35c3CTF Crypto Writeup

robin / 2019-01-08 09:00:00 / 浏览数 2887 安全技术 CTF 顶(0) 踩(0)

35c3CTF Crypto writeup

c3系列CTF题目质量还是很高的。这次比赛有两道Crypto，都属于中等难度。下面是我的Writeup。其中第二道的思路参考了https://sectt.github.io/writeups/35C3CTF/c

unofficial

题目描述

> The NSA gave us these packets, they said it should be just enough to break this crypto.
> Difficulty estimate: medium

题目分析

题目给了两个文件 server.py 和surveillance.pcap
打开server.py发现代码并不复杂。首先看一下加密用的密钥的生成过程：

```
def keygen():
   return [rand() for _ in range(N)]
```

会随机生成４０个数字，其中随机的具体过程如下：

```
def rand():
   return int.from_bytes(os.urandom(bits // 8), 'little')
```

可以看到是调用的操作系统的urandom，而不是python 自己的伪随机函数,因此无法预测出随机数。即密钥是40个无法预测的随机数。
接着分析加密过程：

```
import os, math, sys, binascii
from secrets import key, flag
from hashlib import sha256
from Crypto.Cipher import AES

p = 21652247421304131782679331804390761485569
bits = 128
N = 40

if __name__ == '__main__':
   # key = keygen()   # generated once & stored in secrets.py
   challenge = keygen()
   print(' '.join(map(str, challenge)))
   response = int(input())
   if response != sum(x*y%p for x, y in zip(challenge, key)):
       print('ACCESS DENIED')
       exit(1)

   print('ACCESS GRANTED')
   cipher = AES.new(
           sha256(' '.join(map(str, key)).encode('utf-8')).digest(),
           AES.MODE_CFB,
           b'\0'*16)
   print(binascii.hexlify(cipher.encrypt(flag)).decode('utf-8'))
```

题目告诉我们，key只随机生成一次，除了key以外，还会调用keygen生成４０个随机数作为challenge,challenge每次都是不同的。初始化完成之后，会把生成的challenge
DENIED。如图所示：

```
(python3) robin@robin-Vostro-3268:~/Crypto/35c3/unofficial$ python server.py
215320161381210482268110971767067384423 327459911456481728281188878175197814102
805428940115340976584405046507520211611 156424734079157766785135165053844996476 1
761338232934443401083380031724133290093 351512973738989898227039148521388075702 28
822387922631469446132637326164620348 220101246334438356292932328387065605252 2418
985646614773793833447194021382731381 127539272070427493392282706729219684656 2322
858926549944659853924301930155566804 484433245733383609434598342771476966601 10166
344130300172307505376829339983968402 292490069951773335985615679822563005036 22249
969275690060203640107079693966370 1800918934564154959895026751760562257 286464
723284854549614351834454028733023 211137301713793314407590792568681228215 436024
006040818029698506765049209451252 962479853835461368985463502325082698271 18103495
038560375691935638685772973805612 128445136141316026198396921786183419031 27454638
091460075249214199065485399405 384577274032230577282948936501325156068 8120905543
472641333126598932844241809812 199897802398874399098818749116364938482 32058459623
435721580456434414110783461202 270296416471498326035595320216931606747 92619138282
661394131102887421553006268 286080027638184004878711169154789061137 707048139280
325853004596439543614995872 220519185456350859606586922312048784668 82570705370154
634176561470240448961254 258764989437672252254163498548923753266 9180600987215807
798100869628668744562520 183260369105162135150919176354423225020 8608897196234805
949361815078709870435 3 111811962549308018175062684357217770339 39441593428830582
430223956814352455864 291230889910566536699247500041091835574
111222
ACCESS DENIED
```

