

Hepia

Haute École du Paysage, d'Ingénierie et d'Architecture

Ingénierie et Systèmes de Communication

Année académique 2020/2021

Hacking et pentesting

Série 4 – Modification de code

Genève, 18 Avril 2021

Étudiant :

Sergio Guarino

Professeur :

Stéphane Küng

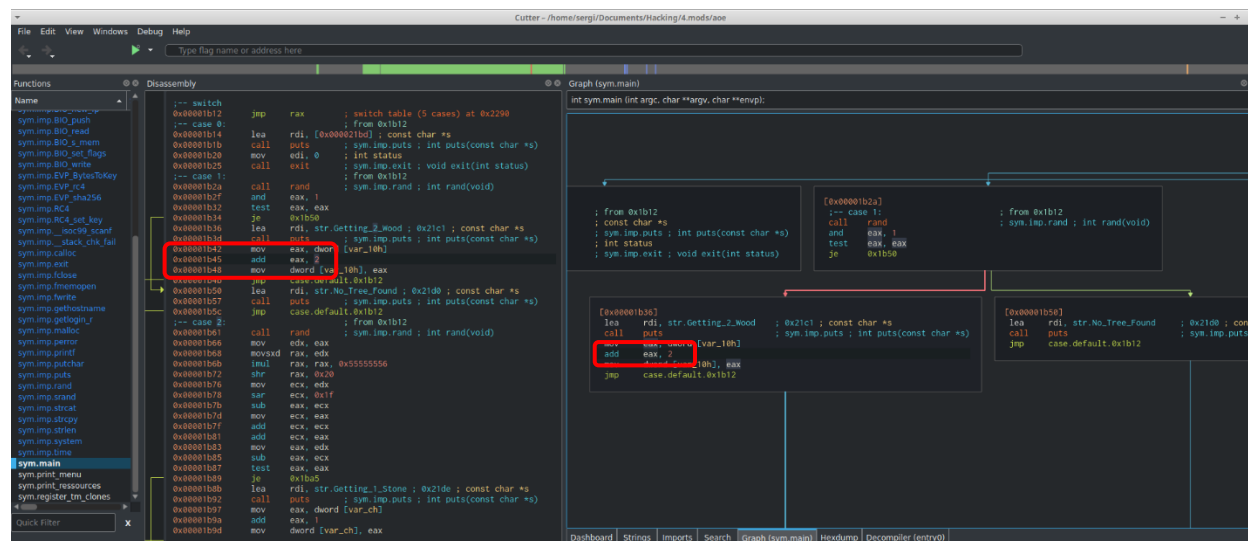
1. Age of Empire 3

Dans ce premier exercice, il faut modifier le binaire pour le faire agir différemment par rapport à la programmation de base.

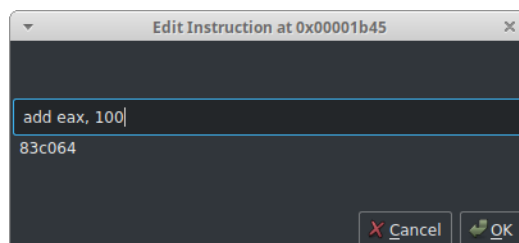
Dans ce cas, on souhaite augmenter le nombre de ressources fournies pendant le jeu, sans modifier les ressources initiales.

Pour cela on a décidé d'utiliser le programme **Cutter**, qui est un éditeur de binaires basé sur **radare2**.

