Challenge : Serial Keyler
Catégorie : Reverse

Énoncé :
On vous demande d'écrire un générateur d'entrées valides pour ce binaire, puis de le valider sur les entrées fournies par le service distant afin d'obtenir le flag.

Fichier(s) : SerialKeyler

Table des matières

1) Première approche

Comme d'habitude, on commence par prendre quelques informations sur le binaire.

```
SoEasY in ~/Bureau [12:49]
> file SerialKeyler
SerialKeyler: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux
-x86-64.so.2, for GNU/Linux 3.2.0, BuildID[sha1]=b307e4dbb397126c0f411da59fd4c4d57304e42e, stripped
```

On a donc ici affaire à un ELF 64bits (x86_64 Intel) qui est cette fois-ci strippé, c'est-à-dire sans symboles de débogage : l'analyse dynamique du code est alors plus compliquée.

On peut ensuite lancer le binaire pour avoir un premier aperçu de son fonctionnement.

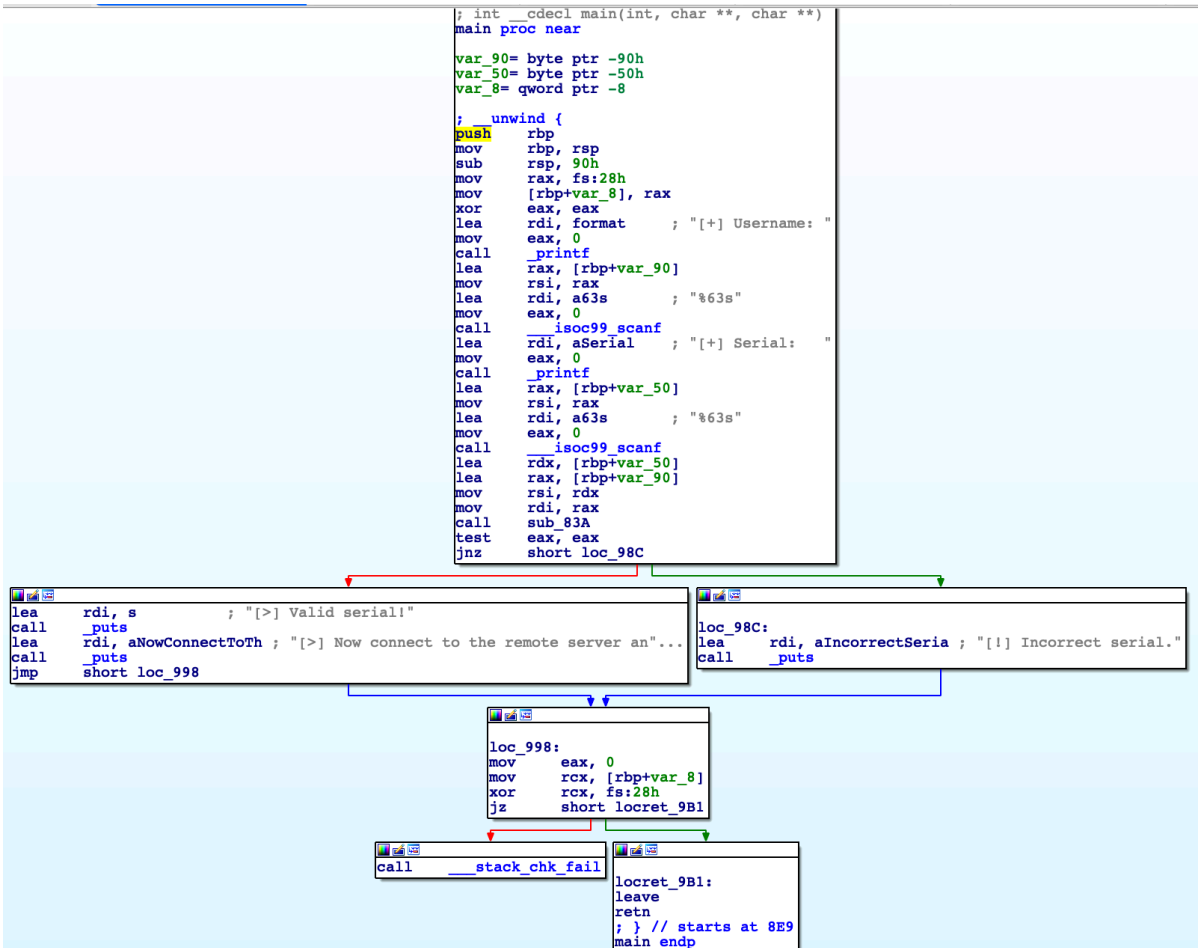
```
SoEasY in ~/Bureau [12:52]
> ./SerialKeyler
[+] Username: SoEasY
[+] Serial:   123456
[!] Incorrect serial.
```

Le programme nous demande donc un nom d'utilisateur puis un serial, une clé correspondant à cet utilisateur : on peut alors imaginer que la clé est calculée à partir du nom de l'utilisateur.