继续分析用户输入需要满足的条件：

```
response ■■ sum(x*y%p for x, y in zip(challenge, key))
```

等式的右边是让key 和 challenge中对应的数字两两相乘并求和，即

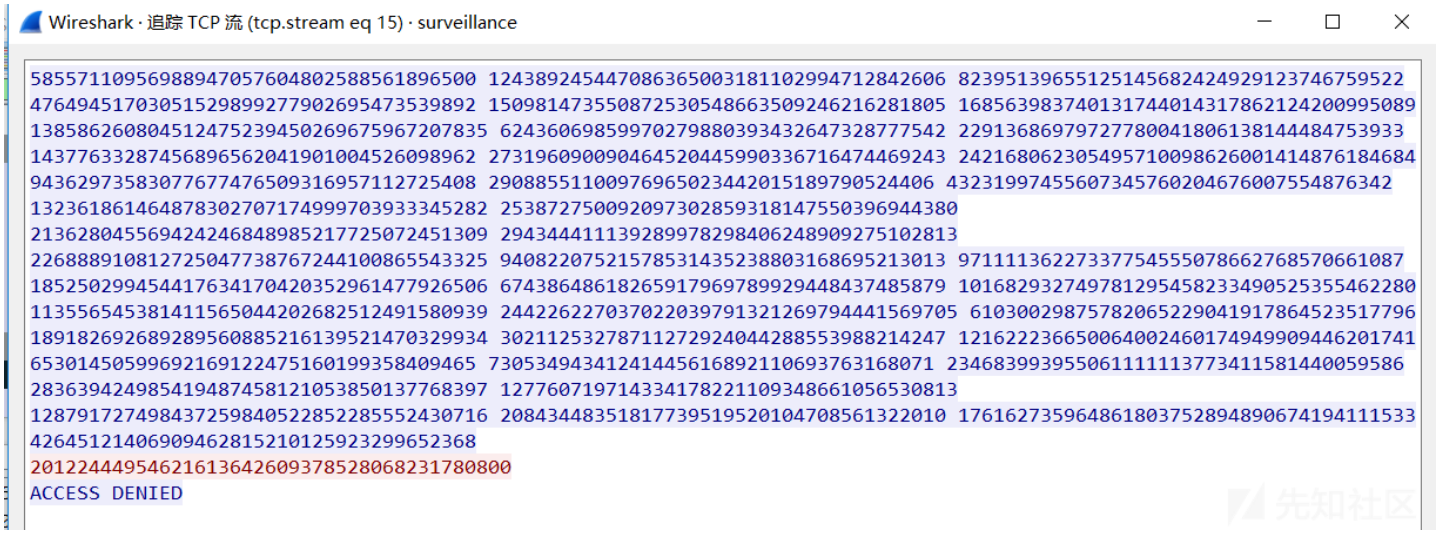
$$response = \sum_{i=1}^{40} (key_i \times challenge_i)$$

如果满足条件，服务器会用对key字符串进行sha256签名的结果作为AES密钥，对flag进行加密，并返回密文。由于key保持不变，所以每次返回的密文其实都是一样的。如

分析完脚本，我们再分析题目给的流量包。用wireshark打开流量包，发现流量包记录了40次客户端与服务端的交互数据。第一次的交互数据如图所示：

```
207799005288176923377991687975274594 91 626702391522718545682813045562991758 00 933536325858385581293393054096121900 36
292752937090308930969063150310934751 17 217443785766847465643965167326471366930 877683948003375346306708687268741692 62
942720218196976388197584764847844374 17 797615098735971935388057745867721525 51 287077043474345536330292000256667722862
338545568968754066451781943973364050747 297369519464034454780797709549121754981 56885490048115174244568542056852168691
125409768592704189418754659727077855243 24890321097180066195790295943752972007 8
105198372128401404878582092383380684270 17134646662561861003027538151057271558 6
155477660830517170755561240929991575154 18267369683661956885449128843244267323 3
262702931683036291589579288667282098519 18870116631740236121991956018407513291 25963971386922396925607504026104622721 1
211759681449153548839999286208573748417 75564698223593131868793110465400517546 39578725483882816971362731562796034 23
216645505172179755607682378174353919850 22996838468306885620265706756952309057 8 667393010270570818767589547532758575 01
116367528061570402100923471061918655467 14375046982536713830614696662054370104 8
183518250254992484830711613992618112334 17765944611261015250261812153402267239 4
294880691868997745862235404753518785618 29681754347812719041082158715888410764 2 589188440882497389060487876715262812 42
191823527475136131674233114644080652238 11515326616725698764347862432114712827 9 330289977318421085452654094423459107 0
320206885550635993814394270345194386 11 136800502151738819775273044156131929386 29726266020084507160452124182038554809 4
48783930546582323400684559460254687477849 1
ACCESS GRANTED
aef8c15e422dfb8443fc94aa9b5234383d8ee523d6da9c4875ccf0d2cf24b1c3fa234e90b9f9757862d242063dbd694806bc54582deddbcbcc
```

