第一次作业

chatgpt简化后的比较key1的逻辑

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main() {
    char key1[12];
    char key2[12];

    // 初始化v3
    char v3[11] = {0};
    char v7[12] = {0};

    // 对v3进行一系列操作
    for (int i = 0; i < 11; ++i) {
        v3[i] = v3[i] - 1;
    }

    // 对v7进行一系列操作
    int v8 = 0;
    int v9 = v3 - v7;
    do {
        v7[v8] = v8 * v8 + v7[v8 + v9] * v7[v8 + v9];
        ++v8;
    } while (v8 < 11);

    // 获取用户输入的key1
    printf("please input key1: ");
    scanf("%11s", key1);
```

```
    // 对用户输入的key1进行一系列操作
    for (int i = 0; i < 11; ++i) {
        key1[i] = i * i + key1[i] * key1[i];
    }
    key1[11] = 0;

    // 比较v7和key1
    if (strcmp(v7, key1) == 0) {
        // 进行相应操作
    }

    return 0;
}
```

观察逻辑可知v3和key1相等。

gpt的简化有一点小问题，ida中可以看到v3来自于0x40fa58（xipbsfzpv@）所有字符减1，也就是：`whoareyou?`

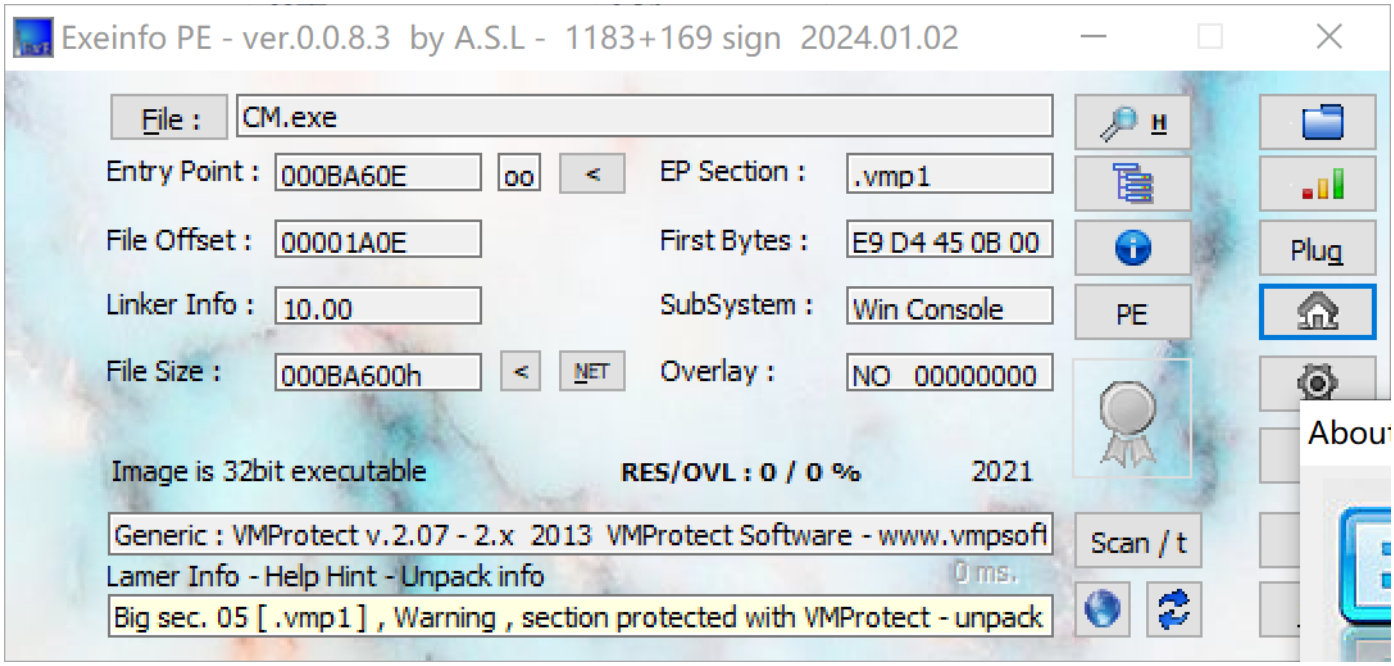
```
def decode(a1, a2):
    v6 = bytearray(11)

    if a2 == 1:
        v6 = bytes([byte ^ 0x77 for byte in a1])
    elif a2 == 2:
