Rapport du devoir maison de Sécurité

BASUALDO Lautaro et LOI Léo

16 avril 2024

Table des matières

1 Exercice 1 :

Avant de pouvoir décrypter les deux mots de passes, nous devons déjà les identifier.
Or, nous savons que la fonction SHA-256 génèrera toujours le même haché pour un mot de passe donné.
Ainsi, nous pouvons détecter deux hachés différents :

- 068be8be83f9bfafd1545d357fd3cd132f8c659effd11e635a698811b796c880 (utilisé par Bart, Homer, Lisa et March)
- 15e2b0d3c33891ebb0f1ef609ec419420c20e320ce94c65fbc8c3312448eb225 (utilisé par Bob, Carlton, John et William)

Pour le premier haché, grâce aux différents indices, nous pouvons deviner que le mot de passe utilisé est le 74ème élément du tableau périodique des élements. A savoir le "tungstène" appelé en anglais "tungsten"
Afin de s'assurer de nos résultats, nous avons haché le mot "tungsten" avec la fonction de hachage SHA-256 ce qui donne le résultat suivant :

```
moi@moi-HP-245-G7-Notebook-PC:~$ echo -n 'tungsten' | openssl sha256
(stdin)= 068be8be83f9bfafd1545d357fd3cd132f8c659effd11e635a698811b796c880
```

Nous pouvons constater que les deux hachés sont similaire, le mot de passe utilisé par Bart, Homer, Lisa et March est donc bien "tungsten"

Pour le second haché, grâce aux différents indices, nous pouvons deviner que le mot de passe utilisé est la suite de chiffres "123456789"
Afin de s'assurer de nos résultats, nous avons haché le mot "123456789" avec la fonction de hachage SHA-256 ce qui donne le résultat suivant :

```
moi@moi-HP-245-G7-Notebook-PC:~$ echo -n '123456789' | openssl sha256
(stdin)= 15e2b0d3c33891ebb0f1ef609ec419420c20e320ce94c65fbc8c3312448eb225
```

Nous pouvons constater que les deux hachés sont similaire, le mot de passe utilisé par Bob, Carlton, John et William est donc bien "123456789"

2 Exercice 2 :

2.1 Question 1 :

```
  curseur.execute("CREATE TABLE utilisateurs ( name
     TEXT, password TEXT)")

def AjoutUtilisateur():
  connection = sqlite3.connect("donnees.db")
  curseur = connection.cursor()
  is_id_incorrect = True
  #tant que le login est deja utilise,
  #on demande a l'utilisateur de re-rentrer un login
  while is_id_incorrect:
    print("Choississez votre identitfiant :")
    id = input()
    res = curseur.execute("SELECT name FROM
       utilisateurs WHERE name = '"+id+"'")
    #si on n'a trouve aucun resultat,
    #le login n'existe pas dans la base de donnee
    if res.fetchone() is None:
      #on sort de la boucle
      is_id_incorrect = False
    else:
      #on redemande son identifiant a l'utilisateur
      print("L'identifiant est deja utilise, veuillez
         en saisir un autre")
  is_mdp_incorrect = True
  #tant que les deux entrees de l'utilisateurs ne
     correspondent pas,
  #on lui demande de re-rentrer un mot de passe et de
     le confirmer
  while is_mdp_incorrect:
    print("choisissez votre mot de passe :")
    mdp = input()
    print("Validez votre mot de passe :")
    mdp2 = input()
    #si les deux entrees correspondent
    if mdp == mdp2:
      #on sort de la boucle
      is_mdp_incorrect = False
    else:
      print("Vos entrees ne correspondent pas")
  return [id, mdp]

def addUtilBDD(ids):
```

```
  connection = sqlite3.connect("donnees.db")
  curseur = connection.cursor()
```

2.2 Question 2 :

Après plusieurs éxecutions du programme ci-dessus, les valeurs suivantes peuvent être récupérées dans la base de donnée

```
moi@moi-HP-245-G7-Notebook-PC:~$ python3 ajoutUtil.py
[('Leo', 'Motdepasse'), ('idRandom', 'mdpRandom')]
```

2.3 Question 3 :

```
  print(res.fetchall())

def Verification():
  connection = sqlite3.connect("donnees.db")
  curseur = connection.cursor()
  print("Connexion␣:")
  is_not_connected = True
  #tant que le login et le mot de passe ne
     correspondent a aucune entree de la base de
     donnee, on demande a l'utilisateur d'en re-
     rentrer
  while is_not_connected:
    print("Entrez␣votre␣identifiant")
    id = input()
    print("Entrez␣votre␣mot␣de␣passe")
    mdp = input()
    res = curseur.execute("SELECT␣name␣FROM␣
       utilisateurs␣WHERE␣name␣=␣'" + id + "'␣AND␣
       password␣='" + mdp +"'")
```

2.4 Question 4 :

Notre fonction d'AjoutUtilisateur renvoyant une liste contenant l'identifiant et le mot de passe de l'utilisateur, il nous suffit de récupérer ces données, et de les renvoyer une fois le mot de passe haché

```
def hachage(ids):
  bmdp = bytes(ids[1], "utf-8")
  m = hashlib.sha256()
  m.update(bmdp)
  ids[1] = m.hexdigest()
```

2.5 Question 5 :

```
def hachageSalage(ids, salt):
    bmdp = bytes(ids[1], "utf-8")
    bsalt = bytes(salt, "utf-8")
    hashed_password = hashlib.pbkdf2_hmac('sha256',
       bmdp, bsalt, 100000)
    ids[1] = hashed_password
    return ids
```

2.6 Question 6 :

```
def randomHachageSalage(ids):
    bmdp = bytes(ids[1], "utf-8")
    salt = random.getrandbits(16*8).to_bytes(16, '
       little')
    hashed_password = hashlib.pbkdf2_hmac('sha256',
       bmdp, salt, 100000)
    ids[1] = hashed_password
    connection = sqlite3.connect("donnes.db")
    curseur = connection.cursor()
    curseur.execute("INSERT␣INTO␣sels␣VALUES('"+ids
       [0]+"','"+salt+"')")
    connection.commit()
    return ids
```

3 Exercice 3 :

3.1 Question 1 :

Grâce aux logs que nous avons pû récupérer, nous pouvons réussir à identifier plusieurs informations concernant l'attaque.
On peut tout d'abord repérer que l'attaque bruteforce a commencé à 8h58 et 35 secondes. On peut l'identifier car à partie de la ligne 268 des logs, nous pouvons observer un nombre anormal de requêtes qui sont toutes efféctuées dans un laps de temps très court.

3.2 Question 2 :

Nous pouvons voir à partir de la ligne 1321 jusqu'à la ligne ... que l'attaquant répète les 2 même requêtes en changeant uniquement l'extension. Il utilise les extensions :

- .php
- .phmtl
- .php3
- .php4
- .php5
- .php6
- .php7
- .phar

3.3 Question 3 :

Il s'arrête à l'extension .phar, ayant sûrement trouvé une faille pour s'introduire dans le système grâce à cette extension. Il s'agit aussi de l'extension qu'il utilisera pour le reste de ses requêtes.

3.4 Question 4 :

Nous pouvons voir que l'attanquant utilise le fichier php pour exécuter les commandes suivantes :

Ligne 1340 : Il télécharge un fichier fiche_de_poste.odt.

Ligne 1341 :

Ligne 1342 : Il modifie les droits d'accès au ficheir qu'il vient de déplacer.

Ligne 1343 : Il entre dans le fichier afin, possiblement, d'en récupérer les données.