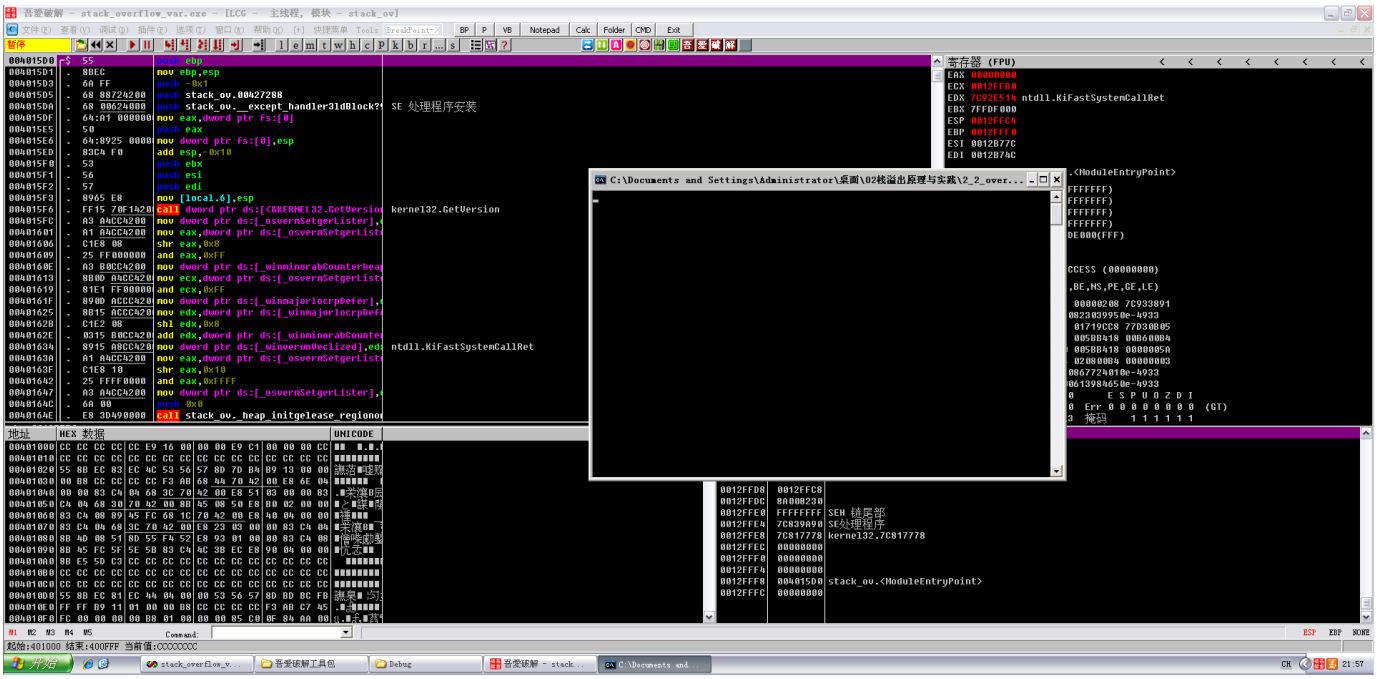
Nearby var overrun in stack

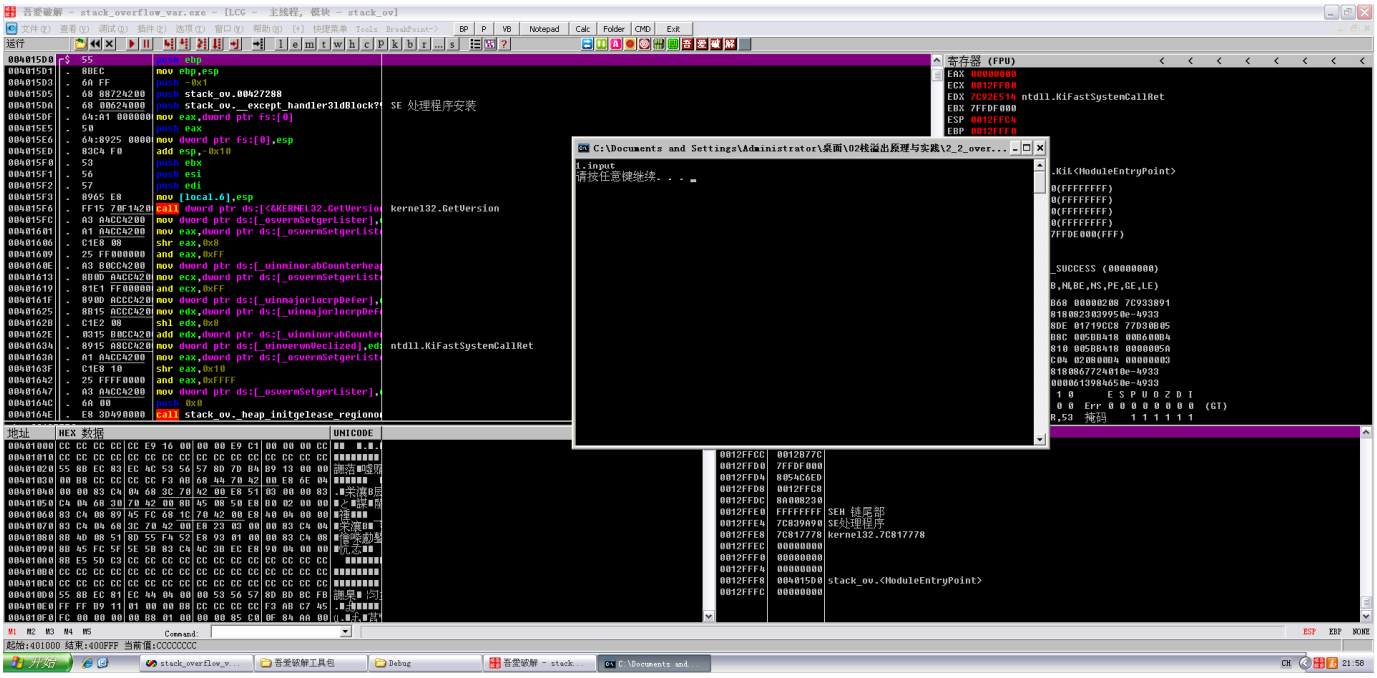
Author：wnagzihxain
Mail：tudouboom@163.com

```
#include <stdio.h>
#include <windows.h>

#define PASSWORD "1234567"

int verify_password (char *password)
{
    int authenticated;
    char buffer[8];// add local buff
    printf("call the strcmp()\n");
    system("pause");
    authenticated = strcmp(password, PASSWORD);
    printf("call the strcpy()\n");
    system("pause");
    strcpy(buffer, password);//over flowed here!
    return authenticated;
}

int main(int argc, char **argv, char **envp)
{
    int valid_flag = 0;
    char password[1024];
    while(1)
    {
        printf("1.input\n");
        system("pause");
        printf("please input password:       ");
        scanf("%s", password);
        printf("call the verify_password()\n");
        system("pause");
        valid_flag = verify_password(password);
        printf("if branch\n");
        system("pause");
        if(valid_flag)
        {
            printf("incorrect password!\n\n");
        }
        else
        {
            printf("Congratulation! You have passed the verification!\n");
            break;
        }
    }
    printf("game over :)\n");
    system("pause");
    return 0;
}
```

