Jarvis OJ Crypto Writeup

原创

WLNY 于 2020-08-09 18:01:16 发布 640 收藏 1

[xman2019]xcaesar

题目给出了提示，和凯撒密码有关，打开下载完的.py文件观察代码

```
def caesar_encrypt(m,k):
r=""
for i in m:
    r+=chr((ord(i)+k)%128)
return r
from secret import m,k
print caesar_encrypt(m,k).encode("base64")
#output:bXNobgJyaHB6aHRwdGgE
```

可以看出加密过程是先进行了一次凯撒，再进行了一次base64加密
反过来base64解密再凯撒解密就可以得出答案

[xman2019]xbase64

提示是base64，观察下文件代码

```
# /usr/bin/python
# encoding: utf-8
#逆排序表
base64_table = ['=','A', 'B', 'C', 'D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z',
            'a', 'b', 'c', 'd','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y','z',
            '0', '1', '2', '3','4','5','6','7','8','9',
            '+', '/'][::-1]

def encode_b64(s):
    l = len(s)
    i = 0
    result = ''
    while i < l:
        # 将字符转换为二进制编码，然后对齐
        s1 = s[i]
        b1 = bin(ord(s1))[2:] #返回字符的二进制编码，bin()函数返回值以0b开头，用切片从index=2位置开始取，去除0b
        cb1 = b1.rjust(8, '0') #rjust()函数是右对齐函数，通过在左边补0将编码扩充至8位

        i += 1
        if i >= l:
            cb2 = '00000000'
        else:
            s2 = s[i]
            b2 = bin(ord(s2))[2:]
            cb2 = b2.rjust(8, '0')

        i += 1
```

```
        if i >= l:
            cb3 = '00000000'
        else:
            s3 = s[i]
            b3 = bin(ord(s3))[2:]
            cb3 = b3.rjust(8, '0')

        # 将三字节转换为四字节
        cb = cb1 + cb2 + cb3

        rb1 = cb[:6]
        rb2 = cb[6:12]
        rb3 = cb[12:18]
        rb4 = cb[18:]

        # 转换后的编码转为十进制备用
        ri1 = int(rb1, 2)
        ri2 = int(rb2, 2)
        ri3 = int(rb3, 2)
        ri4 = int(rb4, 2)

        # 处理末尾为 0 的情况，以'='填充
        if i - 1 >= l and ri3 == 0:
            ri3 = -1

        if i >= l and ri4 == 0:
            ri4 = -1

        result += base64_table[ri1] + base64_table[ri2] + base64_table[ri3] + base64_table[ri4]

        i += 1

    return result

print encode_b64(open("flag","r").read())

#output: mZOemlSXmpOTkKCHkp6Rgv==
```

发现加密思路和base64是一样的，只不过用了个自定义的base64_table
那就改改代码，反过来解密就行了，改完的代码：

```
# /usr/bin/python
# encoding: utf-8
# 逆排序表
base64_table = ['=', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T',
                'U', 'V', 'W', 'X', 'Y', 'Z',
                'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u',
                'v', 'w', 'x', 'y', 'z',
                '0', '1', '2', '3', '4', '5', '6', '7', '8', '9',
                '+', '/'][::-1]

def search(ch):
    for i in range(65):
        if ch == base64_table[i]:
            return i

def encode_b64(s):
    l = len(s)
    i = 0
```

```
    result = ''
    while i < l:
        s1 = s[i]
        b1 = bin(search(s1))[2:]
        cb1 = b1.rjust(6, '0')
        i += 1
        if i >= l:
            cb2 = '000000'
        else:
            s2 = s[i]
            b2 = bin(search(s2))[2:]
            cb2 = b2.rjust(6, '0')

        i += 1
        if i >= l:
            cb3 = '000000'
        else:
            s3 = s[i]
            b3 = bin(search(s3))[2:]
            cb3 = b3.rjust(6, '0')

        i += 1
        if i >= l:
            cb4 = '000000'
        else:
            s4 = s[i]
            b4 = bin(search(s4))[2:]
            cb4 = b4.rjust(6, '0')

        # 将四字节转换为三字节
        cb = cb1 + cb2 + cb3 + cb4

        rb1 = cb[:8]
        rb2 = cb[8:16]
        rb3 = cb[16:24]

        # 转换后的编码转为十进制备用
        ri1 = int(rb1, 2)
        ri2 = int(rb2, 2)
        ri3 = int(rb3, 2)

        result += chr(ri1) + chr(ri2) + chr(ri3)
        i += 1

    return result

print (encode_b64('mZOemlSXmpOTkKCHkp6Rgv'))
```

得到flag

[xman2019]xyf

打开文件一看

n = 31612622552554211332925066943239887112047928187026406660843316344441487124289159506399545409744699314266187020446723181349086787306419814140370340583203591582468139871546791781593918322329901937384541163710459284342399360270065393484883167546115866595876776597916204812007325640683671485412424265338236265865749152752095083001205748191138518959329122087839156527645683197714823093384343640946815791350867031279778705347150390058223128787396116301557143131195456109392533558087426468918154427586602785149764315219337632726156532610446070418762129988837327246624101970384197217732906011090659656741295996261511395663690e = 65537c = 63158391159266065221541268308868878543893838640332332313124753456195853128857061213413928809053361954887615644749862793862641961796891829921286393683970194364373543726430406282820580998453359254759906082945166824056938413013008092829208288852656790269570721566002020139264038851837906324448720488143959181339849528502570428578107298702469813314735423870286180314654805773675600329424879182778228072267045715738520578725997980489296652953690295981367553702887940780236543902471194209112305830546085667691045826809779853290104005050690614154790976609332319736303495992690044042080576871602905288545256062530831428440 6