一开始是服务端发送的40个随机数，然后客户端发送了response(图中红色部分)，response满足条件，服务端打印了ACCESS GRANTED
和加密后的密文aef8c15e422dfb8443fc94aa9b5234383d8ee523d6da9c4875ccf0d2cf24b1c3fa234e90b9f9757862d242063dbd694806bc54582deddbcbcc



解题思路

总结一下题目的已知条件

1. 我们有AES加密的密文
2. 要解密密文必须恢复key，key是40个随机整数
3. key是随机生成的，并且使用的是系统随机数，而不是伪随机算法
4. 我们可以从流量中提取出39个关于key的等式，其中只有key是未知的，challenge和response都是可以从流量中获得的。

这样我们可以把求解key的问题转换为线性代数问题。key为40个未知数 x_1 ... x_{40}。每个等式都是关于 x_i 的方程。

$$response_j = \sum_{i=1}^{40} (key_i \times challenge_{ji})$$

这39个等式构成一个40元一次线性方程组，对该方程组求解即可得到key。用矩阵表示这个乘法为

$$Challenge_{39*40} \times Key_{40*1} = Response_{39*1}$$

常识上说，要解40元一次线性方程组必须要有40个等式，我们只有39个等式，无法求解。但只要有一些线性代数知识就可以知道，非齐次线性方程组的解的情况其实是和矩

解题脚本

```
from sage.all import *
p = 21652247421304131782679331804390761485569
I = Integers(p)

CHM = Matrix(I,CH)
AWM = vector(I,AW)
KM=CHM.solve_right(AWM)

from Crypto.Cipher import AES
from hashlib import sha256
key=list(KM)
print len(key)
print ' '.join(map(str, key)).encode('utf-8')
cipher = AES.new(sha256(' '.join(map(str, key)).encode('utf-8')).digest(),AES.MODE_CFB,b'\0'*16)
c = "aef8c15e422dfb8443fc94aa9b5234383d8ee523d6da9c4875ccf0d2cf24b1c3fa234e90b9f9757862d242063dbd694806bc54582deddbcbcc"
c=c.decode("hex")
print repr(c)
print cipher.decrypt(c)
```

flag 35C3_as_an_att4ck3r_I_am_b1as3d_t0wards_b1ased_r4nd0mness

post quantum

题目描述

> Somebody asked for more crypto challenges, so we made one in the middle of the night. Now you better solve it.

题目分析

题目给了两个脚本challenge.py generate.py和一个文件夹data
先看一下generate.py

```
from challenge import CodeBasedEncryptionScheme
from random import SystemRandom
from os import urandom

if __name__ == "__main__":
    cipher = CodeBasedEncryptionScheme.new()
    random = SystemRandom()
    for i in range(1024 + 512):
        pt = urandom(2)
        ct = cipher.encrypt(pt)
        with open("plaintext_{:03d}".format(i), "wb") as f:
            f.write(pt)
        with open("ciphertext_{:03d}".format(i), "wb") as f:
            f.write(ct)
        assert(pt == cipher.decrypt(ct))

    with open("flag.txt", "rb") as f:
       flag = f.read().strip()

    if len(flag) % 2 != 0:
        flag += b"\0"

    cts = list()
    for i in range(len(flag) // 2):
        cts.append(cipher.encrypt(flag[i*2:i*2 + 2]))

    for i, ct in enumerate(cts):
        with open("flag_{:02d}".format(i), "wb") as f:
            f.write(ct)
```

