

p4-team / ctf

Watch78

Star296

Fork62

<> Code

Issues 0

Pull requests 0

Projects 0

Pulse

Graphs

Branch: master ctf / 2017-02-04-bitsctf / beginners_luck /





Create new file

Find file

History

 Pharisaesus added beginners luck writeup

Latest commit 3d78cb3 on Feb 5

..		
 BITSCTFfullhd.png	added beginners luck writeup	3 months ago
 README.md	added beginners luck writeup	3 months ago
 enc27.py	added beginners luck writeup	3 months ago
 result.png	added beginners luck writeup	3 months ago

 README.md

Beginners Luck (crypto)

tography, and he feels really confident about his skills in this field. Can you break his algorithm and get the flag?

###ENG PL

In the task we get [encryption algorithm](#) and [encrypted flag](#). The algorithm is simple enough:

```
#!/usr/bin/env python

def supa_encryption(s1, s2):
    res = [chr(0)]*24
    for i in range(len(res)):
        q = ord(s1[i])
        d = ord(s2[i])
        k = q ^ d
        res[i] = chr(k)
    res = ''.join(res)
    return res

def add_pad(msg):
    L = 24 - len(msg)%24
    msg += chr(L)*L
    return msg

with open('fullhd.png', 'rb') as f:
    data = f.read()

data = add_pad(data)

with open('key.txt') as f:
    key = f.read()

enc_data = ''
for i in range(0, len(data), 24):
    enc = supa_encryption(data[i:i+24], key)
    enc_data += enc

with open('BITSCTFfullhd.png', 'wb') as f:
    f.write(enc_data)
```

It loads a 24-byte xor key and input png file, adds padding to the input file so that it is a multiple of 24 bytes and then xors every 24 bytes of the input file with the xor key.

Breaking this is simple enough once we know that PNG files have a well known header and trailer. We know that the file has to start with 16 bytes:

```
0x89, 0x50, 0x4e, 0x47, 0xd, 0xa, 0x1a, 0xa, 0x0, 0x0, 0x0, 0xd, 0x49, 0x48, 0x44, 0x52
```

Once we use this key with `0x0` as missing 8 bytes we can already spot where the `IEND` trailer should be and we can use this information to fill the blank spaces in the xor key.

Finally we get:

```
from crypto_commons.generic import xor, xor_string

def main():
    with open('BITSCTFfullhd.png', 'rb') as f:
        data = f.read()
        pngheader_and_trailer = [137, 80, 78, 71, 13, 10, 26, 10, 0, 0, 0, 0xd, 0x49, 0x48, 0x44, 0x52, 0x0, 0x0, 0x7
                                0x80, 0x0,
                                0x0, 0x4, 56]
        result = xor(pngheader_and_trailer, map(ord, data[:len(pngheader_and_trailer)]))
        key = "".join([chr(c) for c in result]) + ("\0" * (24 - len(pngheader_and_trailer)))

        with open('result.png', 'wb') as f:
            f.write(xor_string(data, key * (len(data) // len(key))))

main()
```

And this gives us the [flag file](#)

###PL version

W zadaniu dostajemy [algorytm szyfrowania](#) i [zaszyfrowaną flagę](#).

Algorytm jest dość prosty:

```
#!/usr/bin/env python

def supa_encryption(s1, s2):
    res = [chr(0)]*24
    for i in range(len(res)):
        q = ord(s1[i])
        d = ord(s2[i])
        k = q ^ d
        res[i] = chr(k)
    res = ''.join(res)
    return res

def add_pad(msg):
    L = 24 - len(msg)%24
    msg += chr(L)*L
    return msg

with open('fullhd.png', 'rb') as f:
    data = f.read()

data = add_pad(data)

with open('key.txt') as f:
    key = f.read()

enc_data = ''
for i in range(0, len(data), 24):
    enc = supa_encryption(data[i:i+24], key)
    enc_data += enc

with open('BITSCTFfullhd.png', 'wb') as f:
    f.write(enc_data)
```

Ładujemy 24-bajtowy klucz xora oraz plik png, dodaje do pliku png padding tak żeby jego rozmiar był wielokrotnością 24 bajtów, następnie xoruje 24 bajtowe fragmenty pliku wejściowego z kluczem.

Złamanie tego jest dość proste jeśli wiemy że plik PNG ma dobrze zdefiniowany header i trailer. Wiemy że plik musi zaczynać się od 16 bajtów:

```
0x89, 0x50, 0x4e, 0x47, 0xd, 0xa, 0x1a, 0xa, 0x0, 0x0, 0x0, 0xd, 0x49, 0x48, 0x44, 0x52
```

Kiedy użyjemy tego klucza z `0x0` jako brakujące 8 bajtów klucza to możemy znaleźć pod koniec pliku miejsce gdzie powinien być `IEND` i na podstawie tej informacji możemy wypełnić brakujące elementy klucza.

Finalnie dostajemy:

```
from crypto_commons.generic import xor, xor_string

def main():
    with open('BITSCTFfullhd.png', 'rb') as f:
        data = f.read()
        pngheader_and_trailer = [137, 80, 78, 71, 13, 10, 26, 10, 0, 0, 0, 0xd, 0x49, 0x48, 0x44, 0x52, 0x0, 0x0, 0x7
                                0x80, 0x0,
                                0x0, 0x4, 56]
        result = xor(pngheader_and_trailer, map(ord, data[:len(pngheader_and_trailer)]))
        key = "".join([chr(c) for c in result]) + ("\0" * (24 - len(pngheader_and_trailer)))

        with open('result.png', 'wb') as f:
            f.write(xor_string(data, key * (len(data) / len(key))))

main()
```

A to daje [plik z flagą](#)