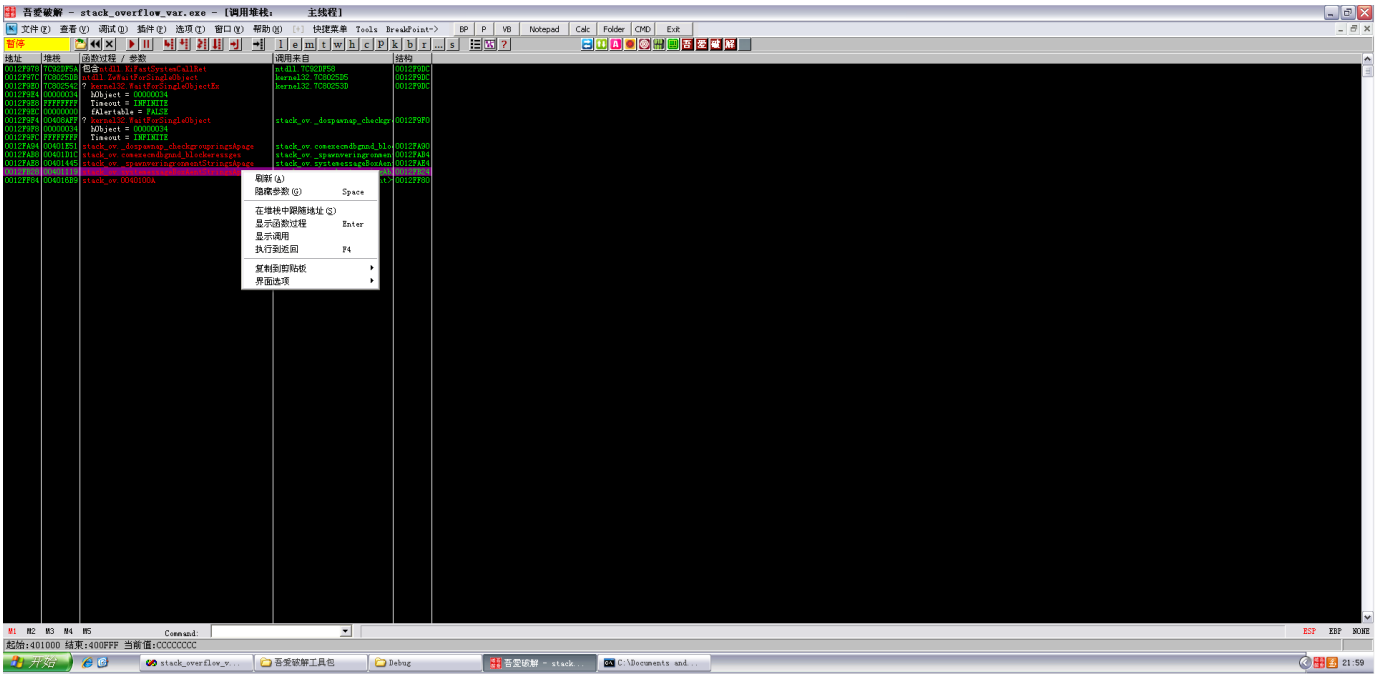
生成PE文件，载入OD



F9跑起来可以看到已经被第一个pause断下来了

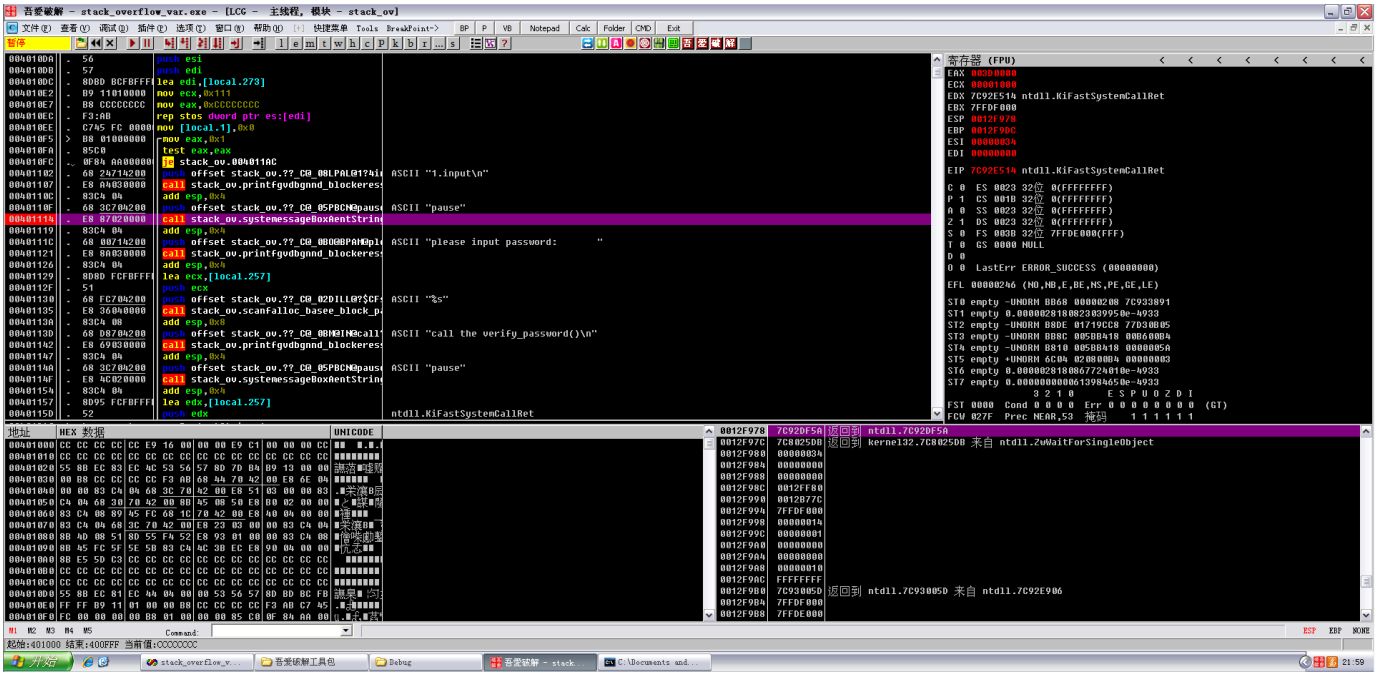


暂停程序，然后点击K，找到刚刚的pause调用，右击显示调用跳到调用的汇编代码



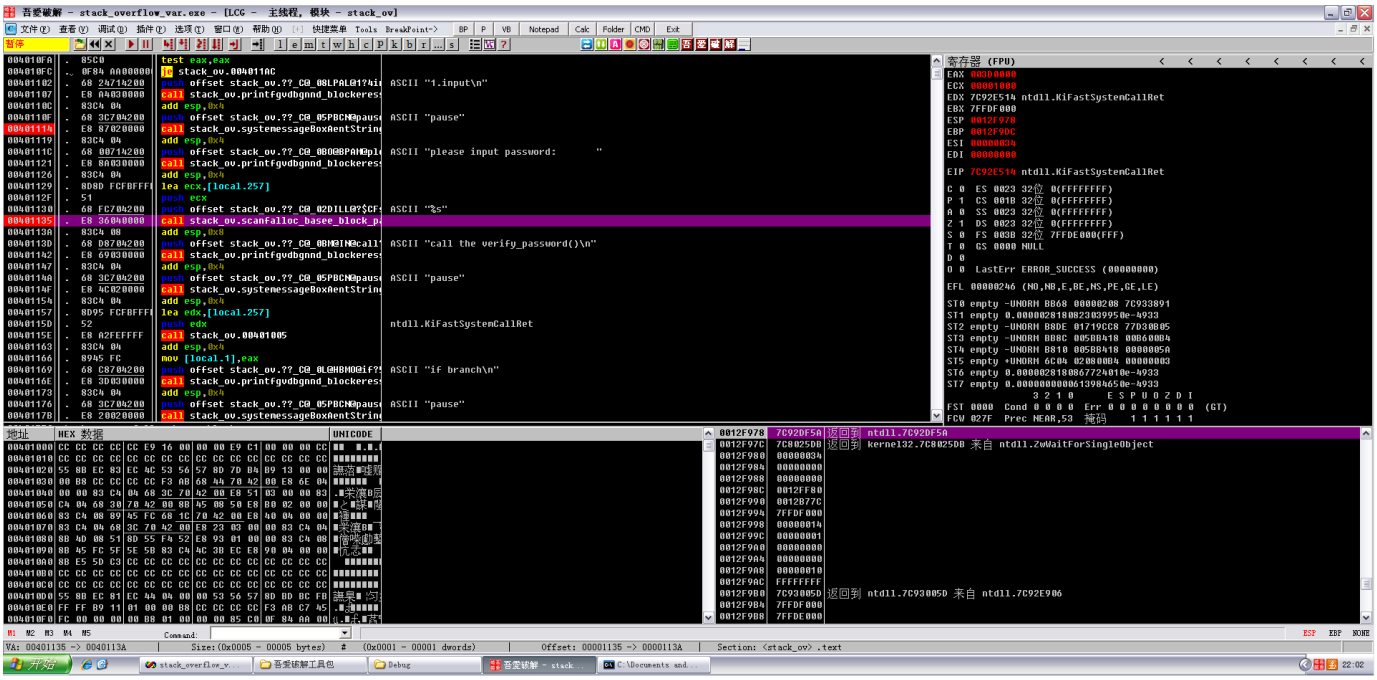
可以看到这是main函数领空，先给下个断点

```
printf("1.input\n");
system("pause");
printf("please input password:       ");
```



往下走，这是调用scanf的代码，从调用名就可以看出来，或者看到上面压了一个%s到栈里也可以判断，输入七个“q”

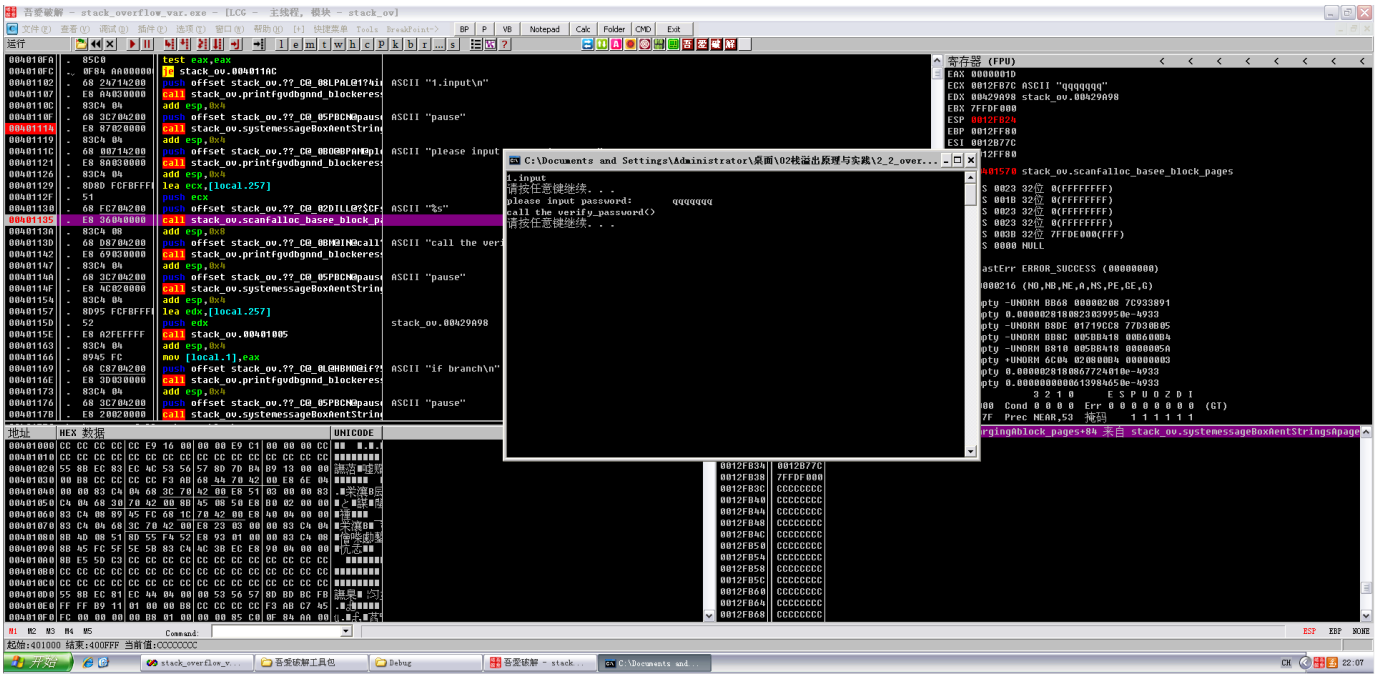
```
scanf("%s", password);
```



回车继续，发现暂停了，暂停的位置在verify_password()

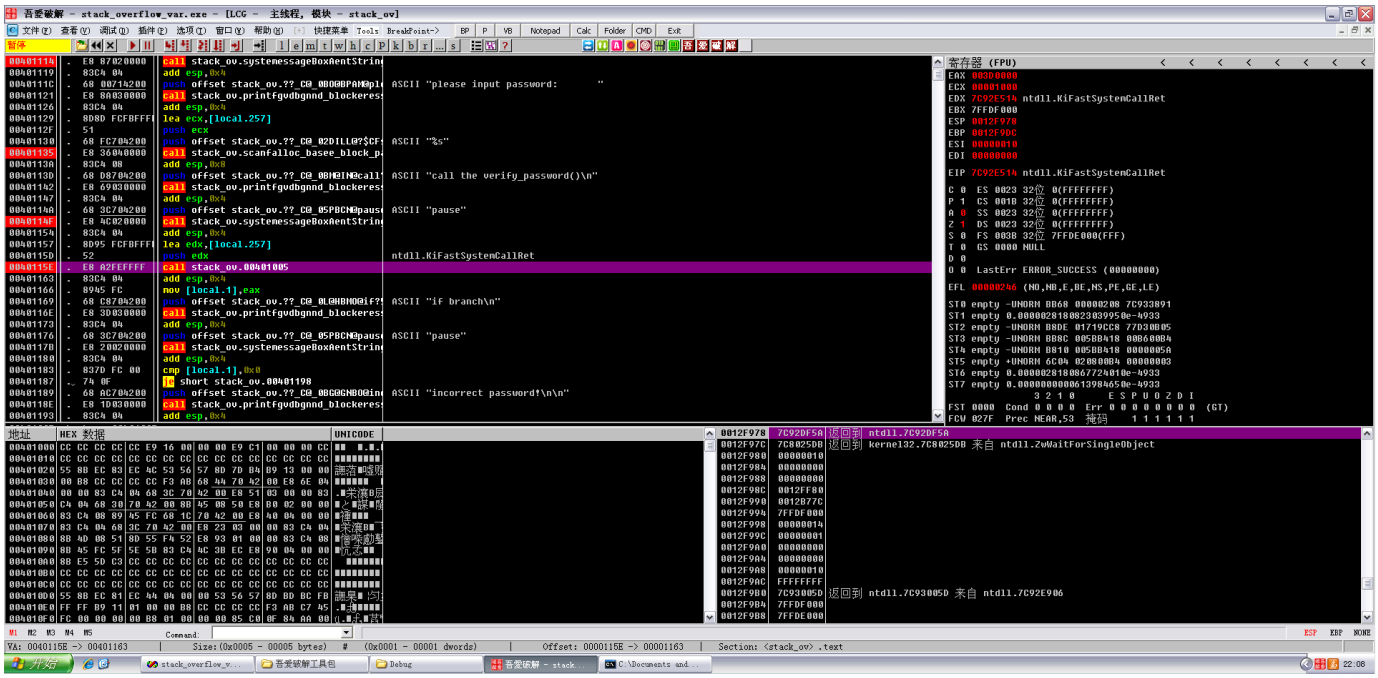
到了判断密码的函数了，C语言代码如下：

```
printf("call the verify_password()\n");
system("pause");
```

