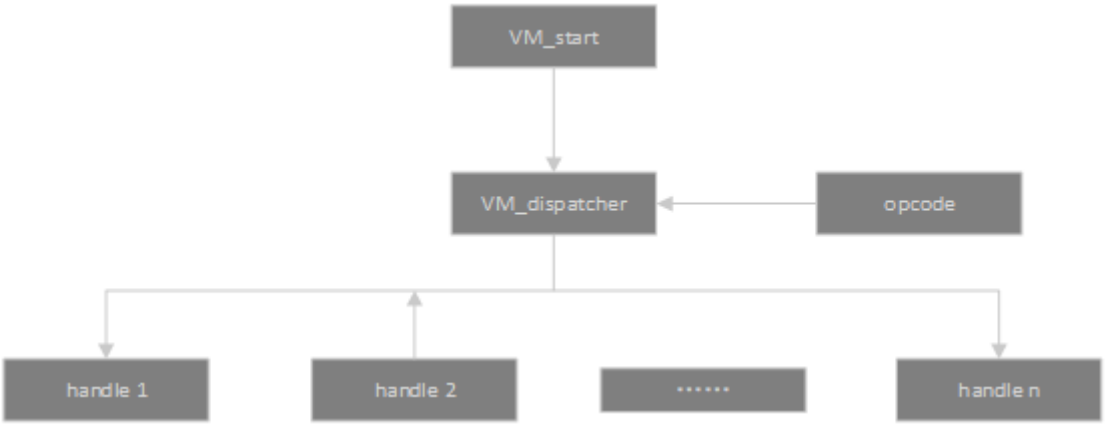
VM知识：

逆向中的虚拟机保护是一种基于虚拟机的代码保护技术。它将基于x86汇编系统中的可执行代码转换为字节码指令系统的代码，来达到不被轻易逆向和篡改的目的。简单点说就是将程序的代码转换自定义的操作码（opcode），然后在程序执行时再通过解释这些操作码，选择对应的函数执行，从而实现程序原有的功能。

分析流程：




```
vm_start :虚拟机入口函数 ，初始化虚拟机
vm_dispatcher: 调度器，解释op_code,并选择相应的函数执行，当函数执行完后会返回这里，形成一个循环，直到执行完
vm_code:程序可执行代码形成的操作码
```

Ponce的简述：

对于Ponce来说只需要关心那里输入，哪里success，哪里wrong。
Ponce是一款IDAPro插件，该工具采用C/C++开发，它可以帮助用户以一种快速简洁的方式对目标代码进行
污点测试以及符号执行。用户只需点一下鼠标或者按一下键盘，剩下的就可以交给Ponce了

```
Ctrl + Shift + M ：符号化（找到输入的参数将其变为符号变量）
```

符号执行：

传统符号执行是一种静态分析技术，最初在1976年由King JC在ACM上提出。即通过使用抽象的符号代替具体值来模拟程序的执行，当遇到分支语句时，它会探索每一个分支，将分支条件加入到相应的路径约束中，若约束可解，则说明该路径是可达的。

在遇到程序分支指令时，程序的执行也相应地搜索每个分支，分支条件被加入到符号执行保存的符号路径约束 PC，PC表示当前路径的约束条件。在收集了路径约束条件之后，使用约束求解器来验证约束的可解性，以确定该路径是否可达。若该路径约束可解，则说明该路径是可达的;

大概原理：

符号：符号是代表一组可能值的抽象。例如，一个符号 x 可能代表任意整数。
路径约束：在执行过程中，程序的控制流会根据条件分支创建不同的执行路径。符号执行会收集这些条件分支的约束，形成路径约束。

