CrackMe

1. Description

- Read the input
- Create pipe, then use it to read and write
- Store the input into the pipe, then it will be read after
- There is a function where read the input in the pipe and check it
- After the check, if the input is correct, there is a function will execute and generate the flag and store it in the pipe; but if not, there is nothing in the pipe
- Finally, the app reads the pipe, and has the length of the string (flag); then check if it is 0. If there is a string in the pipe, the length will be not 0, exactly, it will be 0x23 (35). That is the length of our flag plus 1 which is “/0”, a “mark” of the end of a string.

2. About the check function

- At the beginning of the function, it mallocs, stores a bunch of data which are addresses of functions.
- And those will be used in the loop that checks our input; and we kind of find it hard to know the order of the function will run, and besides, the function can run multiple times. So I will debug it then note what it will do with our input.

2.1. About the loop

To be short, there is just about how our input be checked

- The input will be read 3 times, 4 characters each time. So now, we know how many characters we need in the input.
- In each time, our 4 characters will be “encode” to a value (32bit) in the same way:

```
57              push edi
8BB8 88000000   mov edi,dword ptr ds:[eax+88]
8B50 08         mov edx,dword ptr ds:[eax+8]
8340 14 FC      add dword ptr ds:[eax+14],FFFFFFFC
03D7            add edx,edi
8B58 14         mov ebx,dword ptr ds:[eax+14]
0FB642 02       movzx eax,byte ptr ds:[edx+2]
C0C0 02         rol al,2
0FB6F0          movzx esi,al
0FB602          movzx eax,byte ptr ds:[edx]
04 12           add al,12
0FB6C8          movzx ecx,al
0FB642 01       movzx eax,byte ptr ds:[edx+1]
2C 78           sub al,78
C1E1 08         shl ecx,8
0FB6C0          movzx eax,al
0BC8            or ecx,eax
0FB642 03       movzx eax,byte ptr ds:[edx+3]
C1E1 08         shl ecx,8
0BF1            or esi,ecx
C0C8 04         ror al,4
C1E6 08         shl esi,8
0FB6C0          movzx eax,al
0BF0            or esi,eax
B8 01000000     mov eax,1
89341F          mov dword ptr ds:[edi+ebx],esi
5F              pop edi
```

There is my brief about how it do:

- in[]: input
  buf: the new value of input after “encode”
  out: the finally value from 4 input characters
- buf[0] = in[0] + 12h //18
  buf[1] = in[1] - 78h //120
  buf[2] = rol(in[2], 2)
  buf[3] = ror(in[3], 4) //0xab => 0xba
- out = buf[0]-buf[1]-buf[2]-buf[3]

- **The first 4 characters:**
  1. out += 0xBABECAFE
  2. rol out, 6
  3. out = 0x13371337 - out
  4. out ^= 0x13371337
  5. cmp out, 0x2648ED87 => here is the value we need

- **The next 4 characters:**
  1. out = 0xDEADBEEF - out
  2. cmp out, 0x94C3E659 => here is the value we need

- **The last 4 characters:**
  1. ror out, 4
  2. not out // out = 0xFFFFFFFF - out
  3. out ^= 0xABBAABBA
  4. out += 0x89ABCDEF
  5. not out // out = 0xFFFFFFFF - out
  6. cmp out, 0x5469A57F => here is the value we need

***Note: There is an overflow in the addition and subtraction**

3. Solving

"Bottom-up" is our solution.

- **The first 4 characters:**
  1. out = 0x2648ED87
  2. out ^= 0x13371337 = 0x357FFEB0
  3. out = 2^32 + 0x13371337 - out = 0xDDB71487
  4. out = ror out, 6 = 0x1F76DC52
  5. out -= 0xBABECAFE = 2^32 + out - 0xBABECAFE = 0x64B81154
  6. "Decode" it into the input

     in[0] = 0x64 - 0x12 = 0x52 => R

     in[1] = 0xB8 + 0x78 (- 2^8) = 0x30 => 0

     in[2] = ror 0x11, 2 = 0x44 => D

     in[3] = rol 0x54, 4 = 0x45 => E

- **The next 4 characters:**
  1. out = 0x94C3E659
  2. out = 2^32 + 0xDEADBEEF - out = 0x49E9D896
  3. "Decode" it into the input

     in[4] = 0x49 - 0x12 = 0x37 => 7

     in[5] = 0xE9 + 0x78 (- 2^8) = 0x61 => a

     in[6] = ror 0xD8, 2 = 0x36 => 6

     in[7] = rol 0x96, 4 = 0x69 => i

- **The last 4 characters:**
  1. out = 0x5469A57F
  2. out = 0xFFFFFFFF - out = 0xAB965A80
  3. out -= 0x89ABCDEF = 0x21EA8C91
  4. out ^= 0xABBAABBA = 0x8A50272B
  5. out = 0xFFFFFFFF - out = 0x75AFD8D4
  6. out = rol out, 4 = 0x5AFD8D47
  7. "Decode" it into the input

     in[8] = 0x5A - 0x12 = 0x48 => H

     in[9] = 0xFD + 0x78 (- 2^8) = 0x75 => u

     in[10] = ror 0x8D, 2 = 0x63 => c

     in[11] = rol 0x47, 4 = 0x74 => t

***Note: We can code a .py to solve this.**

4. Result

Now, we have the entire input character, the input is **“R0DE7a6iHuct”**.
Run the app again with found input, we get the flag like this:

“flag{S1mpl3_ST4ck_V1rTu4L_M4ch1n3}"