引入了出题人的加密系统CodeBasedEncryptionScheme,观察其加密过程，发现其一次加密 2 个字节。首先先加密了1024+512=1536组随机内容，并且加明文和对应的密
可以看看所给的明密文对。明文为 2 字节，密文长0x126字节

```
(python3) robin@robin-Vostro-3268:~/Crypto/35c3/post_quantum/data$ xxd ciphertex
t_000
00000000: dc08 accf 0ef5 3775 372e 9da3 8d42 9bb6  ......7u7....B..
00000010: a7f5 97f2 fc98 e28d b3eb 75c3 dc09 eb26  ..........u....&
00000020: d738 ab76 4684 253a 50f5 bdf0 d65c 4f05  .8.vF.%:P....\O.
00000030: bdd9 e42f 7d5c 489a caf7 5385 362b d881  .../}\H...S.6+..
00000040: 993e 5e92 bd94 1365 855b 4e9f e73a dfe1  .>^....e.[N..:..
00000050: 153e 1777 f551 0f2c 19ab 3305 571d c3bc  .>.w.Q.,..3.W...
00000060: e90e 06cf 37bc 28f8 7afa e2c7 df25 1f59  ....7.(.z....%.Y
00000070: 63d3 08c1 4e37 5872 9a2a 8a35 5083 0017  c...N7Xr.*.5P...
00000080: a8a0 66c0 d228 9367 18eb 077b 7853 82fd  ..f..(.g...{xS..
00000090: 973b 083e 223d 6d92 5830 1a72 ee13 c475  .;.>"=m.X0.r...u
000000a0: b353 7d32 7a95 36a4 3a6f a32f 845b d16c  .S}2z.6.:o./.[.l
000000b0: 683f a7a3 6173 0879 024d 2bf6 48cf 78b1  h?..as.y.M+.H.x.
000000c0: 505a 4fea 7988 7dc2 07f9 b4d8 15ff 5357  PZO.y.}.......SW
000000d0: 4ace d663 94fe 0c82 2f39 500a 5ec6 5ad8  J..c..../9P.^.Z.
000000e0: f0c5 0e25 40d4 45a8 9746 b3a6 46df ad69  ...%@.E..F..F..i
000000f0: ff26 267a 3285 60ae c5c7 6ca8 2fd2 9468  .&&z2.`...l./..h
00000100: 861e 24de 0b72 a9d4 4f86 99a4 d87d 44ba  ..$..r..O....}D.
00000110: afcc 6e3f 2e99 9195 3788 36ee e8a7 0de3  ..n?....7.6.....
00000120: dcd1 c858 74e6                           ...Xt.
(python3) robin@robin-Vostro-3268:~/Crypto/35c3/post_quantum/data$ xxd plaintext
_000
00000000: 16db                                     ..
```

先知社区

接着分析具体的加密过程。先看密钥的生成：

```
bitlength=48
def keygen(cls, bitlength):
   key = SystemRandom().getrandbits(bitlength)
   key = BitVector(size=bitlength, intVal = key)
   return key
```

密钥为随机生成的48bit,并构成一个向量。继续分析加密过程，在加密之前，会对输入的内容做一个编码：

```
def add_encoding(self, message):
       message = int.from_bytes(message, 'big')
       message = BitVector(size=self.key_length // 3, intVal=message)
       out = BitVector(size=self.key_length)
       for i, b in enumerate(message):
           out[i*3 + 0] = b
           out[i*3 + 1] = b
           out[i*3 + 2] = b
       return out
```

