快速通道

文档分类
- 红盟公告
- 新手上路
- 漏洞分析
- 工具介绍
- 安全配置
- 编程技术
- **安全技术** <=
- 业界动态

本类热门
- 如何利用黑客技术跟踪并分析一名目标人物
- 有关无线安全与黑客的话题
- 让我们回归(by 默难)
- 通用ShellCode深入剖析[转]
- Format String 漏洞介绍/总结(一)
- 结合实例解读ELF文件-阅读笔记(1)

结合实例解读ELF文件-阅读笔记(2)

提交文章： bkbll
提交日期： 2003-11-12
文章属性： 原创
文档类别： 安全技术
浏览次数： 今3次/总3188次

---

结合实例解读ELF文件(二)
bkbll(bkbll@cnhonker.net, kbll@tom.com)
2003/09/09
3. section
同样的, 我们先给出section的结构:

```
typedef struct
{
Elf32_Word    sh_name;       /* 在section string table中的索引  */
Elf32_Word    sh_type;       /* Section type */
Elf32_Word    sh_flags;      /* Section flags */
Elf32_Addr    sh_addr;       /* Section virtual addr at execution */
Elf32_Off     sh_offset;     /* Section file offset */
Elf32_Word    sh_size;       /* Section size in bytes */
Elf32_Word    sh_link;       /* Link to another section */
Elf32_Word    sh_info;       /* Additional section information */
Elf32_Word    sh_addralign;      /* Section alignment */
Elf32_Word    sh_entsize;        /* Entry size if section holds table */
} Elf32_Shdr;
```

计算一下大小是sizeof(Elf32_Shdr)=0x28 和前面elf header的e_shentsize大小刚好一样.
从elf header的e_shoff 值我们可以得到第一个section在文件中的位置/偏移量0x2178,
从e_shnum可以知道一共有0x22个section.用C语言的算法可以这样表示:

```
fseek(FP, 0x2178, 0);
fread(buffer,0x22,0x28,FP);
```

就可以把所有的section读入到buffer里面.
我们来对照文件结构看看一个section的含义(第4个section(注意:从第0个开始)在文件中的的偏移量应该是: 0x2178+4*0x28=0x2218,我们以第4个section为例子):

```
[netconf@linux1 elf]$ hexdump -s 0x2218 -n 40 -C elf8
00002218 37 00 00 00 0b 00 00 00 02 00 00 00 50 81 04 08 |7...........P...|
00002228 50 01 00 00 50 00 00 00 05 00 00 00 01 00 00 00 |P...P...........|
00002238 04 00 00 00 10 00 00 00 |........|
```

其中:
sh_name: 4字节: 37 00 00 00 表示该section的名称在section string table中的索引.
sh_type: 4字节: 0b 00 00 00 表示该section的类型, 从elf.h中可以看到定义0x0b=11的是:SHT_DYNSYM  11  /* Dynamic linker symbol table */ 表示是一个动态连接符号表.
sh_flags: 4字节: 02 00 00 00 表示该section的类型.0x02是一个ALLOC型的section.
sh_addr: 4字节: 50 81 04 08 执行的时候, 该section的虚拟地址0x08048150
sh_offset: 4字节: 50 01 00 00 表示该section内容在文件的位置/偏移量0x150
sh_size: 4字节: 50 00 00 00 section大小0x50
sh_link: 4字节  01 00 00 00 连接到其他section
sh_info: 4字节: 04 00 00 00 section的其他信息

sh_addralign: 4字节: 04 00 00 00 section的对齐调整值
sh_entsize: 4字节: 10 00 00 00 如果该section定义的是一个表,那么这个值表示该表里面每个结构的大小.

注意: sh_link和sh_info的值随着sh_type不同有不同的含义.
sh_type sh_link sh_info
======= ======= =======
SHT_DYNAMIC The section header index of 0
the string table used by
entries in the section.
SHT_HASH The section header index of 0
the symbol table to which the
hash table applies.
SHT_REL, The section header index of The section header index of
SHT_RELA the associated symbol table. the section to which the
relocation applies.
SHT_SYMTAB, The section header index of One greater than the symbol
SHT_DYNSYM the associated string table. table index of the last local
symbol (binding STB_LOCAL).
other SHN_UNDEF 0

