Reversing(2) ~アセンブリツールとプログラミングを使う問題編~

2年 小俣直史

チャレンジ1:Rev2a(実行ファイル)

復習:

- r2 [ファイル名]
- -> aaaaaa(解析)
- -> VV@[関数名] (命令実行マップ) (最初は main を見て、それから 目星をつけた関数に入るのがおすすめ)

(Esc キーで操作取り消し・戻る)

↑と前回のReversingのパワポ資料を参照しながらやってみましょう [注意]ファイルは最初に sudo chmod +x で権限を付与してあげましょう!!

```
ubuntu@GLANGZEL: ~
[0×00400604]> 0×400604 # int main (int argc, char **argv, char **envp);
                       [0×400604]
                                                    var_8 = input_code():
                      90: int main (int argc, char **
                                                                  入力用の関数で受け付けた値
                      ; var uint64_t var_8h @ rbp-0×8
                      push rbp
                                                    (整数値)を代入
                      mov rbp, rsp
                       call sym.input_code;[oa]
                       mov gword [var_8h], rax
                      mov rax, qword [var_8h]
                      mov rdi, rax
                      call sym.check_code;[ob]
                                                     check_code(var_8):
                      test eax, eax
                                                                   先の整数値をフラグと比
                                                     較して分岐させる関数
               0×400625 [oe]
                                              0×400633 [og]
              lea rdi, qword str.Wrong
                                             lea rdi, qword str.Correct
              call sym.imp.puts;[od]
                                             call sym.imp.puts;[od]
mov rax, qword [var_8h]
                                             mov rsi, rax
                                             lea rdi, qword str.The_flag_is_ctf4b__llu
                                             mov eax, 0
                                             call sym.imp.printf;[of]
                               0×400657 [oh]
                              mov eax, 0
```

```
[0×4005a7]
55: sym.input_code ();
; var int64_t var_8h @ rbp-0×8
push rbp
mov rbp, rsp
sub rsp, 0×10
; const char *format
lea rdi, qword str.LISENCE_CODE:
mov eax, 0
call sym.imp.printf;[oa]
lea rax, qword [var_8h]
mov rsi, rax
; const char *format
lea rdi, qword str.llu
mov eax, 0
call sym.imp.__isoc99_scanf;[ob]
mov rax, qword [var_8h]
leave
```

```
= printf("LISENCE CODE: ")

ubuntu@GLANGZEL:~$ ./rev2a
LISENCE CODE:
```

= scanf("%llu", &var_8) (var_8:符号なし整数値 int)

Return var 8:

```
[0×4005de]
         38: sym.check_code (uint32_t arg1);
          ; var uint32_t var_8h @ rbp-0×8
           arg uint32_t arg1 @ rdi
                                           if(var8 == 0x6aac52eda56eea4a)
         push rbp
                                                    ->整数値と比較して命令を分岐
         mov rbp, rsp
          : arg1
         mov qword [var_8h], rdi
         movabs rax, 0×6aac52eda56eea4a
          cmp qword [var_8h], rax
          ine 0×4005fd
0×4005f6 [ob]
                        0×4005fd [oc]
mov eax, 0
jmp 0×400602
                       mov eax, 1
       0×400602 [od]
      pop rbp
```

入力した値をそのまま比較しているだけ

ただ、scanfで%lluを受け付けているため、16進数を 10進数に変換してから入力(%x であれば16進数値なのでそのまま入力してよい。要確認)

0x6aac52eda56eea4a = 7686609844650830410

ubuntu@GLANGZEL:~\$./rev2a
LISENCE CODE: 7686609844650830410
Correct!
The flag is ctf4b{7686609844650830410}
ubuntu@GLANGZEL:~\$

チャレンジ1:Rev2b(アセンブラファイル)

FLAG形式は ctf4b{処理後のrax レジスタの値}

```
[注意]
```

_[関数]:

処理

• • •

• • •

前回のReversingのパワポ資料を参照しながらやってみましょう [注意]ファイルは最初に sudo chmod +x で権限を付与してあげましょう!!

```
_start:
 mov rax, 1
 mov rbx, 1
 mov rcx, 1
.loop:
 mov rdx, rbx
 add rbx, rax
 mov rax, rdx
 inc rcx
 cmp rcx, 60
 jnz .loop
_end:
 hlt
```

rax, rbx, rcxに1を代入

```
start:
 mov rax, 1
 mov rbx, 1
 mov rcx,
.loop:
 mov rdx, rbx
 add rbx, rax
 mov rax, rdx
 inc rcx
 cmp rcx, 60
 jnz .loop
end:
 hlt
```

```
start:
 mov rax, 1
                  rdx ← rbx
 mov rbx, 1
                  rbx \leftarrow rbx + rax
 mov rcx, 1
                  rax ← rdx
.loop:
 mov rdx, rbx
 add rbx, rax
 mov rax, rdx
 inc rcx
 cmp rcx, 60
 jnz .loop
end:
 hlt
```

```
start:
 mov rax, 1
 mov rbx, 1
 mov rcx, 1
.loop:
                   rcx をインクリメント
 mov rdx, rbx
                 rcx が 60 ならループを終了
 add rbx, rax
 mov rax, rdx
 inc rcx
 cmp rcx, 60
 jnz .loop
end:
 hlt
```

```
つまり、
rax,
rbx=rbx,
rax+rbx
を59回繰り返す
```

-> 59番目の rax を計算する

```
# Python 3

a, b = 1, 1

for i in range(59): #59回繰り返す
    a, b = b, a + b

print("ctf4b{{{}}}".format(a)) #ctf4b{[aの値]}のように出力
```

つまり

```
>>> print("ctf4b{{{}}}".format(a))
ctf4b{1548008755920}
>>>
```

CTF Reversingの問題には、プログラムを使う問題も結構あります。「ctf reversing write up」な感じで調べると各種CTFの解説がみられるのでぜひ参考にしてみてください (大体Python3 を使ってるイメージ?)