该编码过程会把输入的 2 个字节先转换为长为16的比特向量，随后会将这个向量复制 2 次，构成一个与key等长的长为48的比特向量。比如输入的message为b"AC",得到的

```
def encrypt(self, message):

       message = self.add_encoding(message)

       columns = [
           BitVector(
               size=self.key_length,
               intVal=self.random.getrandbits(self.key_length)
           )
           for _ in range(self.key_length)
       ]

       # compute the noiseless mask
       y = matrix_vector_multiply(columns, self.key)

       # mask the message
       y ^= message

       # add noise: make a third of all equations false
       for i in range(self.key_length // 3):
           noise_index = self.random.randrange(inverse_error_probability)
           y[i * 3 + noise_index] ^= 1

       columns = [bitvector_to_bytes(c) for c in columns]
       columns = b"".join(columns)

       return columns + bitvector_to_bytes(y)
```

首先，会随机生成48个长为48比特的向量，构成矩阵columns，随后该矩阵与Key向量相乘再异或编码后的message向量。用公式描述如下：

$$(Columns \times Key) \oplus Message = Y$$

运算完成后还会有一个add noise的过程，简单来说就是对于Y，每3个比特中就会有一个比特被随机翻转，生成Y'。最后会将比特序列columns | Y'转换为字节输出。密文输出的长度为(48*48+48)/8=294=0x126个字节。
同时出题人还给出了解密脚本，其基本原理为，已知columns和key，可以有:

$$(Columns \times Key) \oplus Y' = X'$$

生成的X'中，每三个比特会有 1 个比特的噪音，可以从 3 个比特中挑选出相同的两个来去除噪音，最后获得X即明文。

解题思路

总结一下题目的已知条件

1. 我们有flag加密后的密文
2. 我们有解密算法，需要恢复key
3. 我们有若干组明密文对
4. 如果没有噪音，给我们一组明密文对M,C,w我们有如下等式

$$(Columns \times Key) \oplus M = C$$

根据数学运算性质，我们有

$$Key = Columns^{-1} \times (C \oplus M)$$

也就是说，如果没有噪音，我们可以很方便的恢复key。那我们该如何去除噪音呢？
一个简单的思路是加密一次共引入16个噪音，每个噪音有 3 个可能位置，一共有3^16^种情况。可以通过穷举来去除噪音，对恢复Key。对于每一个恢复出的key，可以用其

1. 选取某一个明密文对C,M,Cols (脚本是选的flag的开头两个字母'35'及其对应的密文)
2. 穷举3^16^种噪音，从中挑选出一个噪音E
3. 用噪音E对C进行去噪得到Y
4. 用Y,M,Cols 根据公式可以恢复出Key，其中求逆矩阵通过sage完成
5. 选取若干组明密文对（脚本选取了 2 组）
6. 用恢复的Key解密这些密文，并把解密结果与明文比较
7. 如果结果与明文存在不匹配，则恢复错误，回到第 2 步
8. 如果匹配则用Key对完整的flag进行解密
9. 对于所有解密出的flag，人工判断是否是正确解

我的脚本选取了 2 组明密文对对Key进行检查，运行10分钟遍历了所有的情况。最后只得到了一个解,并且这个解的确解出了正确的flag。我的脚本如下，其中部分基础函数来

