Reverse - TryAndSolveMe - 150 points

Waïl TAHMAOUI, Kévin DUVERGER

Table des matières

1 Résolution TryAndSolveMe :

Avant de commencer le Write-up, notez que ce challenge est un challenge facile de l'ECW en reverse.

Encore une fois, `strings` ne nous donne pas plus d'informations que le fait que le code teste un mot de passe.

On peut encore le passer dans un décompilateur et cette fois-ci le code est un peu plus lourd :

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
    FILE *v3; // eax
    char Buffer; // [esp+10h] [ebp-430h] BYREF
    char v6; // [esp+11h] [ebp-42Fh]
    ...
    char v22; // [esp+21h] [ebp-41Fh]
    BOOL v23; // [esp+3F8h] [ebp-48h]
    ...
    BOOL v40; // [esp+43Ch] [ebp-4h]

    __main();
    printf("Please enter the password : ");
    v3 = ___acrt_iob_func(0);
    fgets(&Buffer, 999, v3);
    if ( *(&Buffer + strlen(&Buffer) - 1) == 10 )
    *(&Buffer + strlen(&Buffer) - 1) = 0;
    if ( strlen(&Buffer) == 18 ) {
        v40 = 31 * v20 + 27 * v19 + 7 * v18 + 55 * v16 + 71 * v15 + 26 * v14 + 32 * v11 + 91 * v10 +
     38 * v9 + 38 * v8 + 55 * v7 + 32 * Buffer + 15 * v6 + 55 * v12 + 25 * v13 + 22 * v17 + 5 * v21
    + 52 * v22 == 62839;
        v39 = 70 * v21 + 69 * v20 + 98 * v17 + 78 * v16 + 83 * v15 + 97 * v14 + 56 * v13 + 77 * v8 +
     26 * v7 + 90 * Buffer + 4 * v6 + 2 * v9 + 63 * v10 + 59 * v11 + 13 * v12 + 28 * v18 + 6 * v19 +
     (v22 << 6) == 84010;
        v38 = 59 * v19 + 76 * v16 + 51 * v13 + 79 * v12 + (v11 << 6) + 44 * v10 + 5 * v8 + 91 * v6 +
     35 * Buffer + 56 * v7 + 40 * v9 + 82 * v14 + 24 * v15 + 14 * v17 + 33 * v18 + 2 * v20 + 36 *
    v21 + 79 * v22 == 80596;
        v37 = 58 * v21 + 85 * v20 + 60 * v17 + 99 * v16 + 66 * v11 + 51 * v8 + v7 + 82 * v6 + 56 *
    Buffer + 68 * v9 + 3 * v10 + 94 * v12 + 36 * v13 + 23 * v14 + 12 * v15 + 95 * v18 + 40 * v19 +
    57 * v22 == 85317;
        v36 = 4 * v21 + 45 * v20 + 71 * v15 + 87 * v14 + 55 * v11 + 70 * v10 + 42 * v9 + 77 * v8 +
    19 * v6 + 99 * Buffer + 37 * v7 + 70 * v12 + 15 * v13 + 85 * v16 + 13 * v17 + 85 * v18 + 48 *
    v19 + 97 * v22 == 94331;
        v35 = 14 * v21 + 29 * v20 + 58 * v19 + 47 * v16 + 67 * v13 + 8 * v12 + 52 * v11 + 45 * v10 +
     46 * v9 + 51 * v8 + 22 * v7 + 97 * v6 + 61 * Buffer + 79 * v14 + 41 * v15 + 32 * v17 + 20 * v18
     + 61 * v22 == 75345;
        v34 = 8 * v21 + 87 * v20 + 60 * v19 + 71 * v16 + 61 * v15 + 77 * v14 + 57 * v13 + 62 * v12 +
