【CTF WriteUp】2020全国工业互联网安全技术技能大赛密码题（Crypto）wp

原创

EDI安全 于 2020-10-24 23:25:35 发布 2724 收藏 4

分类专栏： CTF-Writeup 文章标签： 安全 经验分享

本文链接：https://blog.csdn.net/qq_45603443/article/details/109267707

CTF-Writeup 专栏收录该内容

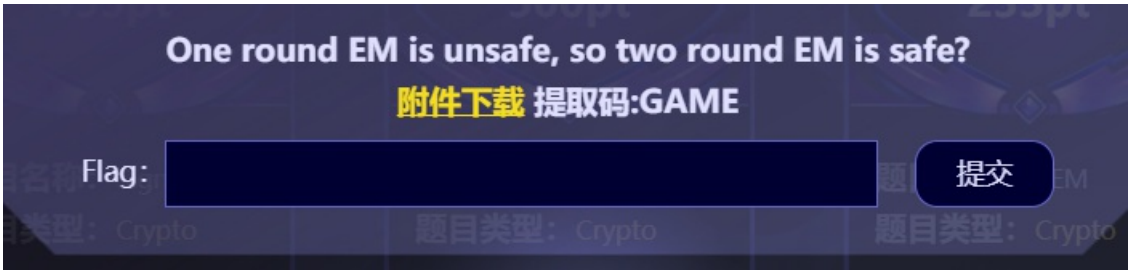
13 篇文章 2 订阅

订阅专栏

提示：文章写完后，目录可以自动生成，如何生成可参考右边的帮助文档

2020全国工业互联网安全技术技能大赛密码题（Crypto）wp

一、题目 2E



二、根据题目给脚本推出EXP如下：

1.Task.py

代码如下（示例）：

```
import random
from random import Random
import mt

# from secret import flag
flag = 'flag:{}'
from Crypto.Util.number import bytes_to_long, long_to_bytes

pbox1 = [22, 28, 2, 21, 3, 26, 6, 14, 7, 16, 15, 9, 17, 19, 8, 11, 10, 1, 13, 31, 23, 12, 0, 27, 4, 18, 30, 29,
24, 20, 5, 25]
pbox2 = [17, 6, 7, 27, 4, 20, 11, 22, 2, 19, 9, 24, 23, 31, 15, 10, 18, 28, 5, 0, 16, 29, 25, 8, 3, 21, 30, 12,
14, 13,1, 26]
def p(data, pbox):
    tmp = bin(data)[2:].rjust(32, '0')
    out = [tmp[x] for x in pbox]
    return int(''.join(out), 2)
def getp(data, pbox):
    tmp = bin(data)[2:].rjust(32, '0')
    _out = {}
    c = 0
    for x in pbox:
        _out[x] = tmp[c]
        c += 1
    out = [_out[x] for x in range(len(pbox))]
    return int(''.join(out), 2)
def genkey(l):
    return random.getrandbits(l)
def encrypt(key, msg):
    tmp1 = p(msg ^ key, pbox1)
    tmp2 = p(tmp1 ^ key, pbox2)
    return tmp2 ^ key
def getcrypt(key, enc):
    tmp1 = enc ^ key
    tmp2 = getp(tmp1, pbox2) ^ key
    return getp(tmp2, pbox1) ^ key
with open('data') as f:
    d = f.readlines()
def getOldStates(states):
    for i in range(3, -1, -1):
        tmp = states[i + 624] ^ states[i + 397]
        if tmp & 0x80000000 == 0x80000000:
            tmp ^= 0x9908b0df
        res = (tmp & 0x40000000) << 1
        tmp = states[i - 1 + 624] ^ states[i + 396]
        if tmp & 0x80000000 == 0x80000000:
            tmp ^= 0x9908b0df
            res |= 1
        res |= (tmp & 0x3fffffff) << 1
        # assert(res == states[i])
        states[i] = res
cur = []
for i in d:
    ii = i.split()
    if len(ii) > 1:
        cur.append(int(ii[0]))
        if len(cur) == 624:
            break
        '''
```

```
        if p(0^int(ii[0]),pbox2) ^ int(ii[0]) == int(ii[1]):
            print(ii)
            key = ii[0]
            break
        '''

states = [0] * 4 + mt.backtrack(cur)
len(states)
getOldStates(states)
random.setstate(tuple([3, tuple(states[:624] + [0]), None]))

key = random.getrandbits(32)
key = random.getrandbits(32)
key = random.getrandbits(32)
key = random.getrandbits(32)
```

2. print(key)

来四次就是正常的key了

代码如下（示例）：

```
key = 1991722937
cc = b''
for i in d:
 ii = i.split()
 if len(ii) > 1:
 break
 else:
 print(long_to_bytes(getcrypt(key, int(ii[0]))))
 cc += long_to_bytes(getcrypt(key, int(ii[0])))
```

3. print(1)

```
key = genkey(32)
getcrypt(key,encrypt(key,3000000000))
flag = flag.ljust(44,'\x00')
for i in range(int(len(flag)/4)):
 pt = bytes_to_long(flag[i*4:i*4+4])
 print(encrypt(key,pt))
''
for i in range(2**22):
 pt = random.getrandbits(32)
 ct = encrypt(key,pt)
 print(pt,ct)
```

