Rapport Exercices Sécurité des Applications : Série 4

Thomas Dagier

October, 20, 2020

1 Age of Empire 1 (MEM1)

Le jeu étant trop compliqué, il faut trouver une astuce pour pouvoir acheter le chateau. Pour cela, on peut faire une édition mémoire ce qui va avoir pour effet de faire croire au jeu que l'on a une certaine somme de bois, pierre ou or alors que ce sont les valeurs en mémoire qui ont changées. Sous Linux, un outil très pratique est `scanmem` qui permet de voir l'utilisation mémoire d'un processus en cours.
Pour modifier les valeurs, on commence par lancer le programme : `./aoe`. De sorte à utiliser `scanmem` il nous faut le PID du processus en cours. On peut l'obtenir avec la commande :
`ps -ef`

```
thomas     60889   60864  0 22:58 pts/1    00:00:00 ./aoe
thomas     60892   60842  0 22:58 pts/0    00:00:00 ps -ef
```

Figure 1: obtention du PID (ici 60889)

Une fois récupéré, on peut lancer scanmem avec la commande : `scanmem 60889`. on arrive donc dans un invite de commande dans lequel on peut utiliser `help` pour trouver une manipulation qui pourrait être utile. La commande `set <match_id>` permet de modifier la valeur qui est à une certaine adresse mais il reste encore à determiner sur quelle adresse le faire. Etant très mauvais joueur avec mon frère, j'ai évidement essayé d'utiliser cheat engine pour modifier l'or sur civilisation V. J'ai donc testé la même méthode, c'est-à-dire :

- modifer le nombre d'or "légalement" dans le jeu

- entrer le nombre actuel dans le terminal (ce qui va retourner un certains nombre d'adresses qui pointent le nombre entré)

- recommencer les deux étapes jusqu'a ce qu'il ne reste qu'un match

- utiliser la commande : `set <new_amount>`.
- faire n'importe quelle modification dans le jeu pour voir que les changements ont bien été fait.

```
> 50
01/08 searching 0x562c68176000 - 0x562c68177000..........ok
02/08 searching 0x562c6a029000 - 0x562c6a04a000..........ok
03/08 searching 0x7f50043c1000 - 0x7f50043c3000..........ok
04/08 searching 0x7f50043e2000 - 0x7f50043e6000..........ok
05/08 searching 0x7f50045da000 - 0x7f50045de000..........ok
06/08 searching 0x7f50049ff000 - 0x7f5004a05000..........ok
07/08 searching 0x7f5004a4b000 - 0x7f5004a4c000..........ok
08/08 searching 0x7ffc2d7c1000 - 0x7ffc2d7e2000..........ok
info: we currently have 32 matches.
32> 52
......ok
info: we currently have 1 matches.
info: match identified, use "set" to modify value.
info: enter "help" for other commands.
1> set 5000
info: setting *0x7ffc2d7df43c to 0x1388...
1> reset
info: maps file located at /proc/60889/maps opened.
info: 8 suitable regions found.
>
```

Figure 2: modification du nombre d'or avec scanmem

On peut donc voir qu'après avoir isolé l'adresse qui contient le nombre d'or récolté et après avoir modifié la valeur, le jeu affiche bien le nouveau montant :

Figure 3: nouveau montant d'or

Dès lors, on répète ces opérations pour le bois et la pierre de sorte à obtenir le minimum dans toutes les ressources pour acheter le chateau tant convoité :

