Quelques exercices de cryptographie et de rétroconception

Jérémy Jean

Laboratoire de cryptographie de l'ANSSI

13 octobre 2022

Plan

Proposition de quelques exercices/TP avec solution.
Contenu reproductible donné dans un fichier *.zip* après cette journée.

- **Cryptographie**
  - ROT13
  - Vigénère
  - RSA avec clé faible
- **Rétroconception (Reverse Engineering)**
  - Reverse simple sur Linux
  - Reverse simple sur Windows

Cryptographie : ROT13

Exercice : le texte ci-dessous a été *chiffré* avec l'algorithme ROT13. Écrire un script Python pour le déchiffrer.

```
PRG NYTBEVGUZR RFG HA PNF CNEGVPHYVRE QH PUVSSER QR PRFNE, HA NYTBEVGUZR
FVZCYVFGR QR PUVSSERZRAG QR GRKGR. PBZZR FBA ABZ Y'VAQVDHR, VY F'NTVG
Q'HA QRPNYNTR QR GERVMR PNENPGRERF QR PUNDHR YRGGER QH GRKGR N PUVSSERE.
FBA CEVAPVCNY NFCRPG CENGVDHR RFG DHR YR PUVSSERZRAG RG YR QRPUVSSERZRAG
FR SBAG RKNPGRZRAG QR YN ZRZR ZNAVRER.
```

Note : seules les lettres ont été chiffrées, tous les autres caractères (espace, ponctuation, etc.) ont été conservés.

Cryptographie : ROT13

```
1  def ROT13(texte): # Fonction a ecrire
2      ALPHABET = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
3      pass
4
5  texte_chiffre = """
6  PRG NYTBEVGUZR RFG HA PNF CNEGVPHYVRE QH PUVSSER QR
7  PRFNE, HA NYTBEVGUZR FVZCYVFGR QR PUVSSERZRAG QR GRKGR.
8  PBZZR FBA ABZ Y'VAQVDHR, VY F'NTVG Q'HA QRPNYNTR QR
9  GERVMR PNENPGRERF QR PUNDHR YRGGER QH GRKGR N PUVSSERE.
10 FBA CEVAPVCNY NFCRPG CENGVDHR RFG DHR YR PUVSSERZRAG RG
11 YR QRPUVSSERZRAG FR SBAG RKNPGRZRAG QR YN ZRZR ZNAVRER.
12 """
13 texte_clair = ROT13(texte_chiffre)
14 print(texte_clair)
```

Cryptographie : ROT13

Solution :

```
1  def ROT13(texte):
2    ALPHABET = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
3    resultat = ""
4    for lettre in texte:
5      if lettre in ALPHABET:
6        position = ALPHABET.index(lettre)
7        resultat += ALPHABET[(position + 13) % 26]
8      else:
9        resultat += lettre
10   return resultat
```

Cryptographie : ROT13

Résultat en exécutant le script Python :

```
$ python3 solution.py

CET ALGORITHME EST UN CAS PARTICULIER DU CHIFFRE DE
CESAR, UN ALGORITHME SIMPLISTE DE CHIFFREMENT DE TEXTE.
COMME SON NOM L'INDIQUE, IL S'AGIT D'UN DECALAGE DE
TREIZE CARACTERES DE CHAQUE LETTRE DU TEXTE A CHIFFRER.
SON PRINCIPAL ASPECT PRATIQUE EST QUE LE CHIFFREMENT ET
LE DECHIFFREMENT SE FONT EXACTEMENT DE LA MEME MANIERE.
```

Cryptographie : Vigénère

Carré de Vigénère

clair

clé	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Notations

- m_i : i-ème lettre du clair
- c_i : i-ème lettre du chiffré
- k_i : i-ème lettre de la clé
- t : longueur de la clé

Opération de chiffrement

$$c_i \leftarrow (m_i + k_{i \bmod t}) \bmod 26$$

Opération de déchiffrement

$$m_i \leftarrow (c_i - k_{i \bmod t}) \bmod 26$$

Cryptographie : Vigénère

Exercice : Le texte donné dans le fichier `ciphertext.txt` a été *chiffré* avec la méthode de Vigénère avec une clé inconnue de 5 caractères. Compléter le script Python pour le déchiffrer de manière automatique.

```
$ cat -n ciphertext.txt
  1  SHJDM GESFL PNIMM BBLIC INEAV UVLVM VNALY UVLLM RAWOG OECSL
  2  EFLAV AGAGV DRTMM NBKSQ RRKJM GASAB LRNSM TIAWV TUSTQ THWDL
  3  UQWJV IRJEW MRFLT EFHSA SNYWZ SRETI RDMSQ EALWA CBJLM SQMFM
  4  FBMDM DNEAA DRKHW RGWMZ SQWLM LRYJI MZWKT APSKY URLLM SHJDW
  5  RRADT EWWLI IRFLL EFFGU SNLJI VRJKT EFKST OAKGV AZWFI IGVWA
  6  MNDDM SRLVM SSDWC RFVWA EAXSV TFUMZ IRMPK OHJSQ EALVC HNMLM
  7  NOSKL UASNQ RRHWV DNFLY URDGZ CUWKB RRSUK OZHSO NNALQ MCWJB
  8  UETSJ LREWV TPWYZ AAVKX EPLSK LRKMZ LRHGV THFHM UNDWK AELVC
  ...
```

Cryptographie : Vigénère

Script Python à compléter :

```
1  ALPHABET = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
2  def dechiffrement(texte):
3      ## On commence par ecrire une fonction
4      ## qui retrouve la cle avec une analyse frequentielle
5      def retrouve_cle(texte):
6          ## ... A COMPLETER ICI ...
7          pass
8
9      ## Puis on retrouve la cle grace au texte
10     cle = retrouve_cle(texte)
11
12     ## On procede enfin au dechiffrement en tant que tel
13     clair = ""
14     ## ... A COMPLETER ICI ...
15     return clair
16
17 texte_chiffre = "".join(open("ciphertext.txt").read().split())
18 texte_clair = dechiffrement(texte_chiffre)
19 print(texte_clair)
```

Cryptographie : Vigénère

Solution :

```
1  def retrouve_cle(texte):
2    from collections import Counter
3    # On commence par calculer la frequence des lettres par paquets de 5,
4    # etant donne que la cle contient 5 caracteres.
5    eeeee_chiffre = "" # Contiendra le chiffrement de "EEEEE"
6    for position in range(5):
7      frequences = Counter(texte[position::5])
8      frequence_max = 0
9      e_chiffre = "" # Contiendra la lettre la plus frequente (le chiffrement de E)
10     for lettre, valeur in frequences.items():
11       if valeur > frequence_max: # Si cette lettre est plus frequente
12         e_chiffre = lettre
13         frequence_max = valeur
14     eeeee_chiffre += e_chiffre # On ajoute une lettre
15   # On recupere desormais la cle en supposant que le clair est "EEEEE"
16   cle = ""
17   for a, b in zip("EEEEE", eeeee_chiffre):
18     x, y = ALPHABET.index(a), ALPHABET.index(b)
19     cle += ALPHABET[(y - x) % 26]
20   return cle
```