        v6 = bytes([(byte >> 4) + ((byte & 0xF) << 4) for byte in a1])
    else:
        v6 = bytes([byte - 1 for byte in a1])

    return v6
```

gpt给出的DLLU.dll中的decode算法

解出`b'iamteacher'`和`b"don'tjoke!"`

第二次作业

**带有vmp**

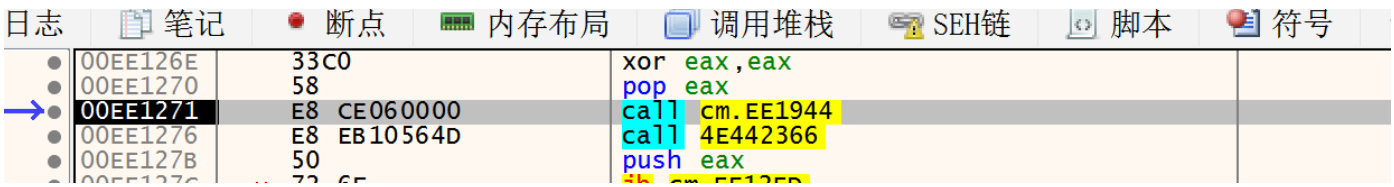



vmp2.07，但其实关键逻辑没有v（异或和memcmp）。

**调试检测**

程序会检测自己是否在调试，这里直接过掉相关检测逻辑后attach上去。