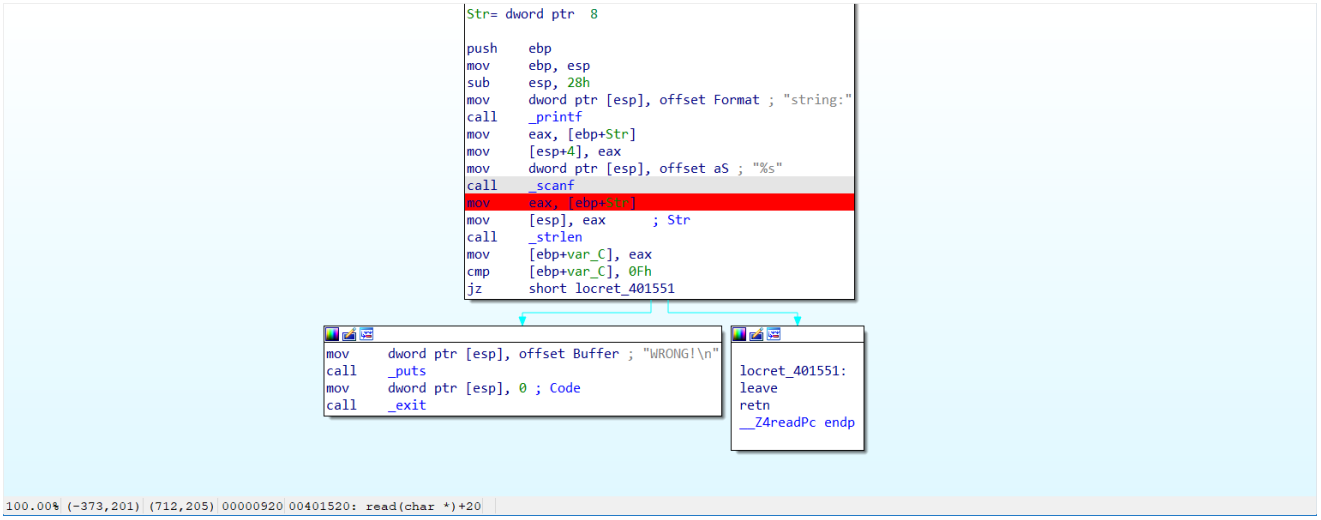
执行过程：

初始化：符号执行开始时，程序输入（如函数参数、全局变量等）被赋予符号值。
执行：程序按照正常流程执行，但所有操作都是对符号值进行的。
路径探索：在遇到条件分支时，符号执行会探索所有可能的路径。对于每个分支，它都会假设条件为真和假，并分别记录下相应的路径约束。
约束求解：符号执行完成后，分析人员可以对这些路径约束进行求解，以找到满足特定路径的具体输入值。

例题讲解：

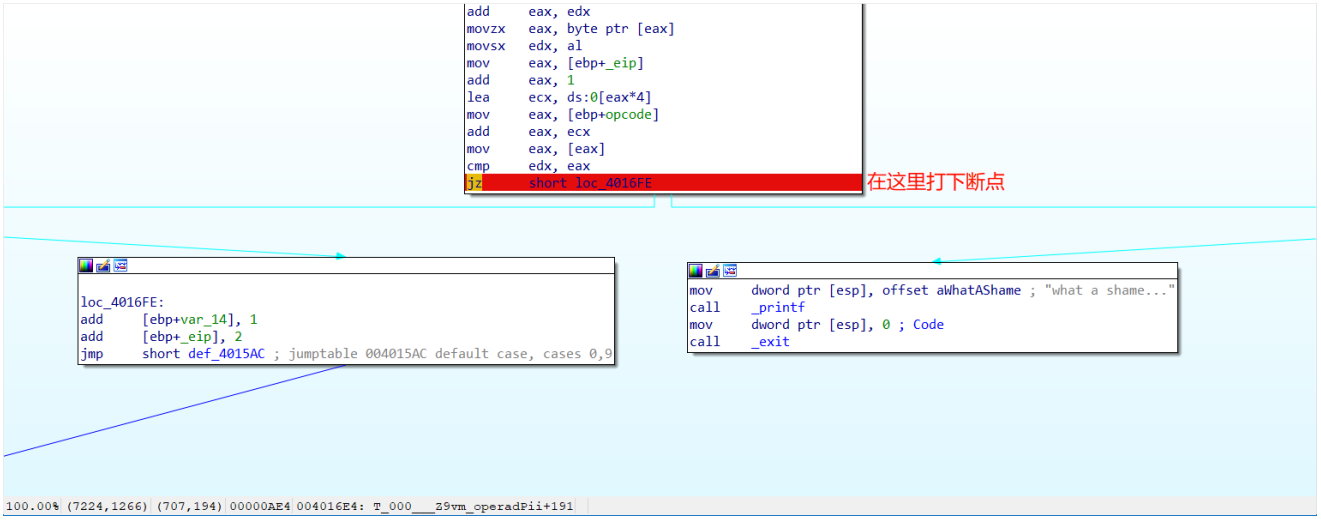
Ponce符号执行：

1、首先一定在输入字符串的地方打下断点，我们要在字符串开始变化之前将他定义为符号变量以记录路径约束，也方便找到我我们输入的字符串的位置，先找到scanf的那段代码，Tab查看他的汇编代码，在call _scanf 下面打下断点