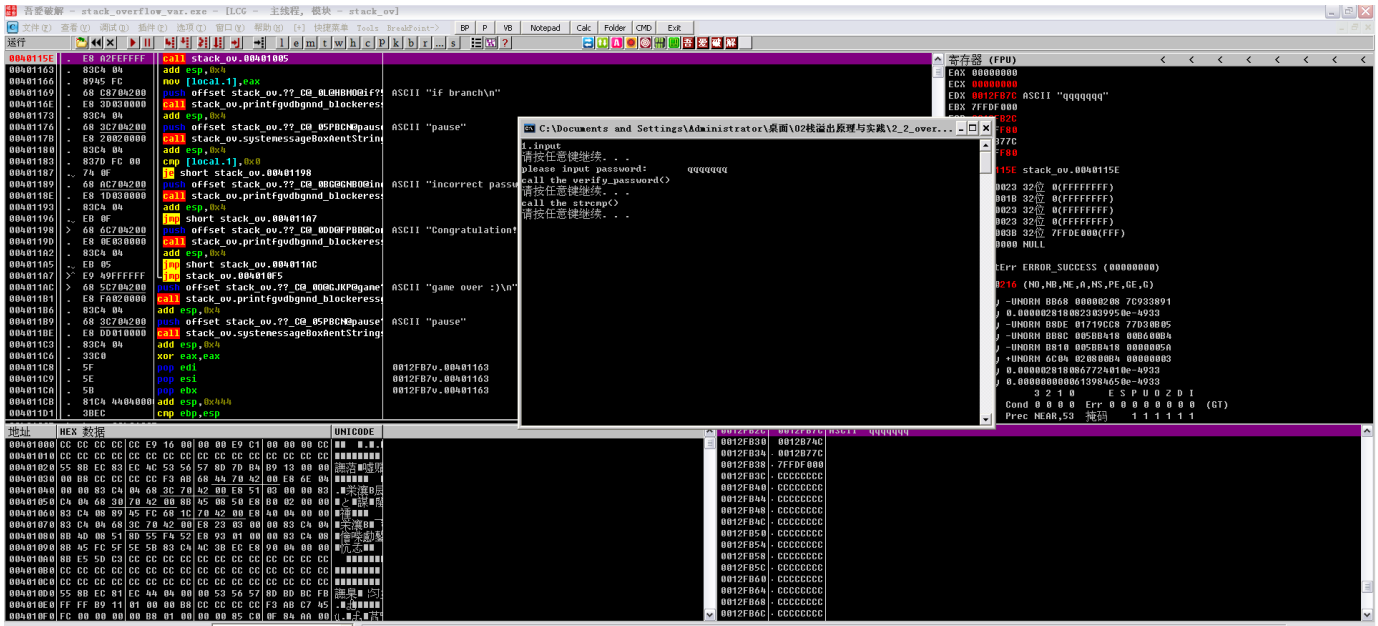


调用verify_password();

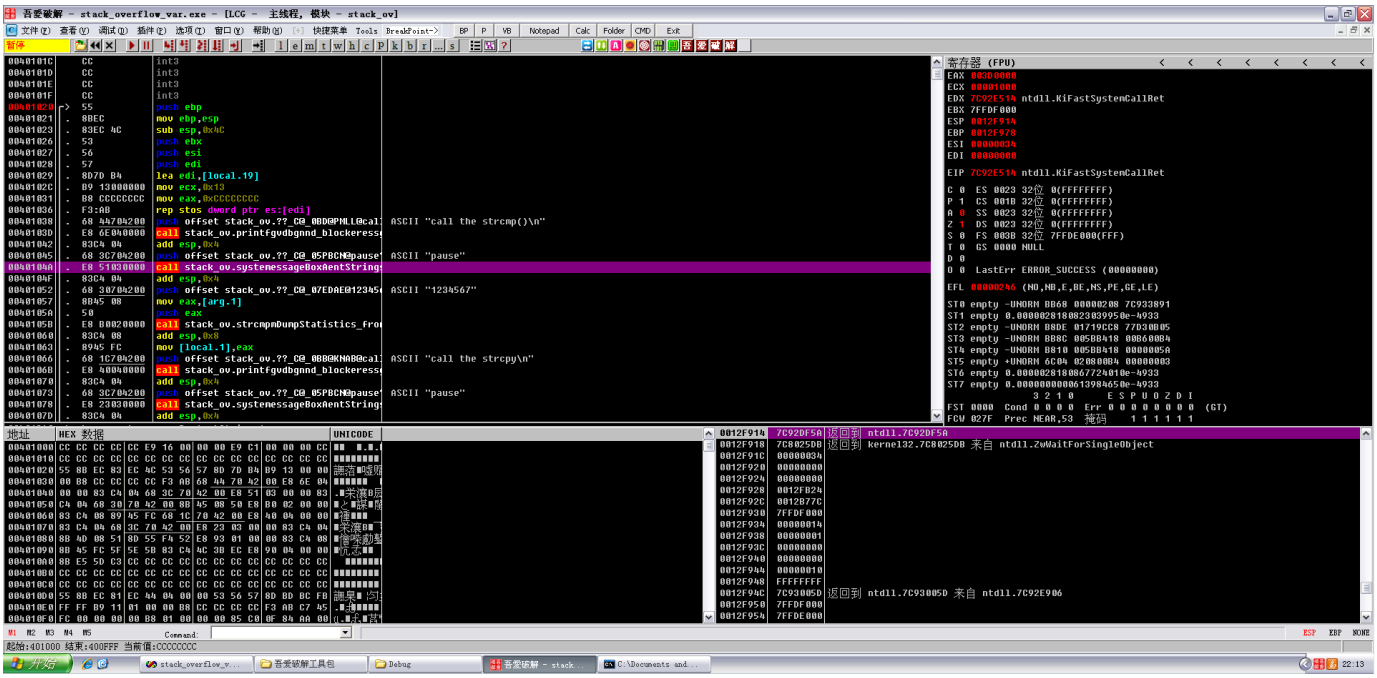
```
valid_flag = verify_password(password);
```



继续回车，碰到pause，依旧暂停，点击k

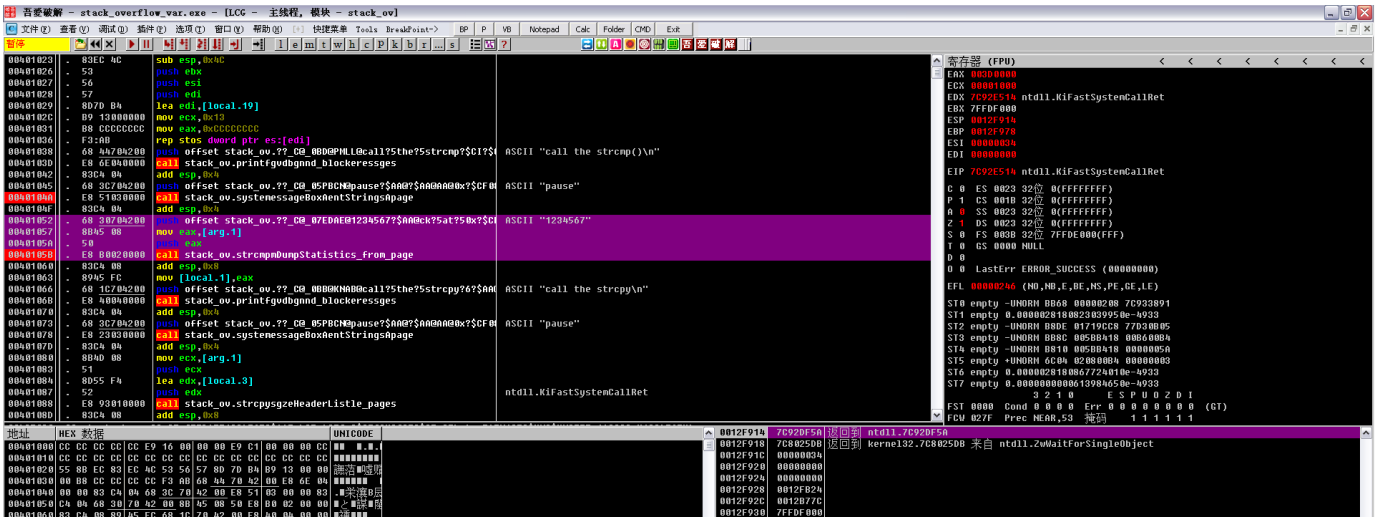


这是verify_password()函数的领空，这是刚才的pause

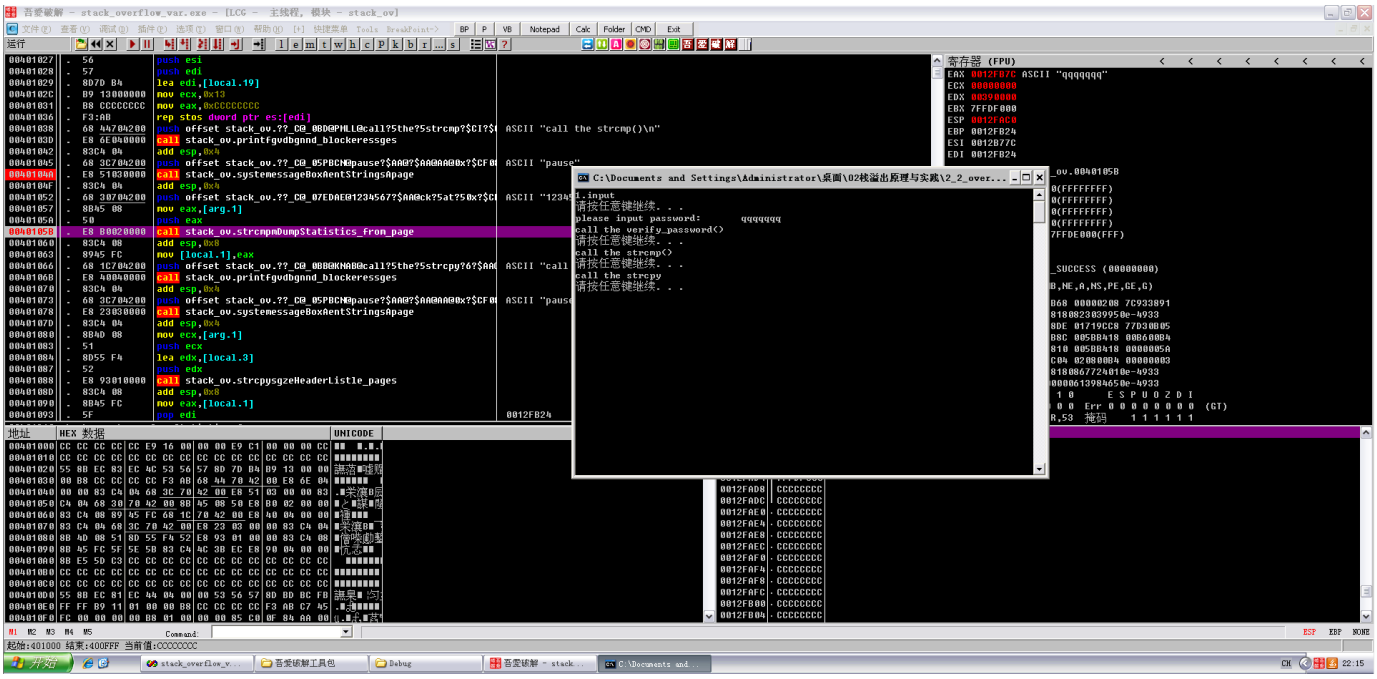


往下看，这个是调用strcmp的压栈操作，C语言代码：

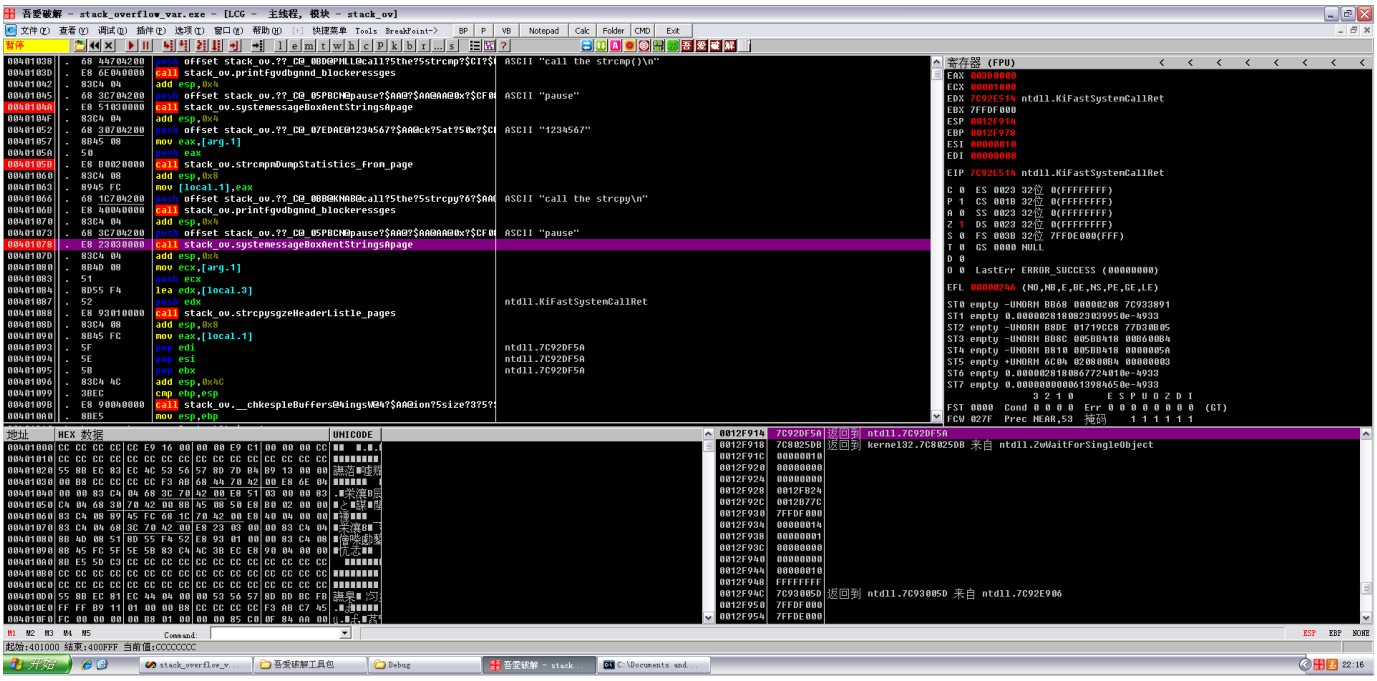
```
authenticated = strcmp(password, PASSWORD);
```