**大量花指令**

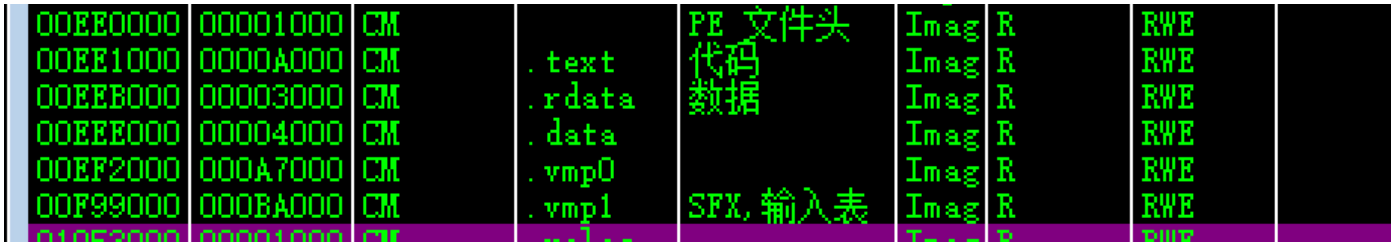



花指令是没有意义的，这里没有恢复，所有的花直接单步过去。

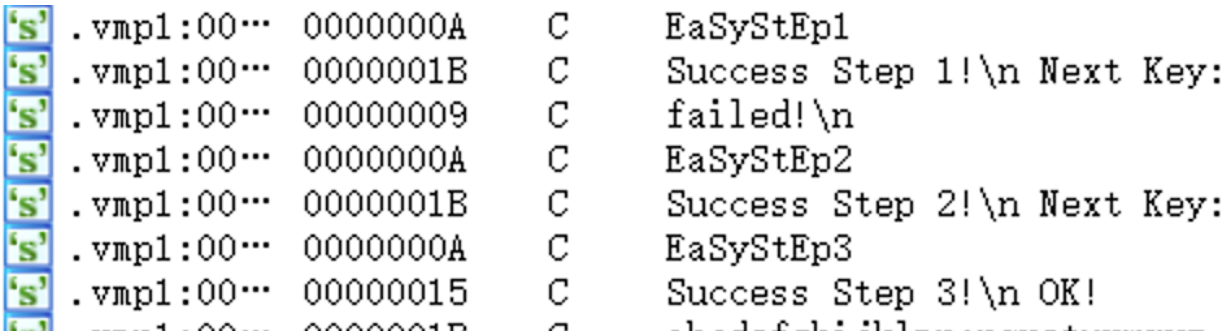
**CRC检测**

程序应该存在使用CRC检测自己代码是否被修改，x32dbg中直接下软件断点不可用，必须使用硬件断点。

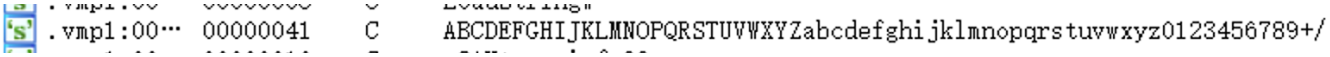
在olldbg中通过FVMP插件找到OEP，此时程序的各个段是解密的并且程序刚开始，我们将每个段dump出来。



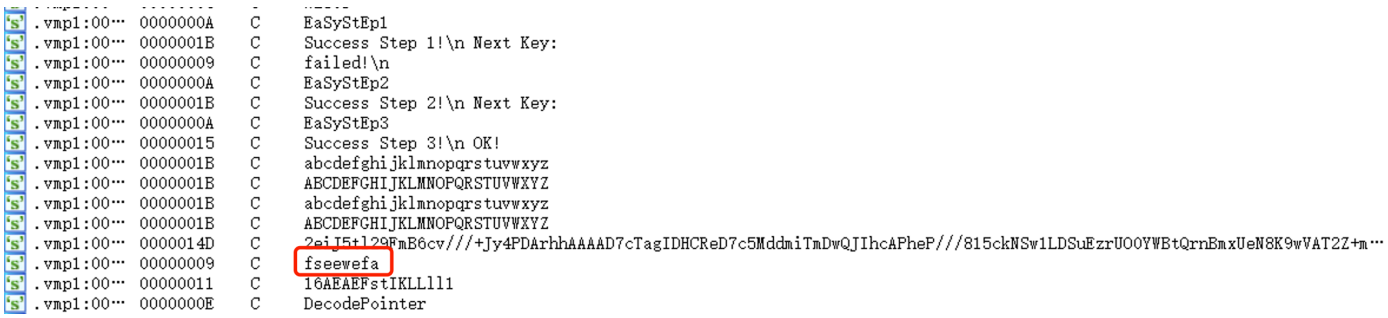

把所有的dump文件合并起来，放到ida中观察字符串。




