Cryptographie - PartitioningOracle - 400 points

Kévin DUVERGER

Table des matières

1 Résolution PartitioningOracle :

Avant de commencer ce write-up, je ne peux que vous conseiller d'aller lire la ressource associée au challenge sur la plateforme de CTF. Elle vous introduira au fonctionnement du mode d'opération GCM sur le chiffrement par bloc AES. Elle vous montre aussi la théorie derrière l'attaque et vous donne les pistes pour la réaliser. Nous ne referons donc pas cette introduction ici.

Pour commencer, on peut se connecter au serveur pour voir comment il fonctionne :

```
C:\Users\keked>ncat 146.59.227.136 23421
libnsock ssl_init_helper(): OpenSSL legacy provider failed to load.

------------------------------------
| PARTITIONING ORACLE ATTACK TRAINER |
------------------------------------

Nonce (IV)     : 437740dad2b214a0a9fb1e14
Encrypted flag : 8693345abe851d31f9c6fff67ec9ae2c3d3b3028c55f873532b8727afbf24e2991d43a81052c1b91c10b3177d7c9af2c
Flag enc. tag  : ca14368c19be2e3e57fe166612ad6433

Now your turn, you are allowed at most 1900 requests !
The password is 3 characters (lowercase letters and digits)
The key is 16 bytes and formed by adding 0 bytes after the 3 bytes of the password

Ciphertext and tag please (in hex strings, separated by a space) : 8693345abe851d31f9c6fff67ec9ae2c3d3b3028c55f87
3532b8727afbf24e2991d43a81052c1b91c10b3177d7c9af2c ca14368c19be2e3e57fe166612ad6433
Valid tag !
Ciphertext and tag please (in hex strings, separated by a space) : 8693345abe851d31f9c6fff67ec9ae2c3d3b3028c55f87
3532b8727afbf24e2991d43a81052c1b91c10b3177d7c9af2c 8693345abe851d31f9c6fff67ec9ae2c
Invalid tag !
Ciphertext and tag please (in hex strings, separated by a space) : ^C
```

On a donc une petite introduction qui est affichée, puis on peut essayer de renvoyer le couple (chiffré, tag) donné par le serveur, et ici on obtient la réponse `Valid tag !`. Si l'on envoie en revanche un tag invalide, on obtient la réponse `Invalid tag !`.

La première chose que l'on peut faire est de parser tout ce que le serveur nous envoie au départ :

```python
server = "146.59.227.136"
port = 23421

#Connecting to the server
maSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
try :
    maSocket.connect((server, port))
except Exception as e :
    print("Probleme de connexion", e.args)
    sys.exit(1)

#Getting the two first messages
data = maSocket.recv(4096).decode("utf-8")

#Getting nonce, ciphertext and tag from first message
nonceIndex    = data.find("Nonce (IV)     : ") + 17
nonceEndIndex = data.find("\n", nonceIndex)
nonce = data[nonceIndex : nonceEndIndex]
flagIndex    = data.find("Encrypted flag : ") + 17
flagEndIndex = data.find("\n", flagIndex)
encryptedFlag = data[flagIndex : flagEndIndex]
tagIndex    = data.find("Flag enc. tag  : ") + 17
tagEndIndex = data.find("\n", tagIndex)
flagTag = data[tagIndex : tagEndIndex]

print(data)
print("Nonce : " + nonce)
print("Flag  : " + encryptedFlag)
print("Tag   : " + flagTag)
print()
```

