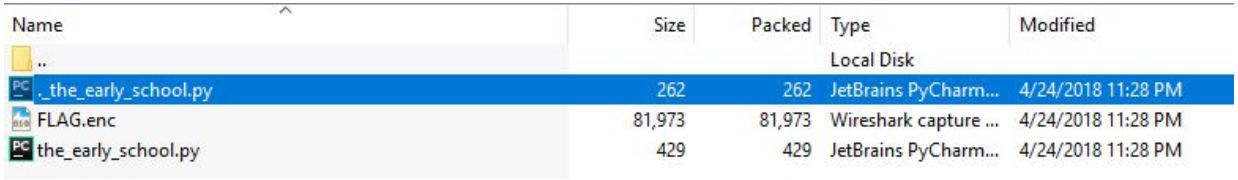
The Early School

Category: Crypto
32 Points
210 Solves
Problem description:
Welcome to ASIS Early School.
Author: Chaocipher
Date: 5/1/2018
Thanks goes to ASIS for organizing the CTF.

Start

I downloaded the file from the description. It's just a 7z file with some files inside. The archive contained three files:

1. A file related to the python script. Didn't need this file so we'll skip that.
2. A file called FLAG.enc.
3. A file called the_early_school.py.

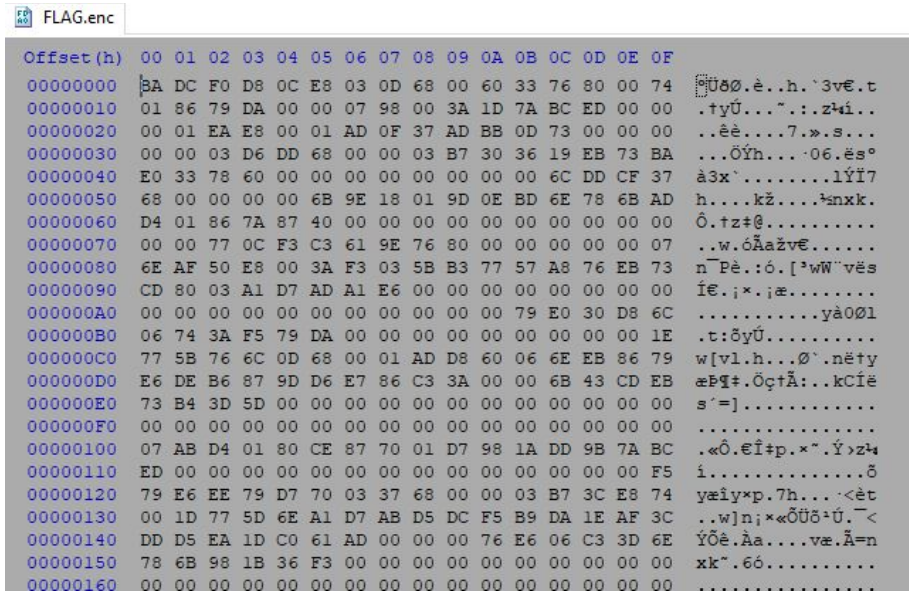


Name	Size	Packed	Type	Modified
..			Local Disk	
._the_early_school.py	262	262	JetBrains PyCharm...	4/24/2018 11:28 PM
FLAG.enc	81,973	81,973	Wireshark capture ...	4/24/2018 11:28 PM
the_early_school.py	429	429	JetBrains PyCharm...	4/24/2018 11:28 PM

