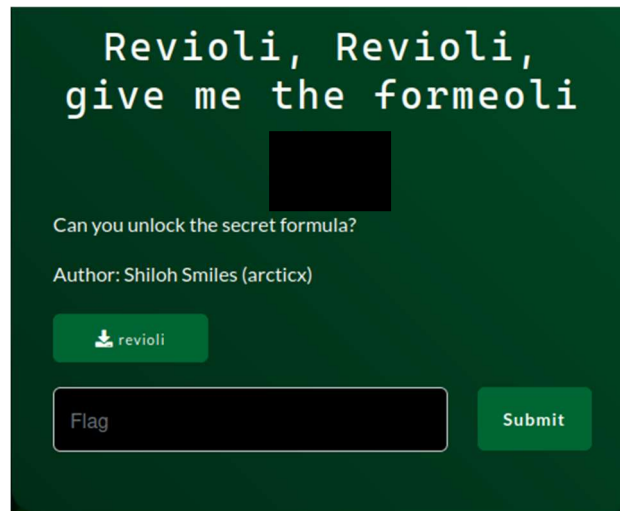


## Muthra

Patriot CTF 2024, Challenge: "Revioli, Revioli, give me the formeoli", Category: rev

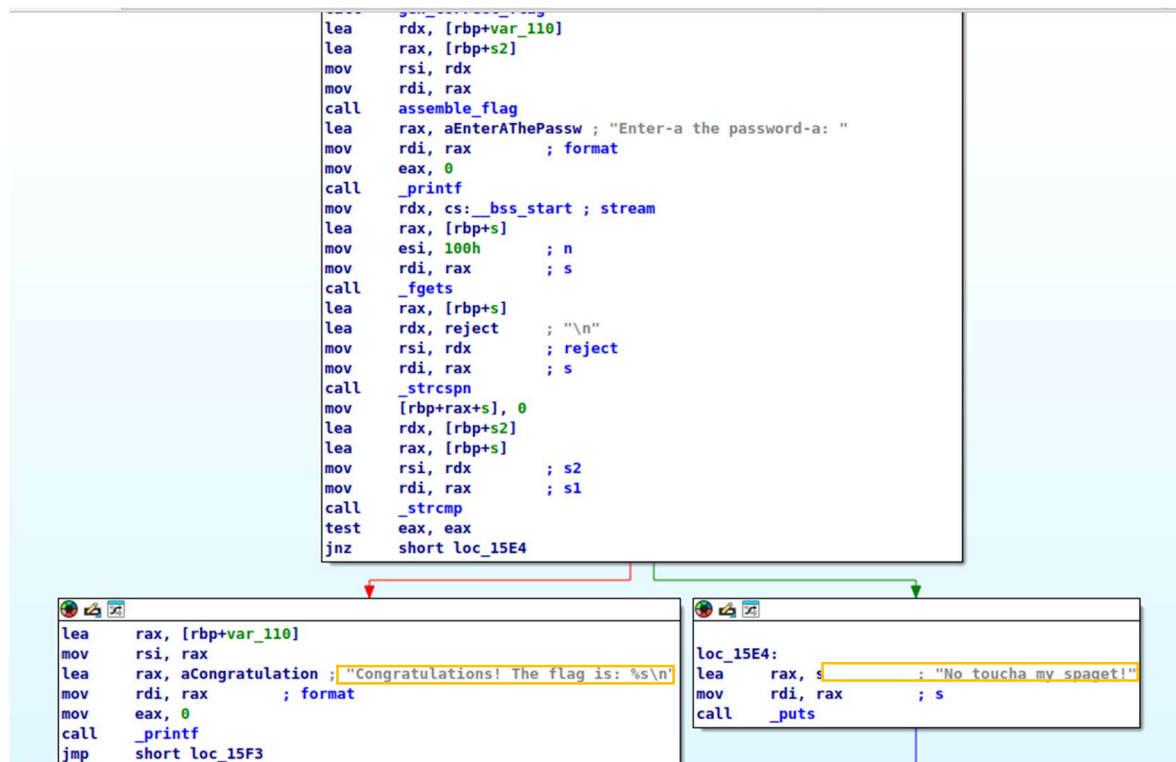
CTF Date :09/20/24-09/22/2024



### Step 1: Disassembly with IDA Free

I opened the executable file in **IDA Free** to disassemble the binary and analyze its logic. The program has a password check where it compares the entered password with a hardcoded string using `strcmp`.

- If the strings match, the program provides the flag.
- If the strings don't match, it prints "No toucha my spaget!" and exits.



## Step 2: Dynamic Analysis with GDB

I proceeded to debug the program using **GDB** to manipulate its behavior and bypass the password check.