2) Reverse engineering

Je vais ici me concentrer sur une analyse statique du code avec IDA ainsi que le decompiler HexRays fournit avec IDA.

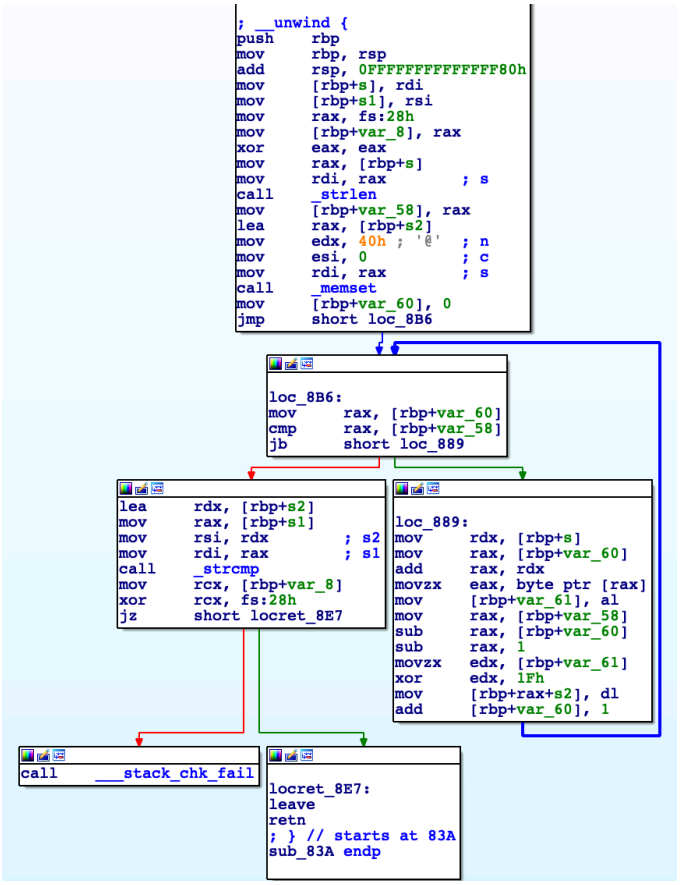
On peut alors commencer par désassembler la fonction main pour avoir un aperçu global du fonctionnement du programme.



On voit bien ici l'affichage avec printf des messages demandant la saisie d'un user et d'un serial comme on voit que leur saisie est effectuée grâce à scanf.

On voit ensuite une fonction nommée « sub_83A » par IDA (car elle se trouve en 0x83A) qui prend en paramètre les deux input (username et serial). Suite à cela, le programme décidera s'il jump en « loc_98c » et affiche le message « [!] Incorrect serial. » ou si au contraire il ne prend pas le jump et affiche le message « [>] Valid serial ! ».

Regardons cette fonction de plus près.



On voit ici que le programme va boucler en vérifiant le serial à partir du username (on voit notamment une opération de XOR et des appels à strlen, strcmp).

Même s'il est possible de comprendre cette fonction depuis l'assembleur, on peut utiliser le decompiler HexRays fournit avec IDA pour obtenir un pseudo code et ainsi comprendre plus vite le fonctionnement de cette fonction.

```
int __fastcall sub_83A(const char *a1, const char *a2)
{
  unsigned __int64 i; // [rsp+20h] [rbp-60h]
  size_t v4; // [rsp+28h] [rbp-58h]
  char s2[72]; // [rsp+30h] [rbp-50h]
  unsigned __int64 v6; // [rsp+78h] [rbp-8h]

  v6 = __readfsqword(0x28u);
  v4 = strlen(a1);
  memset(s2, 0, 0x40uLL);

  for ( i = 0LL; i < v4; ++i )
    s2[v4 - i - 1] = a1[i] ^ 0x1F;

  return strcmp(a2, s2);
}
```

Pour résumer cette fonction, l'username entré va être comparé à l'inverse de la chaine donc chaque caractère est égal au caractère correspondant du username, XORé avec 0x1F.

