simpleDES writeup

原创

seen_in_hw 于 2018-03-20 21:48:51 发布 714 收藏 1

分类专栏： CTF

本文链接：https://blog.csdn.net/qq_23100787/article/details/79632530

题目

题目链接如下:https://ctftime.org/task/5405

我再粘一遍题目吧,因为这个网站似乎访问会经常中断连接

```
Larry is working on an encryption algorithm based on DES.

He hasn't worked out all the kinks yet, but he thinks it works.

Your job is to confirm that you can decrypt a message, given the algorithm and parameters used.

His system works as follows:

    Choose a plaintext that is divisible into 12bit 'blocks'
    Choose a key at least 8bits in length
    For each block from i=0 while i<N perform the following operations
    Repeat the following operations on block i, from r=0 while r<R
    Divide the block into 2 6bit sections Lr,Rr
    Using Rr, "expand" the value from 6bits to 8bits.
    Do this by remapping the values using their index, e.g.
    1 2 3 4 5 6 -> 1 2 4 3 4 3 5 6
    XOR the result of this with 8bits of the Key beginning with Key[iR+r] and wrapping back to the begi
    Divide the result into 2 4bit sections S1, S2
    Calculate the 2 3bit values using the two "S boxes" below, using S1 and S2 as input respectively.

S1  0   1   2   3   4   5   6   7
0   101 010 001 110 011 100 111 000
1   001 100 110 010 000 111 101 011
S2  0   1   2   3   4   5   6   7
0   100 000 110 101 111 001 011 010
1   101 011 000 111 110 010 001 100

    Concatenate the results of the S-boxes into 1 6bit value
    XOR the result with Lr
    Use Rr as Lr and your altered Rr (result of previous step) as Rr for any further computation on blo
    increment r

He has encryped a message using Key="Mu", and R=2. See if you can decipher it into plaintext.

Submit your result to Larry in the format Gigem{plaintext}.

Binary of ciphertext: 01100101 00100010 10001100 01011000 00010001 10000101
```

题目分析

这个题是个密码题,然后是DES的,这肯定没差了,难点我觉得对于我来说最大的就是看题了.
其实题看懂了之后就发现这个题目就是各种流程转来转去的,我看到了最后的R=2,然后结合密文之后6个字节,想到一遍循环只有三个字节,那样就可以用暴力搜索的方法来解决这个问题了,后来证明是可行的.
上面这段话有的人可能觉得是在看天书,那就发一个关于DES的介绍的帖子吧,然后我把题目大致翻译一下.
DES详解:DES详解

然后我对题目的操作过程的翻译:(就是靠这个过程解密的) 建议别看我翻译,还是直接看英文吧…
1,找一个明文,把明文分块,每一个块是12bit
2,找一个长度起码是8bit的key
3,对每个明文块进行如下操作———>这里就是核心啦!!一看就要for循环啦!
4,5步骤放一起 对于每个块,进行R次的重复性操作(R后面给出来了),将块分成Lr跟Re的两个6bit的块.
然后后面基本就是对Rr的操作,对于Lr并没有什么操作.
6,将Rr由原来的6bit映射到8bit
7,根据第4步的循环,将第6步的结果跟一个subkey得到的规则进行异或操作(划重点,这个subkey如何算的很重要!!)
8,将第7步的结果左右划分为2个4bit的s1,s2盒
9,把8的结果分别代入s1,s2盒算
10,第9步结果得到两个3bit的结果,连接起来
11,将第9步的结果跟Lr异或
12,Lr,Rr的替换(自己去看规则吧)
13,继续第4步的迭代过程,直到产出结果

通过对上述加密过程的了解跟观察,我们就可以得出一些东西了,那就是密文跟明文的字节数是一样的.然后对于每一个块要进行R次的迭代过程.
我们来观察密文:
01100101 00100010 10001100 01011000 00010001 10000101
很明显,这是6个字节,然后R=2,这样每次加密的都是三个字节,考虑到另一个CRC32爆破的例子(在我的上几篇文章).那我的思路就变成了:

```
明文字节----->加密------->密文
```

我通过对三字节取全集的方式来获取所有三个可能性字节的遍历,然后通过上述加密过程(翻译的那段)加密明文,最后跟密文相比较,这样遍历所有的可能性结果一定会有所收获的!这就是思路所在了.