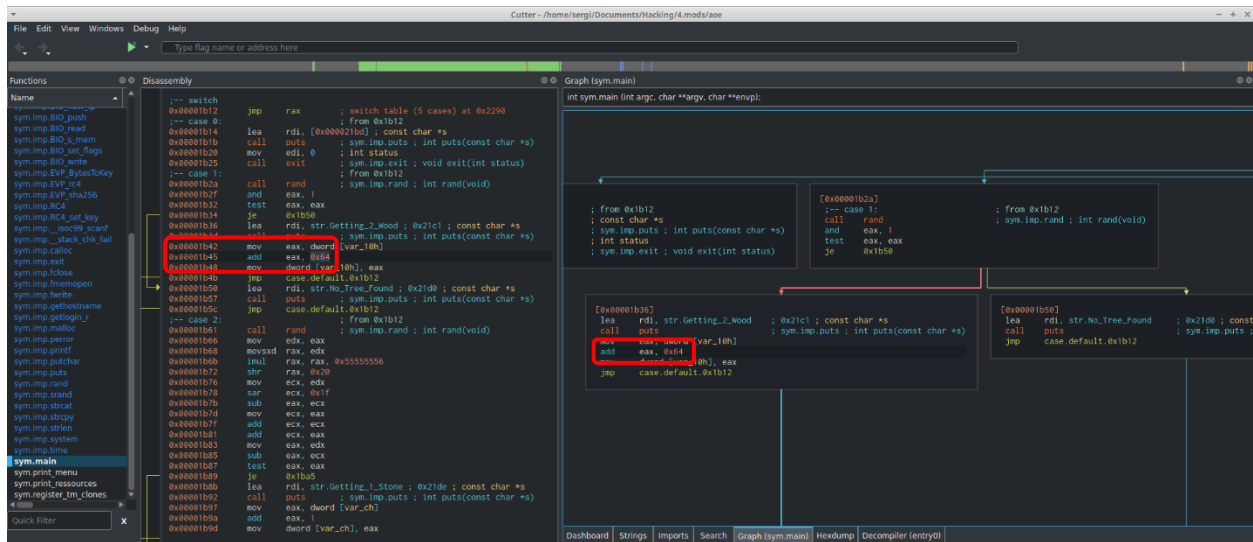
Après avoir importé l'exécutable de Age of Empire dans Cutter en mode écriture, on peut analyser le code et chercher la ou les lignes à modifier pour obtenir le résultat désiré. Ci-dessous on a identifié la ligne qu'il faut modifier pour obtenir plus de bois :



On fait un clic-droit et on modifie la valeur. Dans ce cas on a choisi de la passer à 100 :



Et on peut voir que la modification a bien été prise en compte :



Et pour tester on lance le jeu :

```

Terminal - ./aoe
File Edit View Terminal Tabs Help

Getting 2 Wood
A soldier died from a common disease !

You have : 110 Wood, 20 Stones, 50 Golds, 24 Soldiers
-----
What do you want to do ?
1) Collect Wood
2) Collect Stone
3) Collect Gold
4) Buy a Castle (Wood 200, Stone 1000, Golds 5000)
0) Exit

```

Il suffit de faire la même chose pour les autres ressources pour pouvoir gagner le jeu plus facilement. On peut même remplacer les lignes qui nous font perdre des soldats avec des **NOP** de manière à ne jamais perdre.

```

Terminal - ./aoe
File Edit View Terminal Tabs Help

Getting 1 Stone
A soldier died from a common disease !

You have : 110 Wood, 120 Stones, 50 Golds, 21 Soldiers
-----
What do you want to do ?
1) Collect Wood
2) Collect Stone
3) Collect Gold
4) Buy a Castle (Wood 200, Stone 1000, Golds 5000)
0) Exit

```

```
Terminal - ./aoe
File Edit View Terminal Tabs Help
Getting 2 Gold

You have : 110 Wood, 120 Stones, 150 Golds, 25 Soldiers
-----
What do you want to do ?
1) Collect Wood
2) Collect Stone
3) Collect Gold
4) Buy a Castle (Wood 200, Stone 1000, Golds 5000)
0) Exit
```

2. CrackMe4 C# 1

Dans ce deuxième exercice, il faut trouver un mot de passe caché dans un exécutable Windows écrit en C#.

Avant tout, on le décompile. Pour cela ils existent plusieurs logiciels facilement accessibles en ligne. Pour cet exercice, le module **JetBrains dotPeek** a été utilisé.