输入`EaSyStEp1`可以过第一步，但是EaSyStEp2过不了第二步。




看到了base64表，将EaSyStEp2进行base64编码，`RWFTeVN0RXAy`可以过第二步。

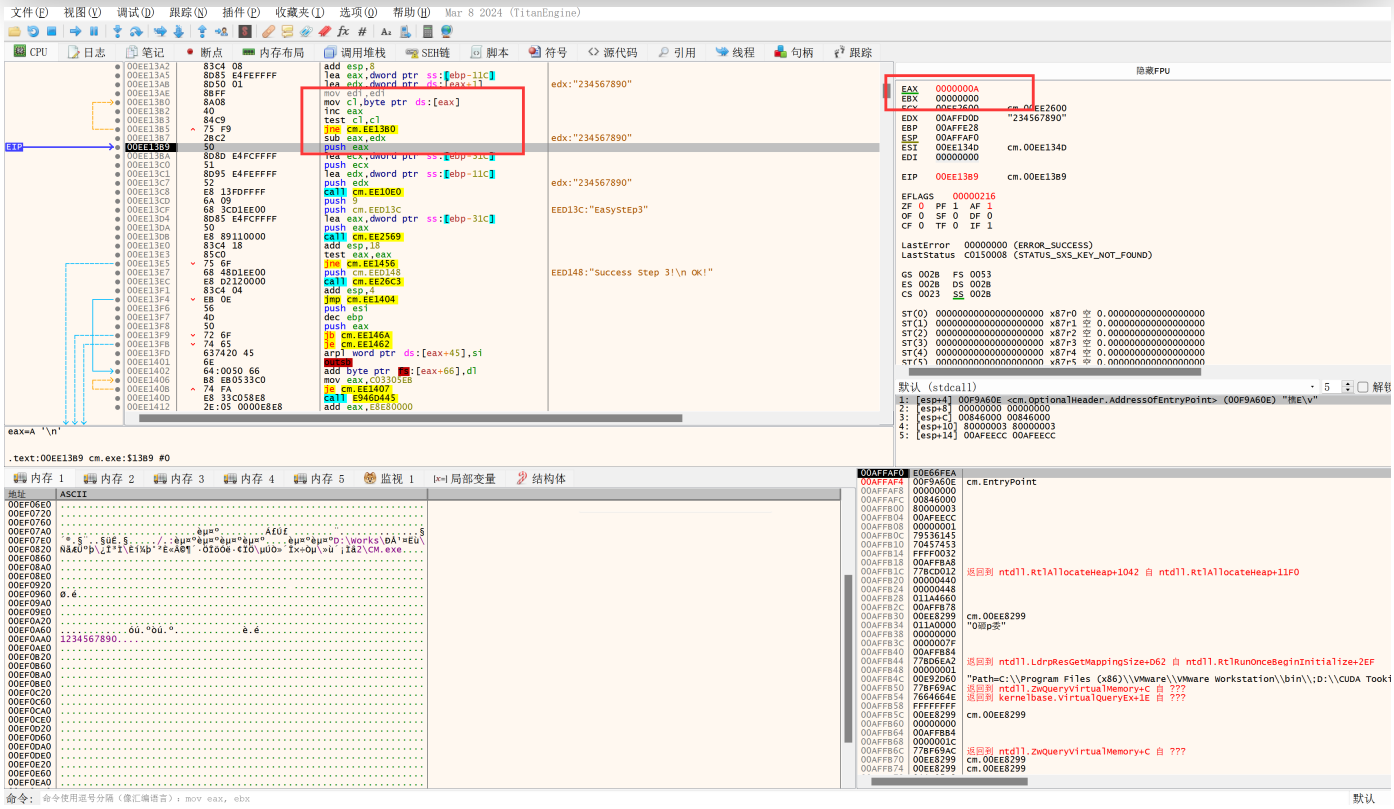



在`Success Step 3!\n`附近有一些关键字符串，其中`fseewefa`与`EaSyStEp3`的长度接近。

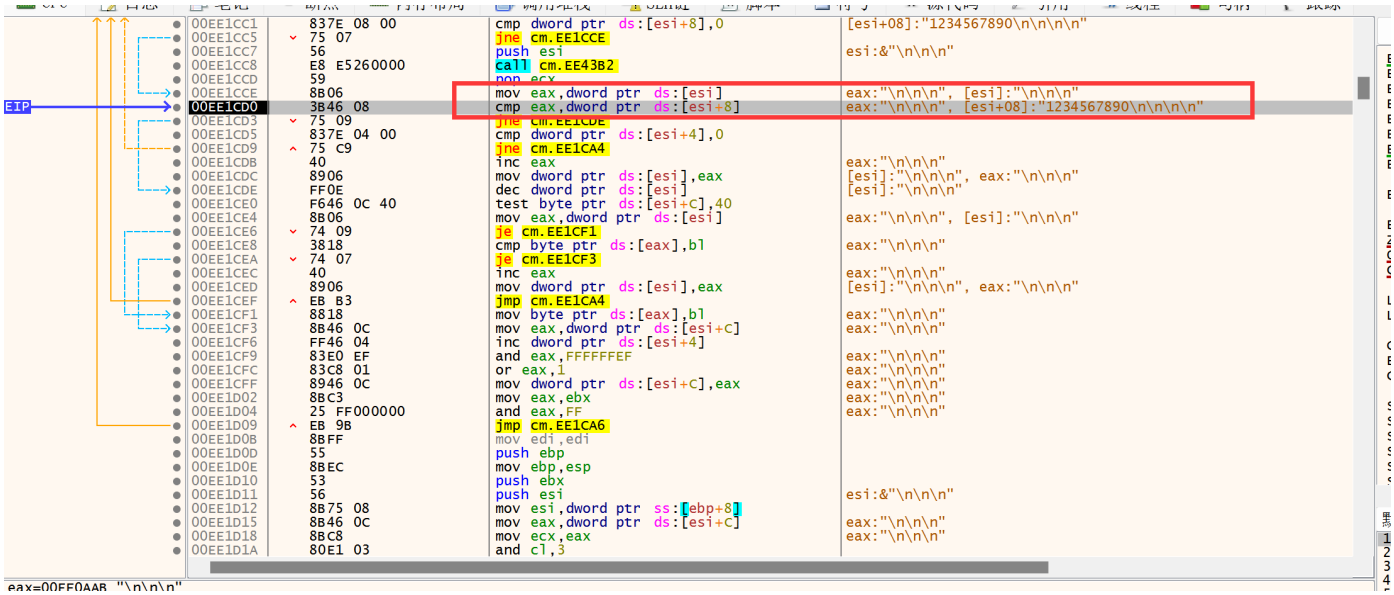
怀疑是循环异或，但是异或的输入有不可见字符，只能调试去看具体的逻辑了。

输入：

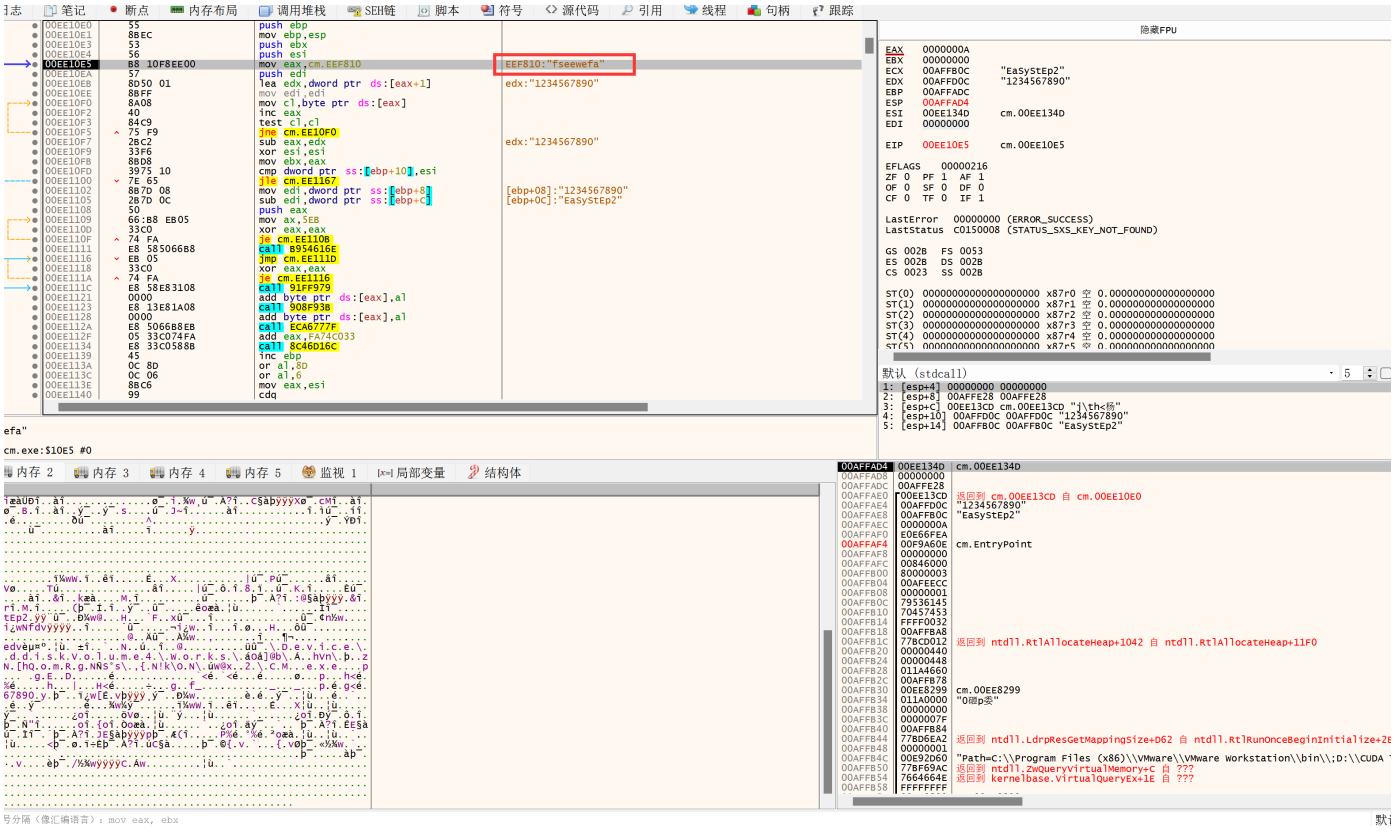
```
EaSyStEp1
