AttackLab

2020/10/26

复习部分

栈帧结构

缓冲区溢出漏洞

对抗缓冲区溢出的保护

栈帧结构

- 书P164（CSAPP 3e中文）
- 前六个参数在寄存器里面
- rdi rsi rdx rcx r8 r9
- （该lab没有push rbp）

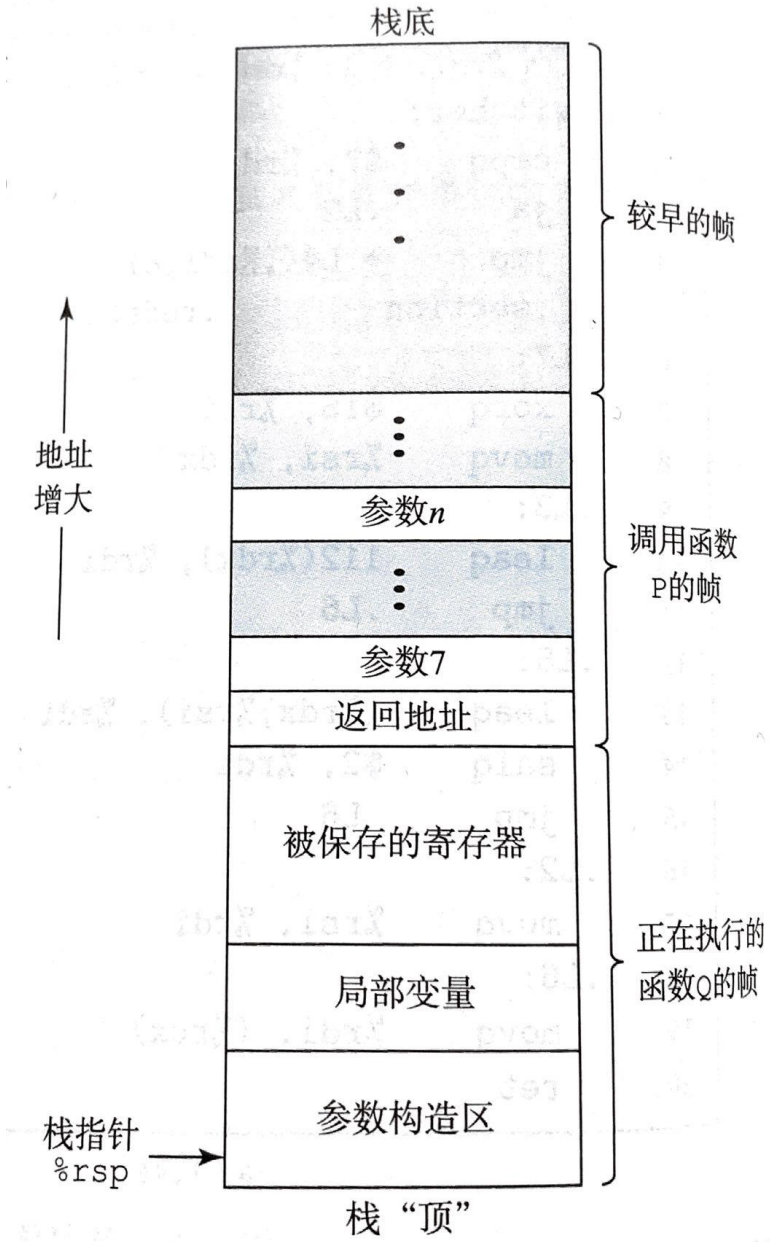


图 3-25 通用的栈帧结构（栈用来传递参数、存储返回信息、保存寄存器，以及局部存储。省略了不必要的部分）

缓冲区溢出攻击

- 当我们读入的字串比缓冲区长的时候，就会用我们的输入覆盖return address，实现可控跳转（想去哪，就去哪）

```
1 unsigned getbuf()
2 {
3       char buf[BUFFER_SIZE];
4       Gets(buf);
5       return 1;
6 }
```

三种可行的防护

- PIE（position-independent executable）：栈地址（确切来说是栈基址）每次运行均不相同。
- NX （Not executable）：栈不可执行。
- Canary：在push了被保存的寄存器之后，push一个canary，离开时检查是否被修改。

可能的绕过保护的手段

- ROP:Return-oriented Programming
- 稍后来讲

各level详解

漏洞所在函数

```
1 unsigned getbuf()
2 {
3       char buf[BUFFER_SIZE];
4       Gets(buf);
5       return 1;
6 }
```

```
1 void test()
2 {
3     int val;
4     val = getbuf();
5     printf("No exploit.  Getbuf returned 0x%x\n", val);
6 }
```