Cryptographie : Vigénère

Solution :

```
1  def dechiffrement(texte):
2    # (...)
3
4    ## Puis on retrouve la cle grace au texte
5    cle = retrouve_cle(texte)
6
7    ## On procede enfin au dechiffrement en tant que tel
8    clair = ""
9    for position, lettre in enumerate(texte):
10     x, y = ALPHABET.index(lettre), ALPHABET.index(cle[position % 5])
11     clair += ALPHABET[(x - y) % 26]
12   return clair, cle
13
14 texte_chiffre = "".join(open("ciphertext.txt").read().split())
15 texte_clair, cle = dechiffrement(texte_chiffre)
16 print(texte_clair)
17 print(cle)
```

Cryptographie : Vigénère

Résultat en exécutant le script Python :

```
$ python3 solution.py
SURLEGRANDPAQUEBOTQUIAMINUITDEVAITQUITTERNEWYORKADESTINATIONDEBUENOSAI
RESREGNAITLEVAETVIENTHABITUELDUDERNIERMOMENTLESPASSAGERSEMBARQUAIENTES
CORTESDUNEFOULEDAMISDESPORTEURSDETELEGRAMMESLACASQUETTESURLOREILLEJETA
IENTDESNOMSATRAVERSLESSALONSONAMENAITDESMALLESETDESFLEURSDESENFANTSCUR
IEUXCOURAIENTDUHAUTENBASDUNAVIREPENDANTQUELORCHESTREACCOMPAGNAITIMPERT
URBABLEMENTCEGRANDSPECTACLESURLEPONTUNPEUALECARTDUMOUVEMENTJEMENTRETEN
(...)
```

Il s'agit de la nouvelle *Le Joueur d'échecs* **de Stefan Zweig, et la clé était** ANSSI**.**

Cryptographie : Vigénère

Idées d'exercices intermédiaires :

- Demander l'implémentation du déchiffrement de Vigénère quand la clé est fournie.
- Chiffrement de César (équivalent à Vigénère avec une clé de longueur 1).

Pour aller plus loin avec Vigénère :

- Attaque sans donner la longueur de la clé.
- Attaque en donnant un texte plus court.
- Attaque en appliquant le chiffrement sur un alphabet contenant également les chiffres.

Cryptographie : RSA

En bref : RSA est un mécanisme de cryptographie *asymétrique* qui permet notamment de faire du chiffrement.

Notations :

- p, q : deux (grands) nombres premiers secrets.
- e : exposant public (usuellement, $e = 2^{16} + 1$).
- $n = p \cdot q$: module public.
- d : exposant privé tel quel $e \cdot d = 1 \pmod{\varphi(n)}$, où $\varphi(n) = (p-1)(q-1)$.
- m : message en clair.
- c : message chiffré.

Fonctionnalités simples :

- **Chiffrement :** $m \to c = m^e \mod n$.
- **Déchiffrement :** $c \to m = c^d \mod n$.

Cryptographie : RSA

Exercice : analyse et attaque d'une génération de clé faible RSA.

```python
from Crypto.Util.number import bytes_to_long, getPrime, isPrime

def generation():
    e = 2 ** 16 + 1             # exposant public
    while True:                 # generation des nombres premiers
        p = getPrime(512)
        q = p + 2
        if isPrime(q):
            return e, p * q     # cle publique

# Le but est de retrouver le contenu du fichier "secret.txt"
e, n = generation()
secret = bytes_to_long(open("secret.txt", "rb").read())
c = pow(secret, e, n)           # chiffrement
print(f"{e = }")                # on donne les valeurs publiques
print(f"{n = }")
print(f"{c = }")
```

Cryptographie : RSA

Avec ces valeurs de sortie, pouvez-vous déchiffrer le message ?

```
e = 65537
n = 669078917619840085379411407749120372978375900140151855669147836499 \
    415994981561969851980179947858980688184084522781143972577059106836 \
    636301482816322493258384997147902251839563631830239097596614079820 \
    536289733437637919011629414878439520270911587080625884275081284425 \
    173405243042107722191197310170906555447745 43
c = 513391109905722466716958360268367453255680247810900121900496218067 \
    400374433865465421503733347199164156706792134361779517138181820705 \
    052214666113591770066879392120343629799472631589574883088683373379 \
    554193964229357029633394554654042569600901904857874192770839212859 \
    464164472020448982329474535460165857981 4783
```

Cryptographie : RSA

Exercice : compléter ce code Python qui retrouve l'exposant privé d.

```
1  from gmpy2 import *          # Calcul sur les grands nombres
2  from Crypto.Util.number import long_to_bytes
3
4  e = 65537
5  n = 669078917619840085379411(...)9731017090655544774543
6  c = 513391109905722466716958(...)745354601658579814783
7
8  def attaque(n, e):
9      # ... A COMPLETER ...
10
11 d = attaque(n, e)
12 secret = long_to_bytes(pow(c, d, n)).decode() # Dechiffrement
13 print(secret)
```

Démo

Ressource possible : Sagemath
`https://www.sagemath.org/fr/`

Cryptographie : RSA

```python
def attaque(n, e):                # Solution de l'exercice
    p = int(isqrt(n))
    assert gcd(n, p) == p

    q = n // p
    assert p * q == n

    phi = (p - 1) * (q - 1)
    d = pow(e, -1, phi)
    return d
```

```
$ python3 solution.py
flag{Bravo!_Vous_avez_réussi_à_factoriser_un_module_RSA_faible!}
```

Fin de la partie cryptographie

Questions ?

Rétroconception

Reverse Engineering en anglais, ou juste "Reverse"

Reverse : les bases

La rétroconception logicielle est l'analyse d'un programme au format binaire.

Exemples :

- Analyse d'un malware.
- Recherche et correction de vulnérabilités.
- Amélioration des antivirus.

Quelques outils :

- Ghidra (gratuit)
- IDA (versions gratuites et payantes)
- x64dbg (Windows)
- objdump (Linux)

Techniques :

- **Désassemblage :** écriture du programme sous forme d'une suite d'instructions exécutées par le processeur de l'ordinateur.
- **Décompilation :** écriture du programme dans un langage plus haut niveau (pseudo C).

