Reverse - MiniVM - 300 points

Noah STAPLE, Kévin DUVERGER

Table des matières

1 Résolution MiniVM :

La première chose que l'on peut tenter est d'exécuter le code python :

```
> python main.py
```

Malheureusement, il n'affiche rien et il faut donc aller voir le code.

1.1 Rendons le code plus lisible :

La première chose que vous pouvez remarquer est que le code du challenge est fortement obfusqué. La première étape sera donc de le rendre un peu plus lisible, ce qui nous permettra de comprendre comment il fonctionne.
La première chose que vous pouvez remarquer est une duplication de la classe E, seule la seconde sera utile (elle va écraser la première définition de E). Nous pouvons maintenant passer à une petite liste des fonctions.

`__init__` :

```
def __init__(A) :
    A._s=[]
    A.uuuuuuuuuuuu=[]
    A.OOOOOOOOOO=0
    A.OOOOOOOOOO={}
```

`i` :

```
def i(A,_):
    B=D.dumps(A.uuuuuuuuuuuu)
    if B!=_:
        exit(0)
```

`ll` :

```
def ll(A):
    A.OOOOOOOOOO=D.loads(b'\x80\x04\x95u\x00\x00\x00\x00\x00\x00\x00}\x94(K\x00\x8c\n@PACG^@T\n\x11\
    x94K\x03\x8c\nb\x02Q@\x02Emy\x02H\x94K\x01\x8c\x1fS]=wi 2\x0b2\x17)66p^\x13\x1a\x0fg\x0b=3?\x190
    <2)OrH\x94K\x04\x8c\x1drCSO_\x10\x12mE\x11SJ\x10E@VEG\xc3\x9b\x19\\T\x12_\\PU\x18\x94K\x05\x8c\
    x07~^BS\x10\n\x0c\x94u.')
```

`l` :

```
def l(A,uuuuuuuuuuu):
    A.uuuuuuuuuuu=uuuuuuuuuuu
    A.OOOOOOOOOO=0
    A._s=[]
```

`OOOOOOOOOO` :

```
def OOOOOOOOOO(D,s,k):
    A=C(k)
    return''.join(chr((B(D)|B(k[C%A]))-(B(D)&B(k[C%A])))for(C,D)in enumerate(s))
```

`r` :
Nous n'allons pas la déobfusquer pour l'instant, elle viendra après !

Et pour finir, voici également le programme principal en un peu plus propre :

```
if __name__=='__main__':
    F=[(10,0),(6,),(10,3),(12,),(10,1),(8,9),(10,5),(5,),(0,),(10,4),(5,),(0,)]
    A=E()
    A.l(F)
    A.i(b'\x80\x04\x95=\x00\x00\x00\x00\x00\x00\x00]\x94(K\nK\x00\x86\x94K\x06\x85\x94K\nK\x03\x86\
    x94K\x0c\x85\x94K\nK\x01\x86\x94K\x08K\t\x86\x94K\nK\x05\x86\x94K\x05\x85\x94K\x00\x85\x94K\nK\
    x04\x86\x94h\x08h\te.')
    A.ll()
    A.r()
```

Grâce à cela, nous pouvons maintenant passer à la compréhension de ces premières fonctions !

1.2 Compréhension du `main` :

Ligne 2 - `A = E()` : c'est la première ligne vraiment intéressante, elle appelle le constructeur `__init__` et met le résultat dans la variable `A` (on ne sait pas encore ce que les varaibles représentent).
Ligne 3 - `A.l(F)` : elle réalise sensiblement la même chose que l'initialisation, met `A.uuuuuuuuuuuu` à `F`, et elle ne modifie pas les autres valeurs (qui restent donc à l'identique).
Ligne 4 - `A.i(...)` : cette ligne semble juste vérifier que la représentation pickle de `A.uuuuuuuuuuuu = F` vaut bien la valeur qui est passée en paramètre (elle n'et pas très intéressante).
Ligne 5 - `A.ll()` : elle est beaucoup plus intéressante et fait un appel à `pickle.loads`, si l'on refait cettte opération dans un autre programme avec la même chaîne / avec `python -m pickle fichier`, on a :

```
A.OOOOOOOOOO = {
    0: b'@PACG^@T\n\x11',
    1: b'S]=wi 2\x0b2\x17)66p^\x13\x1a\x0fg\x0b=3?\x19O<2)OrH',
    3: b'b\x02Q@\x02Emy\x02H',
    4: b'rCSO_\x10\x12mE\x11SJ\x10E@VEGÛ\x19\\T\x12_\\PU\x18',
    5: b'~^BS\x10\n\x0c'
}
```

