ELF 重定位分析

文/0pnlight

0x00 前言

利用重定位注入技术我们可以在 Linux 系统中向正在运行的应用程序增加一些额外功能而不会导致程序退出，如果使用动态链接器，可使用重定位在内存中打热补丁。如果恶意程序拥有重定位注入的能力，可能导致正常的二进制程序受到感染，对企业或个人造成损失。本文主要描述重定位的基本原理，对于注入过程将在后期逐步分析。

0x01 概述

在 Linux 系统下使用 gcc 编译程序时，c 源文件到可执行文件经过了预编译、编译、汇编、链接四个阶段。

- 预编译阶段主要处理源码中"#"开头的指令，比如加载头文件、宏替换等。
- 编译阶段主要对预处理完的文件进行词法、语义分析及优化后生成汇编代码文件。
- 汇编阶段将汇编代码转变为机器可执行的指令。
- 链接阶段将有关的目标文件进行链接，包括地址和空间分配、符号决议、重定位等。

根据 elf(5)中 st_shndx 字段描述可知，重定位就是将符号定义和符号引用进行连接的过程。

0x02 实验

main.c

```c
#include <stdio.h>
int sum(int a, int b)
{
   return (a+b);
}
int main()
{
   int res = sum(3,4);
   printf("sum is %d\n", res);
   return 1;
}
```

使用 gcc 对其进行编译和汇编操作 gcc -c main.c -o main.o -m32，此时已经生成 32 位的机器可执行的指令。

通过 objdump -d main.o 查看汇编指令

```
main.o:     file format elf32-i386


Disassembly of section .text:

00000000 <sum>:
   0:   55                      push   %ebp
   1:   89 e5                   mov    %esp,%ebp
   3:   8b 55 08                mov    0x8(%ebp),%edx
   6:   8b 45 0c                mov    0xc(%ebp),%eax
   9:   01 d0                   add    %edx,%eax
   b:   5d                      pop    %ebp
   c:   c3                      ret
```

```
0000000d <main>:
   d:   8d 4c 24 04             lea    0x4(%esp),%ecx
  11:   83 e4 f0                and    $0xfffffff0,%esp
  14:   ff 71 fc                pushl  -0x4(%ecx)
  17:   55                      push   %ebp
  18:   89 e5                   mov    %esp,%ebp
  1a:   51                      push   %ecx
  1b:   83 ec 14                sub    $0x14,%esp
  1e:   6a 04                   push   $0x4
  20:   6a 03                   push   $0x3
  22:   e8 fc ff ff ff          call   23 <main+0x16>
  27:   83 c4 08                add    $0x8,%esp
  2a:   89 45 f4                mov    %eax,-0xc(%ebp)
  2d:   83 ec 08                sub    $0x8,%esp
  30:   ff 75 f4                pushl  -0xc(%ebp)
  33:   68 00 00 00 00          push   $0x0
  38:   e8 fc ff ff ff          call   39 <main+0x2c>
  3d:   83 c4 10                add    $0x10,%esp
  40:   b8 01 00 00 00          mov    $0x1,%eax
  45:   8b 4d fc                mov    -0x4(%ebp),%ecx
  48:   c9                      leave
  49:   8d 61 fc                lea    -0x4(%ecx),%esp
  4c:   c3                      ret
```