2、接着在判断的地方打一个断点，用来约束求解，去找到我们想要的值

在这里按下Tab键进入汇编代码，然后在空格进入图形界面

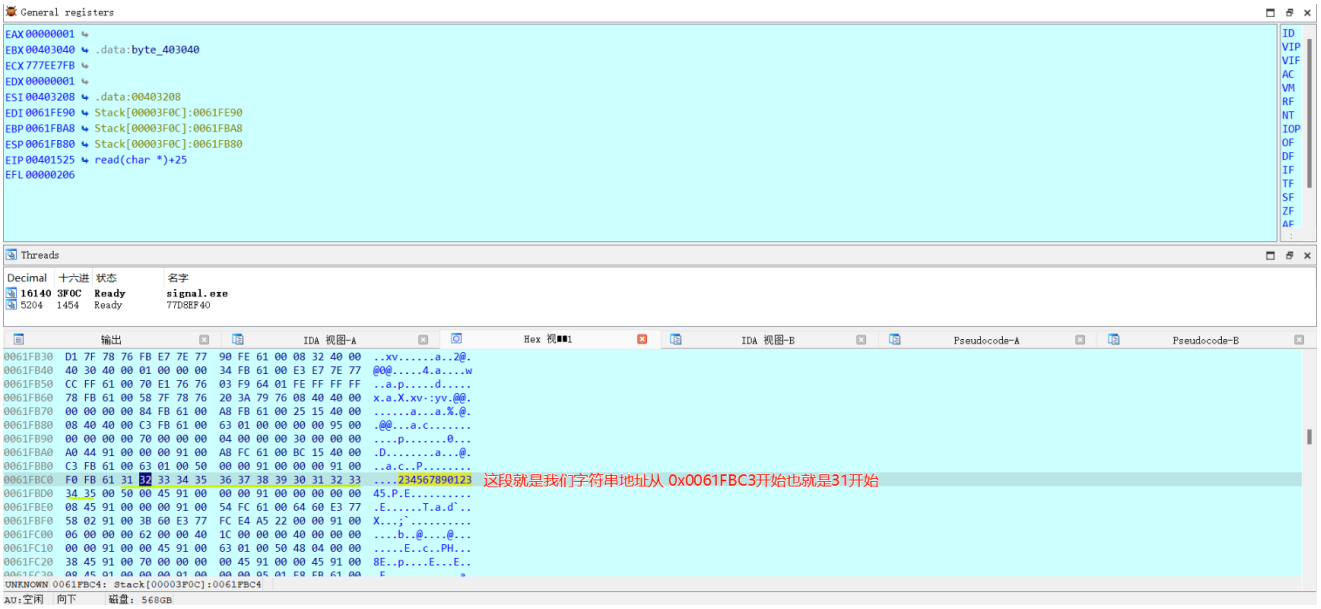


在这里打下断点

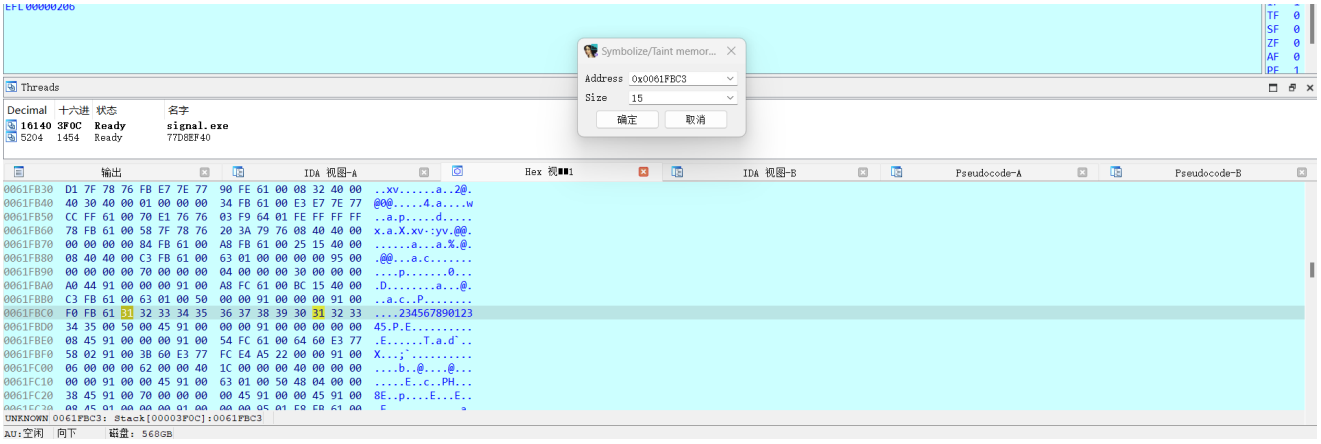
3、紧接着开始调试执行程序

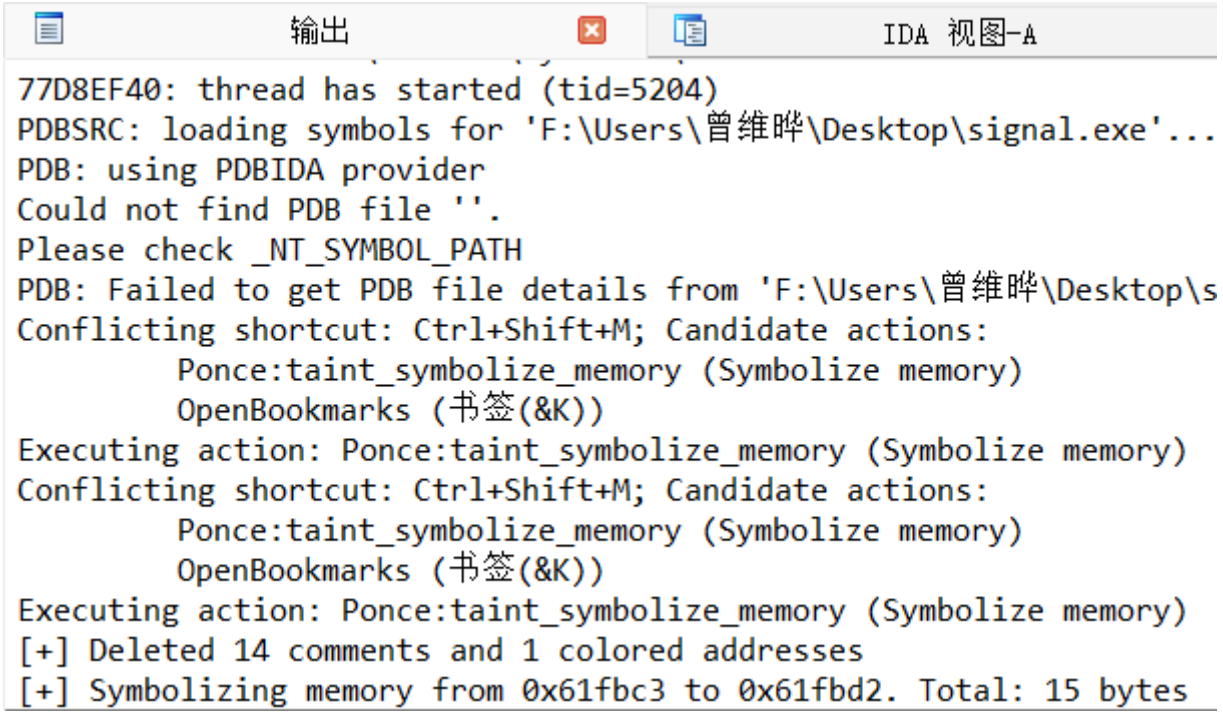
4、输入字符串为了便于识别输入 123456789012345

5、找到字符串在十六进制中的地址，并将其符号化

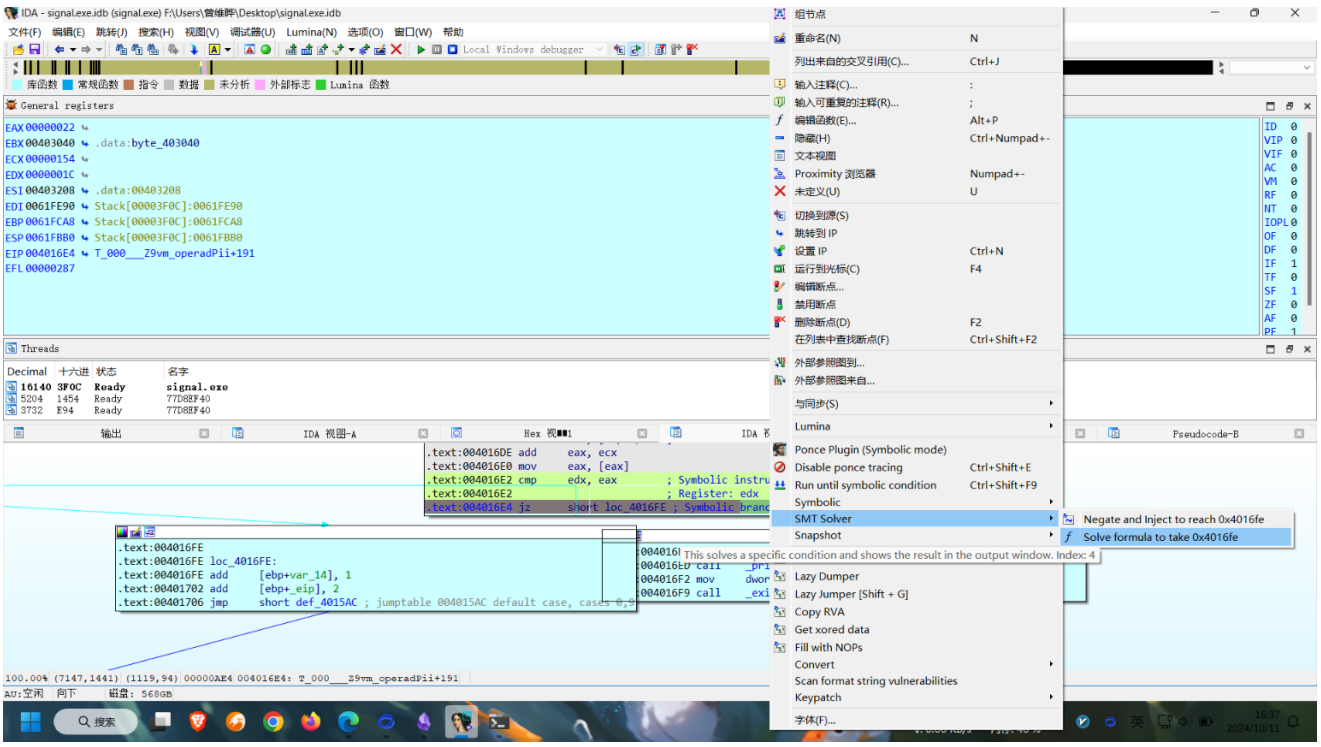


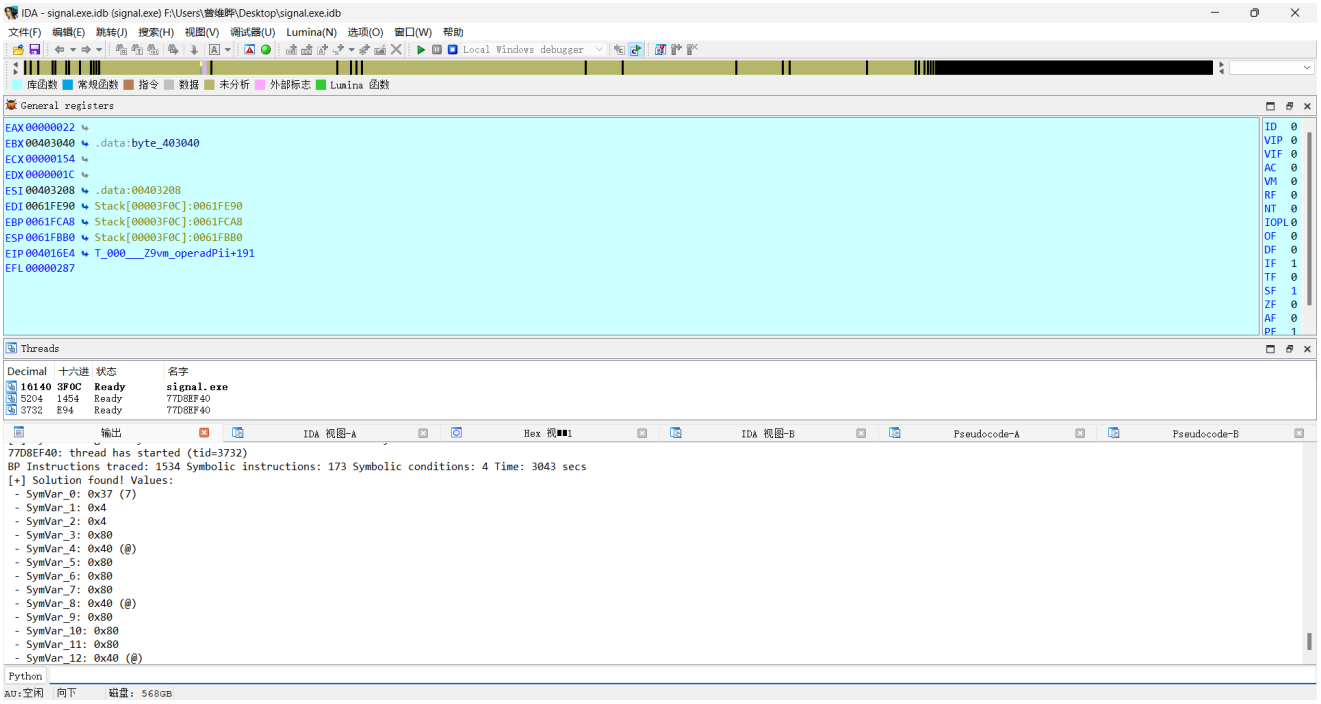
6、选中31，按Ctrl + Shift + M，将其符号化，长度为15



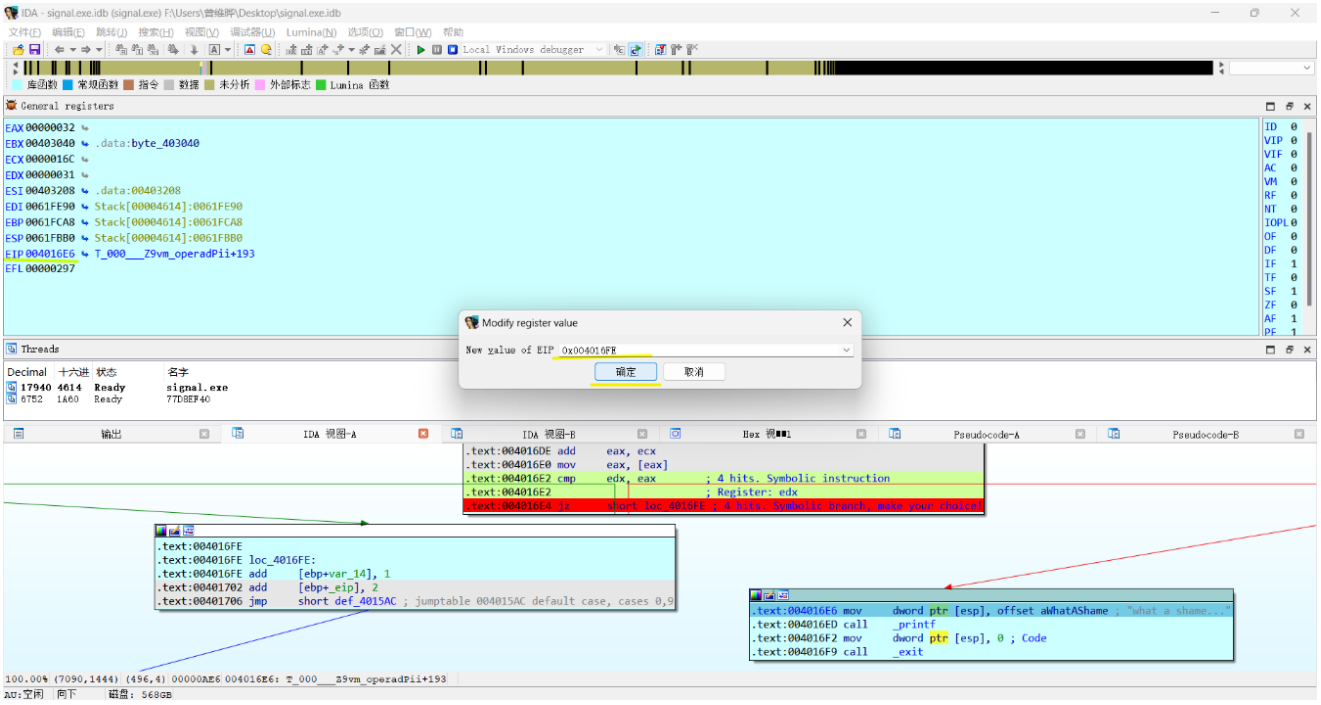


7、F9执行到判断的断点哪里，然后右键选中 SMT Solver开始约束求解，结果在输出窗口那看





8、然后F7单步执行，会跳转到右边0x004016E6这个地址，我们在General registers窗口找到EIP将他的值改为0x004016FE 也就是跳转到左边的地址



9、重复15次即可

代码实现：

```
