Lost

首先用 `tshark` 將 pcap-ng 檔轉成 tcpdump 的格式，再用 scapy 開啟之；scapy 本身支援 pcap-ng 格式，但其外掛 scapy-ssl-tls 似不支援。

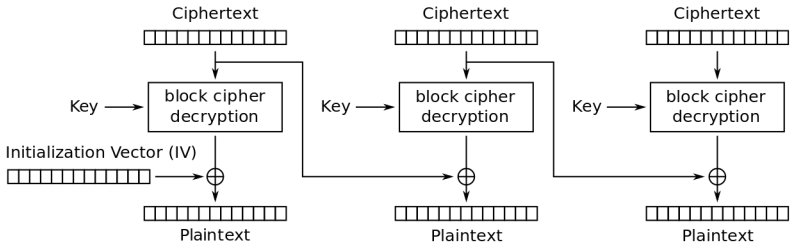
觀察 payload 內容發現，封包內多次傳輸相同內容，但被星號遮罩掉的位置不盡相同，所以第一步先將這些資料組起來，盡量減少星號的數量。

發現：

- KEY 前14碼已知，末兩碼未知，猜測其值域為 `string.printable`，進行暴力破解。且一個 block 為 16bytes。
- Plaintext 已知。
- AES-CBC(KEY, Plaintext)第一個 byte、後17個 byte為已知，即最後一個 block 已知。
- AES-CBC(KEY, FLAG)已知。

依 CBC 解密流程(下圖)，$C_0 = IV$、$P_i = D_K(C_i) \oplus C_{i-1}$，所以 $C_{i-1} = D_K(C_i) \oplus P_i$。

故可由猜測 K 解密 C_i 得到 C_{i-1}，又已知 C_{i-1} 的頭尾 2bytes，故解密出來的結果須符合此限制，才能確定 K 的正確性。由此 C_{i-1}，再解密前一 block，便可得 $C_0 = IV$。



Cipher Block Chaining (CBC) mode decryption

至此，IV、key 皆已知，便可將 flag 解密。

```
ALPHA = string.printable
def keygen(key, placeholder='\x00'):
    assert placeholder in key
    q = [array.array('c', key)]
    while q:
        key, q = q[0], q[1:]
        idx = key.index(placeholder)
        if key.count(placeholder) == 1:
            for i in ALPHA:
                key[idx] = i
                yield key[:]
        else:
            for i in ALPHA:
                key[idx] = i
                q.insert(0, key[:])


def eq(expected, guessed):
    for e, g in zip(expected, guessed):
        if e != '\0':
            if g != e:
                return False
    return True
```

```
def AES_CBCd(key, msg, iv):
    # print('DEC(k=%s, msg=%s, iv=%s)' % (str(key), repr(msg), iv))
    obj = AES.new(key, AES.MODE_CBC, iv)
    return obj.decrypt(msg)


def parse_pcap(pcap):
    KEY = np.zeros(16, np.byte)
    PLAIN = np.zeros(32, np.byte)
    AES_KEY_PLAIN = np.zeros(64, np.byte)  # hex-encoded
    AES_KEY_FLAG = np.zeros(64, np.byte)   # hex-encoded
    tup = (KEY, PLAIN, AES_KEY_PLAIN, AES_KEY_FLAG)

    for packet in rdpcap(pcap):
        # 0x8 is PSH flag
        if not packet[TCP].getfieldval('flags') & 0x8:
            continue
        for store, line in zip(tup, packet.load.splitlines()[:4]):
            store |= np.frombuffer(line.split(' = ', 1)[-1].replace('*', '\0'), np.byte)

    print repr(KEY.tostring())
    print repr(PLAIN.tostring())
    print repr(AES_KEY_PLAIN.tostring())
    print repr(AES_KEY_FLAG.tostring())
    return map(lambda t: t.tobytes(), tup)


def solve(KEY, PLAIN, AES_KEY_PLAIN, AES_KEY_FLAG):
    sPLAIN = do_slice(PLAIN, len(KEY))
    sPLAIN_ENC = do_slice(AES_KEY_PLAIN, len(KEY) * 2)
    assert len(sPLAIN) == len(sPLAIN_ENC)

    for k in keygen(KEY):
        for i, (pn, cn) in reversed(tuple(enumerate(zip(sPLAIN[1:], sPLAIN_ENC[1:]), 1))):
            cn = binascii.unhexlify(cn)   # C_n
            cn_1 = AES_CBCd(k, cn, pn)    # C_{n-1}
            cn_1_hex = binascii.hexlify(cn_1)
            if eq(sPLAIN_ENC[i - 1], cn_1_hex):
                sPLAIN_ENC[i - 1] = cn_1_hex
            else:
                break
        else:
            iv = AES_CBCd(k, cn_1, sPLAIN[0])
            return AES_CBCd(k, binascii.unhexlify(AES_KEY_FLAG), iv)
```