rand

发表于 2021-01-21 分类于 Challenge， 2020， 网鼎杯， 白虎场， Crypto
Challenge | 2020 | 网鼎杯 | 白虎场 | Crypto | rand

点击此处获得更好的阅读体验

WriteUp来源

本WP由Vanish提供

题目考点

- *MT19937预测key*

解题思路

啊，一道憨憨题，mt19937都快给玩烂了。而且这还是最最简单的预测，直接套上现成的工具就能解key。至于后面的一个只给了加密函数没给解密函数用的feistel的密码，可能是DES吧，没细看。直接把密钥流反过来，用他给的加密函数来解密就好了。

```
import random

class MT19937Recover:
    """Reverses the Mersenne Twister based on 624 observed outputs.

    The internal state of a Mersenne Twister can be recovered by observing
    624 generated outputs of it. However, if those are not directly
    observed following a twist, another output is required to restore the
    internal index.

    See also https://en.wikipedia.org/wiki/Mersenne_Twister#Pseudocode .

    """
    def unshiftRight(self, x, shift):
        res = x
        for i in range(32):
            res = x ^ res >> shift
        return res

    def unshiftLeft(self, x, shift, mask):
        res = x
        for i in range(32):
            res = x ^ (res << shift & mask)
        return res

    def untemper(self, v):
        """ Reverses the tempering which is applied to outputs of MT19937 """

        v = self.unshiftRight(v, 18)
        v = self.unshiftLeft(v, 15, 0xefc60000)
        v = self.unshiftLeft(v, 7, 0x9d2c5680)
        v = self.unshiftRight(v, 11)
        return v

    def go(self, outputs, forward=True):
        """Reverses the Mersenne Twister based on 624 observed values.

        Args:
            outputs (List[int]): list of >= 624 observed outputs from the PRNG.
                However, >= 625 outputs are required to correctly recover
                the internal index.
            forward (bool): Forward internal state until all observed outputs
                are generated.

        Returns:
            Returns a random.Random() object.
        """

        result_state = None

        assert len(outputs) >= 624       # need at least 624 values

```

```
        ivals = []
        for i in range(624):
            ivals.append(self.untemper(outputs[i]))

        if len(outputs) >= 625:
            # We have additional outputs and can correctly
            # recover the internal index by bruteforce
            challenge = outputs[624]
            for i in range(1, 626):
                state = (3, tuple(ivals+[i]), None)
                r = random.Random()
                r.setstate(state)

                if challenge == r.getrandbits(32):
                    result_state = state
                    break
        else:
            # With only 624 outputs we assume they were the first observed 624
            # outputs after a twist -->  we set the internal index to 624.
            result_state = (3, tuple(ivals+[624]), None)
        rand = random.Random()
        rand.setstate(result_state)

        if forward:
            for i in range(624, len(outputs)):
                assert rand.getrandbits(32) == outputs[i]

        return rand

def test_PythonMT19937Recover():
    """Just a testcase to ensure correctness"""
    mtb = MT19937Recover()

    #r1 = random.Random(0x31337)

    # just some discarded random numbers to move internal state forward
    #[r1.getrandbits(32) for _ in range(1234)]

    # the actual leak of 1000 values
    #n = [r1.getrandbits(32) for _ in range(1000)]
    with open ("random") as f:
        nn= f.read().split("\n")
        print(nn)
        n = [int(_) for _ in nn[:-1]]
    r2 = mtb.go(n)
    print (r2.getrandbits(32))

    #assert r1.getrandbits(32) == r2.getrandbits(32)

test_PythonMT19937Recover()
```