由于未到链接阶段（重定位操作之前），main() 函数在调用 sum() 函数时使用指令的是 call 23 <main+0x16>。

通过 gcc main.o -o main -m32 进行链接后再次查看 main()函数发现 call 23 <main+0x16> 已经被修改为 call 804840b <sum>。

```
08048418 <main>:
 8048418:   8d 4c 24 04             lea    0x4(%esp),%ecx
 804841c:   83 e4 f0                and    $0xfffffff0,%esp
 804841f:   ff 71 fc                pushl  -0x4(%ecx)
 8048422:   55                      push   %ebp
 8048423:   89 e5                   mov    %esp,%ebp
 8048425:   51                      push   %ecx
 8048426:   83 ec 14                sub    $0x14,%esp
 8048429:   6a 04                   push   $0x4
 804842b:   6a 03                   push   $0x3
 804842d:   e8 d9 ff ff ff          call   804840b <sum>
 8048432:   83 c4 08                add    $0x8,%esp
 8048435:   89 45 f4                mov    %eax,-0xc(%ebp)
 8048438:   83 ec 08                sub    $0x8,%esp
 804843b:   ff 75 f4                pushl  -0xc(%ebp)
 804843e:   68 e0 84 04 08          push   $0x80484e0
 8048443:   e8 98 fe ff ff          call   80482e0 <printf@plt>
 8048448:   83 c4 10                add    $0x10,%esp
 804844b:   b8 01 00 00 00          mov    $0x1,%eax
 8048450:   8b 4d fc                mov    -0x4(%ebp),%ecx
 8048453:   c9                      leave
 8048454:   8d 61 fc                lea    -0x4(%ecx),%esp
 8048457:   c3                      ret
 8048458:   66 90                   xchg   %ax,%ax
 804845a:   66 90                   xchg   %ax,%ax
 804845c:   66 90                   xchg   %ax,%ax
 804845e:   66 90                   xchg   %ax,%ax
```

通过 readelf -r main.o 可以观察到.rel.text 重定位节区信息中记录了 sum 函数的相关信息。

```
Relocation section '.rel.text' at offset 0x200 contains 3 entries:
 Offset     Info    Type            Sym.Value  Sym. Name
00000023  00000902 R_386_PC32        00000000   sum
00000034  00000501 R_386_32          00000000   .rodata
00000039  00000b02 R_386_PC32        00000000   printf
```

- offset 表示需要进行重定位的位置，正好与 main.o 中 call 23 中 23 数值相同。
- Info 字段高 24 位表示重定位的符号表索引，与.symtab 表的 Num 字段的值对应，可通过 readelf -s main.o 查看。低 8 位表示重定位的类型（与 Type 字段对应），用于重定位的计算。
- Type 字段表示重定位类型，每种类型都对应着不同的计算方式，可参考下图。

Name	Value	Field	Calculation
R_386_NONE	0	none	none
R_386_32	1	word32	S + A
R_386_PC32	2	word32	S + A - P
R_386_GOT32	3	word32	G + A - P
R_386_PLT32	4	word32	L + A - P
R_386_COPY	5	none	none
R_386_GLOB_DAT	6	word32	S
R_386_JMP_SLOT	7	word32	S
R_386_RELATIVE	8	word32	B + A
R_386_GOTOFF	9	word32	S + A - GOT
R_386_GOTPC	10	word32	GOT + A - P

S 表示要索引的内容在符号表中对应的值。比如，该例子中 S 等于发生重定位后.symtab 表中 sum 对应的位置，也就是 value 字段的值 0804840b。

通过 readelf -s main 查看：

```
Num:    Value  Size Type    Bind   Vis      Ndx Name
…
…
57: 0804840b    13 FUNC    GLOBAL DEFAULT   14 sum
…
```

A 表示用于计算重定位字段的加数，加数分为隐式加数和显式加数。elf.h 中重定位的结构体如下：

```
typedef struct {
        Elf32_Addr      r_offset;
        Elf32_Word      r_info;
} Elf32_Rel;

typedef struct {
        Elf32_Addr      r_offset;
        Elf32_Word      r_info;
        Elf32_Sword     r_addend;
} Elf32_Rela;
```

```
typedef struct {
        Elf64_Addr      r_offset;
        Elf64_Xword     r_info;
} Elf64_Rel;

typedef struct {
        Elf64_Addr      r_offset;
        Elf64_Xword     r_info;
        Elf64_Sxword    r_addend;
} Elf64_Rela;
```