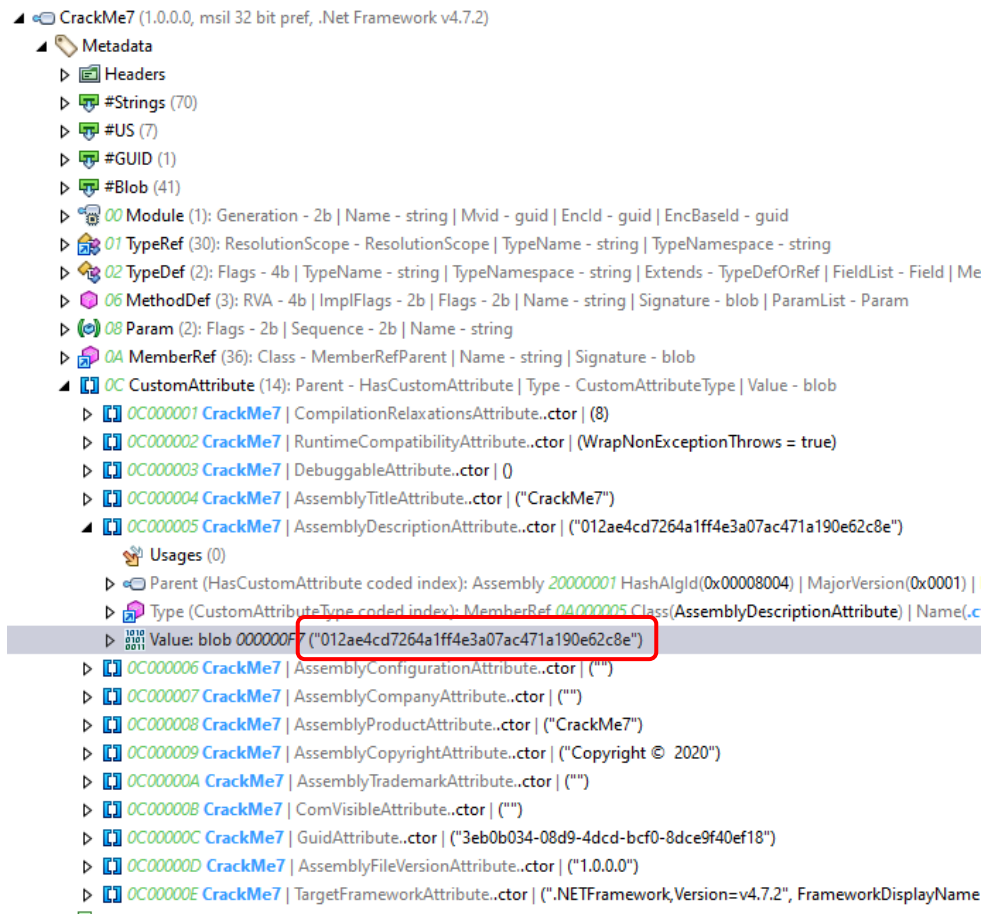
La décompilation nous montre les lignes suivantes :

```
namespace CrackMe7
{
    internal class Program
    {
        private static string sha256(string randomString)
        {
            SHA256Managed sha256Managed = new SHA256Managed();
            StringBuilder stringBuilder = new StringBuilder();
            foreach (byte num in sha256Managed.ComputeHash(Encoding.UTF8.GetBytes(randomString)))
            {
                stringBuilder.Append(num.ToString("x2"));
            }
            return stringBuilder.ToString();
        }

        private static void Main(string[] args)
        {
            if (args.Length < 1)
            {
                Console.WriteLine("Usage : ./CrackMe PASSWORD");
            }
            if (args.Length == 1)
            {
                if (args[0] != null && args[0].Length == 12)
                {
                    string str1 = "6702882accdf7b55591d5e4ef6a";
                    int maxValue = (int) ushort.MaxValue;
                    string str2 = Program.sha256(str1 + ToString());
                    string description = Assembly.GetExecutingAssembly().GetCustomAttributes(typeof(AssemblyDescriptionAttribute), false).OfType<AssemblyDescriptionAttribute>().FirstOrDefault<AssemblyDescriptionAttribute>().Description;
                    string str3 = str2;
                    string str4 = str1 + description;
                    for (int index = 0; index <= maxValue; ++index)
                    {
                        str3 = Program.sha256(str3 + index.ToString());
                        str4 = Program.sha256(str4 + index.ToString());
                        if (str3.StartsWith("0000"))
                        {
                            break;
                        }
                    }
                    if (str3 == str4)
                    {
                        Console.WriteLine("Congratulation, don't forget to explain how you find the password solution !");
                    }
                    else
                    {
                        Console.WriteLine("Wrong Password");
                    }
                }
            }
        }
    }
}
```