FLAG.enc

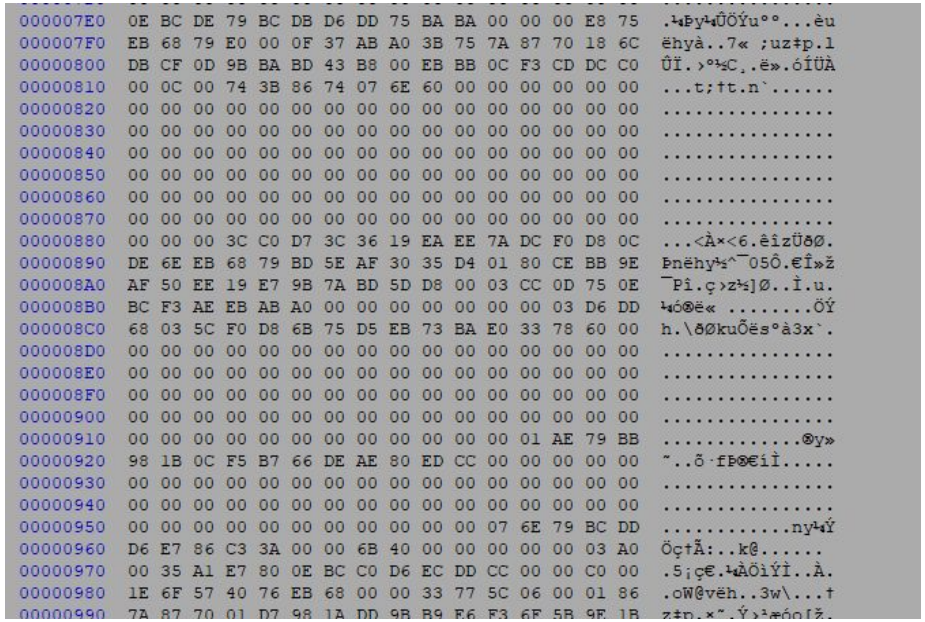
The second file, the FLAG.enc is the encrypted data. I was thinking it's fairly large for a flag, so either it was padded heavily through some operation or it's buried in there somewhere.

This show the top of the file so you can see it doesn't match and magic numbers.


FLAG.enc

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	
00000000	BA	DC	F0	D8	0C	E8	03	0D	68	00	60	33	76	80	00	74	ºÜðØ.è..h.`3v€.t
00000010	01	86	79	DA	00	00	07	98	00	3A	1D	7A	BC	ED	00	00	.†yÚ...˜.:.z¼í..
00000020	00	01	EA	E8	00	01	AD	0F	37	AD	BB	0D	73	00	00	00	..êè....7.».s...
00000030	00	00	03	D6	DD	68	00	00	03	B7	30	36	19	EB	73	BA	...ÖÝh...·06.ës°
00000040	E0	33	78	60	00	00	00	00	00	00	00	00	6C	DD	CF	37	à3x`........lÝÏ7
00000050	68	00	00	00	00	6B	9E	18	01	9D	0E	BD	6E	78	6B	AD	h....kž....½nxk.
00000060	D4	01	86	7A	87	40	00	00	00	00	00	00	00	00	00	00	Ô.†z‡@..........
00000070	00	00	77	0C	F3	C3	61	9E	76	80	00	00	00	00	00	07	..w.óÃažv€.....
00000080	6E	AF	50	E8	00	3A	F3	03	5B	B3	77	57	A8	76	EB	73	n¯Pè.:ó.[³wW¨vës
00000090	CD	80	03	A1	D7	AD	A1	E6	00	00	00	00	00	00	00	00	Í€.¡×.¡æ........
000000A0	00	00	00	00	00	00	00	00	00	00	00	79	E0	30	D8	6C	...........yà0Øl
000000B0	06	74	3A	F5	79	DA	00	00	00	00	00	00	00	00	00	1E	.t:õyÚ..........
000000C0	77	5B	76	6C	0D	68	00	01	AD	D8	60	06	6E	EB	86	79	w[vl.h...Ø`.në†y
000000D0	E6	DE	B6	87	9D	D6	E7	86	C3	3A	00	00	6B	43	CD	EB	æÞ¶‡.Öç†Ã:..kCÍë
000000E0	73	B4	3D	5D	00	00	00	00	00	00	00	00	00	00	00	00	s´=]............
000000F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00000100	07	AB	D4	01	80	CE	87	70	01	D7	98	1A	DD	9B	7A	BC	.«Ô.€Î‡p.×˜.Ý›z¼
00000110	ED	00	00	00	00	00	00	00	00	00	00	00	00	00	00	F5	í..............õ
00000120	79	E6	EE	79	D7	70	03	37	68	00	00	00	03	B7	3C	E8	74	yæîy×p.7h...·<èt
00000130	00	1D	77	5D	6E	A1	D7	AB	D5	DC	F5	B9	DA	1E	AF	3C	..w]n¡×«ÕÜõ¹Ú.¯<
00000140	DD	D5	EA	1D	C0	61	AD	00	00	00	76	E6	06	C3	3D	6E	ÝÕê.Àa....væ.Ã=n
00000150	78	6B	98	1B	36	F3	00	00	00	00	00	00	00	00	00	00	xk˜.6ó..........
00000160	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....