section表是elf文件中最重要的一环, 基本上所有的东东都要查找它.
比如.dynsym, .symtab, .got, .bss等等. 后面我们就要提到的symbol表也要从这里获取相关信息.
我们来看一下elf header里面提到的e_shstrndx, 这个值定义了section string table的位置在第几个section里面的, 这里我们可以计算一下它的位置:
e_shoff+e_shstrndx* e_shentsize (起始位置+序号*每个的大小)
=0x2178+31*0x28= 0x2650
我们看看这个section的内容:
[netconf@linux1 elf]$ hexdump -s 0x2650 -n 40 -C elf8
00002650 11 00 00 00 03 00 00 00 00 00 00 00 00 00 00 00 |................|
00002660 4a 20 00 00 2b 01 00 00 00 00 00 00 00 00 00 00 |J ..+...........|
00002670 01 00 00 00 00 00 00 00 |........|
这里面可以给我们的信息是:
sh_name=0x11, sh_type=0x03,sh_offset=0x204a, sh_size=0x12b
sh_name目前没用.
Sh_type:0x03表示: SHT_STRTAB 3 /* String table */
Ok,我们找到了string table, 让我们看看string table里面有什么吧:
[netconf@linux1 elf]$ hexdump -s 0x204a -n 299 -C elf8
0000204a 00 2e 73 79 6d 74 61 62 00 2e 73 74 72 74 61 62 |..symtab..strtab|
0000205a 00 2e 73 68 73 74 72 74 61 62 00 2e 69 6e 74 65 |..shstrtab..inte|
0000206a 72 70 00 2e 6e 6f 74 65 2e 41 42 49 2d 74 61 67 |rp..note.ABI-tag|
0000207a 00 2e 68 61 73 68 00 2e 64 79 6e 73 79 6d 00 2e |..hash..dynsym..|
0000208a 64 79 6e 73 74 72 00 2e 67 6e 75 2e 76 65 72 73 |dynstr..gnu.vers|
0000209a 69 6f 6e 00 2e 67 6e 75 2e 76 65 72 73 69 6f 6e |ion..gnu.version|
000020aa 5f 72 00 2e 72 65 6c 2e 64 79 6e 00 2e 72 65 6c |_r..rel.dyn..rel|
000020ba 2e 70 6c 74 00 2e 69 6e 69 74 00 2e 74 65 78 74 |.plt..init..text|
000020ca 00 2e 66 69 6e 69 00 2e 72 6f 64 61 74 61 00 2e |..fini..rodata..|
000020da 64 61 74 61 00 2e 65 68 5f 66 72 61 6d 65 00 2e |data..eh_frame..|
000020ea 64 79 6e 61 6d 69 63 00 2e 63 74 6f 72 73 00 2e |dynamic..ctors..|
000020fa 64 74 6f 72 73 00 2e 6a 63 72 00 2e 67 6f 74 00 |dtors..jcr..got.|
0000210a 2e 62 73 73 00 2e 63 6f 6d 6d 65 6e 74 00 2e 64 |.bss..comment..d|
0000211a 65 62 75 67 5f 61 72 61 6e 67 65 73 00 2e 64 65 |ebug_aranges..de|
0000212a 62 75 67 5f 70 75 62 6e 61 6d 65 73 00 2e 64 65 |bug_pubnames..de|
0000213a 62 75 67 5f 69 6e 66 6f 00 2e 64 65 62 75 67 5f |bug_info..debug_|
0000214a 61 62 62 72 65 76 00 2e 64 65 62 75 67 5f 6c 69 |abbrev..debug_li|
0000215a 6e 65 00 2e 64 65 62 75 67 5f 66 72 61 6d 65 00 |ne..debug_frame.|
0000216a 2e 64 65 62 75 67 5f 73 74 72 00 |.debug_str.|