Ces 5 premières étapes semblent donc être une étape d'initialisation, et nous pouvons renommer les variables :

```
A.listeX       = []
A.listeTuples  = [(10,0),(6,),(10,3),(12,),(10,1),(8,9),(10,5),(5,),(0,),(10,4),(5,),(0,)]
A.entier       = 0
A.dico         = {
    0: b'@PACG^@T\n\x11',
    1: b'S]=wi 2\x0b2\x17)66p^\x13\x1a\x0fg\x0b=3?\x19O<2)OrH',
    3: b'b\x02Q@\x02Emy\x02H',
    4: b'rCSO_\x10\x12mE\x11SJ\x10E@VEGÛ\x19\\T\x12_\\PU\x18',
    5: b'~^BS\x10\n\x0c'
}
```

Nous pouvons donc maintenant passer à la fonction `r` !

1.3 La fonction `r` :

La structure principale de cette fonction est la suivante :

```
def r(A):
    J=False
    while A.entier < C(A.listeTuples):
        E=A.listeTuples[A.entier]
        ____________________=E[0]
        # Disjonction de cas sur ___...___
        A.entier=(A.entier^1)+2*(A.entier&1)
```

Grâce à cette fonction, `A.entier` semble donc être un index dans `A.listeTuples`. La ligne à la toute fin peut sembler assez étrange et pour la comprendre il faut l'étudier au cas par cas (ici si A est pair / impair). Si A est pair (dernier bit à 0) cela donne `(A.entier+1)+(2*0)` et si A est impair (dernier bit à 1) c'est équivalent à `(A.entier-1)+(2*1)`. Cette dernière ligne est donc toujours équivalente à `A.entier + 1` et c'est donc une magnifique incrémentation, le nom `A.entier` sera donc changé en `A.idx` !
Pour finir, `___...___` prend la première valeur du tuple et semble être un sélecteur dans la disjonction de cas qui vient après, nous le renommerons donc en `currTuple0` et nous allons étudier les différents cas.

`currTuple0 not in [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12 ]` - sort de la boucle :

```
break
```

`currTuple0 = 1` - ajoute le second nombre du tuple à `A.listeX` :

```
A.listeX.append(currTuple1)
```

`currTuple0 = 2` - retire le dernier élément de `A.listeX`, sort de la boucle si elle est vide :

```
if A.listeX :
    A.listeX.pop()
else :
    break
```

`currTuple0 in [3, 4]` - retire les 2 derniers éléments de `A.listeX` et fait un calcul :

```python
if len(A.listeX) >= 2 :
    x = A.listeX.pop()
    y = A.listeX.pop()
    A.listeX.append((x ^ y) + 2 * (x & y)) # Pour currTuple0 = 3
    A.listeX.append((x ^ y) - 2 * (x & ~y)) # Pour currTuple0 = 4
else :
    break
```

`currTuple0 = 5` - affiche le dernier élément de `A.listeX` s'il existe :

```python
if A.listeX : print(f"{A.listeX[-1]}")
```

`currTuple0 = 6` - affiche le dernier élément de `A.listeX` tout en demandant un input :

```python
if A.listeX :
    A.listeX.append(input(A.listeX.pop()))
else :
    break
```

`currTuple0 = 7` - compare les 2 chaînes à la fin de `listeX` et met le résultat sur `listeX` :

```python
if len(A.listeX) >= 2 :
    x = A.listeX.pop()
    y = A.listeX.pop()
    A.listeX.append(x == y) # on peut dire ça car J vaut toujours False
    # Petite différence : peut comparer des types différents dans la "vraie" implémentation
else :
    break
```

`currTuple0 = 8` - compare les 2 valeurs à la fin de `listeX`, si égales jump à `currTuple1` :

```python
if len(A.listeX) >= 2 :
    x = A.listeX.pop()
    y = A.listeX.pop()
    if x == y :
        A.idx = currTuple1
        continue
else :
    break
```

`currTuple0 = 9` - jump inconditionnel vers `currTuple1` :

```python
A.idx = currTuple1
continue
```

`currTuple0 = 10` - c'est une opération compliquée :

```python
if currTuple1 in A.dico :
    K, L, M, N = A.dico[currtuple1], len(A.listeX), len(A.listeTuples), A.idx
    O = f"{L}{M}{N}"
    max = A.OOOOOOOOOO(K,O)
    print(max.encode("utf-8"))
    A.listeX.append(max)
else :
    break
```

`currTuple0 = 12` - encore une opération compliquée :

```python
if len(A.listeX) >= 2 :
    x = A.listeX.pop()
    y = A.listeX.pop()
    A.listeX.append(A.OOOOOOOOOO(y,x))
else :
    break
```

Conclusion. Avec tout ce que nous venons de voir, le code Python qui nous est donné devrait vous paraître bien plus clair : il s'agit en fait d'une architecture avec des instructions customisées et la liste `listeX` est une liste d'instructions. Il ne nous reste plus qu'à comprendre `A.OOOOOOOOOO` pour finir le challenge !

