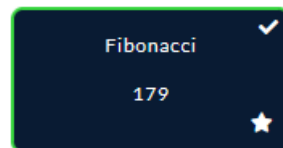


hardware



nc challenges.france-cybersecurity-challenge.fr 2301

SM) <https://www.france-cybersecurity-challenge.fr/vm>

Challenge

168 résolutions

×

Fibonacci

179

algorithmique hardware

★

Cette épreuve fait partie de la série qui utilise la machine virtuelle du FCSC 2023, plus d'informations sur celle-ci ici : <https://www.france-cybersecurity-challenge.fr/vm>

Cette fois, on vous demande de coder en assembleur la suite de Fibonacci.

La machine est initialisée avec une valeur n aléatoire (mise dans le registre $R5$) et devra contenir (dans $R0$) l'élément $Fib(n)$ une fois le code exécuté. Pour rappel :

- $Fib(0) = 0$,
- $Fib(1) = 1$,
- $Fib(n) = Fib(n - 1) + Fib(n - 2)$.

Le code machine (bytecode) sera envoyé sous un format hexadécimal, qu'on pourra générer à l'aide de l'assembleur fourni (fichier `assembly.py`).

nc challenges.france-cybersecurity-challenge.fr 2301

challenge...

machine.py

assembly.py

Flag

Submit

Le code assembleur de la fonction mathématique de la suite de Fibonacci importé doit être d'abord vérifié dans la syntaxe de la machine virtuelle locale imposée puis incluse au code du : `challenge.py`.

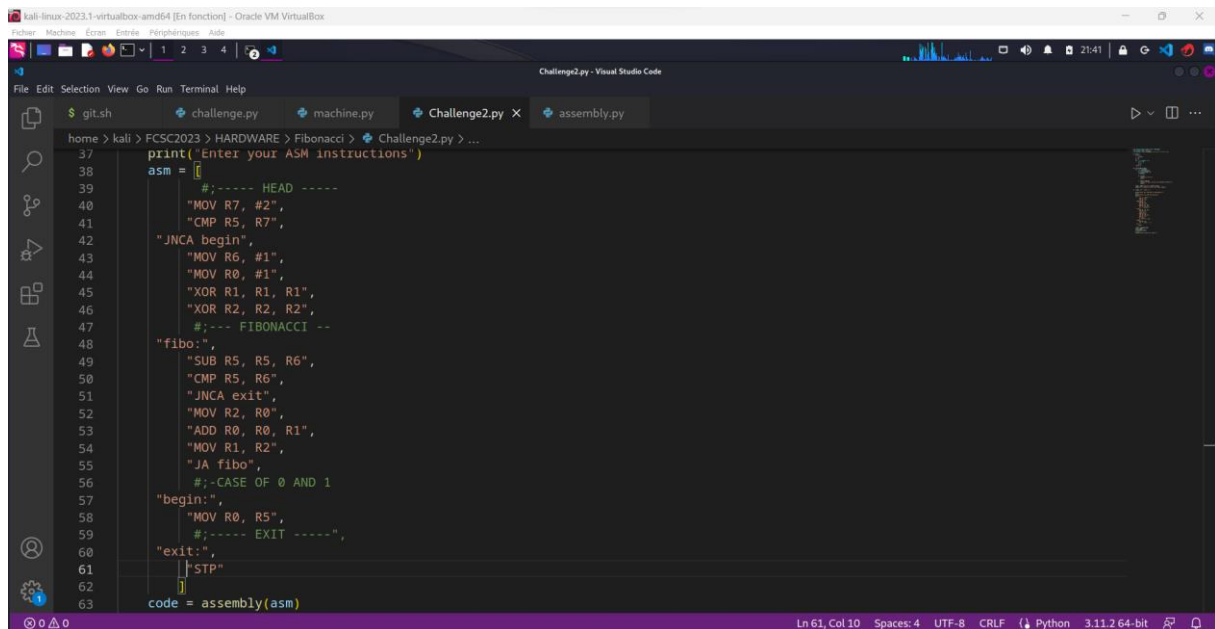
```
asm = [...  
]  
code = assembly(asm)
```

L'exécution d'assembleur produit le résultat attendu de code objet en prérequis de la gestion du flag.