这里就是我们要找的东西了. 通过sh_name的值, 可以在这个string table里面找到自己对应的section name.
比如我们刚才看的第4个section的sh_name是0x37,我们找string table里面偏移量是0x37的内容(到0x00结束).
我们来看看：字符串起始位置0x204a+0x37=0x2081
就先读出10字节内容吧：
[netconf@linux1 elf]$ hexdump -s 0x2081 -n 10 -C elf8
00002081 2e 64 79 6e 73 79 6d 00 2e 64 |.dynsym..d|
0x00前面就是.dynsym了， 这个就是我们要找的section name了.
通过前面的过程我们就可以找到一一对应的section名字, 来看看readelf的结果:
[netconf@linux1 elf]$ readelf -S elf8
There are 34 section headers, starting at offset 0x2178:

```
Section Headers:
[Nr] Name Type Addr Off Size ES Flg Lk Inf Al
[ 0] NULL 00000000 000000 000000 00 0 0 0
[ 1] .interp PROGBITS 080480f4 0000f4 000013 00 A 0 0 1
[ 2] .note.ABI-tag NOTE 08048108 000108 000020 00 A 0 0 4
[ 3] .hash HASH 08048128 000128 000028 04 A 4 0 4
[ 4] .dynsym DYNSYM 08048150 000150 000050 10 A 5 1 4
[ 5] .dynstr STRTAB 080481a0 0001a0 00004c 00 A 0 0 1
[ 6] .gnu.version VERSYM 080481ec 0001ec 00000a 02 A 4 0 2
[ 7] .gnu.version_r VERNEED 080481f8 0001f8 000020 00 A 5 1 4
[ 8] .rel.dyn REL 08048218 000218 000008 08 A 4 0 4
[ 9] .rel.plt REL 08048220 000220 000010 08 A 4 b 4
[10] .init PROGBITS 08048230 000230 000018 00 AX 0 0 4
[11] .plt PROGBITS 08048248 000248 000030 04 AX 0 0 4
[12] .text PROGBITS 08048278 000278 000170 00 AX 0 0 4
[13] .fini PROGBITS 080483e8 0003e8 00001c 00 AX 0 0 4
[14] .rodata PROGBITS 08048404 000404 000050 00 A 0 0 4
[15] .data PROGBITS 08049454 000454 00000c 00 WA 0 0 4
[16] .eh_frame PROGBITS 08049460 000460 000004 00 WA 0 0 4
[17] .dynamic DYNAMIC 08049464 000464 0000c8 08 WA 5 0 4
[18] .ctors PROGBITS 0804952c 00052c 000008 00 WA 0 0 4
[19] .dtors PROGBITS 08049534 000534 000008 00 WA 0 0 4
[20] .jcr PROGBITS 0804953c 00053c 000004 00 WA 0 0 4
[21] .got PROGBITS 08049540 000540 000018 04 WA 0 0 4
[22] .bss NOBITS 08049558 000558 000004 00 WA 0 0 4
[23] .comment PROGBITS 00000000 000558 000132 00 0 0 1
[24] .debug_aranges PROGBITS 00000000 000690 000058 00 0 0 8
[25] .debug_pubnames PROGBITS 00000000 0006e8 000025 00 0 0 1
[26] .debug_info PROGBITS 00000000 00070d 000c85 00 0 0 1
[27] .debug_abbrev PROGBITS 00000000 001392 000127 00 0 0 1
[28] .debug_line PROGBITS 00000000 0014b9 0001f2 00 0 0 1
[29] .debug_frame PROGBITS 00000000 0016ac 000014 00 0 0 4
[30] .debug_str PROGBITS 00000000 0016c0 00098a 01 MS 0 0 1
[31] .shstrtab STRTAB 00000000 00204a 00012b 00 0 0 1
[32] .symtab SYMTAB 00000000 0026c8 0004c0 10 33 37 4
[33] .strtab STRTAB 00000000 002b88 0001dd 00 0 0 1
Key to Flags:
W (write), A (alloc), X (execute), M (merge), S (strings)
I (info), L (link order), G (group), x (unknown)
O (extra OS processing required) o (OS specific), p (processor specific)
[netconf@linux1 elf]$
```

4.Symbol

Symbol结构如下所示:

```
/* Symbol table entry. */

typedef struct
{
Elf32_Word    st_name;        /* Symbol name (string tbl index) */
Elf32_Addr    st_value;       /* Symbol value */
Elf32_Word    st_size;        /* Symbol size */
unsigned char    st_info;        /* Symbol type and binding */
unsigned char    st_other;       /* Symbol visibility */
Elf32_Section    st_shndx;       /* Section index */
} Elf32_Sym;
```

Symbol结构大小是: sizeof(Elf32_Sym)=0x10.

一个elf文件通常包含两个symbol表, 一个是.dynsym, 一个是.symtab, 前者表示程序运行时候需要重新定位/加载的符号(比如函数等), 后一个表示系统所有的符号列表。

该结构位置从上面的section表里面可以看出是: 0x26c8, 就是从文件0x26c8偏移初开始的.

我们读第68个结构看看(0x26c8+68*0x10=0x2b08):

```
[netconf@linux1 elf]$ hexdump -s 0x2b08 -n 16 -C ./elf8
00002b08 77 01 00 00 8e 83 04 08 1d 00 00 00 12 00 0c 00 |w...............|
00002b18
```

结合struct的定义,这样每个字节的内容和含义都可以分析出来.

st_name: 77 01 00 00 : 在symbol string table中的偏移量0x177

st_value: 8e 83 04 08 : 加载地址0x80438e8

st_size: 1d 00 00 00 : symbol大小

st_shndx: 0c 00 :如果st_shndx=SH_UNDEF, 那么表示该symbol不属于任何一个section, 需要重新定位. ” 当链接器将该目标文件和另一个定义该符号的文件相装配的时候，该文件内对该符号的引用将链接到当前实际的定义 ” (from alert7)

像第68个结构的st_shndx=12, 表示属于第12个section, 从前面列表可以知道, 是定义在.text段.

返回上页

Copyright © 2000-2004 **HUC** All Rights Reserved.
中国红客网络技术联盟 www.cnhonker.com
webmaster@cnhonker.net