On remarque que le mot de passe doit être de 12 caractères et est composé d'un hash SHA256 composé de 2 parties :

- **str1** qui vaut **6702882accdf7b55591d5e4ef6a**
- **description** que l'on peut trouver dans les attributs des métadonnées du code et vaut **012ae4cd7264a1ff4e3a07ac471a190e62c8e**



Le hash à retrouver est donc :

6702882accdf7b55591d5e4ef6a012ae4cd7264a1ff4e3a07ac471a190e62c8e

Il n'y a pas besoin d'effectuer une attaque brute-force sur ce code, car on sait qu'en ligne on trouve des tables de hash dans lesquelles on pourrait comparer le hash qu'on vient de trouver. Le site web <https://hashes.com/en/decrypt/hash> a été utilisé pour cette recherche. Le résultat est **goodpassword**. On teste le mot de passe et on peut voir qu'il est correct :

```

C:\> Command Prompt

Sergio's cmd: CrackMe4.exe goodpassword
Congratulation, don't forget to explain how you find the password solution !

Sergio's cmd: CrackMe4.exe wrongpassword
Wrong Password

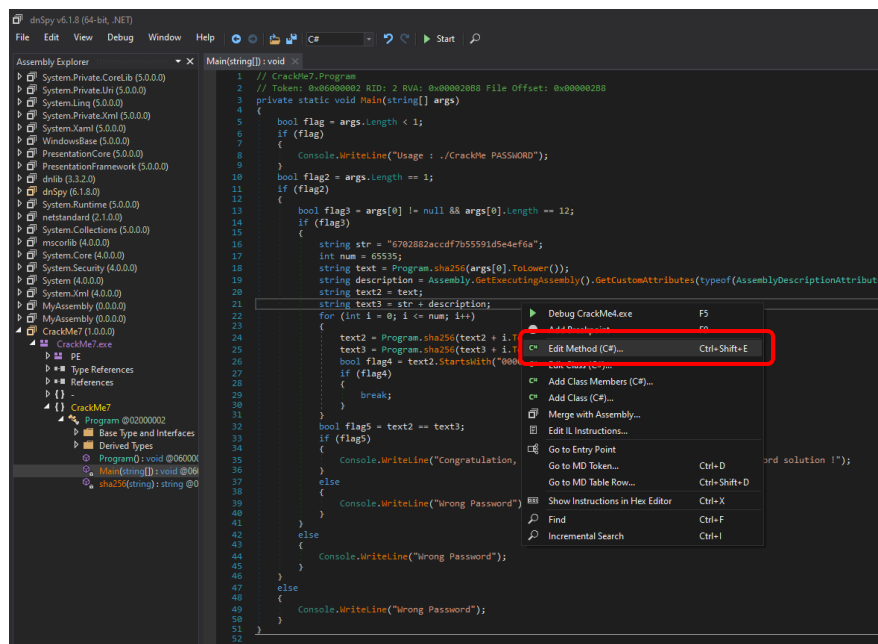
Sergio's cmd:

```

3. CrackMe4 C# 2

Pour cet exercice il faut modifier le code source de l'exécutable et faire en sorte que le mot de passe à rentrer en paramètre soit notre prénom. Dans ce cas, ça sera Sergio.

Il y a plusieurs programmes qui permettent de modifier un exécutable. Celui choisi pour cet exercice est **dnSpy**. On importe le fichier à modifier et on fait un clic-droit pour l'éditer :