回车走下去，碰到pause，依旧暂停找调用

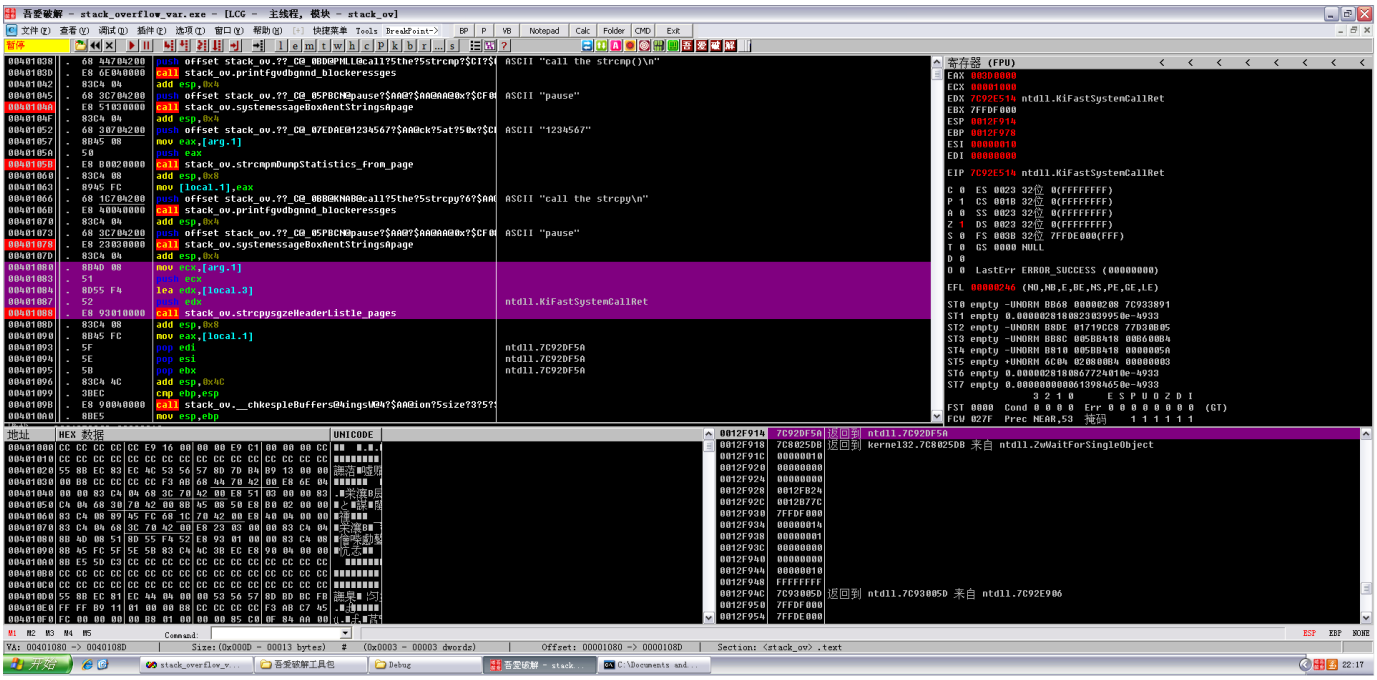


调用pause的位置

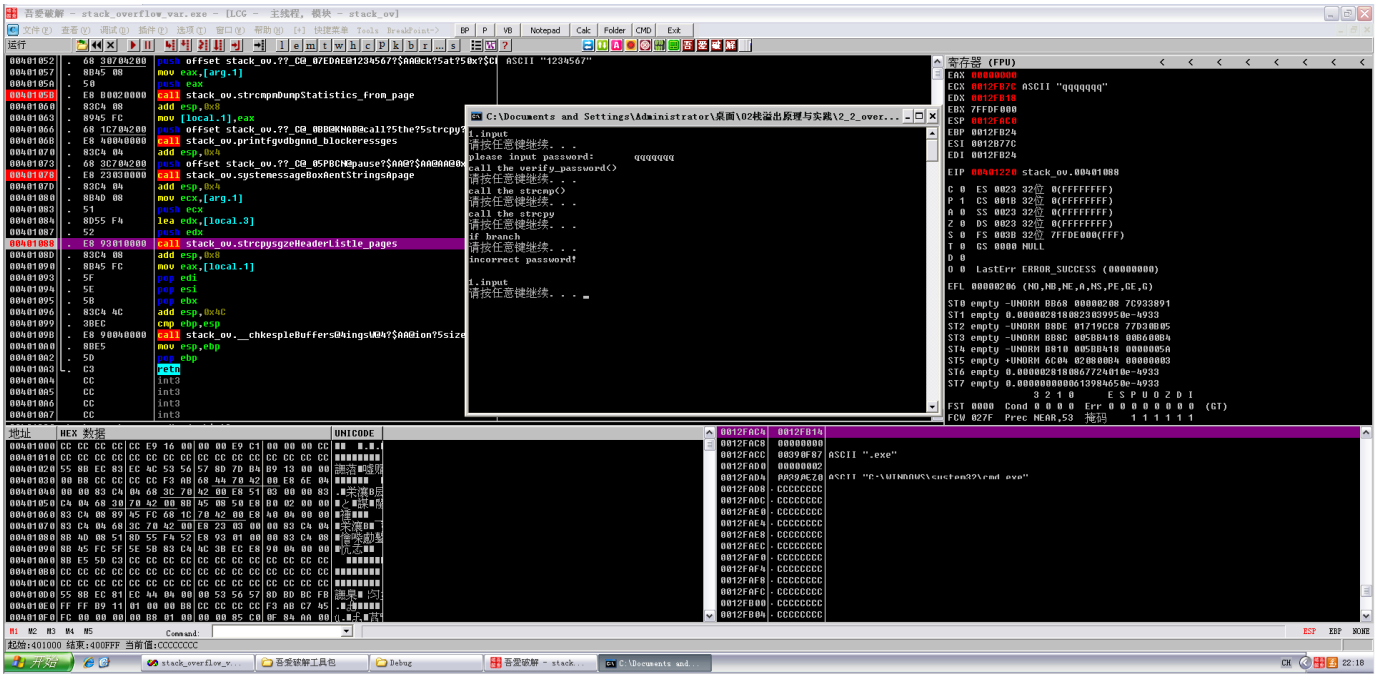


往下拉，这是strcpy，C语言代码：

```
strcpy(buffer, password);
```



然后再回车其实就没了

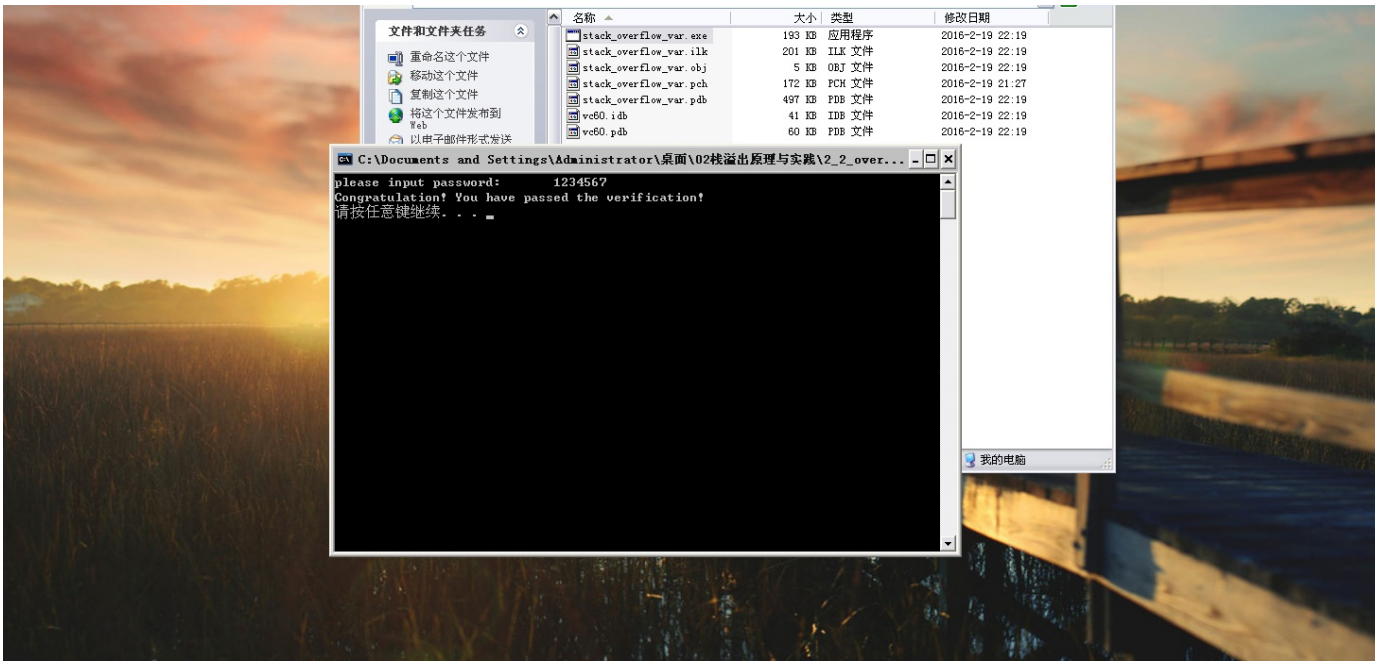


熟悉了一遍整个程序运行的过程之后，重新修改代码，去掉不需要的pause

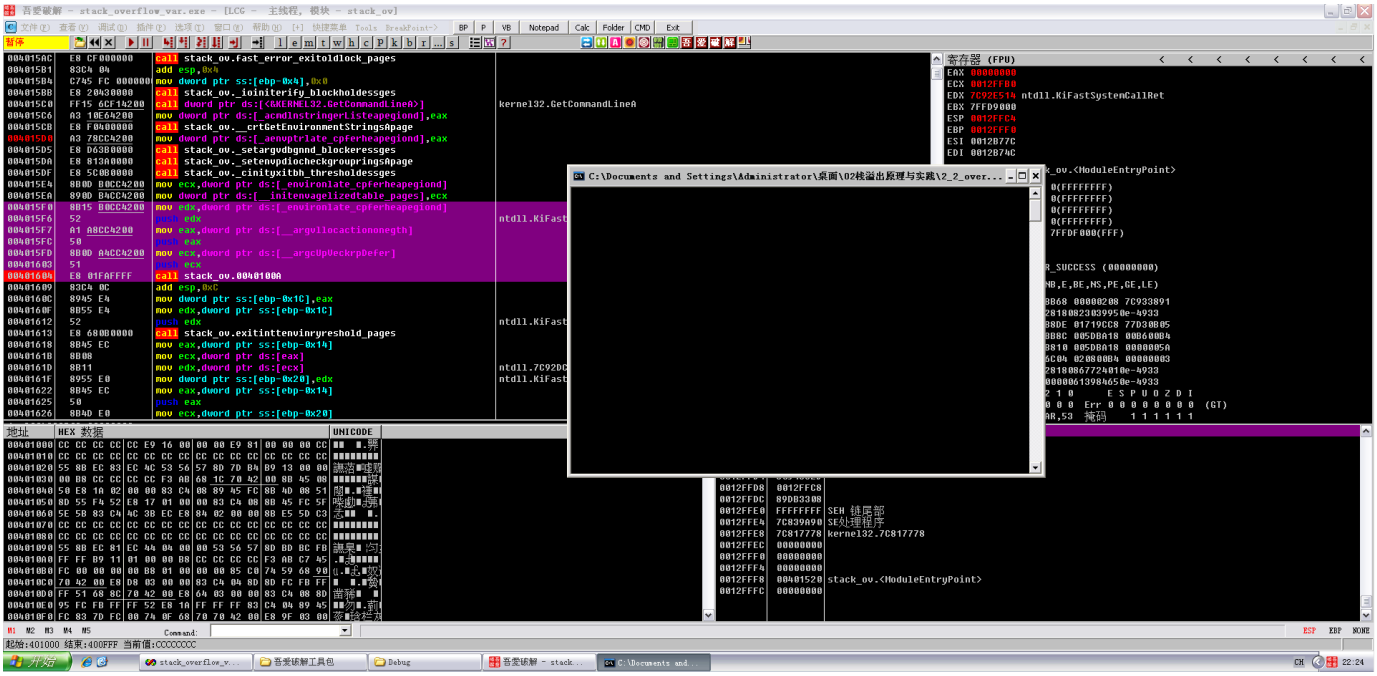
```
#include <stdio.h>

#define PASSWORD "1234567"

int verify_password (char *password)
{
    int authenticated;
    char buffer[8];// add local buff
    authenticated = strcmp(password, PASSWORD);
    strcpy(buffer, password);//over flowed here!
    return authenticated;
}

int main(int argc, char **argv, char **envp)
{
    int valid_flag = 0;
    char password[1024];
    while(1)
    {
```

```
        printf("please input password:       ");
        scanf("%s", password);
        valid_flag = verify_password(password);
        if(valid_flag)
        {
            printf("incorrect password!\n\n");
        }
        else
        {
            printf("Congratulation! You have passed the verification!\n");
            break;
        }
    }
    system("pause");
    return 0;
}
```