This next shot shows a lot of blank space in the file. I noticed this going all the way down the file this is why I expected padding to be performed.



```
000007E0  0E BC DE 79 BC DB D6 DD 75 BA BA 00 00 00 E8 75  .¼Þy¼ÛÖÝu°°...èu
000007F0  EB 68 79 E0 00 0F 37 AB A0 3B 75 7A 87 70 18 6C  ëhyà..7« ;uz‡p.l
00000800  DB CF 0D 9B BA BD 43 B8 00 EB BB 0C F3 CD DC C0  ÛÏ.›°½C¸.ë».óÍÜÀ
00000810  00 0C 00 74 3B 86 74 07 6E 60 00 00 00 00 00 00  ...t;†t.n`......
00000820  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000830  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000840  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000850  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000860  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000870  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000880  00 00 00 3C C0 D7 3C 36 19 EA EE 7A DC F0 D8 0C  ...<À×<6.êîzÜðØ.
00000890  DE 6E EB 68 79 BD 5E AF 30 35 D4 01 80 CE BB 9E  Þnëhy½^¯05Ô.€Î»ž
000008A0  AF 50 EE 19 E7 9B 7A BD 5D D8 00 03 CC 0D 75 0E  ¯Pî.ç›z½]Ø..Ì.u.
000008B0  BC F3 AE EB AB A0 00 00 00 00 00 00 00 03 D6 DD  ¼ó®ë« ........ÖÝ
000008C0  68 03 5C F0 D8 6B 75 D5 EB 73 BA E0 33 78 60 00  h.\ðØkuÕësºà3x`.
000008D0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000008E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000008F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000900  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000910  00 00 00 00 00 00 00 00 00 00 00 00 01 AE 79 BB  ..............®y»
00000920  98 1B 0C F5 B7 66 DE AE 80 ED CC 00 00 00 00 00  ˜..õ·fÞ®€íÌ.....
00000930  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000940  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000950  00 00 00 00 00 00 00 00 00 00 00 00 07 6E 79 BC DD  .............ny¼Ý
00000960  D6 E7 86 C3 3A 00 00 6B 40 00 00 00 00 00 03 A0  Öç†Ã:..k@.......
00000970  00 35 A1 E7 80 0E BC C0 D6 EC DD CC 00 00 C0 00  .5¡ç€.¼ÀÖìÝÌ..À.
00000980  1E 6F 57 40 76 EB 68 00 00 33 77 5C 06 00 01 86  .oW@vëh..3w\...†
00000990  7A 87 70 01 D7 98 1A DD 9B B9 E6 F3 6F 5B 9E 1B  z‡p.×˜.Ý›¹æóo[ž.
```

The_early_school.py

I worked on this file for a while. Time for honesty; I didn't understand right away that this was the file used by ASIS to build the encrypted file in the first place. I thought this would take in the data and overwrite it with good output leaving the flag. In hindsight that was dumb and now I have a much better feel about how these things are done, but never the less, I spent a bunch of time trying to get this thing to run on my machine until it finally hit me what was going on here.

Essentially, what's going on here is that the encryption file is being built from a string flag that has been imported. It's converted to binary and sent to the encryption function. The encryption function is reading the binary in chunks of two bits and performing some math on it.

```
#!/usr/bin/python

from Crypto.Util.number import *
from flag import FLAG, round

def encrypt(msg):
    assert set(msg) == set(['0', '1'])
    enc = [msg[i:i+2] + str(int(msg[i]) ^ int(msg[min(i+1, len(msg)-1)])) for i in range(0, len(msg), 2)]
    return ''.join(enc)

ENC = bin(bytes_to_long(flag))[2:]

for _ in xrange(round):
    ENC = encrypt(ENC)

fp = open('FLAG.enc', 'w')
fp.write(long_to_bytes(int(ENC, 2)))
fp.close()
```

Solver.py

Here is my code to solve the problem. I left some code in there commented showing how I printed out each component to get a better feel for the structure of the data. I also left the encryption function in here just for reference, but it's not used.