Reverse : quelques références

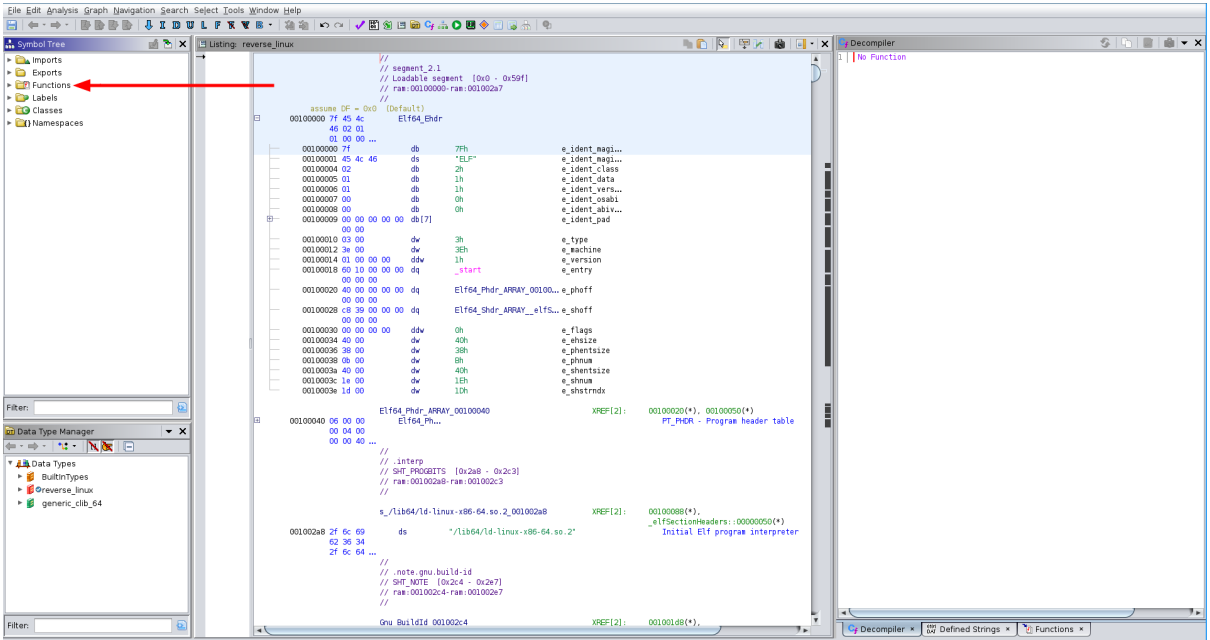
- **IDA :** `https://hex-rays.com/ida-free/`
- **Ghidra :** `https://ghidra-sre.org/`

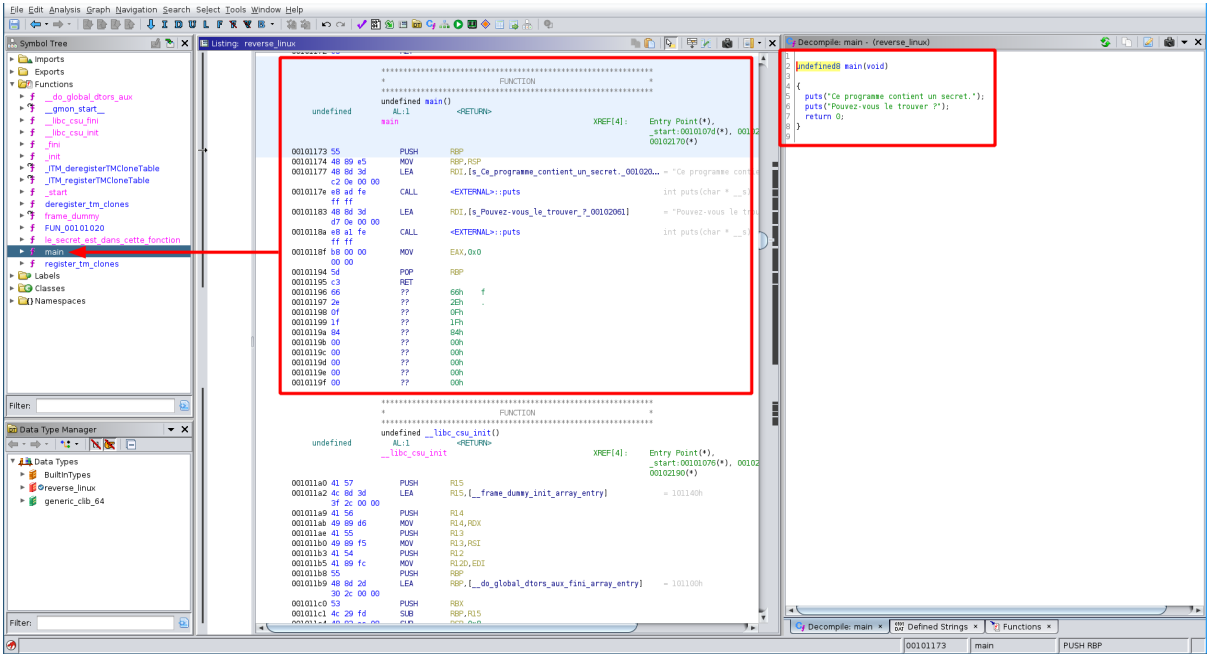
Reverse : exercice pour Linux

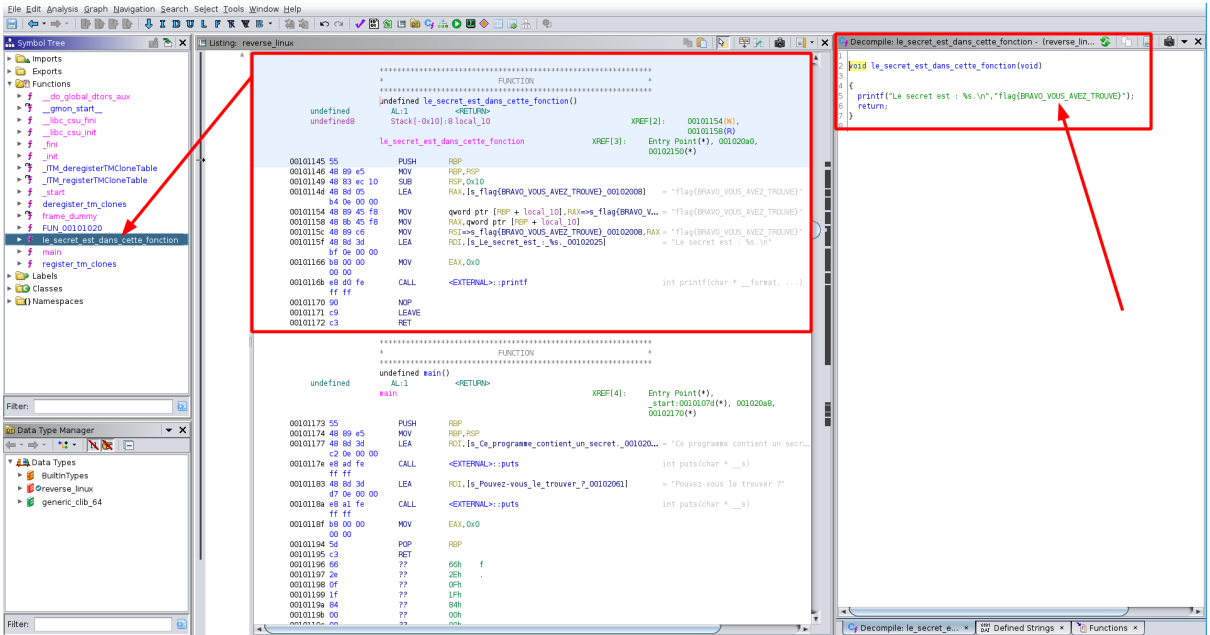
On vous donne le fichier binaire `reverse_linux` qui s'exécute en ligne de commande dans un terminal Linux :

```
$ ls
reverse_linux
$ ./reverse_linux
Ce programme contient un secret.
Pouvez-vous le trouver ?
$
```

Exercice : Charger ce programme dans Ghidra (ou IDA) pour comprendre ce qu'il fait et retrouver le secret.