opcode = [0x00000004, 0x00000010, 0x00000008, 0x00000003, 0x00000005,
0x00000001, 0x00000004, 0x00000020, 0x00000008, 0x00000005, 0x00000003,
0x00000001, 0x00000003, 0x00000002, 0x00000008, 0x0000000B, 0x00000001,
0x0000000C, 0x00000008, 0x00000004, 0x00000004, 0x00000001, 0x00000005,
0x00000003, 0x00000008, 0x00000003, 0x00000021, 0x00000001, 0x0000000B,
0x00000008, 0x0000000B, 0x00000001, 0x00000004, 0x00000009, 0x00000008,
0x00000003, 0x00000020, 0x00000001, 0x00000002, 0x00000051, 0x00000008,
0x00000004, 0x00000024, 0x00000001, 0x0000000C, 0x00000008, 0x0000000B,
0x00000001, 0x00000005, 0x00000002, 0x00000008, 0x00000002, 0x00000025,
0x00000001, 0x00000002, 0x00000036, 0x00000008, 0x00000004, 0x00000041,
0x00000001, 0x00000002, 0x00000020, 0x00000008, 0x00000005, 0x00000001,
0x00000001, 0x00000005, 0x00000003, 0x00000008, 0x00000002, 0x00000025,
0x00000001, 0x00000004, 0x00000009, 0x00000008, 0x00000003, 0x00000020,
0x00000001, 0x00000002, 0x00000041, 0x00000008, 0x0000000C, 0x00000001]
arr1 = [0x22, 0x3F, 0x34, 0x32, 0x72, 0x33, 0x18, 0x000000A7, 0x31,
0x000000F1, 0x00000028, 0x00000084, 0x000000C1, 0x0000001E, 0x0000007A]

def judge(eip, cnt):
    _eip = eip
    for j in range(48, 123):
        flag = j
        while _eip < 83:
            if opcode[_eip] == 2:
```

```
                tmp = opcode[_eip + 1] + flag
                _eip += 2
            elif opcode[_eip] == 3:
                tmp = flag - opcode[_eip + 1]
                _eip += 2
            elif opcode[_eip] == 4:
                tmp = opcode[_eip + 1] ^ flag
                _eip += 2
            elif opcode[_eip] == 5:
                tmp = opcode[_eip + 1] * flag
                _eip += 2
            elif opcode[_eip] == 6:
                _eip += 1
            elif opcode[_eip] == 8:
                flag = tmp
                _eip += 1
            elif opcode[_eip] == 11:
                tmp = flag - 1
                _eip += 1
            elif opcode[_eip] == 12:
                tmp = flag + 1
                _eip += 1
            elif opcode[_eip] == 1:
                if tmp == arr1[cnt]:
                    # print(f"{cnt}:{chr(j)}")
                    print(f"{chr(j)}", end='')
                    _eip += 1
                    return tmp
                else:
                    _eip = eip
                break


if __name__ == '__main__':
    bb = [0, 6, 12, 17, 22, 28, 32, 38, 44, 48, 54, 60, 66, 72, 78, 83]
    for i in range(15):
        mid = judge(bb[i], i)
```