n1ctf（部分复现）

原创

Yu4nzi 于 2020-11-05 20:28:11 发布

N1CTF 2020 部分复现

Before

这半个月一连串的比赛，淦，爷要脑溢血了（两眼一黑，直接去世）

导致我现在才复现`N1CTF`

菜狗表示签到和签退好评，淦

复现的时候学到好多，有些东西要找个时间梳理一下了，这里仅仅放一下我的一些理解和想法好了

继续肝ctf!

淦啊！这已经不是wp了，是我复现题目，阿布，是学习过程记录了，菜啊~

好难啊~

Crypto

VSS

源码

```python
#!/usr/bin/python3
import qrcode  # https://github.com/lincolnloop/python-qrcode
import random
import os
from PIL import Image
from flag import FLAG




def vss22_gen(img):
    m, n = img.size
    share1, share2 = Image.new("L", (2*m, 2*n)), Image.new("L", (2*m, 2*n))
    image_data = img.getdata()
    flipped_coins = [int(bit) for bit in bin(random.getrandbits(m*n))[2:].zfill(m*n)]
    for idx, pixel in enumerate(image_data):
        i, j = idx//n, idx % n
        color0 = 0 if flipped_coins[idx] else 255
        color1 = 255 if flipped_coins[idx] else 0
        if pixel:
            share1.putpixel((2*j, 2*i), color0)
            share1.putpixel((2*j, 2*i+1), color0)
            share1.putpixel((2*j+1, 2*i), color1)
            share1.putpixel((2*j+1, 2*i+1), color1)


            share2.putpixel((2*j, 2*i), color0)
            share2.putpixel((2*j, 2*i+1), color0)
            share2.putpixel((2*j+1, 2*i), color1)
            share2.putpixel((2*j+1, 2*i+1), color1)
        else:
            share1.putpixel((2*j, 2*i), color0)
            share1.putpixel((2*j, 2*i+1), color0)
            share1.putpixel((2*j+1, 2*i), color1)
            share1.putpixel((2*j+1, 2*i+1), color1)
```

```
            share2.putpixel((2*j, 2*i), color1)
            share2.putpixel((2*j, 2*i+1), color1)
            share2.putpixel((2*j+1, 2*i), color0)
            share2.putpixel((2*j+1, 2*i+1), color0)
    share1.save('share1.png')
    share2.save('share2.png')




def vss22_superposition():
    share1 = Image.open('share1.png')
    share2 = Image.open('share2.png')
    res = Image.new("L", share1.size, 255)
    share1_data = share1.getdata()
    share2_data = share2.getdata()
    res.putdata([p1 & p2 for p1, p2 in zip(share1_data, share2_data)])
    res.save('result.png')

def main():
    qr = qrcode.QRCode(
        version=1,
        error_correction=qrcode.constants.ERROR_CORRECT_L,
        box_size=12,
        border=4,
    )
    qr.add_data(FLAG)
    qr.make(fit=True)
    img = qr.make_image(fill_color="black", back_color="white")
    vss22_gen(img._img)
    img.save('res.png')
    vss22_superposition()


if __name__ == '__main__':
    main()
```

思路

python库内置一个 `random` 伪随机函数实现 `flipped_coins` 初始化，然后进而影响 `color0、color1` 的值，影响 `share2.png` 的视觉化密码呈现。然后我们根修复 `qr` 据 `share2.png` 修复回 `img` 的信息，就可以修复 `qrcode` 的信息。
首先的是 `random` 内置伪随机化函数使用 `MT19937` 实现的。（拓： `Mersenne TWister` 算法和 `MT19937` 和 `MT19937-64` ）
关于这个是可以使用python库 `randcrack` 库中的 `RandCrack` 函数预测 `random` 模块里伪随机数的生成的值。
然后就是我们知道 `qrcode` 的边界是白色的，由此我们可以获得第一个624个由 `prng` 产生的32位整数