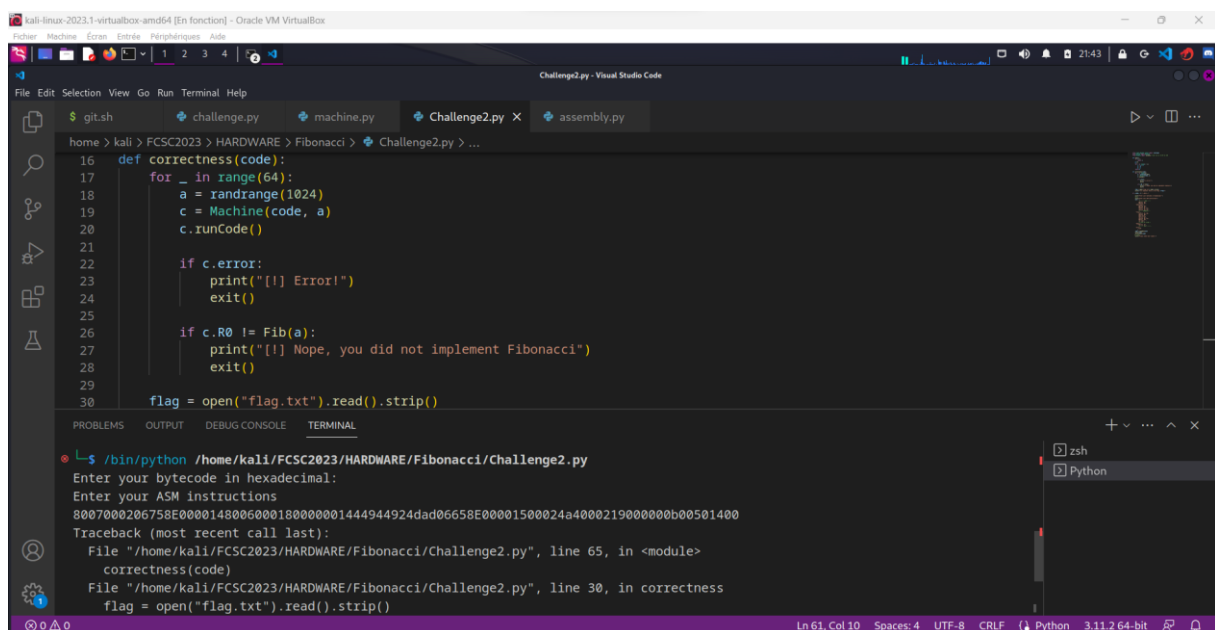
L'issue OK du programme depuis le client fournit le code objet au format attendu du bytecode HEXA.

La commande netcat permet d'entrer ce bytecode hexadécimal en préalable à la récupération de flag.

Il est à noter que la structure du programme par défaut ne peut afficher le flag que depuis le serveur.



```
37 print("Enter your ASM instructions")
38 asm = [
39     #;----- HEAD -----
40     "MOV R7, #2",
41     "CMP R5, R7",
42     "JNCA begin",
43     "MOV R6, #1",
44     "MOV R0, #1",
45     "XOR R1, R1, R1",
46     "XOR R2, R2, R2",
47     #;--- FIBONACCI ---
48     "fibo:",
49     "SUB R5, R5, R6",
50     "CMP R5, R6",
51     "JNCA exit",
52     "MOV R2, R0",
53     "ADD R0, R0, R1",
54     "MOV R1, R2",
55     "JA fibo",
56     #;--CASE OF 0 AND 1
57     "begin:",
58     "MOV R0, R5",
59     #;----- EXIT -----",
60     "exit:",
61     "STP"
62 ]
63 code = assembly(asm)
```



```
16 def correctness(code):
17     for _ in range(64):
18         a = randrange(1024)
19         c = Machine(code, a)
20         c.runCode()
21
22         if c.error:
23             print("[!] Error!")
24             exit()
25
26         if c.R0 != Fib(a):
27             print("[!] Nope, you did not implement Fibonacci!")
28             exit()
29
30     flag = open("flag.txt").read().strip()
```

```
home /kali /FCSC2023 /HARDWARE /Fibonacci /Challenge2.py
Enter your bytecode in hexadecimal:
Enter your ASM instructions
8007000206758E00001480060001800000001444944924dad06658E00001500024a400021900000b00501400
Traceback (most recent call last):
  File "/home/kali/FCSC2023/HARDWARE/Fibonacci/Challenge2.py", line 65, in <module>
    correctness(code)
  File "/home/kali/FCSC2023/HARDWARE/Fibonacci/Challenge2.py", line 30, in correctness
    flag = open("flag.txt").read().strip()
```