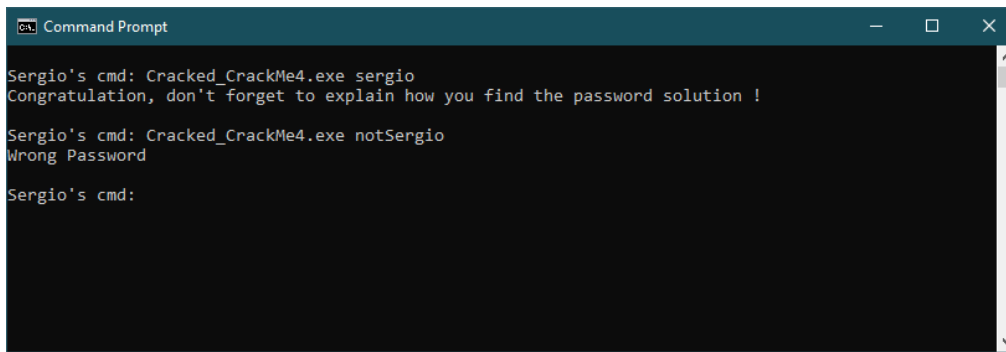
On peut alors remplacer **text3** par le hash du prénom, qui sera :

36033babfb48ec64e197c97fb40d65e6c79f81e04c61aeccef3009e01645ab8d

Le hash SHA256 peut être calculé avec des services en ligne. Le navigateur DuckDuckGo a cette fonctionnalité intégrée, donc la recherche pour « sha256 sergio » nous donne directement le résultat souhaité. Il faut rentrer le mot en minuscules car l'argument passé quand on lance le programme est transformé en minuscules (partie du code : `args[0].ToLower`) et le hash du mot « Sergio » n'est pas le même que celui de « sergio ».

Autre que le hash, il faut également modifier la longueur du mot de passe demandé, puisque le nouveau mot de passe ne fait plus 12 caractères. Ou bien on peut même supprimer la vérification de la longueur.

Après avoir modifié le code, on va depuis **File -> Save Module...** pour sauvegarder le code édité. Si on teste, on peut voir que le nouveau mot de passe fonctionne correctement :



```
Command Prompt

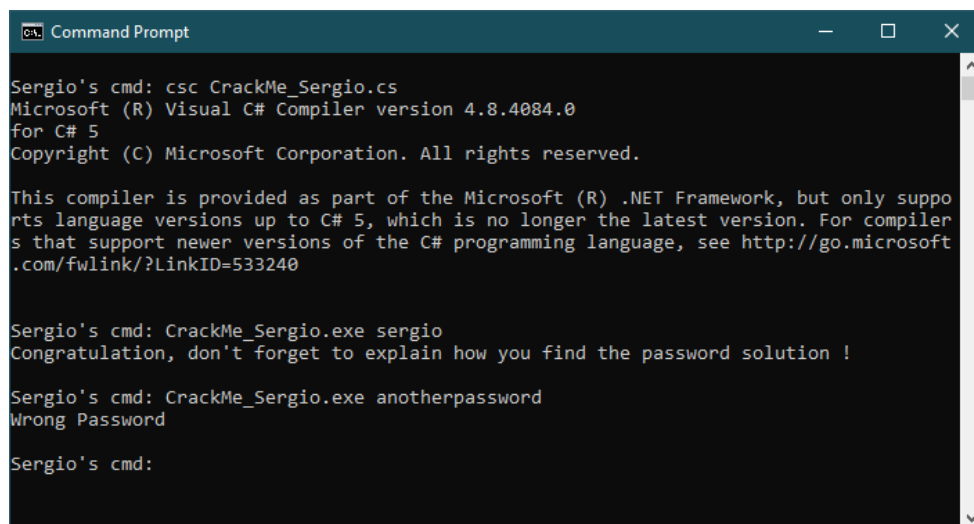
Sergio's cmd: Cracked_CrackMe4.exe sergio
Congratulation, don't forget to explain how you find the password solution !

Sergio's cmd: Cracked_CrackMe4.exe notSergio
Wrong Password

Sergio's cmd:
```