拿到key，改加密函数，解密得到flag

```
# -*- coding: utf-8 -*-
import base64
import random

flag = "flag{**********************************}"

hexadecimalcontrast = {
    '0': '0000',
    '1': '0001',
    '2': '0010',
    '3': '0011',
    '4': '0100',
    '5': '0101',
    '6': '0110',
    '7': '0111',
    '8': '1000',
    '9': '1001',
    'a': '1010',
    'b': '1011',
    'c': '1100',
    'd': '1101',
    'e': '1110',
    'f': '1111',
}
```

```
IP = [58, 50, 42, 34, 26, 18, 10, 2,
      60, 52, 44, 36, 28, 20, 12, 4,
      62, 54, 46, 38, 30, 22, 14, 6,
      64, 56, 48, 40, 32, 24, 16, 8,
      57, 49, 41, 33, 25, 17, 9, 1,
      59, 51, 43, 35, 27, 19, 11, 3,
      61, 53, 45, 37, 29, 21, 13, 5,
      63, 55, 47, 39, 31, 23, 15, 7]
IP_1 = [40, 8, 48, 16, 56, 24, 64, 32,
        39, 7, 47, 15, 55, 23, 63, 31,
        38, 6, 46, 14, 54, 22, 62, 30,
        37, 5, 45, 13, 53, 21, 61, 29,
        36, 4, 44, 12, 52, 20, 60, 28,
        35, 3, 43, 11, 51, 19, 59, 27,
        34, 2, 42, 10, 50, 18, 58, 26,
        33, 1, 41, 9, 49, 17, 57, 25]
E = [32, 1, 2, 3, 4, 5,
     4, 5, 6, 7, 8, 9,
     8, 9, 10, 11, 12, 13,
     12, 13, 14, 15, 16, 17,
     16, 17, 18, 19, 20, 21,
     20, 21, 22, 23, 24, 25,
     24, 25, 26, 27, 28, 29,
     28, 29, 30, 31, 32, 1]
S = [[15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10,
      3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5,
      0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15,
      13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9, ],
     [10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8,
      13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1,
      13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7,
      1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12, ],
     [14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7,
      0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8,
      4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0,
      15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13, ],
     [7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15,
      13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9,
      10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4,
      3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14, ],
     [2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9,
      14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6,
      4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14,
      11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3, ],
     [12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11,
      10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8,
      9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6,
      4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13, ],
     [4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1,
      13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6,
      1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2,
      6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12, ],
     [13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7,
      1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2,
      7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8,
      2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11, ]]
PC_1 = [57, 49, 41, 33, 25, 17, 9, 1,
        58, 50, 42, 34, 26, 18, 10, 2,
        59, 51, 43, 35, 27, 19, 11, 3,
        60, 52, 44, 36, 63, 55, 47, 39,
        31, 23, 15, 7, 62, 54, 46, 38,
        30, 22, 14, 6, 61, 53, 45, 37,
        29, 21, 13, 5, 28, 20, 12, 4, ]
PC_2 = [14, 17, 11, 24, 1, 5, 3, 28,
        15, 6, 21, 10, 23, 19, 12, 4,
        26, 8, 16, 7, 27, 20, 13, 2,
        41, 52, 31, 37, 47, 55, 30, 40,
        51, 45, 33, 48, 44, 49, 39, 56,
        34, 53, 46, 42, 50, 36, 29, 32, ]
P = [16, 7, 20, 21,
     29, 12, 28, 17,
     1, 15, 23, 26,
     5, 18, 31, 10,
     2, 8, 24, 14,
     32, 27, 3, 9,
     19, 13, 30, 6,
     22, 11, 4, 25, ]
movnum = [1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1]