总览

- Level 1-3：ctarget，没有开保护，可自己写可执行代码
- Level 4-5：ratrget，开了保护，使用ROP来实现注入
- hex2raw：辅助工具
- farm.c：ROP gadget来源的源代码
- cookie.txt：记录了你需要修改成的cookie
- 另附：常用汇编指令同机器码的对应

Level 1

```
1 void touch1()
2 {
3     vlevel = 1;         /* Part of validation protocol */
4     printf("Touch1!: You called touch1()\n");
5     validate(1);
6     exit(0);
7 }
```

hex2raw

- 本实验的输入应该是一个字符串，hex2raw则是一个构造该字符串的工具：输入是以空格隔开的两位一组（建议八组一行。想想为什么？）的十六进制数（不含0x)。优点：便于修改，可包含不可见字符
- 如hello则对应的是：“68 65 6c 6c 6f”
- 生成二进制文件：./hex2raw <exploit.txt >exploit_raw.txt
- 运行：./ctarget –q <exploit_raw.txt
  - 也有可能是rtarget
- 生成并运行：cat exploit.txt | ./hex2raw | ./ctarget –q
- 前者的好处是，方便使用gdb调试

Hint for level 1

- 同样可以反汇编来看看汇编代码，确定缓冲区大小,确定相对位置等信息（是否保存了某些寄存器）
- 小心字节顺序：小端法
- 你可以使用gdb来验证的缓冲区溢出是否正确

Level 2

```
1  void touch2(unsigned val)
2  {
3      vlevel = 2;          /* Part of validation protocol */
4      if (val == cookie) {
5          printf("Touch2!: You called touch2(0x%.8x)\n", val);
6          validate(2);
7      } else {
8          printf("Misfire: You called touch2(0x%.8x)\n", val);
9          fail(2);
10     }
11     exit(0);
12 }
```

Hint for Lab2

- 第一个传参是在寄存器rdi里面的。
- 可以自己构造一段可执行的代码放到缓冲区里面，跳转到自己构造的可执行代码中来修改rdi。
- 可考虑使用ret来跳转到touch2，（该ret同样可控，对吗？）
- 不推荐使用jmp和call，因为它们是偏移量寻址，非常麻烦，极难控制。
- 获得某汇编对应的机器码（十六进制表示）的方法如下：

编译汇编以及获得机器码

```
# Example of hand-generated assembly code
        pushq   $0xabcdef           # Push value onto stack
        addq    $17,%rax            # Add 17 to %rax
        movl    %eax,%edx           # Copy lower 32 bits to %edx
```

- gcc –c example.s
- objdump –d example.o >example.d

```
example.o:     file format elf64-x86-64


Disassembly of section .text:

0000000000000000 <.text>:
   0: 68 ef cd ab 00        pushq  $0xabcdef
   5: 48 83 c0 11           add    $0x11,%rax
   9: 89 c2                 mov    %eax,%edx
```

```
68 ef cd ab 00 48 83 c0 11 89 c2
```

Level 3

```
1  /* Compare string to hex represention of unsigned value */
2  int hexmatch(unsigned val, char *sval)
3  {
4      char cbuf[110];
5      /* Make position of check string unpredictable */
6      char *s = cbuf + random() % 100;
7      sprintf(s, "%.8x", val);
8      return strncmp(sval, s, 9) == 0;
9  }
10
11 void touch3(char *sval)
12 {
13     vlevel = 3;        /* Part of validation protocol */
14     if (hexmatch(cookie, sval)) {
15         printf("Touch3!: You called touch3(\"%s\")\n", sval);
16         validate(3);
17     } else {
18         printf("Misfire: You called touch3(\"%s\")\n", sval);
19         fail(3);
20     }
21     exit(0);
22 }
```

Hint for Lab3

- 你需要在栈上构造一个字符串和cookies相同，每个字符恰好一个byte
- 可使用man ascii来获得字符的表示
- 请注意该字符串在栈上的摆放位置，以避免被push XXX覆盖

ROP

- rtarget开了栈随机化和栈不可执行，我们没办法确定栈上代码的位置，也无法在栈上运行。
- 此时可运行的就只有源程序的代码段（.text）。
- 我们不能写新的可执行代码，但可以用原有的：

ROP

- 我们来看一个C代码与其对应的汇编:

```
void setval_210(unsigned *p)
{
    *p = 3347663060U;
}
```

```
0000000000400f15 <setval_210>:
  400f15:       c7 07 d4 48 89 c7       movl   $0xc78948d4,(%rdi)
  400f1b:       c3                      retq
```