脚本

```
from PIL import Image
from randcrack import RandCrack
import random
share = Image.open('share2.png')
width = share.size[0]//2
res = Image.new('L', (width, width))
bits = ''
for idx in range(width*width-624*32, width*width):
    i, j = idx//width, idx % width
    if share.getpixel((2*j, 2*i)) == 255:
        bits += '0'
    else:
        bits += '1'
rc = RandCrack()
for i in range(len(bits), 0, -32):
    rc.submit(int(bits[i-32:i], 2))
flipped_coins = [int(bit) for bit in bin(rc.predict_getrandbits(width*width-624*32))[2:].zfill(width*width-624*3
2)] + list(map(int, bits))

data = []

for idx in range(width*width):
    i, j = idx//width, idx % width
    if share.getpixel((2*j, 2*i)) == 255:
        data.append(0 if flipped_coins[idx] else 255)
    else:
        data.append(255 if flipped_coins[idx] else 0)

res.putdata(data)
res.save('ans.png')

#n1ctf{bf3724e3-c26b-4a63-9b4f-b33024b1db63}
```

其他

视觉密码方案

这是我在写题包括之后复现过程 `VSS` 理解期间所看的一些文献和参考：
基于迭代算法的可验证视觉密码

异或视觉密码方案目标优化研究

[基于异或解密的（k，n）视觉密码方案](file:///C:/Users/MECHREVO/AppData/Local/Temp/MicrosoftEdgeDownloads/441bfae2-b146-48d9-b9b1-b91efb84da84/(kn)%20visual%20cryptography%20scheme%20based%20on%20XOR%20decryption.pdf)

其实还有，但是懒得列了。

关于视觉密码方案的一些学习和结合这道题的理解：

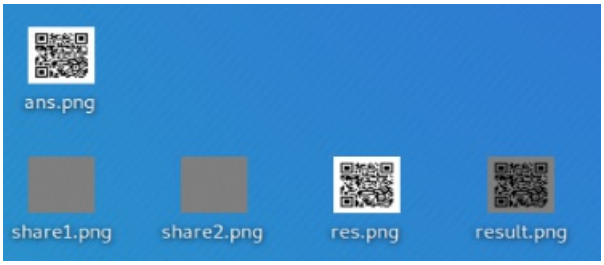
`vss22_gen` 就是秘密图像（flag放进qr里生成秘密图像）通过加密概率矩阵，与加密规则矩阵共同生成两份共享份（`share1.png、share2.png`）

`vss22_superposition` 就是两份共享份通过叠加方式进行恢复（传统的视觉密码恢复）

复现完之后我用 `flag` 再次跑加密脚本，发现通过叠加方式恢复出来的 `result.png` 文件视觉质量较差，然后官方 `wp` 里 `exp`（通过攻击 `random` 随机数，得到加密概率矩阵，进一步恢复秘密图像）恢复出来的 `ans.png` 文件也与原 `res.png` 文件不完全相同，存在一定的相对差且存在白像素不能完全恢复。

另，除叠加方式恢复视觉密码外，还可用异或的解密方式恢复。

就此题而言，无论是叠加还是异或的解密方式，都无法进行解密图片，因为这两种方式都需知道多份共享份，而此题只提供了一份。



Mersenne Twister

马特赛特旋转验算法，伪随机数水生成器，基于有限二进制字段上的矩阵线性再生。
常见变种 `MT19937` 和 `MT19937-64`
算法原理：

> 利用线性反馈寄存器（ `LFSR` ）产生随机数。
> `LFSR` ：
> n
> n位的 `LFSR` 可以在产生周期为 `2^n-1` 的伪随机序列。最大长度的LFSR生成一个m序列，只有具有一定抽头序列的 `LFSR` 才能通过 `2^n-1` 个内部状态，不包括全零。但如果他本身为全零，则其状态不改变。当且仅当抽头序列加上常数1形成的多项式是本原多项式时， `LFSR` 才能最大长度
> 利用上诉产生周期为m的伪随机序列后，将产生的伪随机序列除以序列周期，即可得到（0,10上均匀分布的伪随机序列

梅森旋转

无爆破还原mt_rand()种子

Mersenne Tsister

randcrack库-RandCrack

randcrack库

> 预测python的“random”模块随机生成的值
> cracker不实现 `random（）` 函数的预测，因为它基于基于 `/dev/urandom` 的 `os.urandom` 模块

