. ctxt): int 0x13 ; cd13
91. <bochs:75> u 0x7d19 0x7d29
92. 00007d19: ( ) : int 0x13 ; cd13
93. 00007d1b: ( ) : lea sp, word ptr ds:[si+16] ; 8d6410
94. 00007d1e: ( ) : popa ; 61
95. 00007d1f: ( ) : jnb .+16 ; 7310
96. 00007d21: ( ) : dec di ; 4f
97. 00007d22: ( ) : jz .+12 ; 740c
98. 00007d24: ( ) : pusha ; 60
99. 00007d25: ( ) : xor ah, ah ; 30e4
100. 00007d27: ( ) : mov dl, byte ptr ds:0x7d34 ; 8a16347d
101. <bochs:77> pb 0x7d1b
102. <bochs:78> r
103. rax: 00000000_00004201 rcx: 00000000_00090001
104. rdx: 00000000_00000080 rbx: 00000000_00007e00
105. rsp: 00000000_00007bdc rbp: 00000000_00007dbe
106. rsi: 00000000_000e7bdc rdi: 00000000_00000005
107. r8 : 00000000_00000000 r9 : 00000000_00000000
108. r10: 00000000_00000000 r11: 00000000_00000000
109. r12: 00000000_00000000 r13: 00000000_00000000
110. r14: 00000000_00000000 r15: 00000000_00000000
111. rip: 00000000_00007d19
112. eflags 0x00000202: id vip vif ac vm rf nt IOPL=0 of df IF tf sf zf af pf cf
113. <bochs:79> c
114. (0) Breakpoint 5, 0x0000000000007d1b in ?? ()
115. Next at t=156827497
116. (0) [0x000000007d1b] 0000:7d1b (unk. ctxt): lea sp, word ptr ds:[si+16] ; 8d6410
117. <bochs:80> xp /16bx si
118. [bochs]:
119. 0x0000000000007bdc <bogus+ 0>: 0x10 0x00 0x01 0x00 0x00 0x00 0x7e 0x00 0x00
120. 0x0000000000007be4 <bogus+ 8>: 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
121. <bochs:81> r
122. rax: 00000000_00000001 rcx: 00000000_00090001
123. rdx: 00000000_00000080 rbx: 00000000_00007e00
124. rsp: 00000000_00007bdc rbp: 00000000_00007dbe
125. rsi: 00000000_000e7bdc rdi: 00000000_00000005
126. r8 : 00000000_00000000 r9 : 00000000_00000000
127. r10: 00000000_00000000 r11: 00000000_00000000
128. r12: 00000000_00000000 r13: 00000000_00000000
129. r14: 00000000_00000000 r15: 00000000_00000000
130. rip: 00000000_00007d1b
131. eflags 0x00000202: id vip vif ac vm rf nt IOPL=0 of df IF tf sf zf af pf cf

```

```
132. <bochs:82> xp /32bx 0x7e00
133. [bochs]:
134. 0x00000000000007e00 <bogus+      0>: 0x20    0x00    0x00    0x00    0x00    0x00    0x00    0x00
135. 0x00000000000007e08 <bogus+      8>: 0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x20
136. 0x00000000000007e10 <bogus+     16>: 0x21    0x00    0x00    0x00    0x00    0x00    0x00    0x00
137. 0x00000000000007e18 <bogus+     24>: 0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x21
138. <bochs:83> q
139. (0).[156827497] [0x0000000007d1b] 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。

顶 踩
0 0

- 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 | int 13h 参数大全 |
| div+css视频教程 | int 13H参数详解 |
| DIV+CSS布局 | INT21H的0Ah号功能实现原理 |
| C语言指针与汇编内存地址 | 汇编原理3分析ah4ah时的int 21h |
| ArcGIS for javascript 项目实... | 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

http://blog.csdn.net/LostSpeed/article/details/48878139

8/9

核心技术类目

| | | | | | | | | | | | | | |
|------------|---------------|-----------|------------|---------|-----------|---------|-----------|-----------|----------|------------|----------------|--------|------|
| 全部主题 | Hadoop | AWS | 移动游戏 | Java | Android | iOS | Swift | 智能硬件 | Docker | OpenStack | VPN | Spark | ERP |
| IE10 | Eclipse | CRM | JavaScript | 数据库 | Ubuntu | NFC | WAP | jQuery | BI | HTML5 | Spring | Apache | .NET |
| API | HTML | SDK | IIS | Fedora | XML | LBS | Unity | Splashtop | UML | components | Windows Mobile | Rails | |
| QEMU | KDE | Cassandra | CloudStack | FTC | coremail | OPhone | CouchBase | 云计算 | iOS6 | Rackspace | Web App | | |
| SpringSide | Maemo | Compuware | 大数据 | apttech | Perl | Tornado | Ruby | Hibernate | ThinkPHP | HBase | Pure | Solr | |
| Angular | Cloud Foundry | Redis | Scala | Django | Bootstrap | | | | | | | | |

公司简介 | 招贤纳士 | 广告服务 | 联系方式 | 版权声明 | 法律顾问 | 问题报告 | 合作伙伴 | 论坛反馈

网站客服 杂志客服 微博客服 webmaster@csdn.net 400-600-2320 | 北京创新乐知信息技术有限公司 版权所有 | 江苏知之为计算机有限公司 | 江苏乐知网络技术有限公司

京 ICP 证 09002463 号 | Copyright © 1999-2016, CSDN.NET, All Rights Reserved 

0