File Edit Analysis Graph Navigation Search Select Tools Window Help
Symbol Tree
Imports
Exports
Functions
Labels
Classes
Namespaces
Filter:
Data Type Manager
Data Types
BuiltInTypes
Greverse_linux
generic_clib_64
Filter:
Listing: reverse_linux
//
// segment_2.1
// Loadable segment [0x0 - 0x59f]
// ram:00100000-ram:001002a7
//
assume DF = 0x0 (Default)
00100000 7f 45 4c Elf64_Ehdr
46 02 01
01 00 00 ...
00100000 7f db 7Fh e_ident_magi...
00100001 45 4c 46 ds "ELF" e_ident_magi...
00100004 02 db 2h e_ident_class
00100005 01 db 1h e_ident_data
00100006 01 db 1h e_ident_vers...
00100007 00 db 0h e_ident_osabi
00100008 00 db 0h e_ident_abiv...
00100009 00 00 00 00 00 db[7] e_ident_pad
00 00
00100010 03 00 dw 3h e_type
00100012 3e 00 dw 3Eh e_machine
00100014 01 00 00 00 ddw 1h e_version
00100018 60 10 00 00 00 dq _start e_entry
00 00 00
00100020 40 00 00 00 00 dq Elf64_Phdr_ARRAY_00100... e_phoff
00 00 00
00100028 c8 39 00 00 00 dq Elf64_Shdr_ARRAY__elfS... e_shoff
00 00 00
00100030 00 00 00 00 ddw 0h e_flags
00100034 40 00 dw 40h e_ehsize
00100036 38 00 dw 38h e_phentsize
00100038 0b 00 dw 8h e_phnum
0010003a 40 00 dw 40h e_shentsize
0010003c 1e 00 dw 1Eh e_shnum
0010003e 1d 00 dw 1Dh e_shstrndx
Elf64_Phdr_ARRAY_00100040 XREF[2]: 00100020(*), 00100050(*)
00100040 06 00 00 Elf64_Ph... PT_PHDR - Program header table
00 04 00
00 00 40 ...
//
// .interp
// SHT_PROGBITS [0x2a8 - 0x2c3]
// ram:001002a8-ram:001002c3
//
s_/lib64/ld-linux-x86-64.so.2_001002a8 XREF[2]: 00100088(*),
_elfSectionHeaders::00000050(*)
001002a8 2f 6c 69 ds "/lib64/ld-linux-x86-64.so.2" Initial Elf program interpreter
62 36 34
2f 6c 64 ...
//
// .note.gnu.build-id
// SHT_NOTE [0x2c4 - 0x2e7]
// ram:001002c4-ram:001002e7
//
Gnu BuildId 001002c4 XREF[2]: 001001d8(*),
Decompiler
1 | No Function
Decompiler
Defined Strings
Functions
00100000

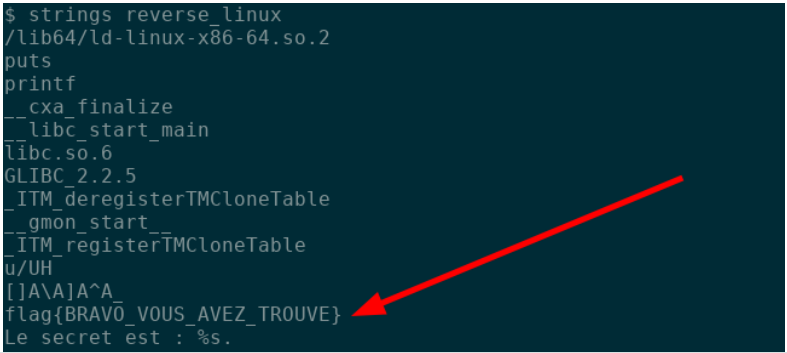

File Edit Analysis Graph Navigation Search Select Tools Window Help
Symbol Tree
Imports
Exports
Functions
__do_global_dtors_aux
__gmon_start__
__libc_csu_fini
__libc_csu_init
_fini
_init
_ITM_deregisterTMCloneTable
_ITM_registerTMCloneTable
_start
deregister_tm_clones
frame_dummy
FUN_00101020
le_secret_est_dans_cette_fonction
main
register_tm_clones
Labels
Classes
Namespaces
Filter:
Data Type Manager
Data Types
BuiltInTypes
reverse_linux
generic_clib_64
Filter:
Listing: reverse_linux
FUNCTION
undefined main()
undefined AL:1 <RETURN>
main XREF[4]: Entry Point(*), _start:00101076(*), 00102190(*)
00101173 55 PUSH RBP
00101174 48 89 e5 MOV RBP,RSP
00101177 48 8d 3d LEA RDI,[s_Ce_programme_contient_un_secret._001020...
c2 0e 00 00
0010117e e8 ad fe CALL <EXTERNAL>::puts int puts(char * __s)
ff ff
00101183 48 8d 3d LEA RDI,[s_Pouvez-vous_le_trouver_?_00102061]
d7 0e 00 00
00101189 e8 a1 fe CALL <EXTERNAL>::puts int puts(char * __s)
ff ff
0010118f b8 00 00 MOV EAX,0x0
00 00
00101194 5d POP RBP
00101195 c3 RET
00101196 66 ?? 66h f
00101197 2e ?? 2Eh .
00101198 0f ?? 0Fh
00101199 1f ?? 1Fh
0010119a 84 ?? 84h
0010119b 00 ?? 00h
0010119c 00 ?? 00h
0010119d 00 ?? 00h
0010119e 00 ?? 00h
0010119f 00 ?? 00h
FUNCTION
undefined __libc_csu_init()
undefined AL:1 <RETURN>
__libc_csu_init XREF[4]: Entry Point(*), _start:00101076(*), 00102190(*)
001011a0 41 57 PUSH R15
001011a2 4c 8d 3d LEA R15,[__frame_dummy_init_array_entry] = 101140h
3f 2c 00 00
001011a9 41 56 PUSH R14
001011ab 49 89 d6 MOV R14,RDX
001011ae 41 55 PUSH R13
001011b0 49 89 f5 MOV R13,RSI
001011b3 41 54 PUSH R12
001011b5 41 89 fc MOV R12D,EDI
001011b8 55 PUSH RBP
001011b9 48 8d 2d LEA RBP,[__do_global_dtors_aux_fini_array_entry] = 101100h
30 2c 00 00
001011c0 53 PUSH RBX
001011c1 4c 29 fd SUB RBP,R15
Decompile: main - (reverse_linux)
undefined8 main(void)
{
puts("Ce programme contient un secret.");
puts("Pouvez-vous le trouver ?");
return 0;
}
Decompile: main
Defined Strings
Functions
00101173
main
PUSH RBP


