# Writeup by Bonfee

## Chall 5

From challenge 4 we have 4 files:

```
0day.py.enc
encrypt.exe
test.py.enc
test.py.raw~
```

`encrypt.exe` is simply a pyinstaller executable, we can unpack it and then obtain the python source code using `uncompyle6`.

```
from datetime import date
from glob import glob
from os import remove

def bytes_to_words(b):
    return [int.from_bytes(b[i:i + 4], 'little') for i in range(0, len(b),
4)]

def words_to_bytes(w):
    return (b'').join([i.to_bytes(4, 'little') for i in w])

def rotate_left(x, n):
    return x << n & 4294967295 | x >> 32 - n & 4294967295

def rotate_right(x, n):
    return x << 32 - n & 4294967295 | x >> n & 4294967295

def pad(b):
    padding = 16 - len(b) % 16
    return b + padding * bytes([padding])

class LEA:

    def __init__(self, key):
        self.deltas = (3287280091, 1147300610, 2044886154, 2027892972,
1902027934,
                       3347438090, 3763270186, 3854829911)
        self.round_keys = self._key_schedule(key)

    def _key_schedule(self, key):
        round_keys = []
        state = bytes_to_words(key)
        for i in range(24):
            state[0] = rotate_left(state[0] ^ rotate_left(self.deltas[(i %
4)], i), 1)
```

```
            state[1] = rotate_left(state[1] ^ rotate_left(self.deltas[(i %
4)], i + 1), 3)
            state[2] = rotate_left(state[2] ^ rotate_left(self.deltas[(i %
4)], i + 2), 6)
            state[3] = rotate_left(state[3] ^ rotate_left(self.deltas[(i %
4)], i + 3), 11)
            round_keys.append((state[0], state[1], state[2], state[1],
state[3], state[1]))

        return round_keys

    def _encrypt_block(self, block):
        state = bytes_to_words(block)
        for i in range(24):
            old_state = state[:]
            state[0] = rotate_left(old_state[0] ^ self.round_keys[i][0] ^
old_state[1] ^ self.round_keys[i][1], 9)
            state[1] = rotate_right(old_state[1] ^ self.round_keys[i][2] ^
old_state[2] ^ self.round_keys[i][3], 5)
            state[2] = rotate_right(old_state[2] ^ self.round_keys[i][4] ^
old_state[3] ^ self.round_keys[i][5], 3)
            state[3] = old_state[0]

        return words_to_bytes(state)

    def encrypt(self, plaintext):
        plaintext = pad(plaintext)
        ciphertext = b''
        for i in range(0, len(plaintext), 16):
            ciphertext += self._encrypt_block(plaintext[i:i + 16])

        return ciphertext


if __name__ == '__main__':
    if date.today() > date.fromisoformat('2022-04-01'):
        try:
            remove('C:\\key.txt')
        except:
            pass

    with open('C:\\key.txt', 'rb') as (f):
        key = f.read()
    lea = LEA(key)
    for path in glob('C:\\exploits\\*.raw'):
        with open(path, 'rb') as (f):
            content = f.read()
        enc = lea.encrypt(content)
        with open(path[:-3] + 'enc', 'wb') as (f):
            f.write(enc)
        remove(path)
```

The script implements a `LEA` cipher, except that during the `_encrypt_block` phase, the modular addition operation has been replaced with a xor operation.
This change now allows us to retrieve the `round_keys` used to encrypt `test.py.raw` and use them to decrypt `0day.py.enc`.
To retrieve the key I used z3 (takes about ~5 seconds):

```
#!/usr/bin/env python3
from z3 import *

def bytes_to_words(b):
    return [int.from_bytes(b[i:i + 4], 'little') for i in range(0, len(b),
4)]

with open("test.py.enc", "rb") as f:
    ct0 = f.read(16)

with open("test.py.raw", "rb") as f:
    pt0 = f.read(16)

solver = Solver()

round_keys = []
for i in range(24):
    a = BitVec(f'key{i}a', 32)
    b = BitVec(f'key{i}b', 32)
    c = BitVec(f'key{i}c', 32)
    d = BitVec(f'key{i}d', 32)
    round_keys.append((a, b, c, b, d, b))

# Encrypt
state = []

pt0_words = bytes_to_words(pt0)
for i in range(len(pt0_words)):
    state.append(BitVecVal(pt0_words[i], 32))

ct0_words = bytes_to_words(ct0)
for i in range(len(ct0_words)):
    ct0_words[i] = BitVecVal(ct0_words[i], 32)

old_state = None

for i in range(24):
    old_state = state[:]
    state[0] = RotateLeft(old_state[0] ^ round_keys[i][0] ^ old_state[1] ^
round_keys[i][1], 9)
    state[1] = RotateRight(old_state[1] ^ round_keys[i][2] ^ old_state[2] ^
round_keys[i][3], 5)
    state[2] = RotateRight(old_state[2] ^ round_keys[i][4] ^ old_state[3] ^
round_keys[i][5], 3)
    state[3] = old_state[0]
```

```
for i in range(4):
    solver.add(state[i] == ct0_words[i])

if solver.check() == unsat:
    print("No solution")
    exit(0)

print("Got solution")

m = solver.model()
for v in sorted ([(d, m[d]) for d in m], key = lambda x: str(x[0])):
    print(v)
```

Once we have the `round_keys` we can write the `_decrypt_block` function and decrypt `0day.py.enc`:

```
def _decrypt_block(self, block):
    state = bytes_to_words(block)
    for i in range(23, -1, -1):
        old_state = state[:]
        state[0] = old_state[3]
        state[1] = rotate_right(old_state[0], 9) ^ self.round_keys[i][0] ^
state[0] ^ self.round_keys[i][1]
        state[2] = rotate_left(old_state[1], 5) ^ self.round_keys[i][2] ^
state[1] ^ self.round_keys[i][3]
        state[3] = rotate_left(old_state[2], 3) ^ self.round_keys[i][4] ^
state[2] ^ self.round_keys[i][5]
    return words_to_bytes(state)
```

**Flag**

```
b"print('Flag:
HXN{1355239c59f759bd56e13b3432a9b49c}')\n\x0b\x0b\x0b\x0b\x0b\x0b\x0b\x0b\x0b\x0
b\x0b"
```