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(1)

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1次/总3970次

结合实例解读ELF文件-阅读笔记(1)
bkbll(bkbll@cnhonker.net, kbll@tom.com)
2003/09/09
一. 预备知识
网上有很多文章讲叙了ELF文件的格式, 加载过程等, 其中我觉得比较实用的是这几篇:
1. breadbox 的< EXECUTABLE AND LINKABLE FORMAT (ELF)> 英文文档很多地方都有下载, alert7主页里面有中文翻译和英文原文.
英文原文: http://elfhack.whitecell.org/mydocs/elf.txt
alert7翻译整理的中文: http://elfhack.whitecell.org/mydocs/ELF_chinese.txt
2. alert7的<ELF动态解析符号过程(修订版)>
http://elfhack.whitecell.org/mydocs/ELF_symbol_resolve_process1.txt
但2只是阐述了一下动态解析符号过程, 1 是列举了一大堆的参数结构, 其中的联系, 结构等如果你开始看肯定会摸不着头脑, 我自己因为要写一个程序, 用到ELF文件的一些东西, 想想肯定还有很多人有和我一样的困惑, 本文权当做我写的笔记, 如果同时也能给你带来帮助, 就不枉此文了.
本文假设你已经看过了1或者2等文章, 熟悉linux系统, gdb等.
二. 分析平台
```
[netconf@linux1 elf]$ uname -a
Linux linux1 2.4.18-14 #1 Wed Sep 4 13:35:50 EDT 2002 i686 i686 i386 GNU/Linux
[netconf@linux1 elf]$ cat /proc/version
Linux version 2.4.18-14 (bhcompile@stripples.devel.redhat.com) (gcc version 3.2 20020903 (Red Hat Linux 8.0 3.2-7)) #1 Wed Sep 4 13:35:50 EDT 2002
[netconf@linux1 elf]$ rpm -qf /usr/bin/readelf /usr/bin/hexdump
binutils-2.13.90.0.2-2
util-linux-2.11r-10
[netconf@linux1 elf]$
```
三. 分析的例子/程序
```
[netconf@linux1 elf]$ cat elf8.c
#include <elf.h>
int foo1()
{
printf("[+] foo1 addr:%p\n",foo1);
foo2();
}
int foo2()
{
printf("[+] foo2 addr:%p\n",foo2);
foo3();
}
int foo3()
{
printf("[+] foo3 addr:%p\n",foo3);
```

```
foo4();
}
int foo4()
{
printf("[+] foo4 addr:%p\n",foo4);
}
main()
{
foo1();
}
[netconf@linux1 elf]$ gcc -o elf8 elf8.c
[netconf@linux1 elf]$ ./elf8
[+] foo1 addr:0x8048328
[+] foo2 addr:0x804834a
[+] foo3 addr:0x804836c
[+] foo4 addr:0x804838e
```

四.分析过程

1. ELF header

首先, elf文件的开头是一个Ehdr的结构. 该结构定义在/usr/include/elf.h中, 我们看看该结构:

```
typedef struct
{
unsigned char    e_ident[EI_NIDENT];    /* Magic number and other info */
Elf32_Half    e_type;            /* 目标文件类型  */
Elf32_Half    e_machine;        /* Architecture */
Elf32_Word    e_version;        /* Object file version */
Elf32_Addr    e_entry;        /* 入口地址  */
Elf32_Off    e_phoff;        /* Program header table文件偏移  */
Elf32_Off    e_shoff;        /* Section header table 文件偏移  */
Elf32_Word    e_flags;        /* Processor-specific flags */
Elf32_Half    e_ehsize;        /* ELF header 大小  */
Elf32_Half    e_phentsize;        /* 每个Program header大小  */
Elf32_Half    e_phnum;        /* 一共多少个Program header */
Elf32_Half    e_shentsize;        /* 每个Section header大小  */
Elf32_Half    e_shnum;        /* 一共多少个  Section header */
Elf32_Half    e_shstrndx;        /* Section的字符表在section header table的索引值  */
} Elf32_Ehdr;
```

除了Elf32_Half是2个字节(16位)外, 其他变量定义等都是4个字节(32位).

从上面的结构可以看出来sizeof(Elf32_Ehdr)=13*4=0x34

我们来看看文件elf8头52字节的内容:

```
[netconf@linux1 elf]$ hexdump -s 0 -n 52 -C elf8
00000000 7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00 |.ELF............|
00000010 02 00 03 00 01 00 00 00 78 82 04 08 34 00 00 00 |........x...4...|
00000020 78 21 00 00 00 00 00 00 34 00 20 00 06 00 28 00 |x!......4. ...(.|
00000030 22 00 1f 00                                     |"...|
```

ok, 我们对照这个结构一个个来分析:

e_ident[EI_NIDENT]: 16字节: 7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00

ELFMAG0 0x7f e_ident[0]

ELFMAG1 'E' e_ident[1]

ELFMAG2 'L' e_ident[2]

ELFMAG3 'F' e_ident[3]

ELFCLASS32 1 e_ident[4]

ELFDATA2LSB 1 e_ident[5]

EI_VERSION 1 e_ident[6]

剩下的全为0

e_type:2字节: 02 00 表示可执行文件(ET_EXEC 2 Executable file)

e_machine: 2字节: 03 00 表示386体系文件(EM_386 3 Intel 80386)

e_version: 4字节: 01 00 00 00 和e_ident里面的EI_VERSION含义一样.