On a donc **pour i dans strlen(username) : serial[strlen(username) - i - 1] = username[i] ^ 0x1F**

On peut alors essayer en local pour l'user abc.
a ^ 0x1F = ~
b ^ 0x1F = }
c ^ 0x1F = |

Donc pour l'user « abc », le serial serait « |}~ ».

```
SoEasY in ~/Bureau [12:55]
> ./SerialKeyler
[+] Username: abc
[+] Serial:   |}~
[>] Valid serial!
[>] Now connect to the remote server and generate serials for the given usernames.
```

Parfait ! on peut alors essayer de lancer la connexion au service pour voir sous quelle forme nous sont donnés les usernames pour lesquels nous allons devoir générer les serials.

```
SoEasY in ~/Bureau [13:37]
> nc challenges2.france-cybersecurity-challenge.fr 3001
What is a valid serial for username: ecsc
>>> |l|z
What is a valid serial for username: ANSSI
>>>
```

3) Script python

Ayant résolu ce challenge ***à la main*** j'ai compris l'importance de scripter cette résolution.

Pour cela, je vais utiliser la libraire pwntools pour me connecter au service, ainsi que time pour mettre des petites pauses entre chaque saisie pour donner au programme le temps d'afficher le message suivant.
De plus, je vais ajouter un compteur de serials générés pour me rentrer dans la tête à quel point il était ***stupide*** de le résoudre à la main…

On commence donc le script en important les librairies dont nous avons besoin puis en établissant la connexion au service.

```python
import time
from pwn import *

r = remote('challenges2.france-cybersecurity-challenge.fr', 3001)
```

On va ensuite définir une fonction chargée de générer le serial pour un username donné.

```python
def keygen(user):
    key = ""
    for i in user:
        key += chr(0x1F ^ ord(i))
    return key[::-1] # Inversion de la chaine
```

On va ensuite devoir extraire l'username du message envoyé, générer son serial puis envoyer celui-ci (sans oublier d'incrémenter le compteur à chaque fois).
Cette opération sera répétée tant que le flag ne sera pas affiché dans la réponse, c'est-à-dire tant que « FCSC{ » ne sera pas dans la réponse.

```
12  time.sleep(0.1)
13  # Recupere le message envoye par le service
14  reponse = r.recv()
15  compteur = 0
16
17  while "FCSC{" not in reponse:
18      # Recupere uniquement le username en enlevant les 37 premiers char
19      username = reponse.splitlines()[0][37:]
20      # Genere le serial correspondand a l'username
21      key = keygen(username)
22
23      print "[+] Username : ",username
24      print "[+] Serial : ",key
25
26      # Envoie le serial comme reponse
27      r.sendline(key)
28      compteur += 1
29
30      time.sleep(0.1)
31      reponse = r.recv()
32      print reponse
33
34  print "[+] Total :",compteur,"serials generated --> too long by hand"
```

On peut alors tester notre script.

```
SoEasY in ~/Bureau/FCSC_2020 [14:00]
> python SerialKeyler.py
[+] Opening connection to challenges2.france-cybersecurity-challenge.fr on port 3001: Done
[+] Username :  ecsc
[+] Serial :  |l|z
What is a valid serial for username: ANSSI
>>>
[+] Username :  ANSSI
[+] Serial :  VLLQ^
What is a valid serial for username: HelloWorld
>>>
[+] Username :  HelloWorld
[+] Serial :  {smpHpsszW
What is a valid serial for username: TeamFrance
```

[…]

```
What is a valid serial for username: MrDHRzBFM6mRIbxLf
>>>
[+] Username :  MrDHRzBFM6mRIbxLf
[+] Serial :  ySg}VMr)RY]eMW[mR
Well done! Here is the flag: FCSC{8f1018d0cfe395018a1c90dbff352e2ba4a6261336fb7c32454cdae4974d4333}

[+] Total : 55 serials generated --> too long by hand
[*] Closed connection to challenges2.france-cybersecurity-challenge.fr port 3001
```

On récupère ainsi le flag. Ce challenge fort sympathique m'aura ouvert les yeux sur l'importance de scripter certaines résolutions !