4. mt.py

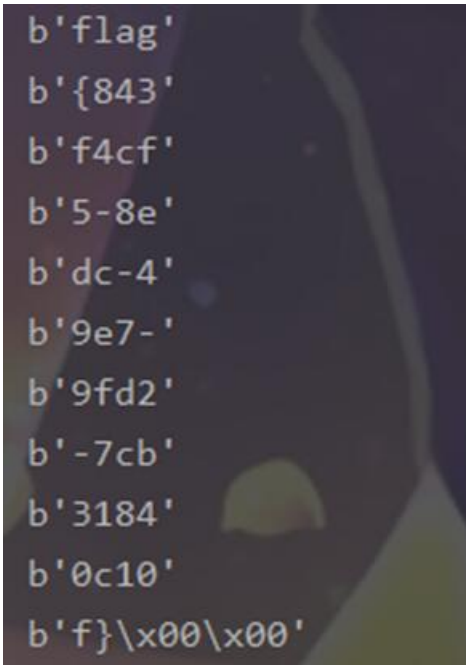
```
from os import urandom
class MersenneTwister(object):
    """
    Implements a basic Mersenne Twister PRNG
    """
    def __init__(self, seed):
        self.index = 624
        self.mt = [seed] + [0] * 623
        for i in range(1, 624):
            self.mt[i] = 0xffffffff & (0x6c078965 * (self.mt[i - 1] ^ self.mt[i - 1] >> 30) + i)
```

```
    def setstate(self, state):
        self.mt = state
    def getstate(self):
        return self.mt
    def random(self):
        """
        Returns floating point number from 0.00 to 1.00
        Mimics Python's random.random()
        """
        a = self.random32() >> 5
        b = self.random32() >> 6
        return float((a * 67108864.0 + b) * (1.0 / 9007199254740992.0))
    def getrandbits(self, n):
        """
        Alias to random 32, only accepts 32 bits
        """
        assert n == 32
        return self.random32()
    def twist(self):
        for k in range(624):
            y = (self.mt[k] & 0x80000000) | (self.mt[(k+1) % 624] & 0x7fffffff)
            n = 0x9908b0df if y % 2 else 0
            self.mt[k] = self.mt[(k+397) % 624] ^ (y >> 1) ^ n
        self.index = 0
    def random32(self):
        """
        Returns the next 32 bit random number
        """
        if self.index >= 624:
            self.twist()
        # generate random number
        y = self.mt[self.index]
        self.index += 1
        y ^= (y >> 11)
        y ^= (y << 7) & 0x9d2c5680
        y ^= (y << 15) & 0xefc60000
        y ^= (y >> 18)
        return y
class PyRand(MersenneTwister):
    """
    Emulates Python's random, seed is required
    """
    def __init__(self, seed):
        key = []
        super(PyRand, self).__init__(19650218)
        while seed:
            key.append(seed & 0xffffffff)
            seed = seed >> 32
        if len(key) == 0:
            key = [0]
        mt = self.mt
        i = 1
        j = 0
        m = max(624, len(key))
        for k in xrange(m, 0, -1):
            mt[i] = 0xffffffff & ((mt[i] ^ ((mt[i-1] ^ (mt[i-1] >> 30)) * 1664525)) + key[j] + j)
            i += 1
            if i >= 624:
                mt[0] = mt[623]
```

```
                i = 1
            j = (j+1) % len(key)
        for k in xrange(623, 0, -1):
            mt[i] = 0xffffffff & ((mt[i] ^ ((mt[i-1] ^ (mt[i-1] >> 30)) * 1566083941)) - i)
            i += 1
            if i >= 624:
                mt[0] = mt[624-1]
                i = 1
        mt[0] = 0x80000000
class MTServer(object):
    def __init__(self, seed, rand=PyRand):
        self.seed = seed
        self.rand = rand
        self.state = []
    def start(self):
        """
        Returns a generator of PRNG numbers
        """
        r = self.rand(self.seed)
        self.state = r.mt
        while 1:
            yield r.getrandbits(32)
def reverse_rshift_xor(y, shift):
    i = 0;
    # iterate over every bit-shifted section
    while (i * shift < 32):
        # Get bits to shift for bit section
        unshift = y & (((0xffffffff << (32 - shift)) & 0xffffffff) >> (shift * i))
        # Reverse right shift
        unshift = unshift >> shift
        # Reverse xor
        y ^= unshift
        i += 1
    return y
def reverse_lshift_xor(y, shift, mask):
    i = 0
    # iterate over every bit-shifted section
    while (i * shift < 32):
        # Git bits to shift for bit section
        unshift = y & (((0xffffffff >> (32 - shift))) << (shift * i))
        # Reverse left shift
        unshift = (unshift << shift)
        # Reverse mask
        unshift &= mask
        # Reverse xor
        y ^=  unshift
        i += 1
    return y
def backtrack(numbers):
    """
    Returns the current state of the MT PRNG based on list of 624 numbers
    """
    assert len(numbers) == 624
    state = []
    for n in numbers:
        n = reverse_rshift_xor(n, 18)               # reverse: y ^= (y >> 18)
        n = reverse_lshift_xor(n, 15, 0xefc60000)   # reverse: y ^= (y << 15) & 0xefc60000
        n = reverse_lshift_xor(n,  7, 0x9d2c5680)   # reverse: y ^= (y <<  7) & 0x9d2c5680
        n = reverse_rshift_xor(n, 11)               # reverse: y ^= (y >> 11)
        state.append(n)
```

```
    return state
```

得到flag!



flag{843f4cf5-8edc-49e7-9fd2-7cb31840c10f}