RWFTeVN0RXAy
1234567890
```



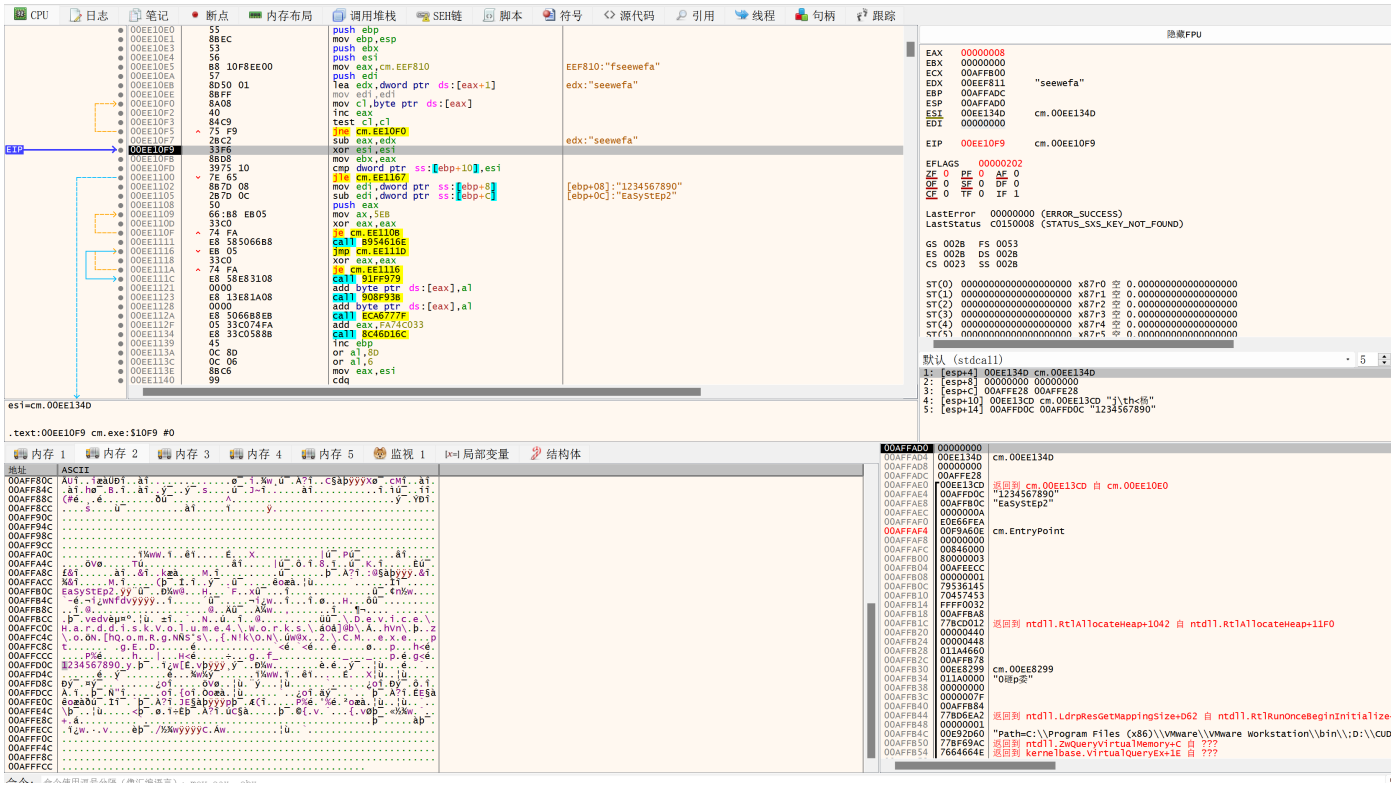
eax是key3的长度。



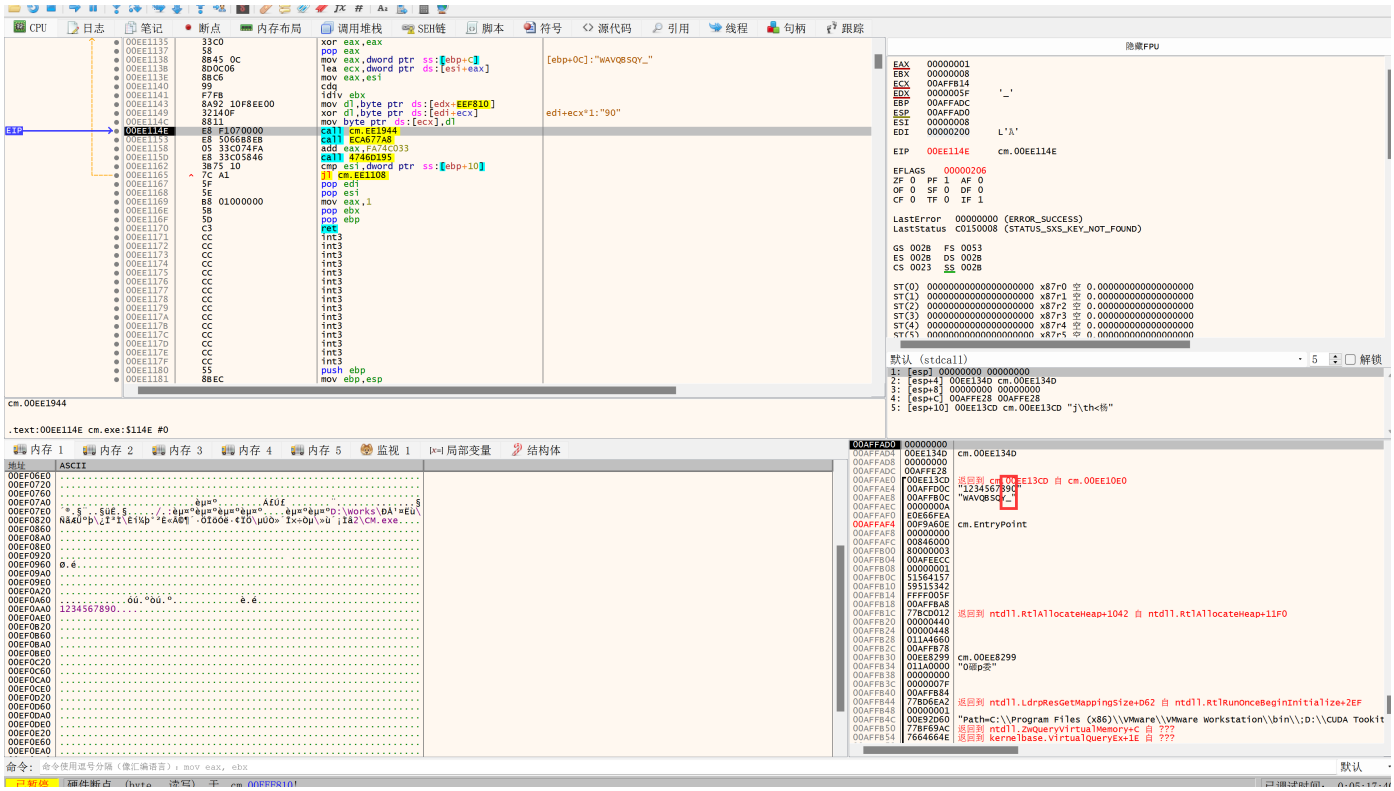
esi+8是输入的`1234567890`这里是判断key3字符串的长度。



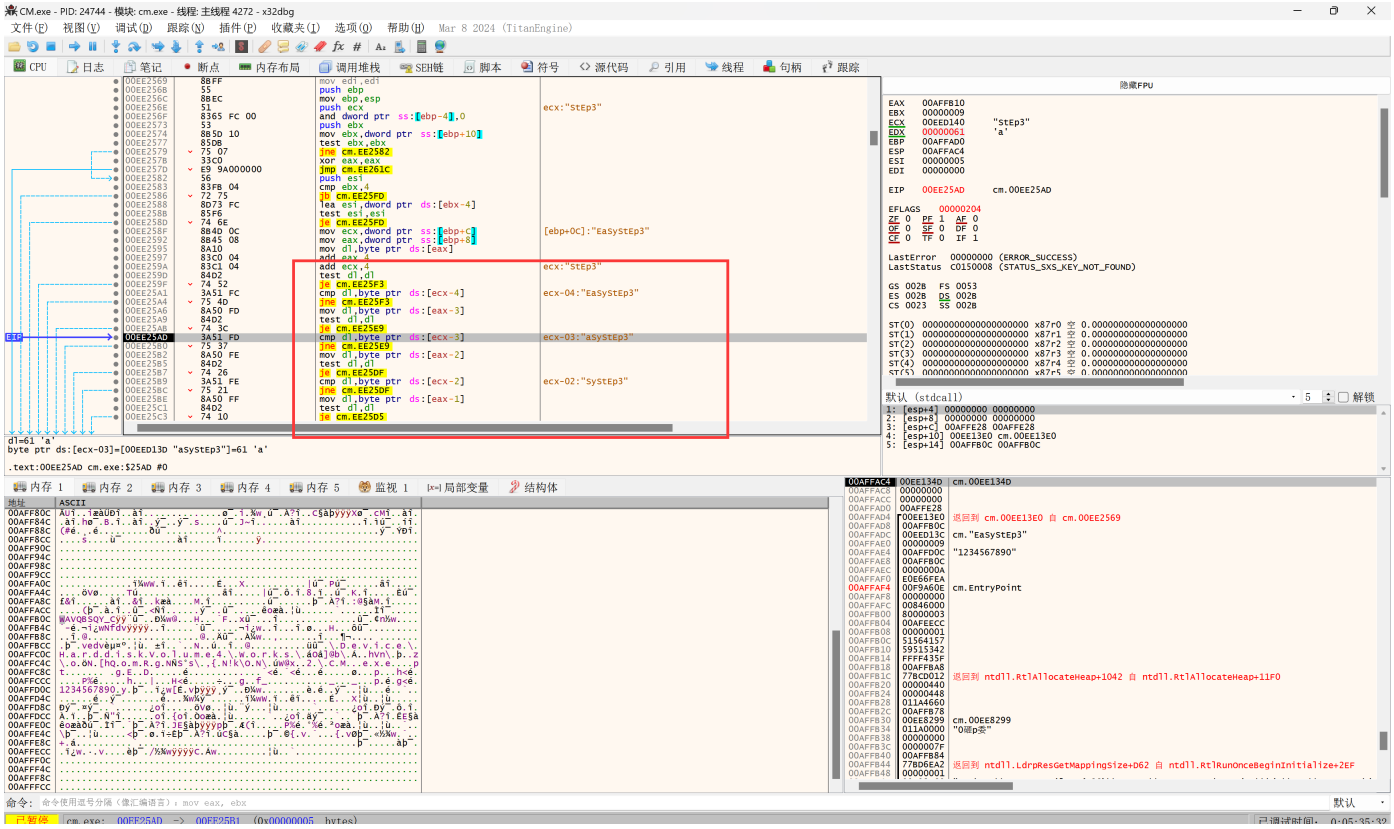
到这里用上了fseewefa。



用 `fseewefa` 与 `1234567890` 进行循环异或。



将key3和异或结果以及key3的长度压入栈中，并调用一个关键函数（猜测为memcmp）。



这里将异或结果与`EaSyStEp3`进行比较，我们手动修改异或结果使得memcpy返回0，最后step3 success。

作业中给出的提交方式是在cmd中直接输入字符，但其实字符在包括不可见字符，在terminal中无法输入不可见字符，可以先将key写入到文本中，在记事本用UTF-8打开，复制到Windows的原始cmd中，即可实现作业中展示的输入方式。

```python
strings = ["EaSyStEp1", "RWFTeVN0RXAy", "#\x126\x1c$\x11#\x11U"]

# 将字符串写入文件
with open("exp.txt", "wb") as file:
    for string in strings:
        file.write(string + "\n")
```