Figure 4: affichage du montant modifé pour chaque ressource

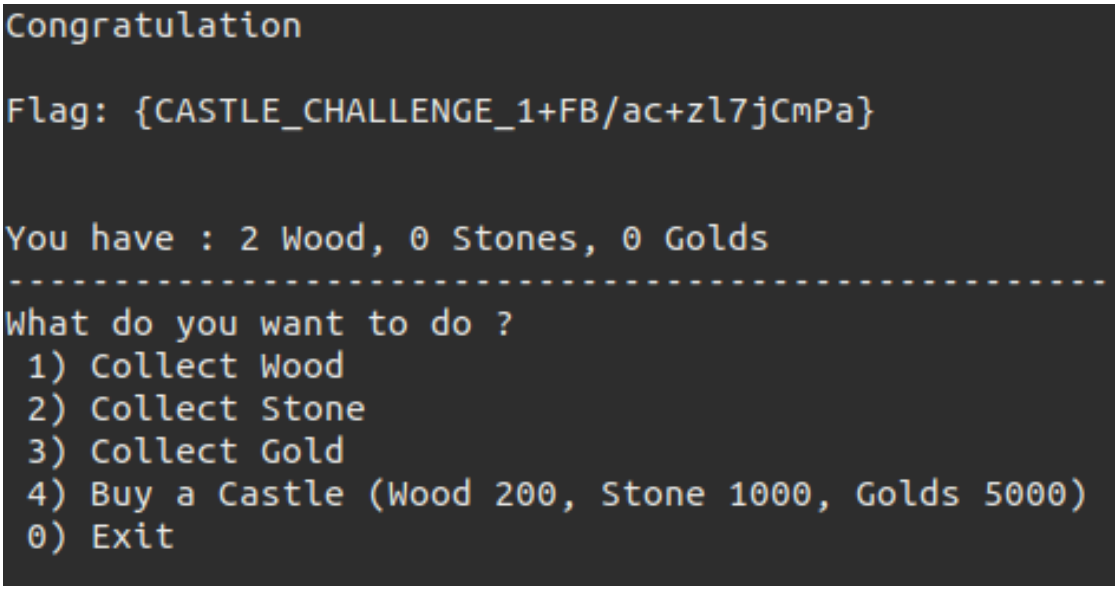

Figure 5: obtention du flag après l'achat du château

2 Age of Empire 3 (EDIT1)

Ayant abandonné l'exercice 2, Ce troisième travail nous invite cette fois à modifier directement le binaire pour simplifier le jeu. Il n'est alors plus question de modifer la mémoire dans le cours du jeu mais bien de changer le montant dès le départ et ceci pour toute la partie. Après quelques recherches, il m'a semblé plus simple d'utiliser `hexcurse`. Pour cela, il suffit de faire : `hexcurse aoe`

Figure 6: affichage du binaire avec hexcurse

Il reste néanmoins à identifer où modifier les données. Pour cela, on peut décompiler le binaire avec ghidra, une nouvelle fois, et identifier les valeurs à changer :

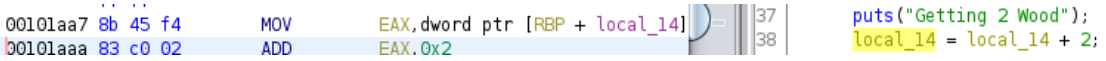

Figure 7: adresse où le nombre de bois semble être stocké

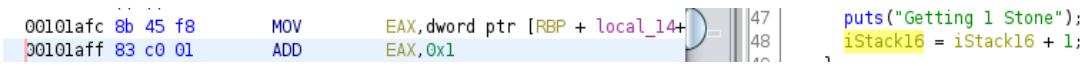

Figure 8: adresse où le nombre de pierres semble être stocké

On voit dans la partie de gauche qu'une commande ADD en assembleur permet de modifier le nombre de bois et de pierres que l'on gagne. Il "suffit alors de modifier" le 02 dans l'hexa pour y mettre la valeur que l'on souhaite et ceci pour la pierre et le bois.

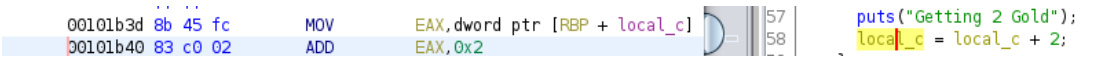

Figure 9: adresse où le nombre d'or semble être stocké

Pour l'or c'est un tout petit peu différent puisque en plus de modifier l'or que l'on gagne, on en gagne plus du tout si le montant dépasse 300. De ce fait, il faut aussi modifier la valeur contenu dans la condition de sorte que l'on puisse incrémenter le nombre d'or malgré tout.

Figure 10: adresse où la limite d'or semble être stockée

On peut alors retourner dans `hexcurse` . Avec la commande `find`, on peut chercher une séquence hexa :

Figure 11: recherche d'une suite hexa spécifique pour modifer le nombre de bois

une fois la séquence trouvée il est possible de modifer la valeur que l'on veut. Ici, on remplace la valeur 0x02 par la valeur 0x64. on ne gagnera plus 2 bois par actions mais 100. Il faut tout de même noter que la première fois, j'ai mis une valeur beaucoup plus grande ce qui a eu pour effet de décrementer le bois plutôt que de l'augmenter. Ceci est dû au fait que le binaire est signé. Si on dépasse la valeur décimale 127, on retombe à -128. De ce fait j'ai pris 0x64 pour être sûr de ne pas avoir de problème.

