Description: Find the correct input to reveal the hidden flag
ELF 64-bit LSB pie executable

- Step 1: Running the File
  ./two
  sh: 1: %cjpka%qm: not found
  Incorrect.
  Note: I suspect that the key entered is related to the given output. I need to input the correct key to get the right word sequence.
- Step 2: Analyzing with radare2
  afl
  0x00001030 1 6 sym.imp.puts
  0x00001040 1 6 sym.imp.system
  ...
  0x00001180 14 382 main
  Note: I found this function in the file. The highlighted chunks are interesting and worth exploring.
- Step 3: Diving into the Main Function using Ghidra
  Exploring the decompiled `main` function reveals interesting code. The default value of `j` is `0x14` (20 in decimal), but if there is an input argument, `j` will be changed according to that argument. Each bit in `j` determines whether certain `chunks` will be modified with an XOR operation of 5. The combined result of `chunk_one` to `chunk_five` is executed as a shell command. If the command result is not 0, it prints "Incorrect."
- Step 4: Examining the Chunks
  chunk_one: "fmj%'Fj"
  chunk_two: "kbwdqv$%"
  chunk_three: "Youve fo"
  chunk_four: "pka%qm%"
  chunk_five: "flag.\""

- Step 5: Decoding XOR

  Let's decode these annoying chunks.

```python
chunks = ["`fmj%'Fj", "kbwdqv$%", "Youve fo", "pka%qm`%", "flag.\""]

def xor_chunk(chunk, xor_val):
    return ''.join(chr(ord(c) ^ xor_val) for c in chunk)

def generate_chunks(j):
    modified_chunks = []
    for i, chunk in enumerate(chunks):
        if j & (1 << i):
            modified_chunks.append(xor_chunk(chunk, 5))
        else:
            modified_chunks.append(chunk)
    return modified_chunks

for j in range(1, 32):  # Mencoba nilai j dari 1 hingga 31
    modified_chunks = generate_chunks(j)
    combined_command = ''.join(modified_chunks)
    print(f'j = {j:02d}: {combined_command}')
```

  Note: But what input should be used to get the complete message? Let's run some tests.

- Step 6: Epic Debugging with GDB

  Using GDB, break into the main function and inspect its variables.

  $ gdb ./two

  $ (gdb) break main

  $ (gdb) run 11

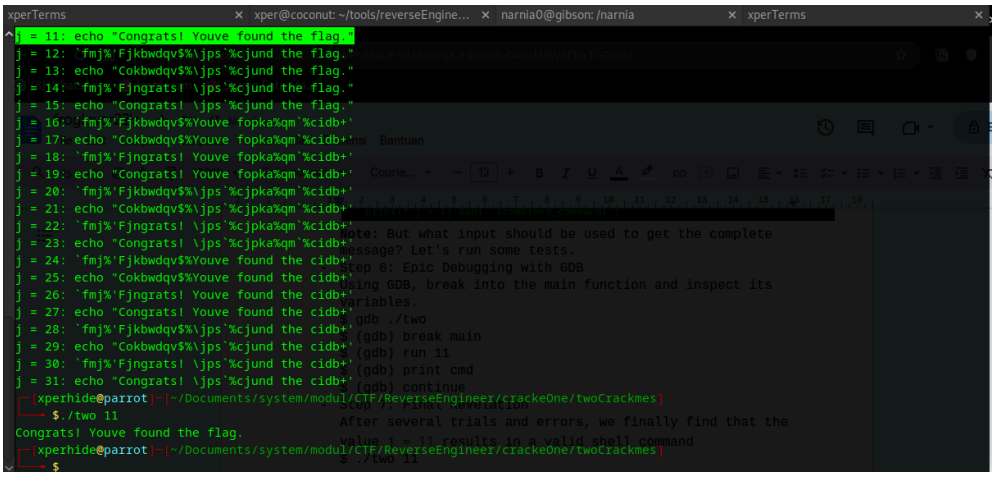
  $ (gdb) print cmd

  $ (gdb) continue

- Step 7: Final Revelation

  After several trials and errors, we finally find that the value `j = 11` results in a valid shell command

  $ ./two 11



  We Done!