Cyber Santa is Coming to Town: Meet Me Halfway

Author crxzii | Contest Date 05.12.2021

Solve Moment During The Contest | Category Cryptography | Score 300

Description

> Evil elves have deployed their own cryptographic service. The keys are unknown to everyone but them. Fortunately, their encryption algorithm is vulnerable. Could you help Santa break the encryption and read their secret message?

Attached Files

- challenge.py

This file contains the same code that's executed on the docker. We can use this to emulate the encryption with a fake flag.

```
flag = b'HTB{dummyflag}'

def gen_key(option=0):
    alphabet = b'0123456789abcdef'
    const = b'cyb3rXm45!@#'
    key = b''
    for i in range(16-len(const)):
        key += bytes([alphabet[randint(0,15)]])

    if option:
        return key + const
    else:
        return const + key

def encrypt(data, key1, key2):
    cipher = AES.new(key1, mode=AES.MODE_ECB)
    ct = cipher.encrypt(pad(data, 16))
    print("Step 1: " + str(ct))
    cipher = AES.new(key2, mode=AES.MODE_ECB)
    ct = cipher.encrypt(ct)
    return ct.hex()


def challenge():
    k1 = gen_key()
    k2 = gen_key(1)

    ct = encrypt(flag, k1, k2)

    print('Super strong encryption service approved by the elves X-MAS spirit.\n'+\
                    'Message for all the elves:\n' +ct + '\nEncrypt your text:\n> ')
    try:
        dt = json.loads(input().strip())
        pt = bytes.fromhex(dt['pt'])
        res = encrypt(pt, k1, k2)
        print(res + '\n')
        exit(1)
    except Exception as e:
        print(e)
        print('Invalid payload.\n')
        exit(1)

if __name__ == "__main__":
    challenge()
```

Summary

We see, that our flag is encrypted twice, by two different AES encryptions, that use a key with 4 random characters each. We also get to enter a plaintext and get its ciphertext.

The initial idea I got was using a Meet-In-The-Middle attack, which would also fit the challenge name 'Meet Me Halfway'.

Flag

```
HTB{m337_m3_1n_7h3_m1ddl3_0f_3ncryp710n}
```

Detailed Solution

Okay, so we want to use a Meet-In-The-Middle attack to find our keys and then decrypt our flag.

But how does a Meet-In-The-Middle attack work?

We want to get the key for the first encryption, by going forward. We encrypt our known plaintext with all possible keys and store them in a dictionary, that maps ciphertext to key.

For the second key we want to go backwards. We decrypt our known ciphertext with all possible keys. Then we check our dictionary. If we find our result in there, we know that our key was correct and we can just take the value for key1 from the dict.

So first, let's get our flag and plaintext-ciphertext pair from our Docker:

```
Super strong encryption service approved by the elves X-MAS spirit.
Message for all the elves:
d8c4cf645e889ec3d21fdaca4083d17acdd31489126f7a4bd48b73ea51675513165a00308497170dc3cd18ee969e4fc24c2534bffa9aa9ae06df
aeae8d025b9ae4ea59fa5c81103ef4ded08f76f88dffd40c7eb15aa21cc1ff88966c3ddcab89
Encrypt your text:
> {"pt":"aaaaaaaaaaaaaaaa"}
dd94b4f4708c5d21ac12ca38d5a0cebe
```

With the variables we got, we can use the following python code to calculate everything for us:

```
import json
from Crypto.Util.Padding import pad
from Crypto.Cipher import AES



def first_half(plaintext, const, printable):
    dt = json.loads('{"pt":"' + plaintext + '"}')
    pt = bytes.fromhex(dt['pt'])
    res = {}
    for a in printable:
        for b in printable:
            for c in printable:
                for d in printable:
                    key1 = const + bytes(a.encode('latin-1')) + bytes(b.encode('latin-1'))
                          + bytes(c.encode('latin-1'))+bytes(d.encode('latin-1'))
                    cipher1 = AES.new(key=key1, mode=AES.MODE_ECB)
                    c1 = cipher1.encrypt(pad(pt, 16))
                    res[c1] = key1
    return res
```

```
def second_half(ciphertext, dict, const, printable):
    ct = bytes.fromhex(ciphertext)
    for a in printable:
        for b in printable:
            for c in printable:
                for d in printable:
                    key2 = bytes(a.encode('latin-1')) + bytes(b.encode('latin-1'))
                          + bytes(c.encode('latin-1'))+bytes(d.encode('latin-1')) + const
                    cipher2 = AES.new(key=key2, mode=AES.MODE_ECB)
                    res = cipher2.decrypt(ct)
                    if res in dict:
                        key1 = dict[res]
                        return key1, key2


def main():
    flag = 'd8c4cf645e889ec3d21fdaca4083d17acdd31489126f7a4bd48b73ea51675513165a00308497170dc3cd18ee969e4fc24c2534b...'
    plaintext = 'aaaaaaaaaaaaaaaa'
    ciphertext = 'dd94b4f4708c5d21ac12ca38d5a0cebe'
    const = b'cyb3rXm45!@#'
    printable = ['0','1','2','3','4','5','6','7','8','9','a','b','c','d','e','f']

    dictionary = first_half(plaintext, const, printable)
    key1, key2 = second_half(ciphertext, dictionary, const, printable)

    cipher1 = AES.new(key=key1, mode=AES.MODE_ECB)
    cipher2 = AES.new(key=key2, mode=AES.MODE_ECB)

    decrypted_flag = cipher1.decrypt(cipher2.decrypt(bytes.fromhex(flag)))
    print(decrypted_flag)

main()
```

The output of our code is:

```
b'https://www.youtube.com/watch?v=DZMv9XO4Nlk\nHTB{m337_m3_1n_7h3_m1ddl3_0f_3ncryp710n}\x0c\x0c\x0c\x0c\x0c\x0c\x0c\x0c
```

We get a nice christmas song and our flag `HTB{m337_m3_1n_7h3_m1ddl3_0f_3ncryp710n}`!