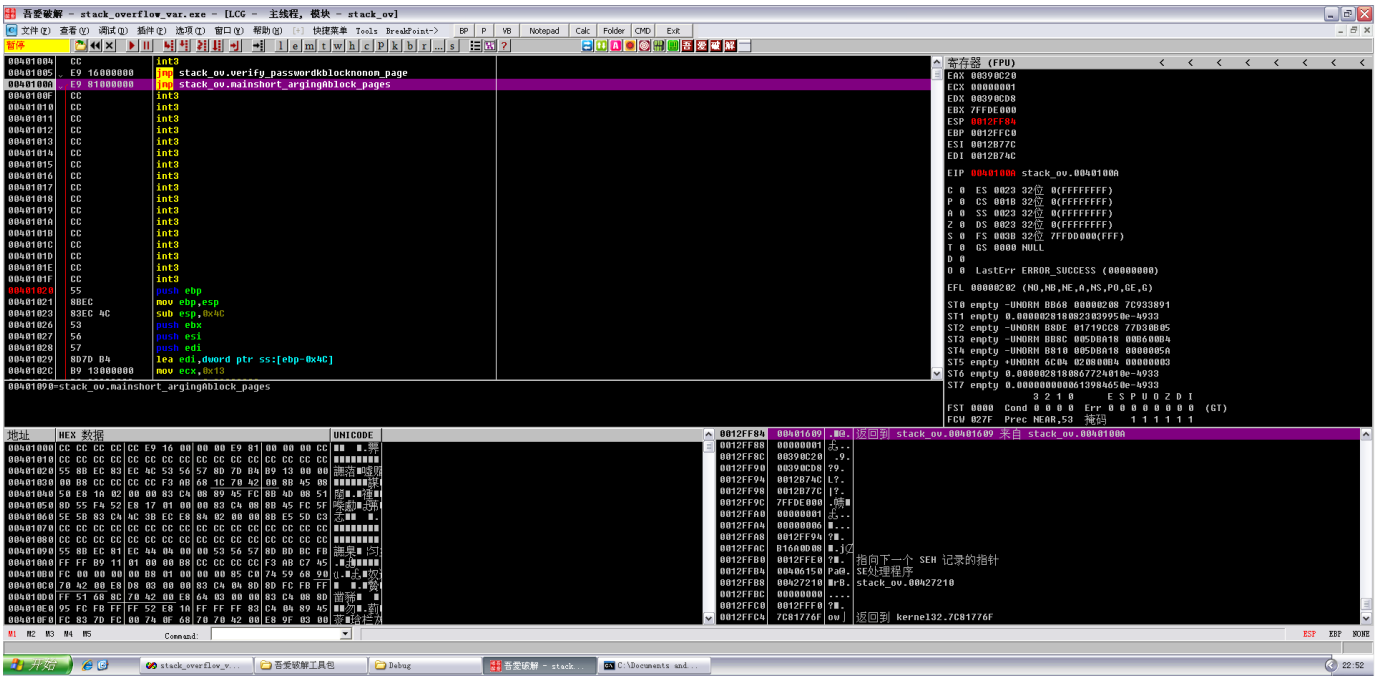
编译后运行



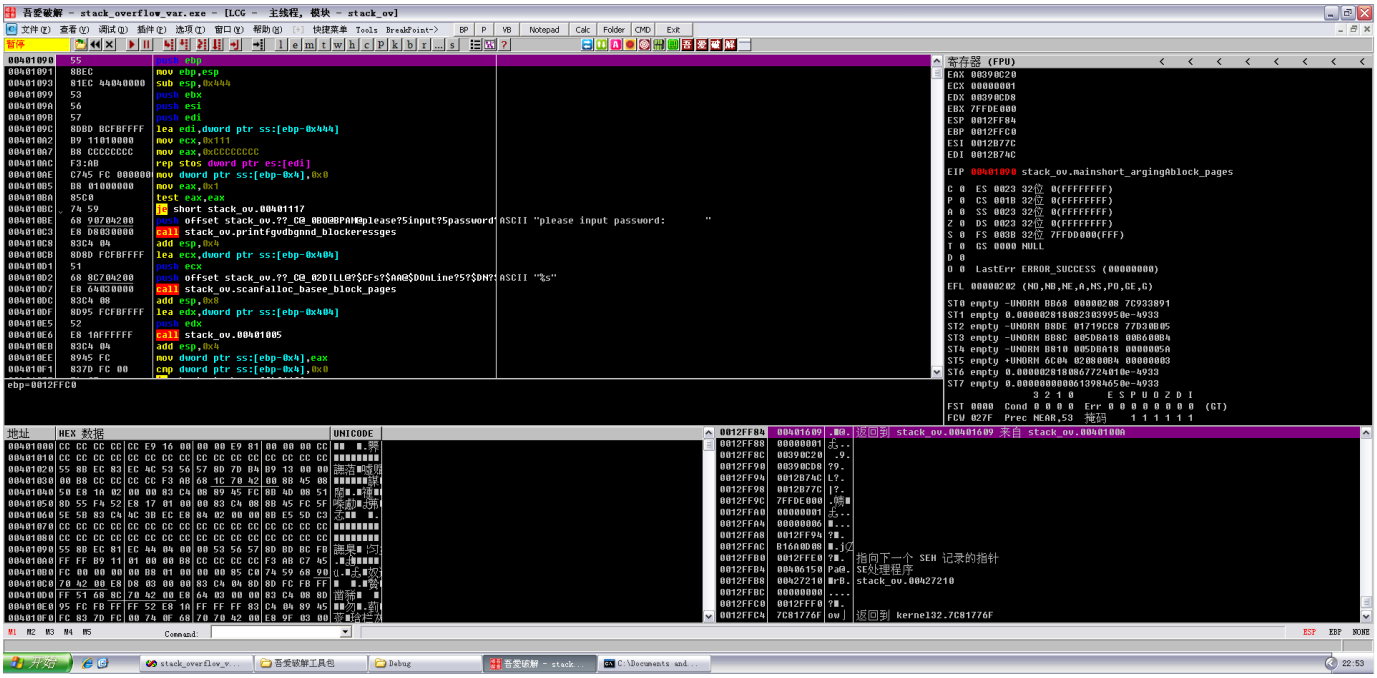
这是主函数的入口，下断，然后重新载入



F9运行到断点，F7跟入，有一个跳转，继续单步走

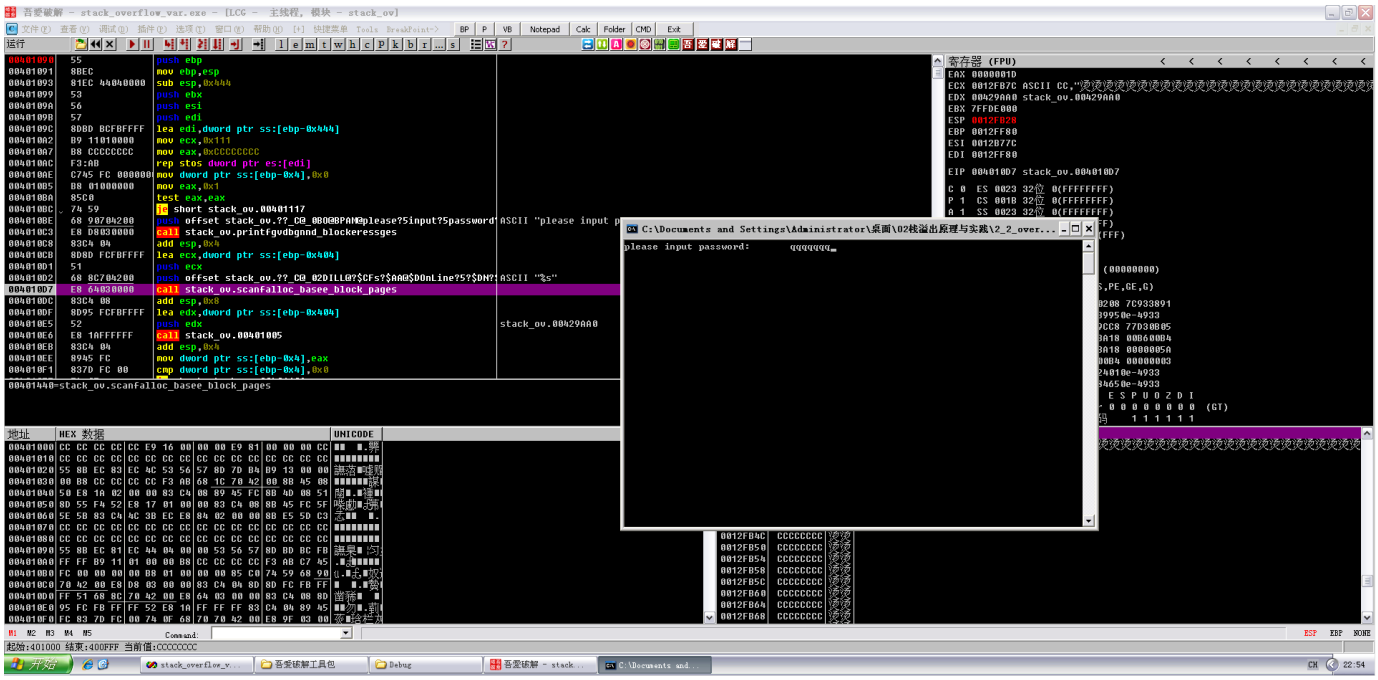


这是main函数的领空

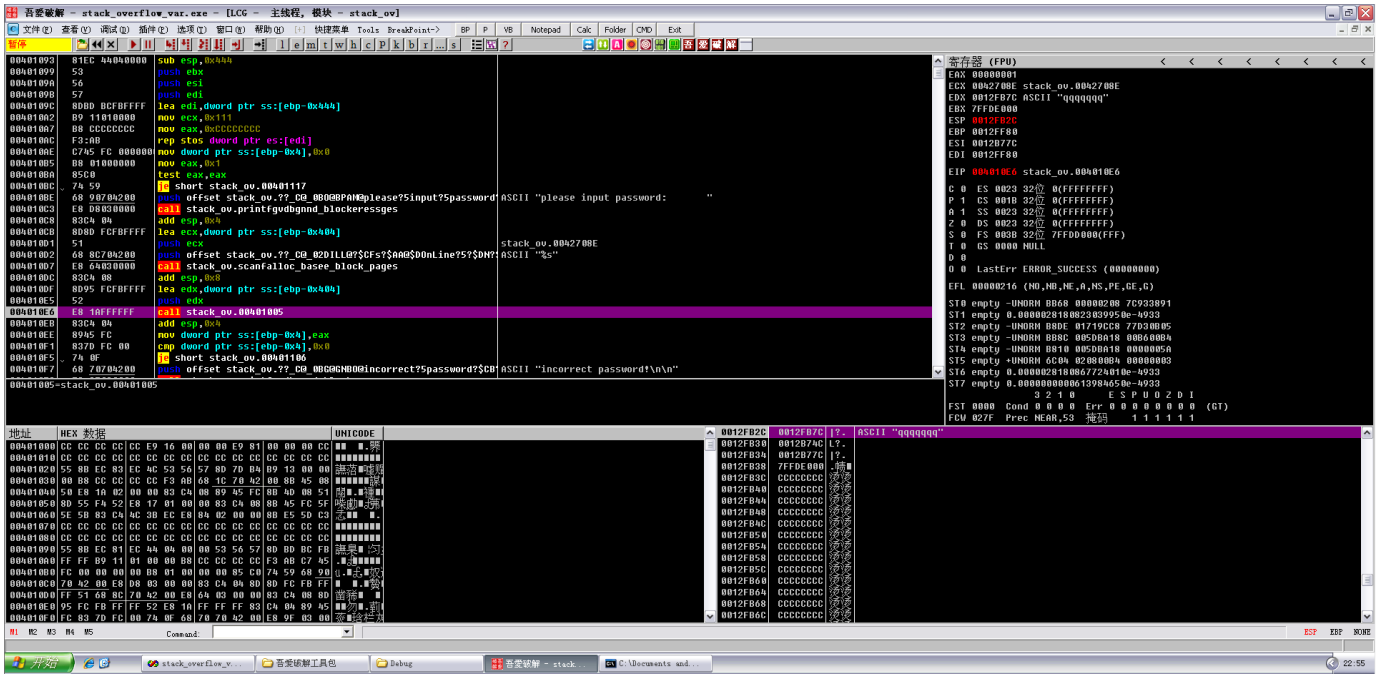


这是调用scanf函数

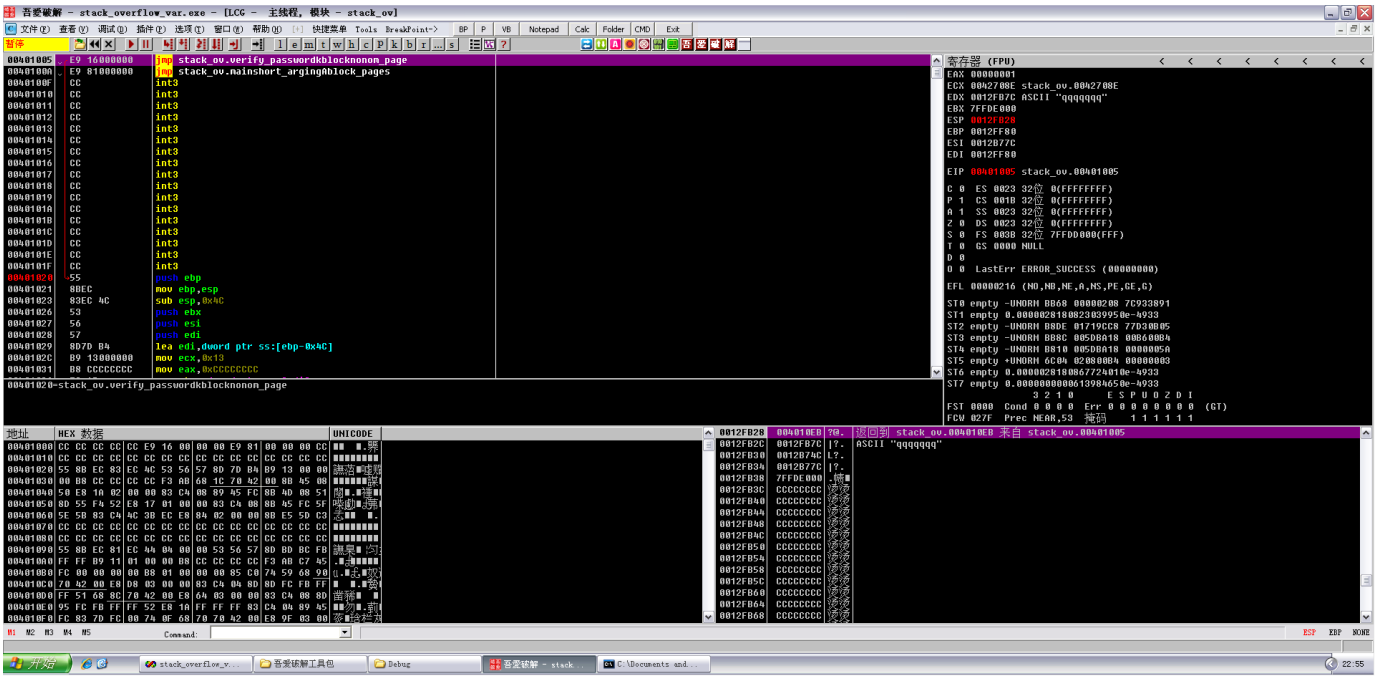
```
scanf("%s", password);
```



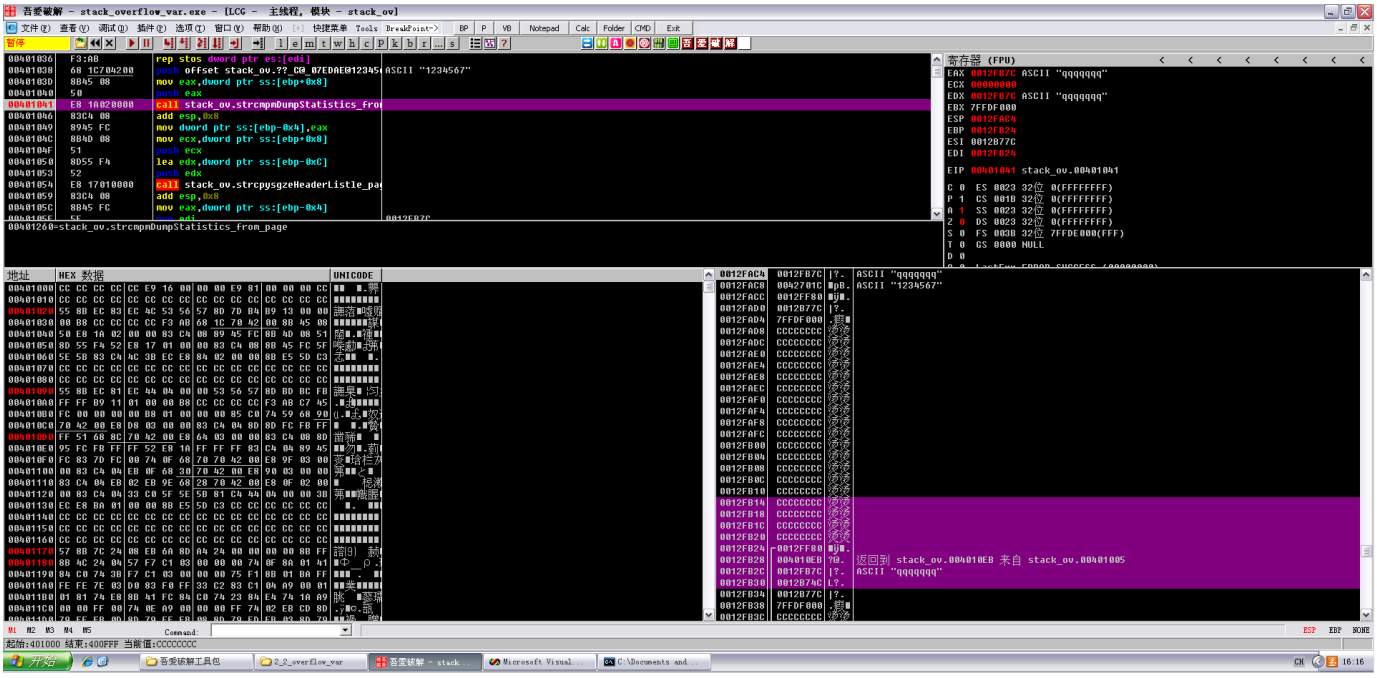
F8单步走下来，开始调用verify_password()，F7单步跟入



碰到跳转，继续单步

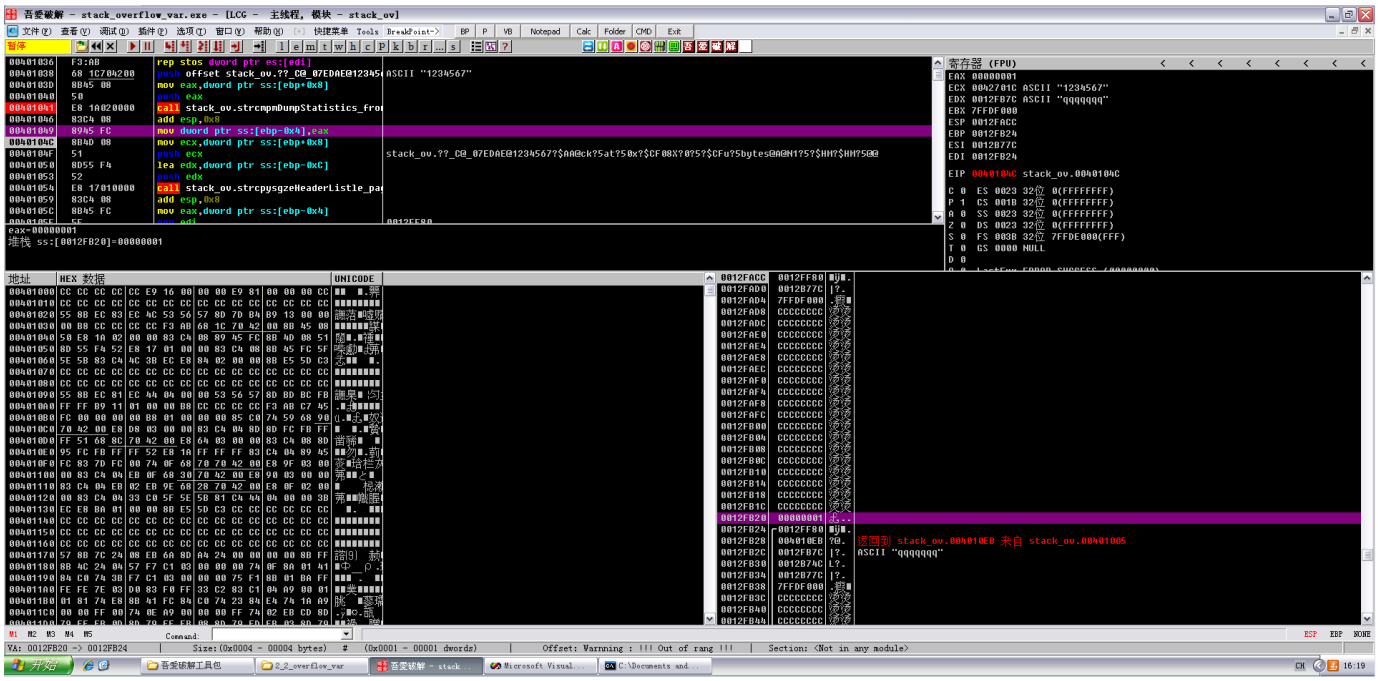


单步走下来注意看栈里的变化



单步走到这，这是将strcmp比较的结果赋值给authenticated变量

同时注意栈里的变化

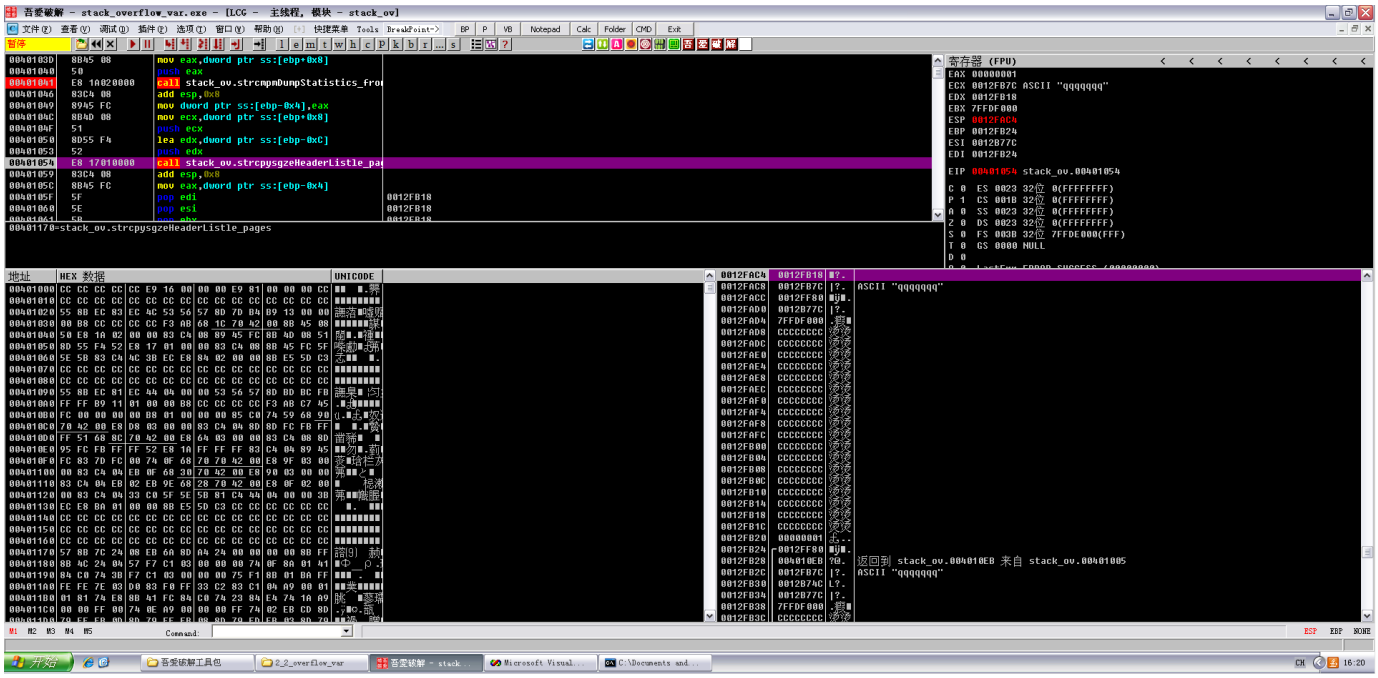