注意事项

密文是十六进制,需要把它转换为十进制,最终转成可见字符,这里需要写一个函数转换

```
def getstr(intnum):                #16 hex to str
    intstr = hex(intnum).lstrip('0x').rstrip('L')
    if len(intstr) % 2 == 1:
        intstr = '0' + intstr
    intstr = intstr.decode('hex')
    return intstr
```

python用来产生全集的写法:

```
for item in itertools.product(string.printable, repeat=3):
```

这个写法是产生string类型的三字节的全集

加密过程是最难的,一定要好好看懂,涉及到各种移位,还是有些难理解的.
加密过程代码:

```
def encrypt(s, key, offset):
    #1, Choose a plaintext that is divisible into 12bit 'blocks'
    plain = s

    #2, Choose a key at least 8bits in length

    #3, For each block from i=0 while i<N perform the following operations
    # Currently no padding
    N = len(plain) * 8 / 12           #block
    R = 2
    result = 0
    pint = int(plain.encode('hex'), 16)
    for i in range(N):
    #4, Repeat the following operations on block i, from r=0 while r<R
        p = (pint >> (12 * (N - 1 - i))) & 0xfff
        Lr = (p >> 6) & 0x3f
        Rr = p & 0x3f                     #step 5
        for r in range(R):
            tmp = ((Rr << 2) & 0xc0) | (Rr & 0x3) | ((Rr & 0x8) << 1) | ((Rr & 0x8) >> 1) | ((Rr & 0x4)
            subkey = getsubkey(int(key.encode('hex'),16), (i + offset) * R + r, len(key) * 8)
            tmp = tmp ^ subkey          #step 7,tmp is 8 bit
            S1 = (tmp >> 4) & 0xf       #step 8
            S2 = tmp & 0xf              #step 8
            tmp = (S1box(S1) << 3) | S2box(S2)      #step 9 10
            tmp = tmp ^ Lr             #step 11
            Lr = Rr
            Rr = tmp
        result = (result << 12) | ((Lr << 6) | Rr)
    return getstr(result)
```

最终代码及结果

```
def S1box(S1):
    box = [[0b101, 0b010, 0b001, 0b110, 0b011, 0b100, 0b111, 0b000],
           [0b001, 0b100, 0b110, 0b010, 0b000, 0b111, 0b101, 0b011]]
    return box[(S1>>3) & 0x1][S1 & 0x7]

def S2box(S2):
    box = [[0b100, 0b000, 0b110, 0b101, 0b111, 0b001, 0b011, 0b010],
           [0b101, 0b011, 0b000, 0b111, 0b110, 0b010, 0b001, 0b100]]
    return box[(S2>>3) & 0x1][S2 & 0x7]

def getsubkey(key, index, n):
    result = 0
    for i in range(8):
        tmp = (((key >> ((n - 1 - (index + i)) % n) & 1) << (7 - i)))
        result = result | tmp
    return result

def encrypt(s, key, offset):
    #1, Choose a plaintext that is divisible into 12bit 'blocks'
    plain = s

    #2, Choose a key at least 8bits in length

    #3, For each block from i=0 while i<N perform the following operations
    # Currently no padding
    N = len(plain) * 8 / 12           #block
```

```
        R = 2
        result = 0
        pint = int(plain.encode('hex'), 16)
        for i in range(N):
        #4, Repeat the following operations on block i, from r=0 while r<R
            p = (pint >> (12 * (N - 1 - i))) & 0xfff
            Lr = (p >> 6) & 0x3f
            Rr = p & 0x3f                       #step 5
            for r in range(R):
                tmp = ((Rr << 2) & 0xc0) | (Rr & 0x3) | ((Rr & 0x8) << 1) | ((Rr & 0x8) >> 1) | ((Rr & 0x4)
                subkey = getsubkey(int(key.encode('hex'),16), (i + offset) * R + r, len(key) * 8)
                tmp = tmp ^ subkey          #step 7,tmp is 8 bit
                S1 = (tmp >> 4) & 0xf       #step 8
                S2 = tmp & 0xf              #step 8
                tmp = (S1box(S1) << 3) | S2box(S2)      #step 9 10
                tmp = tmp ^ Lr             #step 11
                Lr = Rr
                Rr = tmp
            result = (result << 12) | ((Lr << 6) | Rr)
        return getstr(result)
    def getstr(intnum):                 #16 hex to str
        intstr = hex(intnum).lstrip('0x').rstrip('L')
        if len(intstr) % 2 == 1:
            intstr = '0' + intstr
        intstr = intstr.decode('hex')
        return intstr

    enc = 0b011001010010001010001100010110000001000110000101
    encstr = getstr(enc)
    print (encstr)

    import string, itertools, time

    # brute force
    flag = ''
    time1 = time.time()
    for i in range(2):
        for item in itertools.product(string.printable, repeat=3):
            #print item
            if encrypt(''.join(item), 'Mu', i * 2) == getstr((enc >> (1 - i) * 24) & 0xffffff): #twice so w
                print item
                flag += ''.join(item)
                break
    time2 = time.time()
    print ' flag is Gigem{' + flag + '}, passed time:', time2 - time1
```

最终结果:

flag is Gigem{MiN0n!}, passed time: 11.4236609936

我的电脑跑了10多秒就跑出来了,这个数据量不是很大,所以就跑的很快