     62 * v9 + 42 * v8 + 95 * v7 + 97 * v6 + 33 * Buffer + 23 * v10 + 9 * v11 + 28 * v17 + 40 * v18
    + 53 * v22 == 90418;
        v33 = 5 * v21 + 72 * v20 + 53 * v18 + 56 * v17 + 28 * v16 + 61 * v15 + 60 * v14 + 59 * v13 +
     77 * v12 + 57 * v11 + 92 * v10 + 52 * v9 + 71 * v8 + 91 * v7 + 14 * Buffer + 40 * v6 + 79 * v19
     + 3 * v22 == 90466;
        v32 = 80 * v18 + 78 * v16 + 60 * v15 + 30 * v12 + 54 * v11 + 35 * v10 + 51 * Buffer + 2 * v6
     + 41 * v7 + 28 * v8 + 96 * v9 + 29 * v13 + 33 * v14 + 91 * v17 + 80 * v19 + 77 * v20 + 18 * v21
     + 70 * v22 == 82738;
        v31 = 79 * v19 + 97 * v18 + 71 * v17 + 3 * v15 + 90 * v13 + 83 * v12 + 34 * v11 + 97 * v10 +
     (v7 << 6) + 43 * Buffer + 18 * v6 + 82 * v8 + 7 * v9 + (v14 << 6) + 25 * v16 + 74 * v20 + 72 *
    v21 + 42 * v22 == 95605;
        v30 = 42 * v21 + 74 * v20 + 77 * v19 + 87 * v18 + 46 * v17 + 7 * v13 + 13 * v12 + 63 * v11 +
     31 * v10 + 3 * v9 + 14 * v7 + 57 * Buffer + 11 * v6 + 16 * v8 + 81 * v14 + 69 * v15 + 5 * v16 +
     v22 == 60247;
        v29 = 4 * v21 + 56 * v18 + 68 * v15 + 61 * v14 + 93 * v13 + 59 * v12 + 45 * v11 + 51 * v10 +
     57 * v9 + 98 * v8 + 79 * v7 + 76 * v6 + 53 * Buffer + 95 * v16 + 41 * v17 + 34 * v19 + 73 * v20
     + 86 * v22 == 104609;
        v28 = 94 * v19 + 55 * v18 + 49 * v17 + 84 * v16 + 75 * v15 + 34 * v14 + 74 * v13 + 44 * v6 +
     24 * Buffer + 39 * v7 + 40 * v8 + 2 * v9 + 73 * v10 + 95 * v11 + 27 * v12 + 8 * v20 + 73 * v21
    + 67 * v22 == 85506;
        v27 = 31 * v20 + 42 * v18 + 98 * v13 + 46 * v12 + 85 * v11 + 94 * v10 + 50 * v9 + 55 * v8 +
    2 * (48 * Buffer + v6 + 27 * v7) + 53 * v14 + 17 * v15 + 44 * v16 + 18 * v17 + 43 * v19 + 25 *
    v21 + 79 * v22 == 83130;
        v26 = 67 * v19 + 40 * v17 + 57 * v13 + 89 * v12 + 38 * v11 + 97 * v10 + 83 * v9 + 16 * v8 +
    94 * Buffer + 22 * v6 + 63 * v7 + 74 * v14 + 6 * v15 + 29 * v16 + 9 * v18 + 70 * v20 + 27 * v21
    + 38 * v22 == 81503;
        v25 = 6 * v21 + 50 * v19 + 50 * v18 + 71 * v17 + 89 * v16 + 35 * v15 + 62 * v14 + 71 * v13 +
     89 * v12 + 52 * v9 + 52 * v8 + 52 * v7 + 91 * Buffer + 2 * v6 + 14 * v10 + 48 * v11 + 26 * v20
    + 10 * v22 == 76907;
        v24 = 79 * v21 + 79 * v20 + 93 * v19 + 37 * v17 + 21 * v14 + 26 * v10 + 90 * v9 + 42 * v6 +
    49 * Buffer + 88 * v7 + 96 * v8 + 76 * v11 + 96 * v12 + 68 * v13 + 31 * v15 + 89 * v16 + 40 *
    v18 + 78 * v22 == 104158;
```