File Edit Analysis Graph Navigation Search Select Tools Window Help
Symbol Tree
Imports
Exports
Functions
__do_global_dtors_aux
__gmon_start__
__libc_csu_fini
__libc_csu_init
_fini
_init
_ITM_deregisterTMCloneTable
_ITM_registerTMCloneTable
_start
deregister_tm_clones
frame_dummy
FUN_00101020
le_secret_est_dans_cette_fonction
main
register_tm_clones
Labels
Classes
Namespaces
Filter:
Data Type Manager
Data Types
BuiltInTypes
reverse_linux
generic_clib_64
Filter:
Listing: reverse_linux
FUNCTION
undefined le_secret_est_dans_cette_fonction()
undefined AL:1 <RETURN>
undefined8 Stack[-0x10]:8 local_10 XREF[2]: 00101154(W), 00101158(R)
le_secret_est_dans_cette_fonction XREF[3]: Entry Point(*), 001020a0, 00102150(*)
00101145 55 PUSH RBP
00101146 48 89 e5 MOV RBP,RSP
00101149 48 83 ec 10 SUB RSP,0x10
0010114d 48 8d 05 b4 0e 00 00 LEA RAX,[s_flag{BRAVO_VOUS_AVEZ_TROUVE}_00102008] = "flag{BRAVO_VOUS_AVEZ_TROUVE}"
00101154 48 89 45 f8 MOV qword ptr [RBP + local_10],RAX=>s_flag{BRAVO_V... = "flag{BRAVO_VOUS_AVEZ_TROUVE}"
00101158 48 8b 45 f8 MOV RAX,qword ptr [RBP + local_10]
0010115c 48 89 c6 MOV RSI=>s_flag{BRAVO_VOUS_AVEZ_TROUVE}_00102008,RAX = "flag{BRAVO_VOUS_AVEZ_TROUVE}"
00101157 48 8d 3d bf 0e 00 00 LEA RDI,[s_Le_secret_est_:_%s._00102025] = "Le secret est : %s.\n"
00101168 b8 00 00 00 00 MOV EAX,0x0
0010116d e8 d0 fe ff ff CALL <EXTERNAL>::printf int printf(char * __format, ...)
00101170 90 NOP
00101171 c9 LEAVE
00101172 c3 RET
FUNCTION
undefined main()
undefined AL:1 <RETURN>
main XREF[4]: Entry Point(*), _start:00101074(*), 001020a8, 00102170(*)
00101173 55 PUSH RBP
00101174 48 89 e5 MOV RBP,RSP
00101177 48 8d 3d c2 0e 00 00 LEA RDI,[s_Ce_programme_contient_un_secret._001020... = "Ce programme contient un secr...
0010117e e8 ad fe ff ff CALL <EXTERNAL>::puts int puts(char * __s)
00101183 48 8d 3d d7 0e 00 00 LEA RDI,[s_Pouvez-vous_le_trouver_?_00102061] = "Pouvez-vous le trouver ?"
0010118a e8 a1 fe ff ff CALL <EXTERNAL>::puts int puts(char * __s)
0010118f b8 00 00 00 00 MOV EAX,0x0
00101194 5d POP RBP
00101195 c3 RET
00101196 66 ?? 66h f
00101197 2e ?? 2Eh .
00101198 0f ?? 0Fh
00101199 1f ?? 1Fh
0010119a 84 ?? 84h
0010119b 00 ?? 00h
Decompile: le_secret_est_dans_cette_fonction - (reverse_lin...
1 void le_secret_est_dans_cette_fonction(void)
2
3 {
4 printf("Le secret est : %s.\n","flag{BRAVO_VOUS_AVEZ_TROUVE}");
5 return;
6 }
Decompile: le_secret_e...
Defined Strings
Functions
00101145
le_secret_est_dan...
PUSH RBP

Reverse : exercice pour Linux

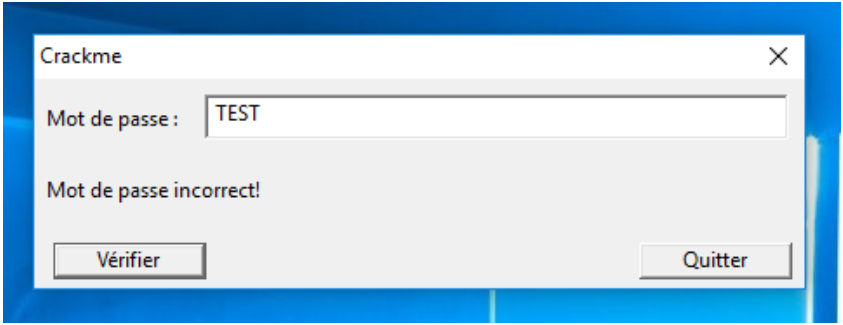
Le secret est donc `flag{BRAVO_VOUS_AVEZ_TROUVE}` et se trouve dans la fonction `le_secret_est_dans_cette_fonction()` qui n'est appelée par aucune autre.

Une autre méthode dans ce cas simple consiste à utiliser la commande `strings` sur Linux qui permet de lister les chaînes de caractères qui apparaissent dans un fichier binaire :