```
0012FB14   CCCCCCCC  烫烫
0012FB18   CCCCCCCC  烫烫
0012FB1C   CCCCCCCC  烫烫
0012FB20   CCCCCCCC  烫烫
0012FB24  /0012FF80  €? .
0012FB28  |004010EB  ?@.   返回到 stack_ov.004010EB 来自 stack_ov.00401005
0012FB2C  |0012FB7C  |?.   ASCII "qqqqqqq"
0012FB30  |0012B74C  L?.
```

变成了

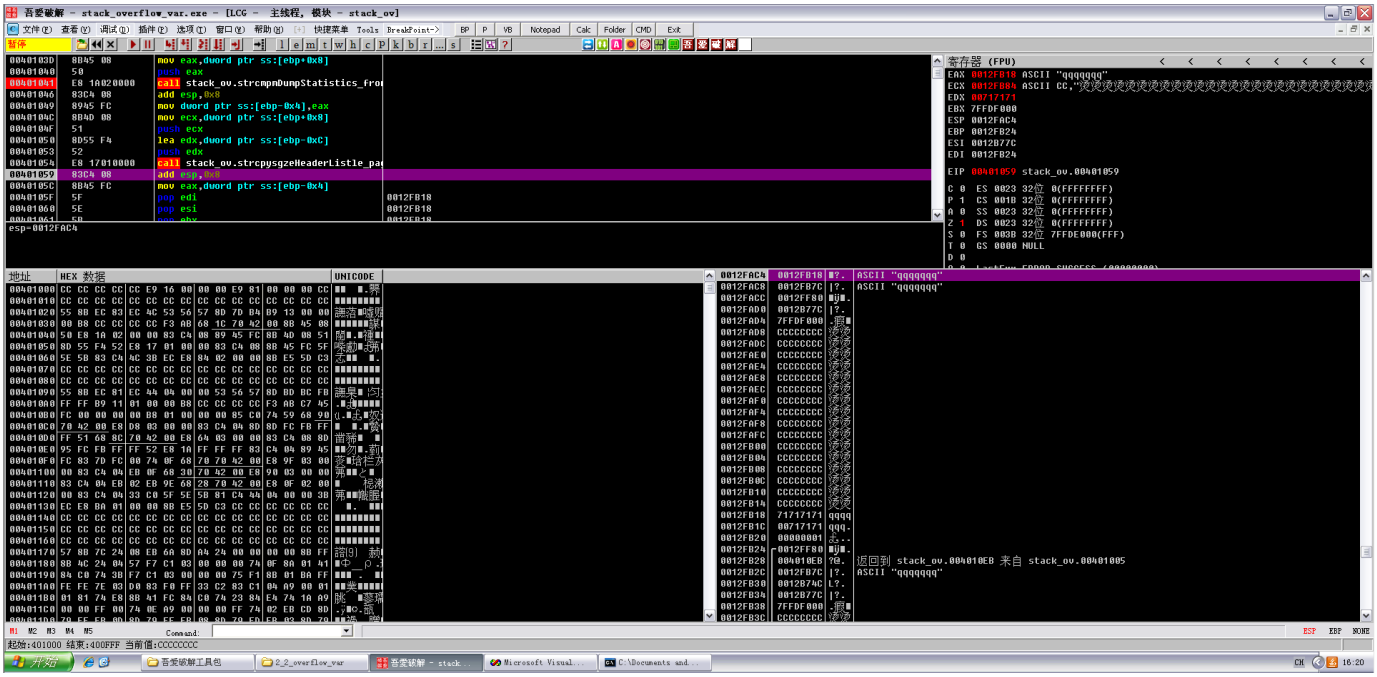
```
0012FB14   CCCCCCCC  烫烫
0012FB18   CCCCCCCC  烫烫
0012FB1C   CCCCCCCC  烫烫
0012FB20   00000001    ...
0012FB24  /0012FF80  €? .
0012FB28  |004010EB  ?@.   返回到 stack_ov.004010EB 来自 stack_ov.00401005
0012FB2C  |0012FB7C  |?.   ASCII "qqqqqqq"