```
        v23 = 78 * v18 + 76 * v17 + 34 * v16 + 82 * v15 + 15 * v11 + 40 * v10 + 29 * v8 + 24 *
    Buffer + 14 * v6 + 31 * v7 + 83 * v9 + 36 * v12 + v13 + 40 * v14 + 35 * v19 + 65 * v20 + 43 *
    v21 + 17 * v22 == 66284;
        if (v40 && v39 && v38 && v37 && v36 && v35 && v34 && v33 && v32 && v31 && v30 && v29 && v28
    && v27 && v26 && v25 && v24 && v23) {
            puts("Good job ! You found the password !");
        } else {
            puts("That's not the password !");
        }
        return 0;
    } else {
        puts("You do not have a password of the right length !");
        return 0;
    }
}
```

Grâce à ce code, vous pouvez voir que le programme semble vérifier un ensemble d'équations qui dépendent de constantes et des caractères donnés par l'utilisateur (faites bien attention à l'ordre des variables qui est quelque peu particulier ici).
Il s'agit en fait d'un système d'équations linéaires à 18 variables que l'on peut représenter par le système suivant :

$$\begin{pmatrix}
32 & 15 & 55 & 38 & 38 & 91 & 32 & 55 & 25 & 26 & 71 & 55 & 22 & 7 & 27 & 31 & 5 & 52 \\
90 & 4 & 26 & 77 & 2 & 63 & 59 & 13 & 56 & 97 & 83 & 78 & 98 & 28 & 6 & 69 & 70 & 64 \\
35 & 91 & 56 & 5 & 40 & 44 & 64 & 79 & 51 & 82 & 24 & 76 & 14 & 33 & 59 & 2 & 36 & 79 \\
56 & 82 & 1 & 51 & 68 & 3 & 66 & 94 & 36 & 23 & 12 & 99 & 60 & 95 & 40 & 85 & 58 & 57 \\
99 & 19 & 37 & 77 & 42 & 70 & 55 & 70 & 15 & 87 & 71 & 85 & 13 & 85 & 48 & 45 & 4 & 97 \\
61 & 97 & 22 & 51 & 46 & 45 & 52 & 8 & 67 & 79 & 41 & 47 & 32 & 20 & 58 & 29 & 14 & 61 \\
33 & 97 & 95 & 42 & 62 & 23 & 9 & 62 & 57 & 77 & 61 & 71 & 28 & 40 & 60 & 87 & 8 & 53 \\
14 & 40 & 91 & 71 & 52 & 92 & 57 & 77 & 59 & 60 & 61 & 28 & 56 & 53 & 79 & 72 & 5 & 3 \\
51 & 2 & 41 & 28 & 96 & 35 & 54 & 30 & 29 & 33 & 60 & 78 & 91 & 80 & 80 & 77 & 18 & 70 \\
43 & 18 & 64 & 82 & 7 & 97 & 34 & 83 & 90 & 64 & 3 & 25 & 71 & 97 & 79 & 74 & 72 & 42 \\
57 & 11 & 14 & 16 & 3 & 31 & 63 & 13 & 7 & 81 & 69 & 5 & 46 & 87 & 77 & 74 & 42 & 1 \\
53 & 76 & 79 & 98 & 57 & 51 & 45 & 59 & 93 & 61 & 68 & 95 & 41 & 56 & 34 & 73 & 4 & 86 \\
24 & 44 & 39 & 40 & 2 & 73 & 95 & 27 & 74 & 34 & 75 & 84 & 49 & 55 & 94 & 8 & 73 & 67 \\
96 & 2 & 54 & 55 & 50 & 94 & 85 & 46 & 98 & 53 & 17 & 44 & 18 & 42 & 43 & 31 & 25 & 79 \\
94 & 22 & 63 & 16 & 83 & 97 & 38 & 89 & 57 & 74 & 6 & 29 & 40 & 9 & 67 & 70 & 27 & 38 \\
91 & 2 & 52 & 52 & 52 & 14 & 48 & 89 & 71 & 62 & 35 & 89 & 71 & 50 & 50 & 26 & 6 & 10 \\
49 & 42 & 88 & 96 & 90 & 26 & 76 & 96 & 68 & 21 & 31 & 89 & 37 & 40 & 93 & 79 & 79 & 78 \\
24 & 14 & 31 & 29 & 83 & 40 & 15 & 36 & 1 & 40 & 82 & 34 & 76 & 78 & 35 & 65 & 43 & 17
\end{pmatrix} \times \begin{pmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \\ P_4 \\ P_5 \\ P_6 \\ P_7 \\ P_8 \\ P_9 \\ P_{10} \\ P_{11} \\ P_{12} \\ P_{13} \\ P_{14} \\ P_{15} \\ P_{16} \\ P_{17} \end{pmatrix} = \begin{pmatrix}
62839 \\ 84010 \\ 80596 \\ 85317 \\ 94331 \\ 75345 \\ 90418 \\ 90466 \\ 82738 \\ 95605 \\ 60247 \\ 104609 \\ 85506 \\ 83130 \\ 81503 \\ 76907 \\ 104158 \\ 66284
\end{pmatrix}$$

Nous avons donc quelque chose de la forme $Ax = v$ ce que l'on peut transformer en $x = A^{-1}v$!

Le programme pour le résoudre est donc le suivant :