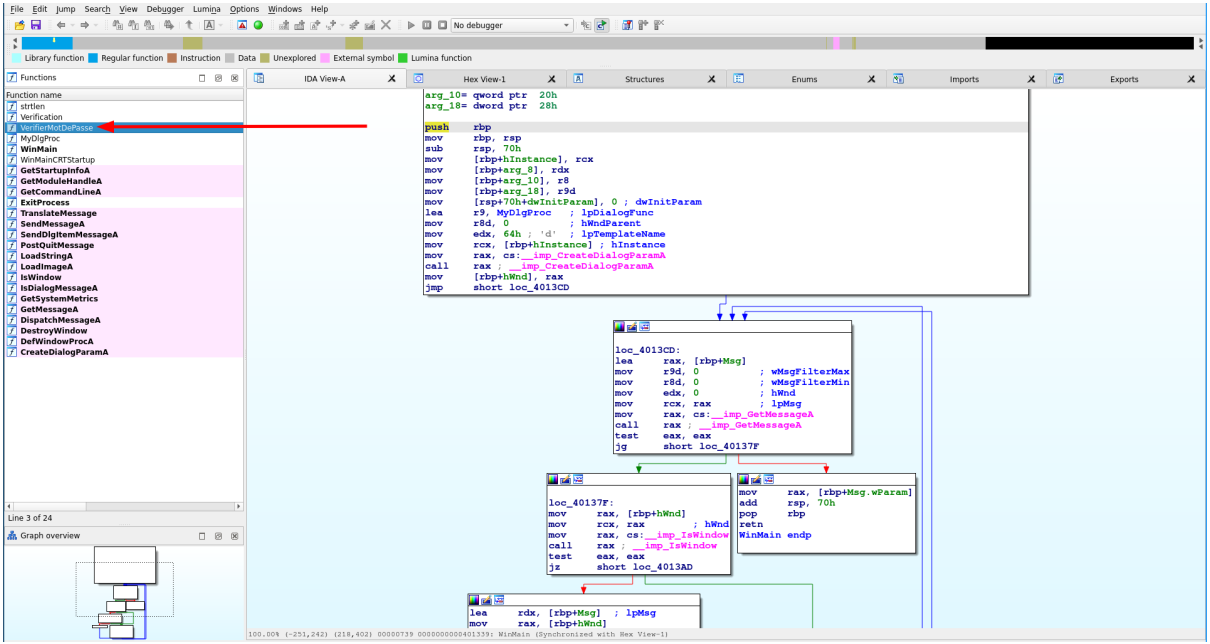


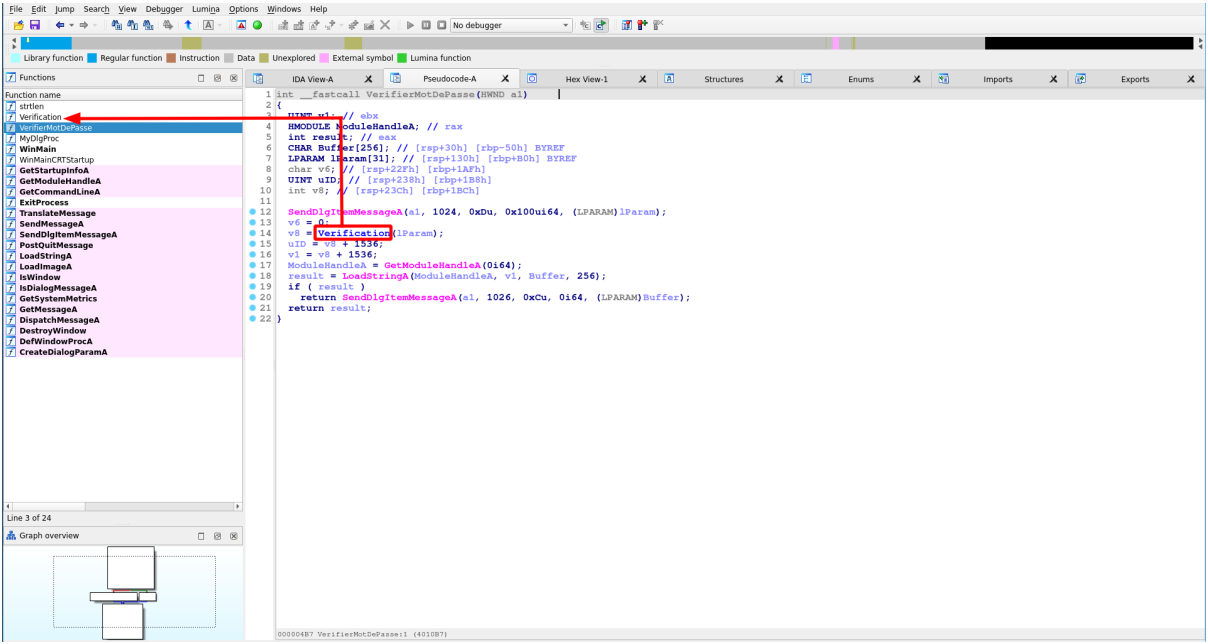
Reverse : exercice pour Windows

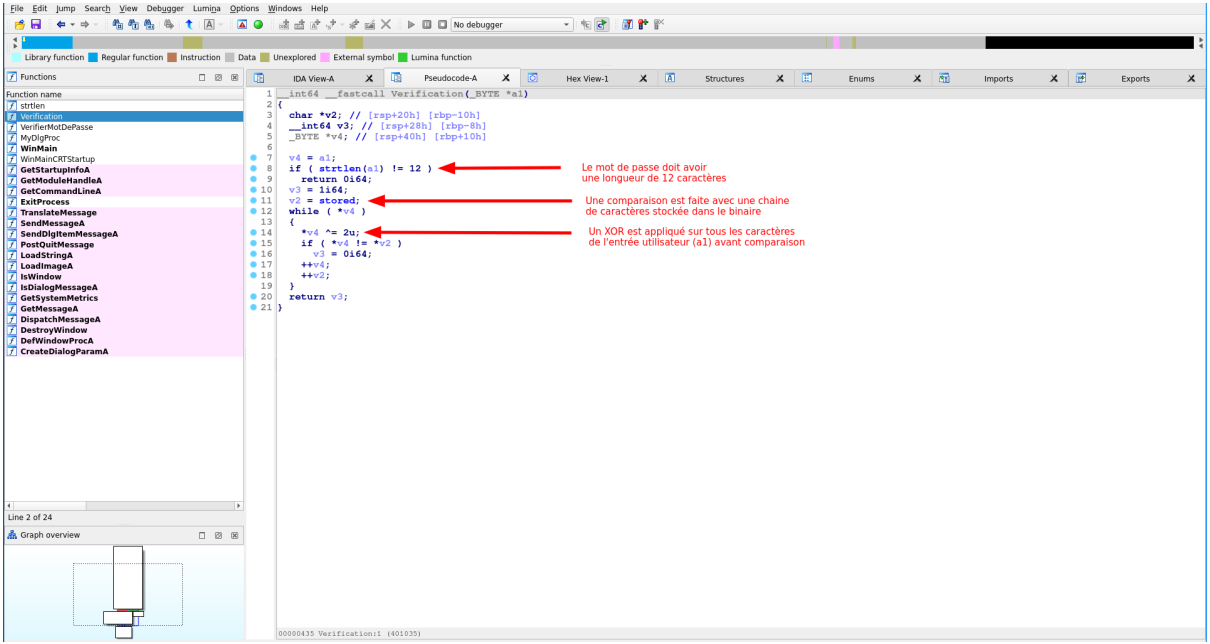
On vous donne le fichier binaire `crackme.exe` qui lance une interface graphique et demande un mot de passe :