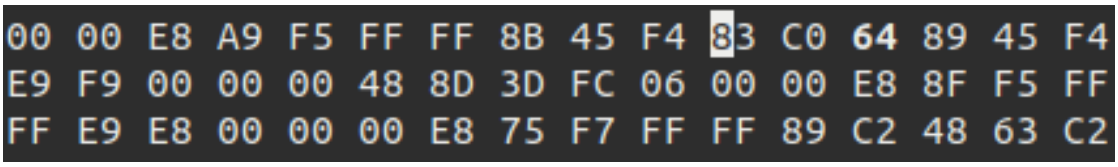

Figure 12: modification du binaire pour le bois

La valeur surlignée est donc la nouvelle valeur prise en compte par le programme. On fait donc de même pour la pierre qui possède la même séquence hexa plus loins dans le binaire, ainsi que pour l'or et la limite de 300 pour pouvoir incrémenter l'or. Une fois cela éxécuté on sauvegarde le binaire puis on le relance comme pour le premier mais avec l'executable modifié :

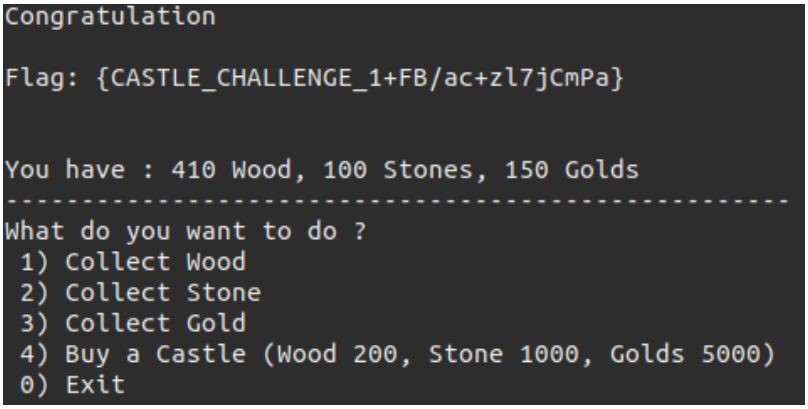

Figure 13: achat du chateau avec le binaire modifié

3 Crackme4 (EDIT2)

Pour ce travail, le fichier est un .exe . On peut utiliser l'outil `ILSpy` pour décompiler le CrackMe4.