Solution alternative

Cet exercice demandait de modifier l'exécutable, mais on aurait également pu faire un copier-coller du code dans un éditeur du texte, le modifier comme ça a été fait avec dnSpy et puis le compiler :



```
Command Prompt

Sergio's cmd: csc CrackMe_Sergio.cs
Microsoft (R) Visual C# Compiler version 4.8.4084.0
for C# 5
Copyright (C) Microsoft Corporation. All rights reserved.

This compiler is provided as part of the Microsoft (R) .NET Framework, but only supports language versions up to C# 5, which is no longer the latest version. For compilers that support newer versions of the C# programming language, see http://go.microsoft.com/fwlink/?LinkID=533240

Sergio's cmd: CrackMe_Sergio.exe sergio
Congratulation, don't forget to explain how you find the password solution !

Sergio's cmd: CrackMe_Sergio.exe anotherpassword
Wrong Password

Sergio's cmd:
```

4. CrackMe5 Time Attack

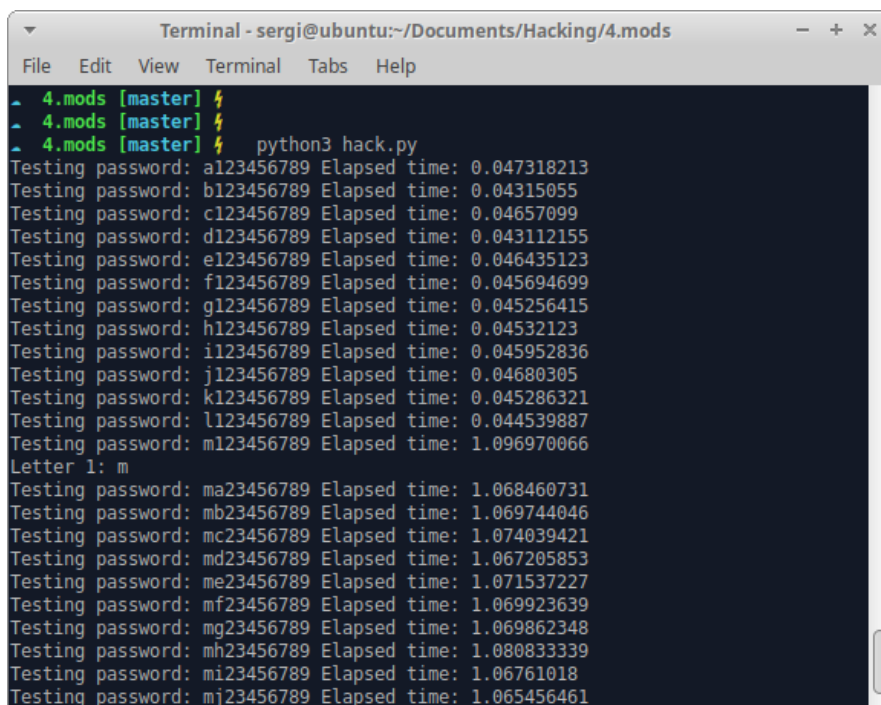
Le dernier exercice demande de trouver un mot de passe en utilisant un Time Attack, soit une attaque qui calcule le temps qu'un programme met à comparer le mot de passe stockée avec celui passé en paramètre.

Comme première possibilité de résolution, on pourrait utiliser la commande bash **time**, qui calcule le temps d'exécution d'un programme, mais ce n'est pas suffisamment précise et ça prendra beaucoup de temps pour trouver le bon mot de passe.

Un script en python a été écrit pour résoudre l'exercice de manière plus simple et rapide. Les modules **subprocess** et **timeit** permettent respectivement d'exécuter un processus et de mesurer le temps pris. En particulier, **timeit** a une fonctionnalité qui permet de lancer le processus plusieurs fois et puis nous retourner la moyenne du temps pris pour l'exécution. Le nombre d'exécutions par défaut est de 100'000, mais pour cet exercice on a pris une valeur de 25.

Le code utilisé est assez simple : on teste tous les mots des passes possibles, une lettre à la fois. Le mot qui prend le plus de temps pour être vérifié, est celui correct.