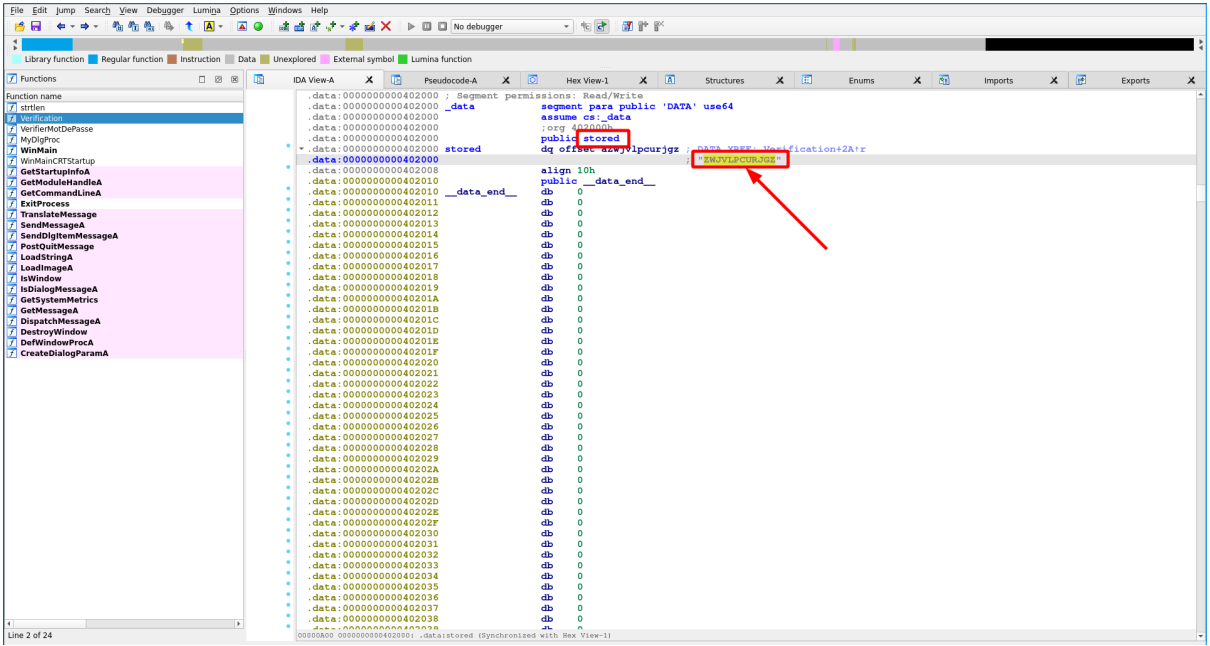


Exercice : Charger ce programme dans IDA (ou Ghidra) pour comprendre ce qu'il fait et trouver le bon mot de passe.


File Edit Jump Search View Debugger Lumina Options Windows Help
No debugger
Library function Regular function Instruction Data Unexplored External symbol Lumina function
Functions
Function name
strtlen
Verification
VerifierMotDePasse
MyDlgProc
WinMain
WinMainCRTStartup
GetStartupInfoA
GetModuleHandleA
GetCommandLineA
ExitProcess
TranslateMessage
SendMessageA
SendDlgItemMessageA
PostQuitMessage
LoadStringA
LoadImageA
IsWindow
IsDialogMessageA
GetSystemMetrics
GetMessageA
DispatchMessageA
DestroyWindow
DefWindowProcA
CreateDialogParamA
Line 3 of 24
Graph overview
IDA View-A
Hex View-1
Structures
Enums
Imports
Exports
arg_10= qword ptr 20h
arg_18= dword ptr 28h
push rbp
mov rbp, rsp
sub rsp, 70h
mov [rbp+hInstance], rcx
mov [rbp+arg_8], rdx
mov [rbp+arg_10], r8
mov [rbp+arg_18], r9d
mov [rsp+70h+dwInitParam], 0 ; dwInitParam
lea r9, MyDlgProc ; lpDialogFunc
mov r8d, 0 ; hWndParent
mov edx, 64h ; 'd' ; lpTemplateName
mov rcx, [rbp+hInstance] ; hInstance
mov rax, cs:__imp_CreateDialogParamA
call rax ; __imp_CreateDialogParamA
mov [rbp+hWnd], rax
jmp short loc_4013CD
loc_4013CD:
lea rax, [rbp+Msg]
mov r9d, 0 ; wMsgFilterMax
mov r8d, 0 ; wMsgFilterMin
mov edx, 0 ; hWnd
mov rcx, rax ; lpMsg
mov rax, cs:__imp_GetMessageA
call rax ; __imp_GetMessageA
test eax, eax
jg short loc_40137F
loc_40137F:
mov rax, [rbp+hWnd]
mov rcx, rax ; hWnd
mov rax, cs:__imp_IsWindow
call rax ; __imp_IsWindow
test eax, eax
jz short loc_4013AD
mov rax, [rbp+Msg.wParam]
add rsp, 70h
pop rbp
retn
WinMain endp
lea rdx, [rbp+Msg] ; lpMsg
mov rax, [rbp+hWnd]
100.00% (-251,242) (218,402) 00000739 0000000000401339: WinMain (Synchronized with Hex View-1)


File Edit Jump Search View Debugger Lumina Options Windows Help
No debugger
Library function Regular function Instruction Data Unexplored External symbol Lumina function
Functions
Function name
strtlen
Verification
VerifierMotDePasse
MyDlgProc
WinMain
WinMainCRTStartup
GetStartupInfoA
GetModuleHandleA
GetCommandLineA
ExitProcess
TranslateMessage
SendMessageA
SendDlgItemMessageA
PostQuitMessage
LoadStringA
LoadImageA
IsWindow
IsDialogMessageA
GetSystemMetrics
GetMessageA
DispatchMessageA
DestroyWindow
DefWindowProcA
CreateDialogParamA
Line 3 of 24
Graph overview
IDA View-A
Pseudocode-A
Hex View-1
Structures
Enums
Imports
Exports
1 int __fastcall VerifierMotDePasse(HWND a1)
2 {
3 UINT v1; // ebx
4 HMODULE ModuleHandleA; // rax
5 int result; // eax
6 CHAR Buffer[256]; // [rsp+30h] [rbp-50h] BYREF
7 LPARAM lParam[31]; // [rsp+130h] [rbp+B0h] BYREF
8 char v6; // [rsp+22Fh] [rbp+1AFh]
9 UINT uID; // [rsp+238h] [rbp+1B8h]
10 int v8; // [rsp+23Ch] [rbp+1BCh]
11
12 SendDlgItemMessageA(a1, 1024, 0xDu, 0x100ui64, (LPARAM)lParam);
13 v6 = 0;
14 v8 = Verification(lParam);
15 uID = v8 + 1536;
16 v1 = v8 + 1536;
17 ModuleHandleA = GetModuleHandleA(0i64);
18 result = LoadStringA(ModuleHandleA, v1, Buffer, 256);
19 if (result)
20 return SendDlgItemMessageA(a1, 1026, 0xCu, 0i64, (LPARAM)Buffer);
21 return result;
22 }
000004B7 VerifierMotDePasse:1 (4010B7)


File Edit Jump Search View Debugger Lumina Options Windows Help
No debugger
Library function
Regular function
Instruction
Data
Unexplored
External symbol
Lumina function
Functions
Function name
strtlen
Verification
VerifierMotDePasse
MyDlgProc
WinMain
WinMainCRTStartup
GetStartupInfoA
GetModuleHandleA
GetCommandLineA
ExitProcess
TranslateMessage
SendMessageA
SendDlgItemMessageA
PostQuitMessage
LoadStringA
LoadImageA
IsWindow
IsDialogMessageA
GetSystemMetrics
GetMessageA
DispatchMessageA
DestroyWindow
DefWindowProcA
CreateDialogParamA
Line 2 of 24
Graph overview
IDA View-A
Pseudocode-A
Hex View-1
Structures
Enums
Imports
Exports
1 __int64 __fastcall Verification(_BYTE *a1)
2 {
3 char *v2; // [rsp+20h] [rbp-10h]
4 __int64 v3; // [rsp+28h] [rbp-8h]
5 _BYTE *v4; // [rsp+40h] [rbp+10h]
6
7 v4 = a1;
8 if (strtlen(a1) != 12)
9 return 0i64;
10 v3 = 1i64;
11 v2 = stored;
12 while (*v4)
13 {
14 *v4 ^= 2u;
15 if (*v4 != *v2)
16 v3 = 0i64;
17 ++v4;
18 ++v2;
19 }
20 return v3;
21 }
Le mot de passe doit avoir une longueur de 12 caractères
Une comparaison est faite avec une chaine de caractères stockée dans le binaire
Un XOR est appliqué sur tous les caractères de l'entrée utilisateur (a1) avant comparaison
00000435 Verification:11 (401035)