0012FB30  |0012B74C  L?.
```

然后就是调用strcpy进行复制，依旧要注意栈的变化



栈变化

```
0012FB14   CCCCCCCC  烫烫
0012FB18   71717171  qqqq
0012FB1C   00717171  qqq.
0012FB20   00000001    ...
0012FB24  /0012FF80  €□ .
0012FB28  |004010EB  ?@.   返回到 stack_ov.004010EB 来自 stack_ov.00401005
0012FB2C  |0012FB7C  |?.   ASCII "qqqqqqq"
0012FB30  |0012B74C  L?.
```

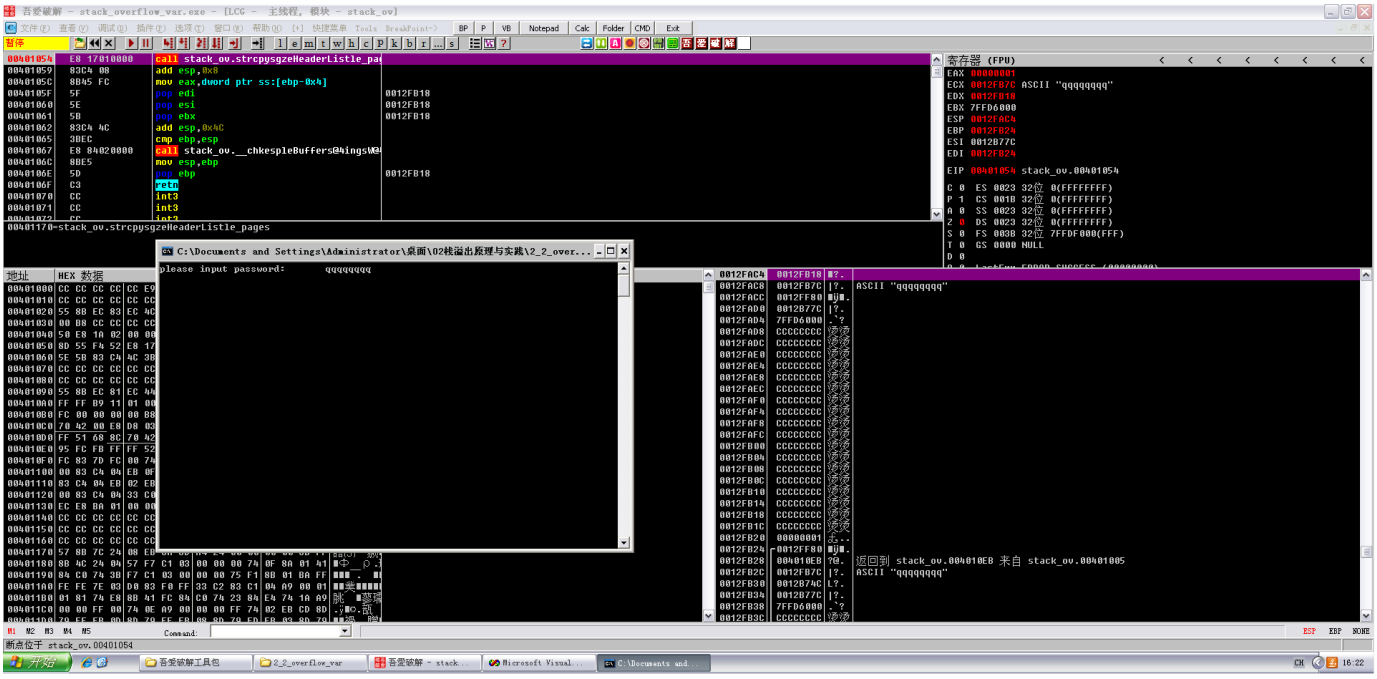


好了，到这里大概就清楚了流程，可以看到authenticated变量在内存中的位置刚好在buffer数组下面，如果buffer数组过长就会覆盖authenticated变量的值

取消其它所有断点，只在

```
00401054    E8 17010000     call stack_ov.strcpysgzeHeaderListle_pag>
```