Il est obtenu à l'exécution le résultat correct de bytecode correspondant à notre suite de Fibonnaci :

8007000206758E00001480060001800000001444944924dad06658E00001500024a400021900000b00501400

Ce bytecode hexadécimal obtenu du côté client est ensuite à injecter du côté du serveur nc.

Remarque : il n'y a aucune autre modification de code induite sur les autres fichiers python.

En fait, les fichiers: assembly.py et machine.py correspondent à des règles d'appel générique.

Seul le fichier challenge.py contient le code de la fonction requise. Le fichier reste à modifier.

Le fichier challenge2.py contient l'import de la séquence en assembleur associée à fibonacci.

```
asm = [
    ";----- HEAD -----",
    "    MOV R7, #2",
    "    CMP R5, R7",
    "    JNCA begin",
    "    MOV R6, #1",
    "    MOV R0, #1",
    "    XOR R1, R1, R1",
    "    XOR R2, R2, R2",
    ";--- FIBONACCI ---",
    "fibonacci:",
    "    SUB R5, R5, R6",
    "    CMP R5, R6",
    "    JNCA exit",
    "    MOV R2, R0",
    "    ADD R0, R0, R1",
    "    MOV R1, R2",
    "    JA fibo",
    ";-CASE OF 0 AND 1-",
    "begin:",
    "    MOV R0, R5",
    "    STP",
    ";----- EXIT -----",
    "exit:",
    "    STP",
    ]
```

```
global fibonacci

section .text

;-----[ HEAD FUNCTION ]-----;
fibonacci:
    cmp rdi, 0x0 ; if N < 0 ;
    jl negative ; call negative ;
    cmp rdi, 0x2 ; if N < 2 ;
    jl begin ; call begin ;
    mov rax, 1 ; init A to 1 ;
    xor rbx, rbx ; init B to 0 ;
    jmp fibo ; call the loop ;
;-----[ FIBONNACI ]-----;
fibo:
    dec rdi ; N = N - 1 ;
    cmp rdi, 0x0 ; if N = 0 ;
    jle exit ; call exit ;
    mov rcx, rax ; C = A ;
    add rax, rbx ; A = A + B ;
    mov rbx, rcx ; B = C ;
    jmp fibo ; looping ;
;-----[ EXIT ]-----;
exit:
    ret ; return ;
;-----[ CASE OF 0 AND 1 ]-----;
begin:
    mov rax, rdi ; A = N ;
    ret ; return A ;
;-----[ CASE OF NEGATIVE ]-----;
negative:
    mov rax, -0x1 ; A = -1 ;
    ret ; return A ;
;-----;
```

Fichier : code assembleur local obtenu de la virtual machine (vm) pour une suite de Fibonacci

```
(kali@kali)-[~]
$ cd FCSC2023
code = assembly(asm)
(kali@kali)-[~/FCSC2023]
$ ls
asm.docx 00_workspace_visual_studio Archives FORENSICS Git HARDWARE INTRO MISC Readme.txt WEB WELCOME
(kali@kali)-[~/FCSC2023]
$ cd HARDWARE
(kali@kali)-[~/FCSC2023/HARDWARE]
$ ls
asm.docx exportcanflag.csv Fibonacci
(kali@kali)-[~/FCSC2023/HARDWARE]
$ cd Fibonacci
(kali@kali)-[~/FCSC2023/HARDWARE/Fibonacci]
$ ls
assembly.py challenge.py machine.py
(kali@kali)-[~/FCSC2023/HARDWARE/Fibonacci]
$ code
[+] Congrats! Here is the flag: FCSC{770ac04f9f113284eeee2da655eba34af09a12dba789c19020f5fd4eff1b1907}
```

Le flag est FCSC{770ac04f9f113284eeee2da655eba34af09a12dba789c19020f5fd4eff1b1907}