[xman2019]xbk

打开文件观察

```
n=47966708183289639962501363163761864399454241691014467172805658518368423135168025285144721028476297179341434450931955275325060173656301959484440112740411109153032840150659
e=3
c=10968126341413081941567552025256642365567988931403833266852196599058668508079150528128483441934584299102782386592369069626088211004467782012298322278772376088171342152839
```

先去factordb看看能不能分解，发现不能，再尝试用yafu分解，需要几个小时，看来不能直接分解。再仔细观察，发现 e = 3 ，百度一下 e = 3 时RSA怎么解密，在freebuf找到一篇讲解低加密指数攻击的文章，根据 c = m +k*n这个表达式来爆破求解，编写爆破脚本

```
from gmpy2 import iroot
from libnum import s2n, n2s
c = 10968126341413081941567552025256642365567988931403833266852196599058668508079150528128483441934584299102782386592369069626088211004467782012298322278772376088171342152839
n = 47966708183289639962501363163761864399454241691014467172805658518368423135168025285144721028476297179341434450931955275325060173656301959484440112740411109153032840150659
k = 0
while 1:
    res = iroot(c+k*n,3)
    if(res[1]==True):
        print (res)
        break
    print("k="+str(k))
    k=k+1
m = 25114135108419253959322585150558844347940933210764528446 69
print (libnum.n2s(m))
```

得到flag，具体rsa各种攻击方式原理以后会单独写一篇文章总结一下

Medium RSA

打开下载的压缩包，里面有两个文件，flag.enc, pubkey.pem
这种文件需要用到openssl打开，下载安装完后，在题目文件夹中使用如下cmd指令读取公钥：

```
openssl rsa -pubin -in pubkey.pem -text -modulus
```

结果如图
[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-BJZj4GSQ-1596967243231)
(file://E:/Gridea/blog/post-images/1596620259635.png)]
其中

```
e = 65537
n = C2636AE5C3D8E43FFB97AB09028F1AAC6C0BF6CD3D70EBCA281BFFE97FBE30DD(16进制)
```

通过如下代码从文件中读取 n，e，c

```
from Crypto.PublicKey import RSA
with open('./pubkey.pem', 'r') as f:
    key = RSA.importKey(f)
    N = key.n
    e = key.e
with open('flag.enc', 'r') as f:
    c = f.read().encode('hex')
    c = int(c, 16)
```

之后步骤同其他rsa题

hard RSA

打开压缩包，里面有两个文件，flag.enc, pubkey.pem，读取公钥
[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-17c05BTx-1596967243270)
(file://E:/Gridea/blog/post-images/1596620597481.png)]

```
e = 2
n = C2636AE5C3D8E43FFB97AB09028F1AAC6C0BF6CD3D70EBCA281BFFE97FBE30DD
```

这里公钥为2，利用rabin算法破解

[xman2019]xfz

打开压缩包发现两个文件，先看fez.py，这是加密过程

```
import os
def xor(a,b):
    assert len(a)==len(b)
    c=""
    for i in range(len(a)):
        c+=chr(ord(a[i])^ord(b[i]))
    return c
def round(M,K):
    L=M[0:27]
    R=M[27:54]
    new_l=R
    new_r=xor(xor(R,L),K)
    return new_l+new_r
def fez(m,K):
    for i in K:
        m=round(m,i)
    return m

K=[]
for i in range(7):
    K.append(os.urandom(27))
m=open("flag","rb").read()
assert len(m)<54
m+=os.urandom(54-len(m))

test=os.urandom(54)
print test.encode("hex")
print fez(test,K).encode("hex")
print fez(m,K).encode("hex")
```

简单来说就是把字符串分成两节交替和密钥进行异或操作
画出流程图，发现最后 fez(test,K) 和 fez(m,K) 都是形如
r ^ p + r ^ l ^ q
的字符串结构，这样只要把 fez(test,K) 和 fez(m,K) 以及 test 左右两边单独拿出来互相异或就能得到 m 的左右两边，编写代码如下

```
import os
import libnum
#异或操作照抄
def xor(a,b):
    assert len(a)==len(b)
    c=""
    for i in range(len(a)):
        c+=chr(ord(a[i])^ord(b[i]))
    return c
#从fez.log文件得到
test='50543fc0bca1bb4f21300f0074990f846a8009febded0b2198324c1b31d2e2563c908dcabbc461f194e70527e03a807e9a478f9a56f7'
utest='66bbd551d9847c1a10755987b43f8b214ee9c6ec2949eef01321b0bc42cffce6bdbd604924e5cbd99b7c56cf461561186921087fa1e9'
um='44fc6f82bdd0dff9aca3e0e82cbb9d6683516524c245494b89c272a83d2b88452ec0bfa0a73ffb42e304fe3748896111b9bdf4171903'
#编码为字符
test = test.decode('hex')
utest = utest.decode('hex')
um = um.decode('hex')
#分成左右两段
um_l = um[0:27]
um_r = um[27:54]
utest_l = utest[0:27]
utest_r = utest[27:54]
test_l = test[0:27]
test_r = test[27:54]
#异或得到m左右两段
m_r = xor(xor(um_l, utest_l), test_r)
m_l = xor(xor(xor(xor(um_r, utest_r), test_r), test_l), m_r)
#合并
m = m_l + m_r
print m
```