Par exemple, ci-dessous on teste la première lettre et on peut voir que, quand on arrive à la **m**, le temps d'exécution est nettement supérieur. Le programme a été écrit de façon à ne pas vérifier les lettres suivantes si on a trouvé celle qui semble être correcte.



```
Terminal - sergi@ubuntu:~/Documents/Hacking/4.mods
File Edit View Terminal Tabs Help
4.mods [master] $
4.mods [master] $
4.mods [master] $ python3 hack.py
Testing password: a123456789 Elapsed time: 0.047318213
Testing password: b123456789 Elapsed time: 0.04315055
Testing password: c123456789 Elapsed time: 0.04657099
Testing password: d123456789 Elapsed time: 0.043112155
Testing password: e123456789 Elapsed time: 0.046435123
Testing password: f123456789 Elapsed time: 0.045694699
Testing password: g123456789 Elapsed time: 0.045256415
Testing password: h123456789 Elapsed time: 0.04532123
Testing password: i123456789 Elapsed time: 0.045952836
Testing password: j123456789 Elapsed time: 0.04680305
Testing password: k123456789 Elapsed time: 0.045286321
Testing password: l123456789 Elapsed time: 0.044539887
Testing password: m123456789 Elapsed time: 1.096970066
Letter 1: m
Testing password: ma23456789 Elapsed time: 1.068460731
Testing password: mb23456789 Elapsed time: 1.069744046
Testing password: mc23456789 Elapsed time: 1.074039421
Testing password: md23456789 Elapsed time: 1.067205853
Testing password: me23456789 Elapsed time: 1.071537227
Testing password: mf23456789 Elapsed time: 1.069923639
Testing password: mg23456789 Elapsed time: 1.069862348
Testing password: mh23456789 Elapsed time: 1.080833339
Testing password: mi23456789 Elapsed time: 1.06761018
Testing password: mj23456789 Elapsed time: 1.065456461
```

Le programme marche plutôt bien jusqu'à la 8^{ème} lettre, mais pour les deux dernières il n'est pas possible d'identifier le bon caractère. En effet, à cause du long temps d'exécution du programme, d'autres processus interfèrent avec l'exécution et faussent la mesure. On peut voir ci-dessous que plusieurs combinaisons ont pris plus de **5.1** secondes :


```
Terminal - sergi@ubuntu:~/Documents/Hacking/4.mods
File Edit View Terminal Tabs Help
4.mods [master] $ python3 hack.py
Testing password: mFT8nHb2a1 Elapsed time: 5.028392747
Testing password: mFT8nHb2b1 Elapsed time: 5.1424396
Testing password: mFT8nHb2c1 Elapsed time: 5.307730765
Testing password: mFT8nHb2d1 Elapsed time: 5.087701125
Testing password: mFT8nHb2e1 Elapsed time: 5.053382392
Testing password: mFT8nHb2f1 Elapsed time: 5.051394042
Testing password: mFT8nHb2g1 Elapsed time: 5.050929586
Testing password: mFT8nHb2h1 Elapsed time: 5.056662493
Testing password: mFT8nHb2i1 Elapsed time: 5.045566023
Testing password: mFT8nHb2j1 Elapsed time: 5.066648428
Testing password: mFT8nHb2k1 Elapsed time: 5.065284671
Testing password: mFT8nHb2l1 Elapsed time: 5.04671907
Testing password: mFT8nHb2m1 Elapsed time: 5.048829665
Testing password: mFT8nHb2n1 Elapsed time: 5.045757059
Testing password: mFT8nHb2o1 Elapsed time: 5.054208079
Testing password: mFT8nHb2p1 Elapsed time: 5.300407025
Testing password: mFT8nHb2q1 Elapsed time: 5.060449457
Testing password: mFT8nHb2r1 Elapsed time: 5.043933807
Testing password: mFT8nHb2s1 Elapsed time: 5.051187422
Testing password: mFT8nHb2t1 Elapsed time: 5.035451566
Testing password: mFT8nHb2u1 Elapsed time: 5.155869093
Testing password: mFT8nHb2v1 Elapsed time: 5.067517686
Testing password: mFT8nHb2w1 Elapsed time: 5.081684646
Testing password: mFT8nHb2x1 Elapsed time: 5.052731097
Testing password: mFT8nHb2y1 Elapsed time: 5.038506211
Testing password: mFT8nHb2z1 Elapsed time: 5.027009346
Testing password: mFT8nHb2A1 Elapsed time: 5.031287021
Testing password: mFT8nHb2B1 Elapsed time: 5.083808084
Testing password: mFT8nHb2C1 Elapsed time: 5.208924341
Testing password: mFT8nHb2D1 Elapsed time: 5.022161622
Testing password: mFT8nHb2E1 Elapsed time: 5.016013138
Testing password: mFT8nHb2F1 Elapsed time: 5.04293424
Testing password: mFT8nHb2G1 Elapsed time: 5.030738796
Testing password: mFT8nHb2H1 Elapsed time: 5.050465745
Testing password: mFT8nHb2I1 Elapsed time: 5.039524807
Testing password: mFT8nHb2J1 Elapsed time: 5.095211767
Testing password: mFT8nHb2K1 Elapsed time: 5.058062428
Testing password: mFT8nHb2L1 Elapsed time: 5.029568209
Testing password: mFT8nHb2M1 Elapsed time: 5.03592159
Testing password: mFT8nHb2N1 Elapsed time: 5.049576549
Testing password: mFT8nHb2O1 Elapsed time: 5.109456527
Testing password: mFT8nHb2P1 Elapsed time: 5.232810878
Testing password: mFT8nHb2Q1 Elapsed time: 5.048078895
```