Easy_RSA

源码

```
from Crypto.Util.number import *
import numpy as np

mark = 3**66

def get_random_prime():
    total = 0
    for i in range(5):
        total += mark**i * getRandomNBitInteger(32)
    fac = str(factor(total)).split(" * ")
    return int(fac[-1])
#factor是因式分解 fac[-1]是取其中一个数，但我不清楚他是取小的还是取大的数

def get_B(size):
    x = np.random.normal(0, 16, size)
    return np.rint(x)
#np.random.normal:[正态分布](https://blog.csdn.net/qq_21763381/article/details/95914372)
#np.rint:四舍五入取整

p = get_random_prime()
q = get_random_prime()
N = p * q
e = 127

flag = b"N1CTF{**********************************}"
secret = np.array(list(flag))
#np.array:[产生数组](https://blog.csdn.net/sinat_28576553/article/details/89047893#1.1%20%E5%87%BD%E6%95%B0%E5%BD
%A2%E5%BC%8F)

upper = 152989197224467
A = np.random.randint(281474976710655, size=(e, 43))
B = get_B(size=e).astype(np.int64)
linear = (A.dot(secret) + B) % upper
# astype:变量类型转换

result = []
for l in linear:
    result.append(pow(l, e, N))

print(result)
print(N)
np.save("A.npy", A)
```

思路

题目说了`RSA`，但并不easy，tm完全不会做，淦

复现的时候，官方给的wp是英文的，真的不习惯，更淦

只能一点一点的复，一点一点的学，顺便巩固一下数论知识，另去年强网杯有一题六个challenge 全是RSA各种攻击，可以一起学了，淦

先分析一下题目给的sage源码，首先是一个get_random_prime的函数，说白他就是一个随机素数发生器，很脆弱，为啥说他脆弱呢，他好像是一个多项式方程

我们将它转换成数学公式表示一下

> `getRandomNBitInteger(32)` 所产生的数设为 v
> mark = 3 * * 66 为 x
> total 就是 a * b

那么数学表达式就是

> a * p = v0 + v1 * x + v2 * x^2 + v3 * x^3 + v4 * x^4

N的数学表达式为

> a *b * N =
> w0 + w1 * x + w2 * x^2 + w3 * x^3 + w4 * x^4 + w5 * x^5 + w6 * x^6 + w7 * x^7 + w8 * x^8

然后我们用`polynomialRing`帮助我们解决。

到这里我们就基本可以分解N，解出p、q了。

然后就是往下看，get_B函数就是一个高斯分布随机数生成器，然后就是p、q、N、e、flag 取值赋值，再往下，就是把flag以一个list的形式放入secret数组，继续搞A、搞B，用A和B处理一下含有flag的secret数组，并将其存入linear里，下面那个for循环就是主要就是rsa处理元素，最后输出就行了。

基本上sage源码我们分析完了。

上面我们已经可以分解N，得到p、q，然后题目的txt文件给了result的输出，我们可以用他和rsa解密得到linear

然后从这一步`linear = (A.dot(secret) + B) % upper`解出secret，官方wp说这个是一个LWE，但我真的真的没有学过格密码，淦

而且看了挺多的，目前只能理解到这里。也只会前面RSA的部分，当时做的时候也只能出来rsa的部分，然后上面解有关N的那个多项式方程好像也是有格的一些知识在，但我打的时候只是因为以前做到过一个也是解类似方程的题，然后解决的。

这个并不Easy的RSA就只能在这了，等之后学了格在更吧，淦

这里涉及一个LLL算法和格的一些知识

脚本

（还不会呢~~只懂了一部分）

其他

基于密码学中格的基础知识

LLL算法：

应用于寻找格中的近似最短向量，本质上说或者更白话一点就是用来寻找离原点比较近的一个格向量。

最初应用在实数域分解多项式（也是这题所涉及到的）以及在固定维度下解决整数规划问题

除此之外

，可以用来寻找整数关系，例如你拿到一个数，like 6.73205080756887……，你觉得很熟悉，那么你可以用这个算法来实现找到该数为 根号3+5的结果这样的一个类似功能，即对于给定数，该算法可以帮助你知道这可能是由哪些数字哪些运算得来的，这让我想到了114514 极其恶臭的数字论证，但对于数字论证目前我所比较明确的思路是枚举或者二叉树中序遍历，不知道LLL可不可以实现，因为两者有一定的相似性，数字论证是给定数n，并给定数组a，求解数组a中各元素之间通过怎样运算得到n，即数和数字元素是已知，运算为未知，而LLL中数已知，数字元素和运算均未知。但我不知道可不可以。（刚学刚接触不久，也不太清楚，路过的大佬轻喷教教窝~）