File Edit Jump Search View Debugger Lumina Options Windows Help
No debugger
Library function Regular function Instruction Data Unexplored External symbol Lumina function
Functions
Function name
strtlen
Verification
VerifierMotDePasse
MyDlgProc
WinMain
WinMainCRTStartup
GetStartupInfoA
GetModuleHandleA
GetCommandLineA
ExitProcess
TranslateMessage
SendMessageA
SendDlgItemMessageA
PostQuitMessage
LoadStringA
LoadImageA
IsWindow
IsDialogMessageA
GetSystemMetrics
GetMessageA
DispatchMessageA
DestroyWindow
DefWindowProcA
CreateDialogParamA
Line 2 of 24
IDA View-A
Pseudocode-A
Hex View-1
Structures
Enums
Imports
Exports
.data:0000000000402000 ; Segment permissions: Read/Write
.data:0000000000402000 _data segment para public 'DATA' use64
.data:0000000000402000 assume cs:_data
.data:0000000000402000 ;org 402000h
.data:0000000000402000 public stored
.data:0000000000402000 stored dq offset aZwjvlpcurjgz ; DATA XREF: Verification+2A↑r
.data:0000000000402000 "ZWJVLPCURJGZ"
.data:0000000000402008 align 10h
.data:0000000000402010 public __data_end__
.data:0000000000402010 __data_end__ db 0
.data:0000000000402011 db 0
.data:0000000000402012 db 0
.data:0000000000402013 db 0
.data:0000000000402014 db 0
.data:0000000000402015 db 0
.data:0000000000402016 db 0
.data:0000000000402017 db 0
.data:0000000000402018 db 0
.data:0000000000402019 db 0
.data:000000000040201A db 0
.data:000000000040201B db 0
.data:000000000040201C db 0
.data:000000000040201D db 0
.data:000000000040201E db 0
.data:000000000040201F db 0
.data:0000000000402020 db 0
.data:0000000000402021 db 0
.data:0000000000402022 db 0
.data:0000000000402023 db 0
.data:0000000000402024 db 0
.data:0000000000402025 db 0
.data:0000000000402026 db 0
.data:0000000000402027 db 0
.data:0000000000402028 db 0
.data:0000000000402029 db 0
.data:000000000040202A db 0
.data:000000000040202B db 0
.data:000000000040202C db 0
.data:000000000040202D db 0
.data:000000000040202E db 0
.data:000000000040202F db 0
.data:0000000000402030 db 0
.data:0000000000402031 db 0
.data:0000000000402032 db 0
.data:0000000000402033 db 0
.data:0000000000402034 db 0
.data:0000000000402035 db 0
.data:0000000000402036 db 0
.data:0000000000402037 db 0
.data:0000000000402038 db 0
00000A00 0000000000402000: .data:stored (Synchronized with Hex View-1)

Reverse : exercice pour Windows

Exercice : Après cette phase de reverse, réécrire l'algorithme de vérification en Python.

```
1 def Verification(chaine_utilisateur):
2     # ... A COMPLETER ...
3     pass
```

Reverse : exercice pour Windows

Solution :

```
1  def Verification(chaine_utilisateur):
2      # Verification de la longueur
3      if len(chaine_utilisateur) != 12:
4          return 0
5      # Chaine de caracteres stockee en dur
6      stored = b"ZWJVLPCURJGZ"
7      v3 = 1
8      for i in range(12):
9          c = chaine_utilisateur[i] ^ 2
10         if c != stored[i]:
11             v3 = 0
12     return v3
```

Reverse : exercice pour Windows

Exercice : Retrouver le mot de passe qui affiche le message de succès. Compléter pour cela la fonction Python suivante.

```
1 def Motdepasse():
2     stored = b"ZWJVLPCURJGZ"
3     pass
```

Reverse : exercice pour Windows

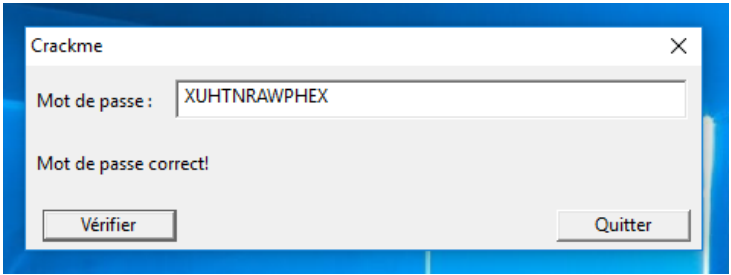
Solution :

```
1 def Motdepasse():
2     stored = b"ZWJVLPCURJGZ"
3     mdp = bytes([x ^ 2 for x in stored])
4     return mdp.decode()
5
6 print(Motdepasse())
```

À l'exécution, ce script donne le mot de passe : XUHTNRAWPHEX.

Reverse : exercice pour Windows

En entrant ce mot de passe `XUHTNRAWPHEX` dans le binaire :



Questions ?

Pour aller plus loin

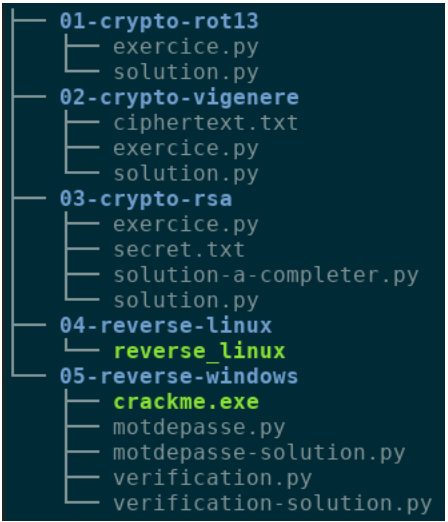
FCSC : *France Cyber Security Challenge*

- Compétition nationale individuelle de cybersécurité (*Capture The Flag, CTF*)
- Organisée tous les ans par l'ANSSI depuis 2019.
- Public prioritairement visé : moins de 25 ans.
- Plus d'informations : `https://www.ssi.gouv.fr/agence/cybersecurite/france-cybersecurity-challenge-2022/`

Épreuves techniques de différents niveaux et catégories :

- Cryptographie
- Reverse engineering
- Exploitation binaire
- Sécurité web
- Forensics
- Hardware (radio, side channel)
- Programmation
- etc.

Fournitures



Merci pour votre attention !