```

```
104 def HexToBin(string):
105     "Convert sixteen to binary"
106 
107     Binstring = ""
108     string = string.lower()
109     for i in string:
110         try:
111             Binstring += hexadecimalcontrast[i]
112         except:
113             return -1
114     return Binstring
115 
116 def BinToStr(strbin):
117     "Turn the binary string to a ASCII string"
118 
119     strten = ""
120     for i in range(len(strbin) // 8):
121         num = 0
122         test = strbin[i * 8:i * 8 + 8]
123         for j in range(8):
124             num += int(test[j]) * (2**(7 - j))
125         strten += chr(num)
126     return strten
127 
128 def StrToHex(string):
129     "Converts a string to HEX"
130 
131     hexStr = ''
132     for i in string:
133         tmp = str(hex(ord(i)))
134         if len(tmp) == 3:
135             hexStr += tmp.replace('0x', '0')
136         else:
137             hexStr += tmp.replace('0x', '')
138     return hexStr
139 
140 def Binxor(string1, string2):
141     "If the length is different, only the short one is returned."
142 
143     strlen = 0
144     xorstr = ""
145     if len(string1) > len(string2):
146         strlen = len(string2)
147     else:
148         strlen = len(string1)
149     for i in range(strlen):
150         if string1[i] == string2[i]:
151             xorstr += '0'
152         else:
153             xorstr += '1'
154     return xorstr
155 
156 def SubstitutionBox(keyfield, sub):
157 
158     newkeyfield = ''
159     for i in range(len(sub)):
160         newkeyfield += keyfield[sub[i] - 1]
161     return newkeyfield
162 
163 def SubkeyGeneration(freq, C, D):
164 
165     for i in range(movnum[freq]):
166         C = C[1:] + C[:1]
167         D = D[1:] + D[:1]
168     return C, D, SubstitutionBox(C + D, PC_2)
169 
170 def enkey(secretkey):
171 
172     netss = SubstitutionBox(HexToBin(StrToHex(secretkey)), PC_1)
173     C, D = netss[:28], netss[28:]
174     key = []
175     for i in range(16):
176         C, D, keyone = SubkeyGeneration(i, C, D)
177         key.append(keyone)
178     return key
179 
180 def Sbox(plaintext, sub):
181 
182     return HexToBin("%x" % S[sub][int(plaintext[:1] + plaintext[-1:], 2) * 16 + int(plaintext[1:-1], 2)])
```

```
183
184 def Function(plaintext, secretkey):
185
186     plaintext = Binxor(SubstitutionBox(plaintext, E),
187                        secretkey)
188     sout = ''
189     for i in range(8):
190         sout += Sbox(plaintext[i * 6:(i + 1) * 6], i)
191     sout = SubstitutionBox(sout, P)
192     return sout
193
194 def endecrypt(plaintext, secretkey):
195
196     netss = SubstitutionBox(HexToBin(StrToHex(plaintext)), IP)
197     L, R = netss[:32], netss[32:]
198     for i in range(16):
199         L, R = R, Binxor(L, Function(R, secretkey[i]))
200     return SubstitutionBox(R + L, IP_1)
201
202 def encryption(plaintext, secretkey):
203
204     plaintext = plaintext + (8 - len(plaintext) % 8) * '\0'
205     keys = enkey(secretkey[:8])
206     print(keys)
207     ciphertext = ''
208     for i in range(len(plaintext) / 8):
209         ciphertext += endecrypt(plaintext[i * 8:(i + 1) * 8], keys)
210     return base64.b64encode(BinToStr(ciphertext))
211
212 def decryption(plaintext, secretkey):
213     plaintext = "t2GzuEfVDaJsNLBqC8N7C3/2UgIeCoQLC2qLkT1ukFULskzMc1u9QeFizVUfFYfA"
214     plaintext = base64.b64decode(plaintext)
215     print plaintext
216     keys = enkey(secretkey[:8])[::-1]
217     #print(keys)
218     ciphertext = ''
219     for i in range(len(plaintext) / 8):
220         ciphertext += endecrypt(plaintext[i * 8:(i + 1) * 8], keys)
221     return (BinToStr(ciphertext))
222
223 def generate():
224     fw = open("random", "w")
225     for i in range(700):
226         fw.write(str(random.getrandbits(32))+"\n")
227     fw.close()
228
229 generate()
230 f = open("flag.txt", "r")
231
232 key = str(random.getrandbits(32))
233 key = '2990136190'
234 ciphertext = decryption(flag, key)
235 print ciphertext
```

Flag

- ***本文作者：*** *CTFHub*
- ***本文链接：*** *https://writeup.ctfhub.com/Challenge/2020/网鼎杯/白虎场/Crypto/uDCoGsHmHe8SfLoTMwQ5SU.html*
- ***版权声明：*** *本博客所有文章除特别声明外，均采用 BY-NC-SA 许可协议。转载请注明出处！*

 # Challenge # 2020 # Crypto # 网鼎杯 # 白虎场
PWN5
simple