So, I first tried to reverse the math, but couldn't get that figure out very easily since the chunk and the math output is concatenated. So, I figured I would just bruteforce it. By taking in only two bits, the output was finite. It really could only be four different strings coming out. So, I decide to write my solver to look for those strings of 3 bits and "case select" my way to the original binary chunk. Running that 19 rounds gives out the flag. See below for the solving code.

```
from Crypto.Util.number import *
import binascii
import argparse

# Parser structure
varArgParser = argparse.ArgumentParser()
varArgParser.add_argument("--flag", help="nothing to do here")
varArgParser.add_argument("--rounds", help="nothing to do here")
varArgParser.add_argument("--Debug", help="Set the level of debugging. {DEBUG|INFO|WARNING|ERROR|CRITICAL}")
varArgs = varArgParser.parse_args()

def encrypt(msg):
    assert set(msg) == set(['0', '1'])
    enc = [msg[i:i+2] + str(int(msg[i]) ^ int(msg[min(i+1, len(msg)-1)])) for i in range(0, len(msg), 2)]
    return ''.join(enc)

def decrypt(msg):
    # dnc = [msg[i:i+2] + str(int(msg[i]) ^ int(msg[min(i+1, len(msg)-1)])) for i in range(0, len(msg), 2)]
    # for i in range(0, len(msg), 2):
        # print("i: {}".format(i))
        # varOp1 = msg[i:i+2]
        # print("varOp1 {}".format(varOp1))
        # varOp2 = int(msg[i])
        # print("varOp2 {}".format(varOp2))
        # varOp3 = int(msg[min(i+1, len(msg)-1)])
        # print("varOp3 {}".format(varOp3))
```

```
        # varOp4 = varOp1 + str(varOp2 ^ varOp3)
        # print("varOp4 {}".format(varOp4))

    # print("Len of msg: {}".format(len(msg)))
    # print("msg: {}".format(msg))

    varNewBin = ''
    # print("First few bits: {}".format(msg[0:30]))
    for i in range(0, len(msg)-1, 3):
        varChnk = str(msg[i:i+3])
        if varChnk == '000':
            # print('00')
            varNewBin = varNewBin + '00'
        elif varChnk == '001':
            # print('01')
            varNewBin = varNewBin + '01'
        elif varChnk == '110':
            # print('11')
            varNewBin = varNewBin + '11'
        elif varChnk == '101':
            # print('10')
            varNewBin = varNewBin + '10'
        elif varChnk == '011':
            # print('01')
            varNewBin = varNewBin + '01'
        elif varChnk == '100':
            # print('10')
            varNewBin = varNewBin + '10'
        else:
            # print("Else hit!!!")
            # print(varChnk)
            varNewBin = varNewBin + varChnk

    return varNewBin


# Open the encrypted file and read binary format out.
```

```
with open('FLAG.enc', 'rb') as fp:
    varFLAGENC = fp.read()

# varArgs to manually inject data I wanted and to change the rounds as I was tinkering.
if varArgs.flag is not None:
    FLAG = bytes(varArgs.flag, encoding='utf-8')
else:
    FLAG = b"ASIS{is_awesome}"
    FLAG = varFLAGENC

if varArgs.rounds is not None:
    round = int(varArgs.rounds)
else:
    round = 19    # Found after hitting a length of binary I though would fit the flag then I deleted
                  # one bit at a time until the rest of the flag popped. That confirmed the 19 rounds.


DNC = bin(bytes_to_long(FLAG))[2:]    # strip off the "0b" in the front of the data.

print(DNC[:30])    # 101110101101110011110000110110

# Send the data to the decryption function.
for _ in range(round):
    DNC = decrypt(DNC)


DNC = '0' + DNC    # I found this manually through deleteing each bit in front of the string until
                   # I saw the rest of the flag. For the life of me I can't figure out how I'm losing
                   # that first bit. It just never gets changed and according to the pattern it shouldn't
                   # be changed into a zero as the first bit.

print("DNC after decrypt call: {}".format(DNC))   # 01000001010100110100100101010011011110

# I really struggled with some of the other simple techniques for this binary to ascii conversion.
# They all ended in some kind of error. So, in the end I just hacked this line together to convert it
# chunk by chunk. Sorry, to those python purists reading this. ;)
```

```
varFLAGBinaryToASCII = ''.join([chr(int(DNC[i:i+8], 2)) for i in range(0, len(DNC), 8)])
print(varFLAGBinaryToASCII)    # ASIS{50_S1mPl3_CryptO__4__warmup____}
```