e_entry: 4字节: 78 82 04 08 表示程序入口地址0x08048278

e_phoff: 4字节: 34 00 00 00 表示program head table在文件中的偏移量(开始位置)

e_shoff: 4字节: 78 21 00 00 表示section head table在文件中的偏移量(开始位置)

e_flags: 4字节: 00 00 00 00

e_ehsize: 2字节: 34 00 表示elf header大小, 其实就是sizeof(Elf32_Ehdr)

e_phentsize: 2字节: 20 00 表示每个program header 的大小(0x20)

e_phnum: 2字节: 06 00 表示一共多少个program header(0x06)

e_shentsize: 2字节: 28 00 表示每个section header的大小(0x28)

e_shnum: 2字节: 22 00, 表示有多少个section header

e_shstrndx: 2字节: 1f 00 表示section string table在section header table中的索引值.(即第几个section描述了section string table的位置和大小)

是不是很简单?我们来看看readelf的分析结果:

```
[netconf@linux1 elf]$ readelf -h elf8
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: EXEC (Executable file)
Machine: Intel 80386
Version: 0x1
Entry point address: 0x8048278
Start of program headers: 52 (bytes into file)
Start of section headers: 8568 (bytes into file)
Flags: 0x0
Size of this header: 52 (bytes)
Size of program headers: 32 (bytes)
Number of program headers: 6
Size of section headers: 40 (bytes)
Number of section headers: 34
Section header string table index: 31
```

2. Program header

除了开始的elf header位置是固定的外, 其他位置都是相关联的, 或者说都是要到elf header里面读取的.

看elf.h里面关于这个结构的定义:

```
typedef struct
{
Elf32_Word    p_type;           /* Segment type */
Elf32_Off     p_offset;        /* Segment file offset */
Elf32_Addr    p_vaddr;         /* Segment virtual address */
Elf32_Addr    p_paddr;         /* Segment physical address */
Elf32_Word    p_filesz;        /* Segment size in file */
Elf32_Word    p_memsz;         /* Segment size in memory */
Elf32_Word    p_flags;         /* Segment flags */
Elf32_Word    p_align;         /* Segment alignment */
} Elf32_Phdr;
```

我们来计算一下这个结构的大小, sizeof(Elf32_Phdr)=0x20, 从前面的elf header信息我们也同样可以知道这个长度是0x20, elf文件中一共有6个这样的Phdr, 最开始的一个Phdr在文件的0x34偏移处. 我们可以这样读取:

```
lseek(FP,0x34,0);
fread(buffer,6,0x20,FP);
```

将所有的Phdr都读入到buffer里面.

我们分析其中的一个结构:

```
[netconf@linux1 elf]$ hexdump -s 52 -n 32 -C elf8
00000034 06 00 00 00 34 00 00 00 34 80 04 08 34 80 04 08 |....4...4...4...|
00000044 c0 00 00 00 c0 00 00 00 05 00 00 00 04 00 00 00 |?..?.........|
```

按结构进行分析:

p_type: 4字节: 06 00 00 00, 表示该段是PT_PHDR类型(自己的入口)
p_offset: 4字节: 34 00 00 00 在文件中的偏移量.
p_vaddr: 4字节: 34 80 04 08 虚拟地址:0x08048034
p_paddr: 4字节: 34 80 04 08 物理地址: 0x08048034
p_filesz: 4字节: c0 00 00 00 段的大小:0xc0
p_memsz: 4字节: c0 00 00 00 在内存中的大小: 0xc0
p_flags: 4字节: 05 00 00 00 段标记
p_align: 4字节: 04 00 00 00

Phdr结构大概和上面差不多, 每一个结构都定义了该elf文件的特性以及每个segment的属性,大小等.

我们来看一看PT_INTERP段的信息:

```
[netconf@linux1 elf]$ hexdump -s 84 -n 32 -C elf8
00000054 03 00 00 00 f4 00 00 00 f4 80 04 08 f4 80 04 08 |....?..?..?..|
00000064 13 00 00 00 13 00 00 00 04 00 00 00 01 00 00 00 |................|
```

这个segment指出了程序依赖解释器的路径/文件名, 定义在0xf4的偏移量处, 大小是0x13字节.

```
[netconf@linux1 elf]$ hexdump -s 0xf4 -n 19 -C elf8
000000f4 2f 6c 69 62 2f 6c 64 2d 6c 69 6e 75 78 2e 73 6f |/lib/ld-linux.so|
00000104 2e 32 00 |.2.|
```

该elf文件依赖/lib/ld-linux.so.2来解释.

我们可以用readelf来检测看看:

```
[netconf@linux1 elf]$ readelf -l elf8

Elf file type is EXEC (Executable file)
Entry point 0x8048278
```

```
There are 6 program headers, starting at offset 52

Program Headers:
Type Offset VirtAddr PhysAddr FileSiz MemSiz Flg Align
PHDR 0x000034 0x08048034 0x08048034 0x000c0 0x000c0 R E 0x4
INTERP 0x0000f4 0x080480f4 0x080480f4 0x00013 0x00013 R 0x1
[Requesting program interpreter: /lib/ld-linux.so.2]
LOAD 0x000000 0x08048000 0x08048000 0x00454 0x00454 R E 0x1000
LOAD 0x000454 0x08049454 0x08049454 0x00104 0x00108 RW 0x1000
DYNAMIC 0x000464 0x08049464 0x08049464 0x000c8 0x000c8 RW 0x4
NOTE 0x000108 0x08048108 0x08048108 0x00020 0x00020 R 0x4

Section to Segment mapping:
Segment Sections...
00
01 .interp
02 .interp .note.ABI-
tag .hash .dynsym .dynstr .gnu.version .gnu.version_r .rel.dyn .rel.plt .init .plt .text .fini .rodata
03 .data .eh_frame .dynamic .ctors .dtors .jcr .got .bss
04 .dynamic
05 .note.ABI-tag
[netconf@linux1 elf]$
```

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