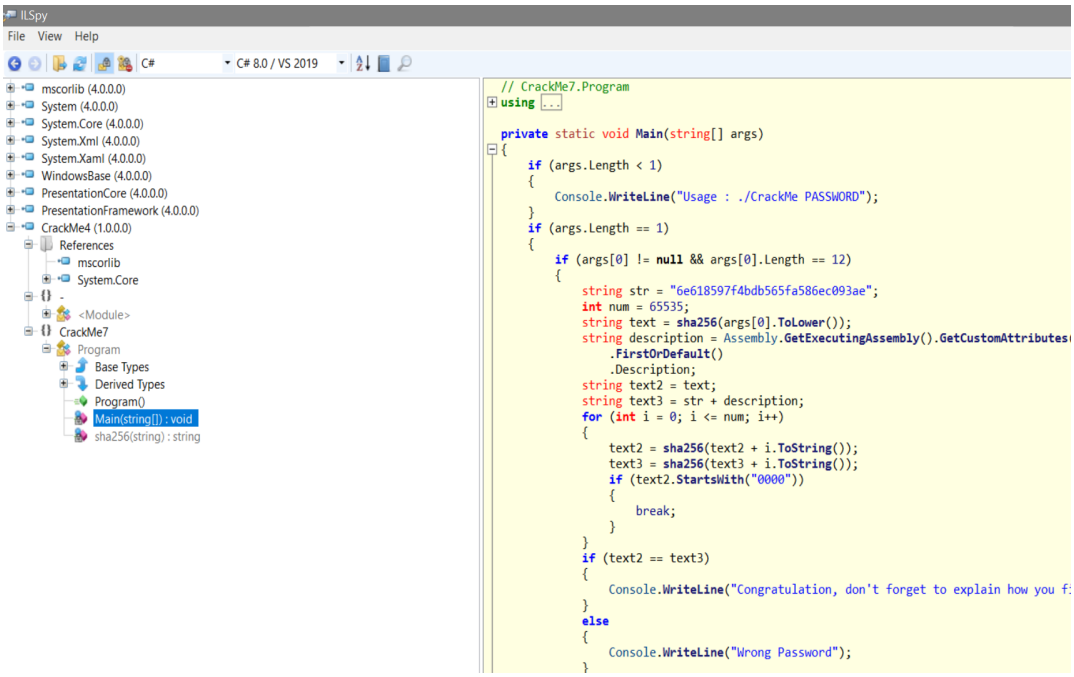

Figure 14: affichage du main décompilé grace à ILSpy

Une fois décompilé, on peut exporter le contenu dans un projet C. Ce dernier doit être ouvert avec Visual Studio en tant que projet pour pouvoir le modifier et trouver le mot de passe.

```
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 str = "6e618597f4bdb565fa586ec093ae";
            int num = 65535;
            string text = sha256(args[0].ToLower());
            string description = Assembly.GetExecutingAssembly().GetCustomAttributes(typeof(AssemblyDescriptionAtt
                .FirstOrDefault()
                .Description;
            Console.WriteLine("description: " + description);
            Console.WriteLine("text: " + text);
            string text2 = text;
            string text3 = str + description; // toujours le meme
            Console.WriteLine("text3: " + text3);
            System.Environment.Exit(0);
            for (int i = 0; i <= num; i++)
            {
                text2 = sha256(text2 + i.ToString());
                text3 = sha256(text3 + i.ToString());
                if (text2.StartsWith("0000"))
                {
                    break;
                }
            }
            if (text2 == text3)
            {
                Console.WriteLine("Congratulation, don't forget to explain how you find the password solution !");
            }
```

Figure 15: affichage du main sur Visual Studio

Pour comprendre comment fonctionne le code, j'ai fait des `Console.WrtieLine()` sur les variables intéressantes. J'ai aussi remarqué que le mot de passe attendu est de 12 caractères. Il est donc possible de lancer le programme depuis l'invite de commande Windows avec un mot de passe de 12 caractères pour voir les données intéressantes :

```
C:\Users\thoma\Desktop\P_CrackMe4\bin\Debug>CrackMe7.exe PASSWORD1234
description: fa0506aa073a2158380895204df81f0fcc29
text: b9c950640e1b3740e98acb93e669c65766f6670dd1609ba91ff41052ba48c6f3
text3: 6e618597f4bdb565fa586ec093aefa0506aa073a2158380895204df81f0fcc29
```

Figure 16: affichage des variables intéressantes

On remarque alors que `text3` est en faite le hash du mot de passe attendu. Ce dernier est comparé à `text` , hash du mot de passe donné comme paramètre. Avant cette comparaison, il y a une modification du hash qui est équivalente pour les 2 variables. Il suffit alors de trouver le message qui se cache derrière `text3` pour réussir l'exercice.
En cherchant un peu sur internet, j'ai remarqué qu'il existe des outils de reverse hashing / decryption qui permettent de trouver le message. J'ai utilisé le site `md5hashing.net` disponible à l'adresse :
`https://md5hashing.net/hash/sha256`

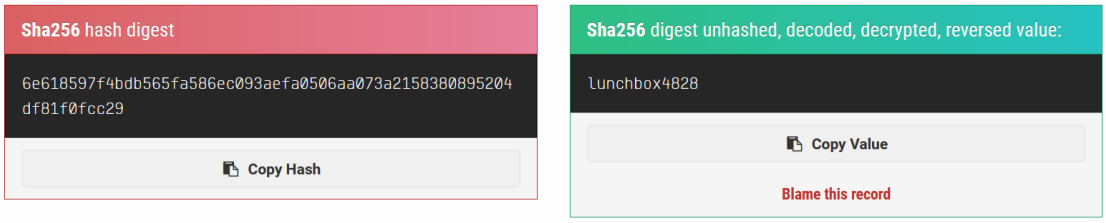

Figure 17: obtention du message depuis le hash

On peut alors tester le mot de passe lors d'une nouvelle éxécution :

```
C:\Users\thoma\Desktop\P_CrackMe4\bin\Debug>CrackMe7.exe lunchbox4828
Congratulation, don't forget to explain how you find the password solution !
```

Figure 18: résolution du problème avec le bon mot de passe