使用 Rela 类型时采用显式加数，Rel 类型时采用隐式加数。在 32 位 SPARC 环境仅使用 Elf32_Rela 类型，64 位 SPARC 和 64 位 x86 仅使用 Elf64_Rela 类型，32 位 x86 仅使用 Elf32_Rel 类型。本人环境是 x86-64 但是在编译时使用了-m32 参数，所以采用 Elf32_Rel 隐式加数，隐式加数存储在目标本身。

比如，在 main.o 中调用 sum 函数时的指令如下：

```
22: e8 fc ff ff ff          call   23 <main+0x16>
```

call 指令对应 e8，23 对应 fc ff ff ff，其中 fc ff ff ff 就是要找的隐式加数 A，根据小端序排序得到 A=0xfffffffc。

P 表示经过重定位后所在的地址。使用 objdump -d main 查看 main 方法调用 sum 函数的指令，如下：

```
08048418 <main>:
 8048418:   8d 4c 24 04             lea    0x4(%esp),%ecx
 804841c:   83 e4 f0                and    $0xfffffff0,%esp
 804841f:   ff 71 fc                pushl  -0x4(%ecx)
 8048422:   55                      push   %ebp
 8048423:   89 e5                   mov    %esp,%ebp
 8048425:   51                      push   %ecx
 8048426:   83 ec 14                sub    $0x14,%esp
 8048429:   6a 04                   push   $0x4
 804842b:   6a 03                   push   $0x3
 804842d:   e8 d9 ff ff ff          call   804840b <sum>
 8048432:   83 c4 08                add    $0x8,%esp
 8048435:   89 45 f4                mov    %eax,-0xc(%ebp)
 8048438:   83 ec 08                sub    $0x8,%esp
 804843b:   ff 75 f4                pushl  -0xc(%ebp)
 804843e:   68 e0 84 04 08          push   $0x80484e0
 8048443:   e8 98 fe ff ff          call   80482e0 <printf@plt>
 8048448:   83 c4 10                add    $0x10,%esp
 804844b:   b8 01 00 00 00          mov    $0x1,%eax
 8048450:   8b 4d fc                mov    -0x4(%ebp),%ecx
 8048453:   c9                      leave
 8048454:   8d 61 fc                lea    -0x4(%ecx),%esp
 8048457:   c3                      ret
 8048458:   66 90                   xchg   %ax,%ax
 804845a:   66 90                   xchg   %ax,%ax
 804845c:   66 90                   xchg   %ax,%ax
 804845e:   66 90                   xchg   %ax,%ax
```

call 804840b <sum>所在的地址是 804842d，但是 call 指令(e8)还占用一个字节，所以 sum 函数是从 804842e 开始，也就是 P=0x804842e。

经过计算得到 S、A、P 的值分别为 0x804840b、0xfffffffc、0x804842e，由于 sum 函数采用的重定位类型属于 R_386_PC32。通过计算，sum 在执行完重定位后的数值为：S + A - P = 0xffffffd9，也就是 d9 ff ff ff。与实际程序重定位后的结果一致。

```
8048429:       6a 04                   push   $0x4
804842b:       6a 03                   push   $0x3
804842d:       e8 d9 ff ff ff          call   804840b <sum>
8048432:       83 c4 08                add    $0x8,%esp
8048435:       89 45 f4                mov    %eax,-0xc(%ebp)
8048438:       83 ec 08                sub    $0x8,%esp
804843b:       ff 75 f4                pushl  -0xc(%ebp)
```

0x03 总结

当将源代码(main.c)编译成中间文件时(main.o)，call 的操作数为 fc ff ff ff，由于还无法知道 sum 函数的具体位置所以预先存放 0xfffffc，当进行链接操作时这个地址就需要根据重定位表(.rel.text)和符号表(.symtab)进行修正。但是并不会直接修改汇编指令，仅修改指令的操作数。

0x04 参考链接

https://linux.die.net/man/5/elf
http://www.skyfree.org/linux/references/ELF_Format.pdf
https://docs.oracle.com/cd/E26926_01/html/E25910/chapter6-54839.html#gentextid-16503
https://docs.oracle.com/cd/E19683-01/817-3677/chapter6-54839/index.html