解题脚本

```
#include <assert.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
void print_array(int *mask, int len) {
 for (int i = 0; i < len; i++) {
   printf("%d, ", mask[i]);
 }
 puts("");
}
int cols0[48][48] = {
   {0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0,
    1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0},
   {1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0,
    0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0},
   {0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1,
    1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0},
   {0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0,
    1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1},
   {0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1,
    0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1},
   {0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1,
    1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1},
   {1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1,
    0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0},
   {0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1,
    1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0},
   {1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1,
    1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0},
   {0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1,
    1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1},
   {1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0,
    0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0},
   {0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0,
    0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0},
   {1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1,
    0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1},
   {0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0,
    1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1},
   {0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0,
    0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0},
   {1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0,
    1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1},
```

```
   {0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0,
    1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0},
   {0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1,
    1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0},
   {1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1,
    1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1},
   {0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1,
    1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0},
   {0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0,
    0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1},
   {1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1,
    1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0},
   {0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1,
    0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0},
   {0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0,
    1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1},
   {1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1,
    0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0},
   {0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1,
    1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0},
   {1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0,
    1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0},
   {1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0,
    0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0},
   {1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0,
    0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1},
   {0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1,
    1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1},
   {0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1,
    1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0},
   {0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0,
    0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0},
   {1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0,
    0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0},
   {0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1,
    0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0},
   {0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0,
    0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1},
   {0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0,
    0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0},
   {1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0,
    0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1},
   {1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1,
    0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0},
   {0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0,
    1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0},
   {1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0,
    0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1},
   {1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1,
    1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1},
   {1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0,
    0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1},
   {0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1,
    1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1},
   {1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1,
    0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1},
   {0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1,
    0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1},
   {1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0,
    0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1},
   {1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0,
    0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0}};

int cols0Inv[48][48] = {
   {1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0,
    0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0},
   {1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0,
```

```
 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0},
{0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1,
 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1},
{1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1,
 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1},
{0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0,
 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1},
{0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0,
 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0},
{0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0,
 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1},
{0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1,
 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0},
{1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0},
{0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0,
 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0},
{1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1,
 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1},
{1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0,
 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0},
{0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0,
 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0},
{1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0,
 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0},
{0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0,
 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1},
{0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1,