下一个断点，重新运行起来，输入8个q



回车，注意栈

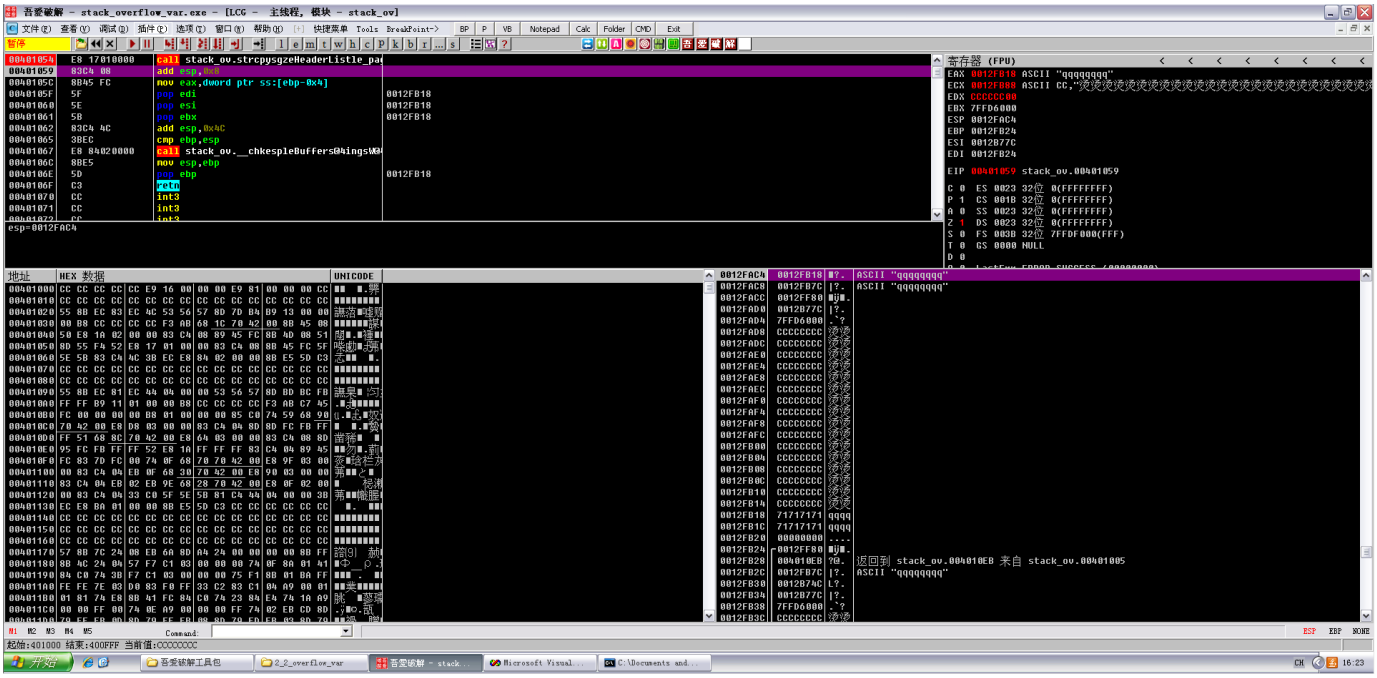
```
0012FB14   CCCCCCCC  烫烫
0012FB18   CCCCCCCC  烫烫
0012FB1C   CCCCCCCC  烫烫
0012FB20   00000001   ...
0012FB24  /0012FF80  €? .
0012FB28  |004010EB  ?@.   返回到 stack_ov.004010EB 来自 stack_ov.00401005
0012FB2C  |0012FB7C  |?.   ASCII "qqqqqqqq"
0012FB30  |0012B74C  L?.
```

F8单步走，可以看到栈里的变化

```
0012FB14   CCCCCCCC  烫烫
0012FB18   71717171  qqqq
0012FB1C   71717171  qqqq
0012FB20   00000000  ....
0012FB24  /0012FF80  €? .
0012FB28  |004010EB  ?@.   返回到 stack_ov.004010EB 来自 stack_ov.00401005
0012FB2C  |0012FB7C  |?.   ASCII "qqqqqqqq"
0012FB30  |0012B74C  L?.
```

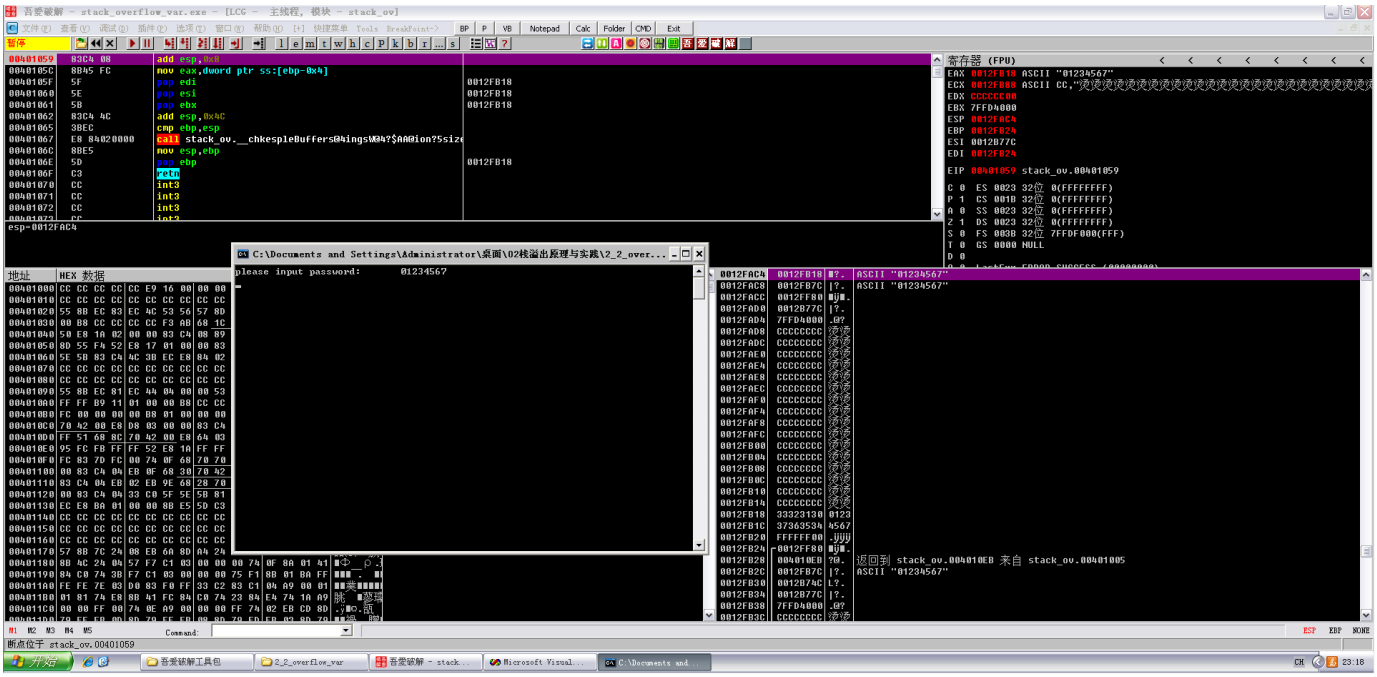
已经覆盖掉了authenticated变量，返回的结果为0，因为字符串会在最后面加上一个结束符

而buffer定义的时候只有8字节，所以8个q加上结束符，结束符就会覆盖authenticated变量在内存中的位置

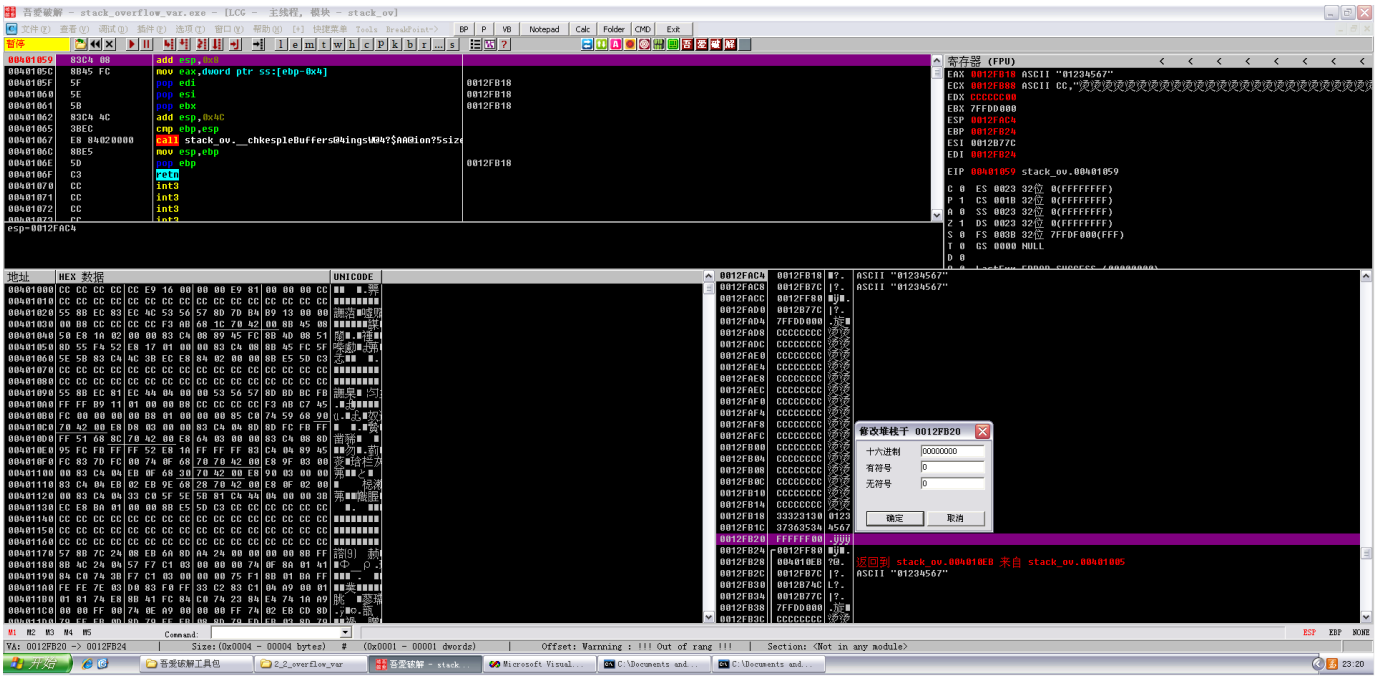


至于最后留下的关于输入比"1234567"小的字符串的问题，挺有意思的

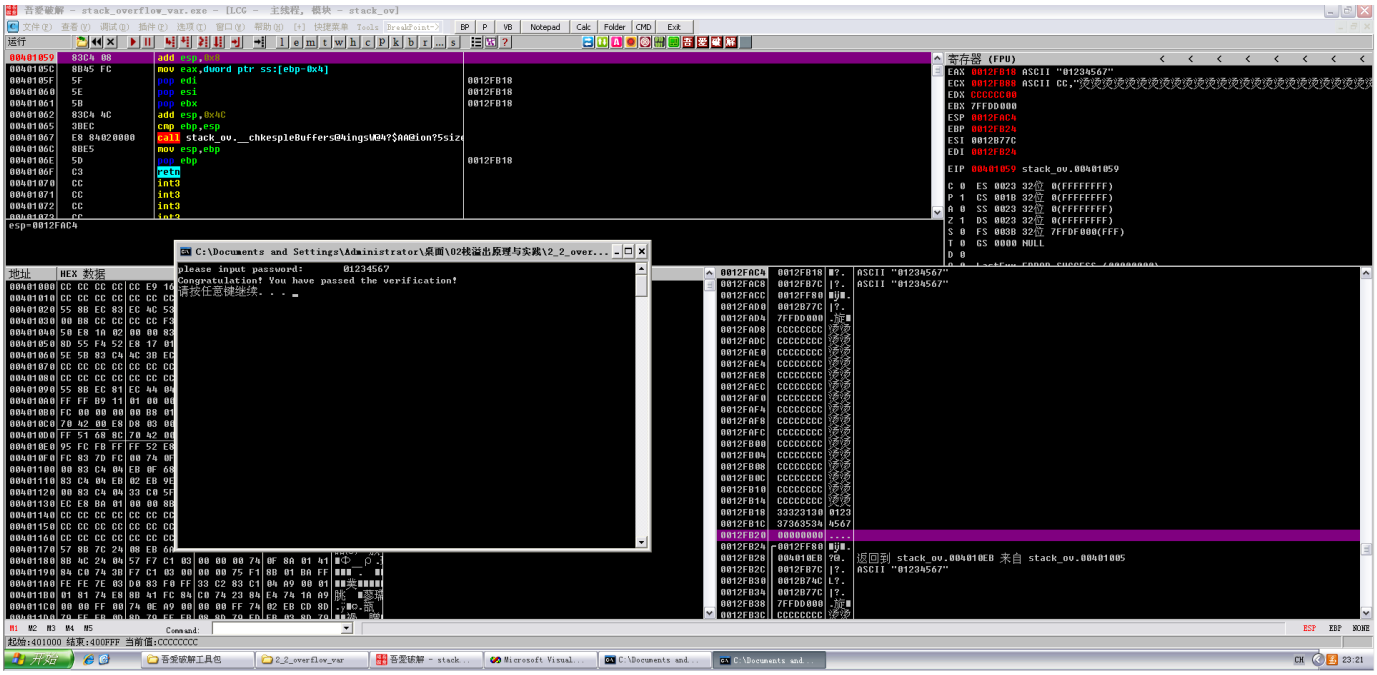
先在调用strcpy的地方下断点，重新载入，F9运行，输入"01234567"，观察栈的变化



如图右击，修改堆栈值，F9继续运行



妥了


C:\Documents and Settings\Administrator\桌面\02栈溢出原理与实践\2_2_over...
please input password:	01234567
Congratulation! You have passed the verification!
请按任意键继续. . .