- First, I disassembled the main function by running:
- Next, I identified the strcmp call, which compares the entered password with the correct one.

```
(gdb) disassemble *main
Dump of assembler code for function main:
0x0000000000001511 <+0>:      endbr64
0x0000000000001515 <+4>:      push    rbp
0x0000000000001516 <+5>:      mov     rbp, rsp
0x0000000000001519 <+8>:      sub     rsp, 0x310
0x0000000000001520 <+15>:     mov     rax, QWORD PTR fs:0x28
0x0000000000001529 <+24>:     mov     QWORD PTR [rbp-0x8], rax
0x000000000000152d <+28>:     xor     eax, eax
0x000000000000152f <+30>:     lea     rax, [rbp-0x210]
0x0000000000001536 <+37>:     mov     rdi, rax
0x0000000000001539 <+40>:     call    0x1298 <gen_correct_flag>
0x000000000000153e <+45>:     lea     rdx, [rbp-0x110]
0x0000000000001545 <+52>:     lea     rax, [rbp-0x210]
0x000000000000154c <+59>:     mov     rsi, rdx
0x000000000000154f <+62>:     mov     rdi, rax
0x0000000000001552 <+65>:     call    0x14d6 <assemble_flag>
0x0000000000001557 <+70>:     lea     rax, [rip+0xac1]          # 0x201f
0x000000000000155e <+77>:     mov     rdi, rax
0x0000000000001561 <+80>:     mov     eax, 0x0
0x0000000000001566 <+85>:     call    0x10e0 <printf@plt>
0x000000000000156b <+90>:     mov     rdx, QWORD PTR [rip+0x2a9e] # 0x4010 <stdin@GLIBC_2.2.5>
0x0000000000001572 <+97>:     lea     rax, [rbp-0x310]
0x0000000000001579 <+104>:    mov     esi, 0x100
0x000000000000157e <+109>:    mov     rdi, rax
0x0000000000001581 <+112>:    call    0x1110 <fgets@plt>
0x0000000000001586 <+117>:    lea     rax, [rbp-0x310]
0x000000000000158d <+124>:    lea     rdx, [rip+0xaa4]          # 0x2038
0x0000000000001594 <+131>:    mov     rsi, rdx
0x0000000000001597 <+134>:    mov     rdi, rax
0x000000000000159a <+137>:    call    0x1100 <strcspn@plt>
0x000000000000159f <+142>:    mov     BYTE PTR [rbp+rax*1-0x310], 0x0
0x00000000000015a7 <+150>:    lea     rdx, [rbp-0x210]
0x00000000000015ae <+157>:    lea     rax, [rbp-0x310]
0x00000000000015b5 <+164>:    mov     rsi, rdx
0x00000000000015b8 <+167>:    mov     rdi, rax
0x00000000000015bb <+170>:    call    0x1120 <strcmp@plt>
0x00000000000015c0 <+175>:    test    eax, eax
0x00000000000015c2 <+177>:    jne     0x15e4 <main+211>
0x00000000000015c4 <+179>:    lea     rax, [rbp-0x110]
0x00000000000015cb <+186>:    mov     rsi, rax
0x00000000000015ce <+189>:    lea     rax, [rip+0xa6b]          # 0x2040
0x00000000000015d5 <+196>:    mov     rdi, rax
0x00000000000015d8 <+199>:    mov     eax, 0x0
0x00000000000015dd <+204>:    call    0x10e0 <printf@plt>
0x00000000000015e2 <+209>:    jmp     0x15f3 <main+226>
0x00000000000015e4 <+211>:    lea     rax, [rip+0xa77]          # 0x2062
0x00000000000015eb <+218>:    mov     rdi, rax
0x00000000000015ee <+221>:    call    0x10c0 <puts@plt>
0x00000000000015f3 <+226>:    mov     eax, 0x0
0x00000000000015f8 <+231>:    mov     rdx, QWORD PTR [rbp-0x8]
0x00000000000015fc <+235>:    sub     rdx, QWORD PTR fs:0x28
```

### Step 3: Breakpoint at main

I set a breakpoint at the start of the main function to step through the program and understand its flow:

I then ran the program and used the ni (next instruction) command to step through it line by line.

```
quit
(gdb) break *main
Breakpoint 1 at 0x1511
(gdb) run
Starting program: /Patriot ctf/revioli
Downloading separate debug info for system-supplied DSO at 0x7ffff7fc3000
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, 0x0000555555555511 in main ()
(gdb) ni
0x0000555555555515 in main ()
(gdb)
0x0000555555555516 in main ()
(gdb)
0x0000555555555519 in main ()
(gdb)
0x0000555555555520 in main ()
(gdb)
0x0000555555555529 in main ()
(gdb)
0x000055555555552d in main ()
(gdb)
```

```
(gdb)
0x0000555555555581 in main ()
(gdb)
Enter-a the password-a: dskfdsjk
0x0000555555555586 in main ()
(gdb)
0x000055555555558d in main ()
(gdb)
0x0000555555555594 in main ()
(gdb)
0x0000555555555597 in main ()
(gdb)
0x000055555555559a in main ()
(gdb)
0x000055555555559f in main ()
(gdb)
0x00005555555555a7 in main ()
(gdb)
0x00005555555555ae in main ()
(gdb)
0x00005555555555b5 in main ()
(gdb)
0x00005555555555b8 in main ()
(gdb)
0x00005555555555bb in main ()
(gdb)
0x00005555555555c0 in main ()
(gdb)
0x00005555555555c2 in main ()
(gdb)
0x00005555555555e4 in main ()
(gdb)
0x00005555555555eb in main ()
(gdb)
0x00005555555555ee in main ()
(gdb)
No toucha my spaget!
```

#### Step 4: Manipulating the eax Register

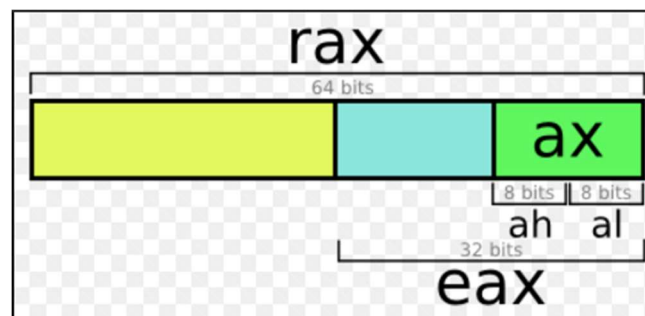
After the strcmp call, I found a test eax, eax instruction. This is where the result of the comparison (strcmp) is checked. If the eax register contains 0 (indicating the strings matched), the program jumps to the section that prints the flag.

So, I set a breakpoint at this instruction.

Then, I continued execution until the breakpoint was hit. At this point, I modified the value of the eax register to 0, forcing the program to believe that the password was correct.

```
0x000000000000015b5 <+164>:  mov    rsi, rdx
0x000000000000015b8 <+167>:  mov    rdi, rax
0x000000000000015bb <+170>:  call   0x1120 <strcmp@plt>
0x000000000000015c0 <+175>:  test   eax, eax
0x000000000000015c2 <+177>:  jne    0x15e4 <main+211>
```

Note : Eax is a part of rax



Note: I used info register command to find the register values



```

(gdb) break *0x00005555555555c0
Breakpoint 2 at 0x5555555555c0
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /Patriot ctf/revioli
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, 0x0000555555555511 in main ()
(gdb) continue
Continuing.
Enter-a the password-a: dfdsf

Breakpoint 2, 0x00005555555555c0 in main ()
(gdb) info register
rax                0x1b                27
rbx                0x7fffffffdec8        140737488346824
rcx                0x49                 73
rdx                0x7fffffffdb90        140737488346000
rsi                0x7fffffffdb90        140737488346000
rdi                0x7fffffffda90        140737488345744
rbp                0x7fffffffdda0        0x7fffffffdda0
rsp                0x7fffffffda90        0x7fffffffda90
r8                 0x5555555596b6        93824992253622

```

```

(gdb) set $eax=0
(gdb) info register
rax                0x0                 0
rbx                0x7fffffffdec8        140737488346824
rcx                0x49                 73
rdx                0x7fffffffdb90        140737488346000
rsi                0x7fffffffdb90        140737488346000
rdi                0x7fffffffda90        140737488345744
rbp                0x7fffffffdda0        0x7fffffffdda0
rsp                0x7fffffffda90        0x7fffffffda90
r8                 0x5555555596b6        93824992253622

```

### Step 5: Continuing Execution to Retrieve the Flag

After setting `eax` to 0, I used the `ni` command to continue stepping through the program, which now jumped to the flag-printing section. As a result, the program printed the flag, successfully bypassing the password check.

```
(gdb) ni
0x00005555555555c2 in main ()
(gdb)
0x00005555555555c4 in main ()
(gdb)
0x00005555555555cb in main ()
(gdb)
0x00005555555555ce in main ()
(gdb)
0x00005555555555d5 in main ()
(gdb)
0x00005555555555d8 in main ()
(gdb)
0x00005555555555dd in main ()
(gdb)
Congratulations! The flag is: PCTF{ITALY_01123581321345589144233377}
```

## Conclusion

By analyzing the logic in IDA and using GDB to manipulate the execution flow, I was able to bypass the password check and retrieve the flag without knowing the correct password.