Ensuite, on peut passer à la partie plus compliquée qui consiste à créer un chiffré valide pour plusieurs clés :

```python
def byteArrayToHex(byte_values) :
    return ''.join('{:02x}'.format(x) for x in byte_values)

def byteArrayToBin(byte_values) :
    return ''.join('{:08b}'.format(x) for x in byte_values)

def craftCiphertext(passwordList, GF, nonce, tag) :
```

```
    # Creating all of the different H values (which are encryptions of all zeros) :
    HlistInGF = []
    data = bytes.fromhex("00" * 16)
    for i in range(0, len(passwordList)) :
        cipher = AES.new(passwordList[i].encode("utf-8") + bytes.fromhex("00" * (16 - len(passwordList[i]))), AES.MODE_ECB)
        H = cipher.encrypt(data)
        HlistInGF.append(GF(int(byteArrayToBin(H)[::-1], 2)))

    # Creating all of the values of J0enc
    J0encListInGF = []
    data = bytes.fromhex(nonce) + bytes.fromhex("00" * 3 + "01")
    for i in range(0, len(passwordList)) :
        cipher = AES.new(passwordList[i].encode("utf-8") + bytes.fromhex("00" * (16 - len(passwordList[i]))), AES.MODE_ECB)
        J0enc = cipher.encrypt(data)
        J0encListInGF.append(GF(int(byteArrayToBin(J0enc)[::-1], 2)))

    # Creating the two other values that we will need
    TGF    = GF(int(byteArrayToBin(tag)[::-1], 2))
    sizeGF = GF(int(("{:0128b}".format(128 * len(passwordList)))[::-1], 2))

    # Creating the vandermonde matrix
    matrix = [ [ GF(0) for j in range(len(passwordList)) ] for i in range(len(passwordList)) ]
    for i in range(0, len(passwordList)) :
        currentValue = GF(1)
        for j in range(len(passwordList) - 1, -1, -1) :
            matrix[i][j] = currentValue + GF(0) # Otherwise, won't do a copy of currentValue which is what we want here (so that the whole line / column is not the same object in memory)
            currentValue *= HlistInGF[i]
    matrix = GF(matrix)

    # Creating the vector on the right of the equal sign
    vector = [ GF(0) for i in range(len(passwordList)) ]
    for i in range(0, len(passwordList)) :
        vector[i] = (sizeGF * HlistInGF[i] + J0encListInGF[i] + TGF) * (HlistInGF[i] ** -2)
    vector = GF(vector)

    # Inverting the matrix
    startTime = time.time()
    inverse = np.linalg.inv(matrix)
    print("Time taken for inversion : " + str(time.time() - startTime))

    # By multiplying the matrix and the vector we can get the ciphertext
    resultVector = [ GF(0) for i in range(len(passwordList)) ]
    for i in range(0, len(passwordList)) :
        currentValue = GF(0)
        for j in range(0, len(passwordList)) :
            currentValue += inverse[i][j] * vector[j]
        resultVector[i] = currentValue

    #Creation du texte chiffre
    ciphertext = b""
    for i in range(0, len(passwordList)) :
        C = int(("{:0128b}".format(int(resultVector[i])))[::-1], 2)
        C = C.to_bytes(16, "big")
        ciphertext += C
    return ciphertext, tag
```

Grâce à cela, nous pouvons maintenant trouver un batch de mots de passe dans lequel le mot de passe cible se trouve :

```
GF = galois.GF(2 ** 128, irreducible_poly = "x^128 + x^7 + x^2 + x + 1")
myTag = bytes.fromhex("00" * 16)

# First part : getting a batch of 25 passwords containing the password of the server via the partitioning oracle
inversionTimeList = []
pwdsPerBatch      = 25
currentBatch      = []
pwdLength = 3
characters = "abcdefghijklmnopqrstuvwxyz0123456789"
for i in range(0, len(characters) ** pwdLength) :
    # Getting the current pwd by interpreting i as a base 36 integer
    iCopy, currentPwd = i, ""
    for j in range(0, pwdLength) :
        currentPwd += characters[iCopy % len(characters)]
        iCopy //= len(characters)
    currentBatch.append(currentPwd)

```

```
    if len(currentBatch) == pwdsPerBatch or i == (len(characters) ** pwdLength - 1) :
        startTime = time.time()
        ciphertext, tag = craftCiphertext(currentBatch, GF, nonce, myTag)
        inversionTime = time.time() - startTime
        inversionTimeList.append(inversionTime)
        print("Testing batch " + str(len(inversionTimeList)) + " / " + str(math.ceil((len(characters) ** pwdLength) / pwdsPerBatch)))

        maSocket.sendall((byteArrayToHex(ciphertext) + " " + byteArrayToHex(tag)).encode("utf-8") + b"\n")
        response = maSocket.recv(1024)
        while b"space)" not in response :
            response += maSocket.recv(1024)

        if b"Invalid" in response :
            print("Not the right batch !")
        else :
            print("Right batch found !")
            break
        currentBatch = []
```

Une fois que l'on a le batch, on peut soit tester mot de passe par mot de passe "linéairement" soit faire une recherche dichotomique pour trouver le mot de passe en encore moins de requêtes, voici ce que cela donnerait :

```
#Isolation du "vrai" mot de passe
while len(currentBatch) > 1 :
    leftHalf = currentBatch[0 : len(currentBatch) // 2]
    rightHalf = currentBatch[len(currentBatch) // 2 : len(currentBatch)]
    ciphertext, tag = craftCiphertext(leftHalf, GF, nonce, myTag)

    maSocket.sendall((byteArrayToHex(ciphertext) + " " + byteArrayToHex(tag)).encode("utf-8") + b"\n")
    response = maSocket.recv(1024)
    while b"space)" not in response :
        response += maSocket.recv(1024)

    if b"Invalid" in response :
        currentBatch = rightHalf
    else :
        currentBatch = leftHalf
    print(currentBatch)

print("The password is : " + str(currentBatch[0]))

cipher = AES.new(currentBatch[0].encode("utf-8") + bytes.fromhex("00" * 13), AES.MODE_GCM, nonce=bytes.fromhex(nonce))
plaintext = cipher.decrypt_and_verify(bytes.fromhex(encryptedFlag), bytes.fromhex(flagTag))
print(plaintext)
```

Ce qui nous permet d'obtenir le flag !