Réseau - ReplayAttack - 100 points

Table des matières

1 Résolution ReplayAttack :

Pour ce nouveau challenge, commençons par intéragir avec le serveur pour voir ce qu'il nous propose. Pour ce faire, nous pouvons donc faire un netcat vers `146.59.227.136` avec pour port `23411` :

```
Please make a choice :
        [1] : see the user authenticate himself
        [2] : try to authenticate
Your choice :
```

Cela semble donc être un protocole d'authentification.

Tentons donc le premier choix pour voir comment ce protocole fonctionne :

```
Server sends challenge 62002
Client sends back Enc_AES_K(challenge) : ct = 8055470e63658be46012b7a24e5ba55c
Server verifies that Dec_AES_K(ct) = challenge : client is authenticated !
```

Comme on peut le voir, le serveur envoie un challenge, le client doit ensuite chiffrer ce challenge et l'authentification sera réussie si le client a bien réussi à chiffrer le challenge (prouvant la connaissance de la clé K).

On peut voir que les challenges sont assez petits, il est peut être possible qu'il y ait des collisions de challenges auquel cas on pourra tout simplement rejouter le texte chiffré de la communication précédente (cela suppose aussi que K est le même à travers toutes les communications, ce qui ne semble pas déraisonnable). Nous pouvons donc tenter une dizaine d'espionnages sur les valeurs et voici ce que l'on obtient : `49844`, `77384`, `70374`, `82051`, `90451`, `3366`, `5535`, `18031`, `87518` et `63727`.
Il semblerait donc que la valeur maximale du challenge soit de 100000 L'idée de l'attaque se fait donc en 2 temps : une première phase pour récupérer des chiffrés associés à des challenges (1000 dans mon cas, le meilleur choix serait $\sqrt{100000}$), et une secodne pour tenter de s'authentifier (on peut espérer 100 connexions pour le choix de 1000, et $\sqrt{100000}$ pour l'autre choix). Voici le code de cette attaque :

```python
import math
import random
import os, sys, socket



server = "146.59.227.136"
port = 23411

# First step, getting auth values
numberOfAuthValuesToGet = 1000
authValues = []
for i in range(0, numberOfAuthValuesToGet) :
    print("Getting auth value " + str(i))
    ma_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    try:
        ma_socket.connect((server, port))
    except Exception as e:
        print("Problème de connexion", e.args)
        sys.exit(1)
    # Getting data
    data = ma_socket.recv(16384).decode("utf-8")
    # Sending choice
    ma_socket.sendall(("1" + "\n").encode("utf-8"))
    # Getting data
    data = ma_socket.recv(16384).decode("utf-8").split("\n")
    challenge = int(data[1][23:])
    ciphertext = bytes.fromhex(data[2][46:])
    authValues.append([ challenge, ciphertext ])
    # Closing the socket
    ma_socket.close()



# Now, the replay attack in itself
tryIndex = 0
while True :
    tryIndex += 1
    print("Authentication try " + str(tryIndex))
    ma_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    try:
        ma_socket.connect((server, port))
    except Exception as e:
        print("Problème de connexion", e.args)
        sys.exit(1)
    # Getting data
```

```
    data = ma_socket.recv(16384).decode("utf-8")
    # Sending choice
    ma_socket.sendall(("2" + "\n").encode("utf-8"))
    # Getting data
    data = ma_socket.recv(16384).decode("utf-8").split("\n")
    challenge = int(data[1][23:])
    # Seeing whether the challenge was already seen
    foundIndex = -1
    for i in range(0, numberOfAuthValuesToGet) :
        if authValues[i][0] == challenge :
            foundIndex = i
            break
    # If we have already seen it, we can return the same ciphertext, otherwise we continue
    if foundIndex != -1 :
        print("Challenge collision on " + str(challenge) + ", sending " + authValues[foundIndex][1].
    hex() + " !")
        ma_socket.sendall((authValues[foundIndex][1].hex() + "\n").encode("utf-8"))
        print(ma_socket.recv(16384).decode("utf-8"))
        ma_socket.close()
        break
    # Closing the socket
    ma_socket.close()
```