 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0},
{0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0,
 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1},
{0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0,
 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1},
{0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1,
 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1},
{1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1,
 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1},
{1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0,
 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1},
{1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1,
 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1},
{0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0,
 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1},
{1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0,
 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0},
{1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0,
 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1},
{0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1},
{0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1,
 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0},
{0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1,
 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1},
{1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0,
 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0},
{1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1},
{1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1,
 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1},
{1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0,
 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0},
 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1},
```

```
   {1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1,
    0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1},
   {0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0,
    1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1},
   {1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0,
    1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1},
   {1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1,
    0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1},
   {1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0,
    1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0},
   {1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0,
    1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0},
   {1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1,
    1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0},
   {0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0,
    1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0},
   {0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
    1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1},
   {0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1,
    1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1},
   {1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1,
    0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1},
   {1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1,
    0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1}};
int flag0xorflagcipher0[48] = {1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1,
                   1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1,
                   1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0};


int *matrix_vector_multiply(int a[48][48], int v[48]) {
 int sz = 48;
 int *res = calloc(sz, sizeof(int));

 for (int i = 0; i < sz; i++) {
   for (int j = 0; j < sz; j++) {
     res[i] ^= a[j][i] * v[j];
   }
 }

 return res;
}

int *bytes_to_bits(char *message, int sz) {
 int *res = calloc(sz * 8, sizeof(int));

 for (int i = 0; i < sz; i++) {
   for (int j = 0; j < 8; j++) {
     res[(i * 8) + j] = (message[i] >> (7 - j)) & 0x1;
   }
 }

 return res;
}

char *bits_to_bytes(int *message, int sz) {
 char *res = calloc(sz / 8, 1);

 for (int i = 0; i < sz; i += 8) {
   for (int j = 0; j < 8; j++) {
     res[i / 8] |= ((message[i + j] << (7 - j)));
   }
 }

 return res;
}

char *decode(int *message) {
 int *out = malloc((48 / 3) * sizeof(int));
 int decoded_bit;
```

```
 for (int i = 0; i < 48 / 3; i++) {
   if (message[i * 3] == message[i * 3 + 1])
     decoded_bit = message[i * 3];
   else if (message[i * 3] == message[i * 3 + 2])
     decoded_bit = message[i * 3];
   else if (message[i * 3 + 1] == message[i * 3 + 2])
     decoded_bit = message[i * 3 + 1];
   else
     assert(0);
   out[i] = decoded_bit;
 }

 char *res = bits_to_bytes(out, 48 / 3);
 free(out);

 return res;
}

void hex(char *s, int len) {
 for (int i = 0; i < len; i++) {
   printf("%02hhx", s[i]);
 }
 puts("");
}

int* readCipher(int colls[48][48],const char* filename){
 FILE *fp;
 int* y;
 char buffer[294 - 6];
 char y_buf[6];
 fp = fopen(filename, "rb");

 for (int j = 0; j < 294 - 6; j += 6) {
   char b[6];
   fread(b, 6, 1, fp);
   int *cb = bytes_to_bits(b, 6);
   memcpy(colls[j / 6], cb, 48 * sizeof(int));
   free(cb);
 }
 fread(y_buf, sizeof(y_buf), 1, fp);
 y = bytes_to_bits(y_buf, sizeof(y_buf));
 fclose(fp);
 return y;
}
char* decrypt(int key[48],int* y,int colls[48][48]) {


 int *aux = matrix_vector_multiply(colls, key);
 int m_xor_e[48];
 for (int i = 0; i < 48; i++) {
   m_xor_e[i] = y[i] ^ aux[i];
 }
 char *m = decode(m_xor_e);
 free(aux);

 return m;
}
void decryptflag(int key[48]) {
 FILE *fp;
 char filename[64];
 char buffer[294 - 6];
 char y_buf[6];

 int colls[48][48];

 for (int i = 0; i < 31; i++) {
   sprintf(filename, "./data/flag_%02d", i);
   fp = fopen(filename, "rb");
```

```
   for (int j = 0; j < 294 - 6; j += 6) {
     char b[6];
     fread(b, 6, 1, fp);
     int *cb = bytes_to_bits(b, 6);
     memcpy(colls[j / 6], cb, 48 * sizeof(int));
     free(cb);
   }

   fread(y_buf, sizeof(y_buf), 1, fp);
   int *y;
   y = bytes_to_bits(y_buf, sizeof(y_buf));
   int *aux = matrix_vector_multiply(colls, key);
   int m_xor_e[48];

   for (int i = 0; i < 48; i++) {
     m_xor_e[i] = y[i] ^ aux[i];
   }
   char *m = decode(m_xor_e);
   // hex(m, 2);
   printf("%s", m);
   free(aux);
   free(y);
   fclose(fp);
 }
 puts("");
}

int* y_1;
int colls1[48][48];
int* y_2;
int colls2[48][48];
int check(int* key){
 //only read once
 if(!y_1){
   y_1 = readCipher(colls1,"./data/ciphertext_001");
   y_2 = readCipher(colls2,"./data/ciphertext_000");
 }
 char* result = decrypt(key,y_1,colls1);

 if (result[0] == (char)0x62 && result[1] == (char)0xd4) {

   result = decrypt(key,y_2,colls2);
   if(result[0]==(char)0x16 && result[1]==(char)0xdb){
     free(result);
     return 1;
   }
 }
 free(result);
 return 0;

}
int* recover_key(int mask[48]) {
 int key_start[48];
 for (int i = 0; i < 48; i++) {
   key_start[i] = flag0xorflagcipher0[i] ^ mask[i];
 }

 int *key = matrix_vector_multiply(cols0Inv, key_start);

 return key;

}

int main() {
 unsigned long total=pow(3,16);
 int mask[48];
 unsigned long i = 0;
 for(;i<total;i++){
```

```
  if(i%400000 ==0){
    printf("%ld\n",i);
  }
  //clear mask
  for(int tmp=0;tmp<48;tmp++){
    mask[tmp]=0;
  }

  //set mask
  int num=i;
  for(int tmp=0;tmp<16;tmp++){
    mask[tmp*3+num%3]=1;
    num=num/3;
  }
  int* key=recover_key(mask);
  if(check(key)){
    decryptflag(key);
  }
  free(key);
 }

}
```

flag 35C3_let's_hope_these_Q_computers_will_wait_until_we're_ready

点击收藏 | 0 关注 | 1

上一篇：使用基于Web的攻击手法发现并入侵... 下一篇：从Self-XSS到可利用的xss

1. 1 条回复

actanble 2019-01-08 11:23:47

兄弟，这个网站是用的那个开源工具搭建的啊， 我搭建过 nodebb, phpbb, flarum 好像都没你这个好看，

0 回复Ta

登录 后跟帖

先知社区

现在登录

热门节点

技术文章

社区小黑板

目录

RSS 关于社区 友情链接 社区小黑板