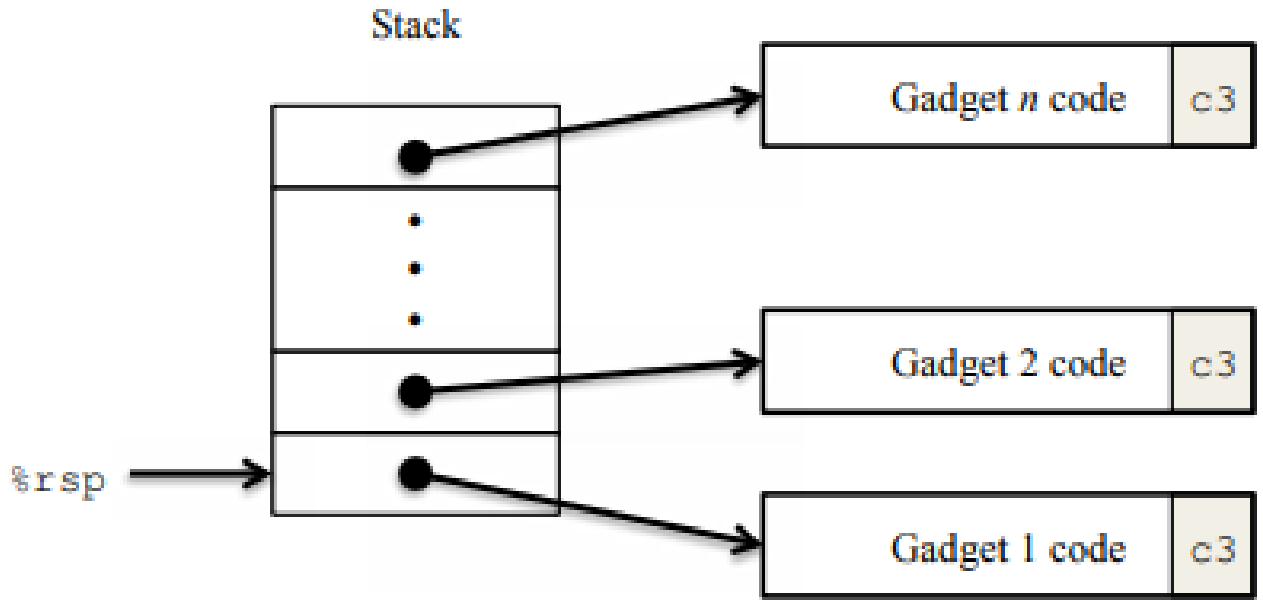
ROP

- 注意到48 89 c7也对应一个汇编指令：movq %rax, %rdi
- 那么48 89 c7 c3就对应 movq %rax, %rdi　ret
- 那我们跳到0x400f18，则会执行一句把rax的值放到rdi中，然后继续返回。
- Rtarget中提供了很多这样的函数：代码详见farm.c，可自己编译来获得机器码（前面讲过了）。

```
0000000000400f15 <setval_210>:
  400f15:       c7 07 d4 48 89 c7       movl   $0xc78948d4,(%rdi)
  400f1b:       c3                      retq
```

ROP链

- 注意，第一个ROP的位置，已经是溢出到return address了，后面的ROP，则溢出的更多



ROP Hint

- 一定是以c3结尾
- 另附常用指令与机器码的对应于elearning上
- 90 对应的机器码是nop，相当于没有
- 你们的目标是修改rdi
- 注意ret会使用掉一个栈顶元素，pop也会。
- 有些别的指令虽然并不是nop，但也在一定程度上相当于没有
- 有些gadget farm里面的gadget的画风和别的不太一致
- 如果你愿意花时间去整理有哪些可用的gadget，那你一定能通过这个lab

Level4

- rtarget里的level2
- 和ctarget里的level2的区别在于开了保护，但多了gadget farm

Level5

- rtarget里的level3
- 和ctarget里的level3的区别在于开了保护，但多了gadget farm

Lab提交

- ddl：2020/10/29
- 每一关的通过截图
- 每一关的十六进制字符串
- 对构造的字符串的解释与思路（重要）
- 体会等

Lab占比

- Level1（10%）
- Level2-4（20%）
- Level5（10%）->太难了，作为加分项
- 实验报告（20%）

Lab中需要注意的地方

你被允许跳转到的位置

- touch1,touch2,touch3的首地址
- 你注入代码的任意位置
- Gadget farm中的某一个gadget

Payload字符串

- 你的十六进制字符串，不能包含字节 （0x）0a，因为它的ascii码是“\n”，会截断输入
- Amd64大多都是小端法，本lab也是小端法。
- 构造数：0xdeadbeef，对应的输入应当是“ef be ad de”
- 辅助记忆：大端法就是大的在偏移为0的字节。小端法就是小的在偏移为0的字节。