1.4 Terminons le chall :

Pourquoi le code ne faisait rien. Si vous rentrez dans la fonction qui teste la liste des instructions (la fonction `i`), vous verrez que le test ne passe pas. Pour le faire passer, il faut changer le second octet par 5 :

```
A.i(b'\x80\x05\x95=\x00\x00\x00\x00\x00\x00]\x94(K\nK\x00\x86\x94K\x06\x85\x94K\nK\x03\x86\x94K\
    x0c\x85\x94K\nK\x01\x86\x94K\x08K\t\x86\x94K\nK\x05\x86\x94K\x05\x85\x94K\x00\x85\x94K\nK\x04\
    x86\x94h\x08h\te.')
```

Une fois cela fait, voici ce que l'on obtient :

```
> python main.py
password: test
Nope ;>
```

Instructions. Voici donc le programme qui est exécuté par cette MiniVM :

```
fonctionCompliquee10(0)
input() # => affiche password
fonctionCompliquee10(3)
fonctionCompliquee12()
fonctionCompliquee10(1)
Si inegalite() : # => jump a 9 si egales
    fonctionCompliquee10(5) # => probablement ce qui affiche mauvais password
    print()
Sinon :
    fonctionCompliquee10(4) # => la partie pour le bon mot de passe !
    print()
```

Allons donc maintenant comprendre la fonction `A.OOOOOOOOOO` pour terminer !

La dernière fonction. Voici tout d'abord son code :

```
def OOOOOOOOOO(D,s,k) :
    return ''.join(
        chr((ord(carac) | ord(k[i % len(k)])) - (ord(carac) & ord(k[i % len(k)])))
        for (i, carac) in enumerate(s)
    )
```

On dirait donc une fonction de chiffrement / déchiffrement ! Pour l'instruction 10, la chaîne `s` est une des chaînes de `dico` et la clé est la concaténation `(len(state), len(instructions), idx)`. Pour l'instruction 12, la clé est la dernière valeur du state et la chaîne l'avant dernière valeur de ce dernier.

Finalisation. Voici le détail de ce qui se passe dans le programme :

1. `(10, 0)` - Charge la première chaîne du dico et la déchiffre (via `decrypt(strfromhex("40504143475e40540a11"), "0120")`) ce qui va ensuite placer `"password: "` dans le state.
2. `(6,)` - La chaîne est retirée de l'état interne, elle est affichée et input est demandé à l'utilisateur. Le résultat de cet input sera remis dans l'état interne de la VM.
3. `(10, 3)` - Charge la quatrième chaîne en la déchiffrant (via `decrypt(strfromhex("6202514002456d790248"), "1122")`), qui donne la chaîne `S3cr3t_K3y` qui sera mise sur le state.
4. `(12,)` - Fait le déchiffrement avec comme clé la chaîne de bytes précédente et comme chaîne le mot de passe entrée par l'utilisateur, c'est l'opération qui va nous permettre de gagner.
5. `(10, 1)` - Charge la seconde chaîne en la déchiffrant (via `decrypt(strfromhex("535d3d776920320b3217293636705e131a0f670b3d333f194f3c32294f7248"), "1124")`) ce qui donne la chaîne hexadécimale `626c0f435811003f03261b0207416c272b3e553f0c020d2d7e0d001d7e437a` dont les bytes seront mis sur le state.

Il faut donc que l'on arrive à trouver un mot de passe tel que le résultat de 12 donne la même chaîne hexadécimale que l'opération 5. Pour ce faire, on peut la chercher octet par octet avec la clé `Secret_Key`.
Voici le code :

```
toFind = bytes.fromhex("626c0f435811003f03261b0207416c272b3e553f0c020d2d7e0d001d7e437a")
found = b""
for i in range(0, len(toFind)) :
    for j in range(0, 256) :
        newFound = found + j.to_bytes(1, "big")
        if OOOOOOOOOO(newFound.decode("utf-8"), "S3cr3t_k3y")[i] == chr(toFind[i]) :
            found = newFound
            break
print(found)
```

Nous obtenons donc enfin le flag qui est : `1_l1ke_TO_H1d3_StUfF_1n_My_vM:)` !