La meilleure stratégie serait de les tester une par une, mais la vérification du 10^{ème} caractère fausse encore plus la mesure. On a donc opté pour une attaque brute-force sur les 2 dernières lettres. Comme il y a 62 caractères en total (26 lettres minuscules, 26 majuscules et 10 chiffres), on a « que » 62² combinaisons à tester (soit 3844), qui est un nombre relativement petit pour un ordinateur. En vérité, si on analyse le code, on remarque que le string de caractère utilisée pour générer le mot de passe n'inclue pas toutes les lettres de l'alphabet.

Ainsi faisant, le mot de passe trouvé a été **mFT8nHb2uq**.

Bizarrement, si on exécute le code en testant uniquement la dernière lettre, on peut bien remarquer une différence dans les temps d'exécution pour la lettre **q** :

```
Terminal - sergi@ubuntu:~/Documents/Hacking/4.mods
File Edit View Terminal Tabs Help
4.mods [master] $ python3 hack.py
Testing password: mFT8nHb2ua Elapsed time: 5.173710784
Testing password: mFT8nHb2ub Elapsed time: 5.196786382
Testing password: mFT8nHb2uc Elapsed time: 5.159589894
Testing password: mFT8nHb2ud Elapsed time: 5.170836013
Testing password: mFT8nHb2ue Elapsed time: 5.507046094
Testing password: mFT8nHb2uf Elapsed time: 5.221344329
Testing password: mFT8nHb2ug Elapsed time: 5.190302004
Testing password: mFT8nHb2uh Elapsed time: 5.229850045
Testing password: mFT8nHb2ui Elapsed time: 5.455903795
Testing password: mFT8nHb2uj Elapsed time: 5.29384108
Testing password: mFT8nHb2uk Elapsed time: 5.14833412
Testing password: mFT8nHb2ul Elapsed time: 5.288381551
Testing password: mFT8nHb2um Elapsed time: 5.802162521
Testing password: mFT8nHb2un Elapsed time: 5.231356848
Testing password: mFT8nHb2uo Elapsed time: 5.206413975
Testing password: mFT8nHb2up Elapsed time: 5.773168083
Testing password: mFT8nHb2uq Elapsed time: 7.334661048
Testing password: mFT8nHb2ur Elapsed time: 5.131568184
Testing password: mFT8nHb2us Elapsed time: 5.758757061
Testing password: mFT8nHb2ut Elapsed time: 5.486711102
Testing password: mFT8nHb2uu Elapsed time: 5.551736377
Testing password: mFT8nHb2uv Elapsed time: 5.151862609
Testing password: mFT8nHb2uw Elapsed time: 5.128141827
```

Annexes

Cracked_CrackMe4.exe -> exécutable CrackMe4 modifié avec dnSpy

CrackMe_Sergio.cs -> code C# de CrackMe4. Modifié pour que le mot de passe soit « sergio »

CrackMe_Sergio.exe -> exécutable généré à partir de « CrackMe_Sergio.cs »

hack.py -> code python utilisé pour résoudre CrackMe5