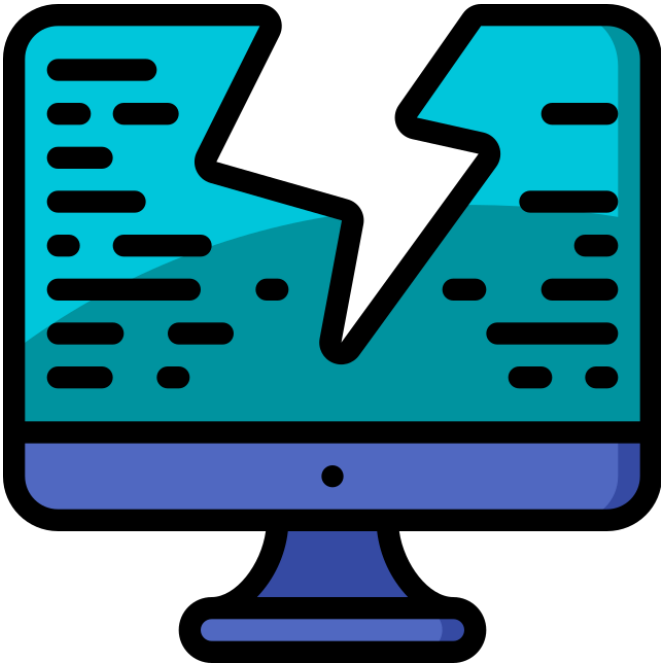
Projet ISO-03

YDays - Séance 2

Delbende Thomas 18/11/2020



Cracking sur Root-Me.org

Comme lors de la dernière séance, avec des membres de mon groupe nous sommes allé sur root-me, mais cette fois pour pratiquer le “cracking”.

Le Cracking, dans notre cas, consiste à rechercher des informations (comme des mots de passe) dans le code d’une application.

Exercice 1 - *0 protection*

Dans cet exercice, un fichier "*.bin*" nous est donné.

Lorsqu'on l'exécute, le programme nous demande alors un mot de passe:

```
###########################################################

##    Bienvennue dans ce challenge de cracking        ##

###########################################################

Veuillez entrer le mot de passe :
```

À l'aide de la commande strings, qui affiche toutes les chaînes de caractères présentes dans le binaire, on peut établir une liste, dont voici une partie:

- `%s : "%s"`
- `Allocating memory`
- `Reallocating memory`
- `123456789`
- `###########################################################`
- `##    Bienvennue dans ce challenge de cracking        ##`
- `###########################################################`
- `Veuillez entrer le mot de passe :`
- `Bien joue, vous pouvez valider l'epreuve avec le pass : %s!`
- `Dommage, essaye encore une fois.`

Nous pouvons que dans cette liste, certaines strings font parties du programme, comme `*Allocating memory*` ou encore `*%s : "%s"*`, ainsi que ses phrases de réponses. Cependant, on peut voir `*123456789*`, qui ne semble pas faire partie du programme, et qui n'a pas non plus de raison d'être affiché. On peut donc penser qu'il s'agit du mot de passe.

Exercice 2 - *Basique*

Même procédé que pour l'exercice 1, le mot de passe est marqué en clair dans le code, on peut le retrouver à l'aide de la commande strings:

```
username:

password:

987654321

Bien joue, vous pouvez valider l'epreuve avec le mot de passe : %s !

Bad password

Bad username

FATAL: kernel too old

FATAL: cannot determine kernel version
```

Le mot de passe est donc *987654321*.

Exercice 3 - *Basic Crackme*

Cet exercice est un peu différent, on nous donne toujours un binaire, mais cette fois le mot de passe n'est pas contenu en clair dedans. Il est vérifié par le programme grâce à des if successifs, comme on peut le voir dans sa version décompilée:

```
if (local_42 == 's') {

      if (local_43 == 'p') {

        if (local_4d == 'm') {

          if ((local_54[2] == 'n') && (local_4e == 'n')) {

            if (local_54[0] == 'c') {

              if (local_54[1] == 'a') {

                if (local_54[3] == 't') {

                  if (local_50 == 'r') {

                    if (local_4f == 'u') {

                      FUN_004007c0();

                      return local_c;
```

On peut donc reconstituer le mot de passe caractère par caractère, (tout les caractères du mot de passe ne sont pas présents ci-dessus), et on obtient le mot *cantrunmiiiiiiiiiips*.

Exercice 4 - *Golang basique*

Cet exercice est le plus dur que nous ayons eu à résoudre. En effet, non seulement le mot de passe est stocké dans le code sous forme de caractères en code ASCII (par exemple *0x3B*), mais en plus il faut combiner ces caractères avec la chaîne de caractères "*rootme*".

Une fois que que avons trouvé la liste des codes ASCII qui composent le mot de passe, on obtient "*0x3B 0x02 0x23 0x1B 0x1B 0x0C 0x1C 0x08 0x28 0x1B 0x21 0x04 0x1C 0x0B*".

Il faut donc ensuite les décrypter à l'aide de la string "*rootme*", et pour cela j'ai utilisé un simple algorithme Python:

```
crypted = [0x3B, 0x02, 0x23, 0x1B, 0x1B, 0x0C, 0x1C, 0x08, 0x28, 0x1B, 0x21,
0x04, 0x1C, 0x0B]

key = "rootme"

res = ""


def itr(s):
   for idx in range(len(s)):
       yield s[idx], idx


for c, i in itr(crypted):
   char = c ^ ord(key[i % len(key)])
   res += chr(char)


print(res)
```

L'algorithme nous affiche la solution: "*ImLovingGoLand*".

C'est ainsi que s'est achevée cette journée de projet YDays.