```
import numpy as np

matrix = np.array([
    [ 32, 15, 55, 38, 38, 91, 32, 55, 25, 26, 71, 55, 22,  7, 27, 31,  5, 52 ],
    [ 90,  4, 26, 77,  2, 63, 59, 13, 56, 97, 83, 78, 98, 28,  6, 69, 70, 64 ],
    [ 35, 91, 56,  5, 40, 44, 64, 79, 51, 82, 24, 76, 14, 33, 59,  2, 36, 79 ],
    [ 56, 82,  1, 51, 68,  3, 66, 94, 36, 23, 12, 99, 60, 95, 40, 85, 58, 57 ],
    [ 99, 19, 37, 77, 42, 70, 55, 70, 15, 87, 71, 85, 13, 85, 48, 45,  4, 97 ],
    [ 61, 97, 22, 51, 46, 45, 52,  8, 67, 79, 41, 47, 32, 20, 58, 29, 14, 61 ],
    [ 33, 97, 95, 42, 62, 23,  9, 62, 57, 77, 61, 71, 28, 40, 60, 87,  8, 53 ],
    [ 14, 40, 91, 71, 52, 92, 57, 77, 59, 60, 61, 28, 56, 53, 79, 72,  5,  3 ],
    [ 51,  2, 41, 28, 96, 35, 54, 30, 29, 33, 60, 78, 91, 80, 80, 77, 18, 70 ],
    [ 43, 18, 64, 82,  7, 97, 34, 83, 90, 64,  3, 25, 71, 97, 79, 74, 72, 42 ],
    [ 57, 11, 14, 16,  3, 31, 63, 13,  7, 81, 69,  5, 46, 87, 77, 74, 42,  1 ],
    [ 53, 76, 79, 98, 57, 51, 45, 59, 93, 61, 68, 95, 41, 56, 34, 73,  4, 86 ],
    [ 24, 44, 39, 40,  2, 73, 95, 27, 74, 34, 75, 84, 49, 55, 94,  8, 73, 67 ],
    [ 96,  2, 54, 55, 50, 94, 85, 46, 98, 53, 17, 44, 18, 42, 43, 31, 25, 79 ],
    [ 94, 22, 63, 16, 83, 97, 38, 89, 57, 74,  6, 29, 40,  9, 67, 70, 27, 38 ],
    [ 91,  2, 52, 52, 52, 14, 48, 89, 71, 62, 35, 89, 71, 50, 50, 26,  6, 10 ],
    [ 49, 42, 88, 96, 90, 26, 76, 96, 68, 21, 31, 89, 37, 40, 93, 79, 79, 78 ],
    [ 24, 14, 31, 29, 83, 40, 15, 36,  1, 40, 82, 34, 76, 78, 35, 65, 43, 17 ]
])

matrix_inv = np.linalg.inv(matrix)

vector = [
    0xf577, 0x1482a, 0x13ad4, 0x14d45, 0x1707b, 0x12651, 0x16132, 0x16162, 0x14332, 0x17575, 0xeb57,
     0x198a1, 0x14e02, 0x144ba, 0x13e5f, 0x12c6b, 0x196de, 0x102ec
]

```

```
for i in range(0, 18) :
    letter = 0
    for j in range(0, 18) :
        letter += matrix_inv[i][j] * vector[j]
    print(chr(round(letter)))
```

1.1 Une autre façon de résoudre le challenge :

On peut aussi voir la liste de conditions précédentes comme une liste de contraintes que l'on peut résoudre avec un bon algorithme d'optimisation entière ! Python dispose d'une librairie (à installer) qui est faite pour ça : **z3**. Il suffit de rentrer les différentes contraintes, qui sont les conditions de la boucle if, ce qui donne le code suivant (notez que cette partie a été faite avec Ghidra, d'où les noms de variables différents dans la condition) :

```
from z3 import *

x = [ Int(f"x{i}") for i in range(18) ]
s = Solver()

# bornes ASCII lisibles
for v in x:
    s.add(v >= 32, v <= 126)
ac1, ac2, ac3 = x[0], x[1], x[2]
cs = x[3:17]
c43d, c43c, c43b, c43a, c439, c438, c437, c436, c435, c434, c433, c432, c431, c430 = cs
c42f = x[17]

# === Contraintes ===
s.add(c42f*0x34 + ac1*0x20 + ac2*0xf + ac3*0x37 + c43d*0x26 + c43c*0x26 + c43b*0x5b + c43a*0x20 +
    c439*0x37 + c438*0x19 + c437*0x1a + c436*0x47 + c435*0x37 + c434*0x16 + c433*7 + c432*0x1b +
    c431*0x1f + c430*5 == 0xf577)
#.
#.
#. Les autres contraintes ...
#.
#.
s.add(c42f*0x11 + ac1*0x18 + ac2*0xe + ac3*0x1f + c43d*0x1d + c43c*0x53 + c43b*0x28 + c43a*0xf +
    c439*0x24 + c438*1 + c437*0x28 + c436*0x52 + c435*0x22 + c434*0x4c + c433*0x4e + c432*0x23 +
    c431*0x41 + c430*0x2b == 0x102ec)

# === Solve ===
if s.check() == sat:
    m = s.model()
    pw = "".join(chr(m[v].as_long()) for v in x)
    print("Mot de passe trouve:", pw)
else:
    print("Pas de solution :(")
```