额……扯远了，继续LLL学习，淦

目前常用密码RSA

LLL算法可以破解基于背包问题的密码学系统，可以破解RSA变种问题，

格密码学方案可以抵御量子计算机的攻击

FlagBot

源码

```
from hashlib import sha256
from Crypto.Cipher import AES
from Crypto.Util.number import long_to_bytes, bytes_to_long
from Crypto.Util.Padding import pad, unpad
import base64
from secret import flag

RECEIVER_NUM = 7

def generate_safecurve():
    while True:
        p = random_prime(2 ^ 256-1, False, 2 ^ 255)
        a = randint(-p, p)
        b = randint(-p, p)

        if 4*a^3 + 27*b^2 == 0:
            continue

        E = EllipticCurve(GF(p), [a, b])

        fac = list(factor(E.order()))

        # Prevent rho method
        if fac[-1][0] < 1 << 80:
            continue

        # Prevent transfer
        for k in range(1, 20):
            if (p ^ k - 1) % fac[-1][0] == 0:
                break
        else:
            return E

class Sender:
    def __init__(self, curves, receivers):
        self.secret = randint(1 << 254, 1 << 255)
        self.curves = curves
        self.receivers = receivers
        self.shared_secrets = [None for _ in range(len(receivers))]
```

```
    def setup_connections(self):
        for idx, receiver in enumerate(self.receivers):
            curve = self.curves[idx]
            print(f"curves[{idx}] : {curve}")
            g = self.curves[idx].gens()[0]
            print(f"g[{idx}] = {g.xy()}")
            receiver.set_curve(curve, g)
            public = self.secret * g
            print(f"S_pub[{idx}] = {public.xy()}")
            yours = receiver.key_exchange(public)
            print(f"R_pub[{idx}] = {yours.xy()}")
            self.shared_secrets[idx] = yours * self.secret

    def send_secret(self):
        msg = b'Hi, here is your flag: ' + flag
        for idx, receiver in enumerate(self.receivers):
            px = self.shared_secrets[idx].xy()[0]
            _hash = sha256(long_to_bytes(px)).digest()
            key = _hash[:16]
            iv = _hash[16:]
            encrypted_msg = base64.b64encode(AES.new(key, AES.MODE_CBC, iv).encrypt(pad(msg, 16)))
            print(f"encrypted_msg[{idx}] = {encrypted_msg}")
            receiver.receive(encrypted_msg)


class Receiver:
    def __init__(self):
        self.secret = randint(1 << 254, 1 << 255)
        self.curve = None
        self.g = None
        self.shared_secret = None

    def set_curve(self, curve, g):
        self.curve = curve
        self.g = g

    def key_exchange(self, yours):
        self.shared_secret = yours * self.secret
        return self.g * self.secret

    def receive(self, encrypted_msg):
        px = self.shared_secret.xy()[0]
        _hash = sha256(long_to_bytes(px)).digest()
        key = _hash[:16]
        iv = _hash[16:]
        msg = AES.new(key, AES.MODE_CBC, iv).decrypt(base64.b64decode(encrypted_msg))
        msg = unpad(msg, 16)
        assert msg.startswith(b'Hi, here is your flag: ')


receivers = [Receiver() for _ in range(RECEIVER_NUM)]
curves = [generate_safecurve() for _ in range(RECEIVER_NUM)]

A = Sender(curves, receivers)
A.setup_connections()
A.send_secret()
```

思路

害，n1ctf的题太难了，我只能一边学一边分析,最后找思路，还是菜狗啊

好，开始学习

一看源码，直接先看到AES，真不戳，这题我应该会做了，美滋滋，结果在线？？？？？

首先是`generate_safecurve`这个函数，他就是用来check一下椭圆曲线序列上是否存在大素数因子（不能存在大素数因子，但可以存在小素数因子）

谢谢，复现不下去了，等格学完之后,再弄吧

After

好难啊~，自卑