每天一个技术点

(2023.7.15)软件加密与解密-番外1-PWN2REVERSE[XDbg]

本文作者：XDbg(小吧唧)
发布时间：2023年7月15日

内容概要：初学 PWN 后对 VMP Handler 的快速定位

1. 准备

加壳工具：VMP3.6

测试代码：

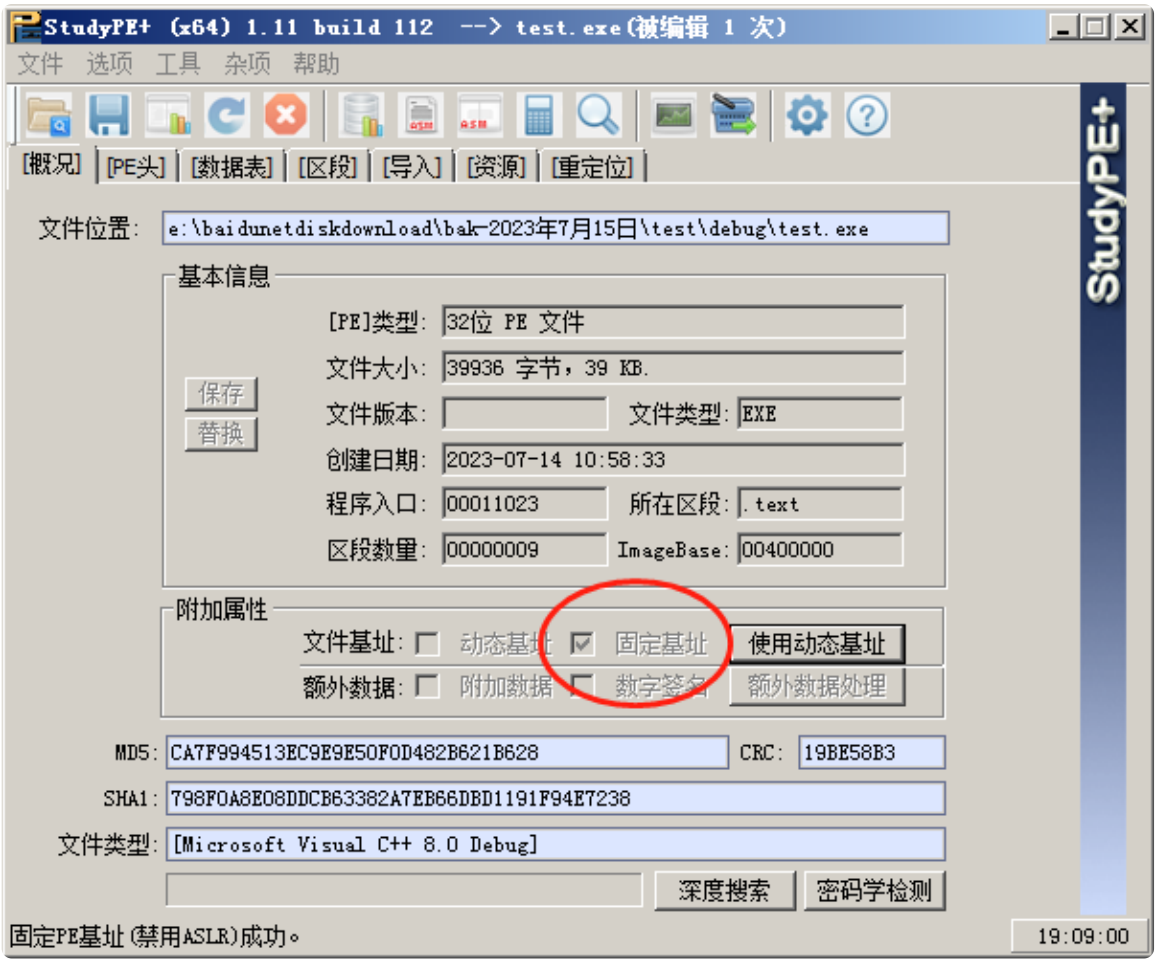
```
// test.cpp : 此文件包含 "main" 函数。程序执行将在此处开始并结束。
// 编译为 32 位程序
#include <iostream>

int main()
{
    // vmp mov handle
    int* p = 0x0;
    *p = 1;

    // vmp add handle
    *p = 1 + 2;

    // vmp sub handle
    *p = 9 - 9;
}
```

调试工具：OllyDbg_P.Y.G

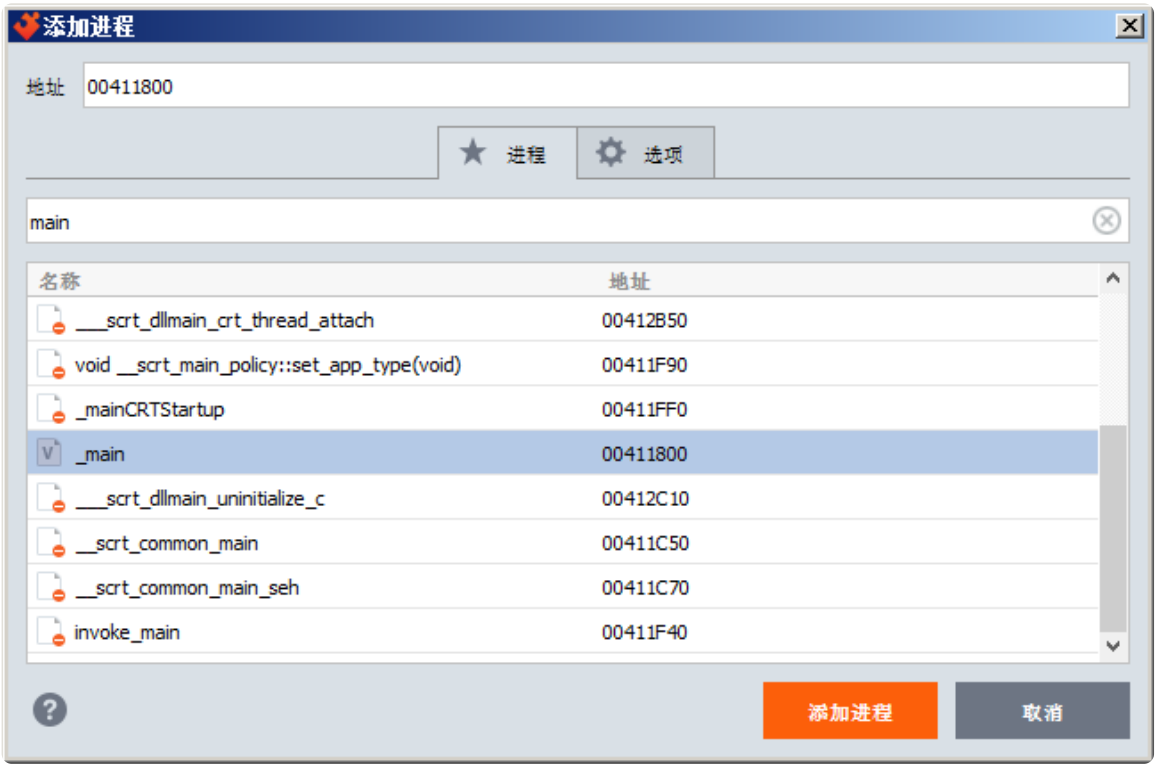
PE工具：StudyPE+ x64

2. 加壳

用 StudyPE 固定程序基址，然后保存，扔到 VMP 中。

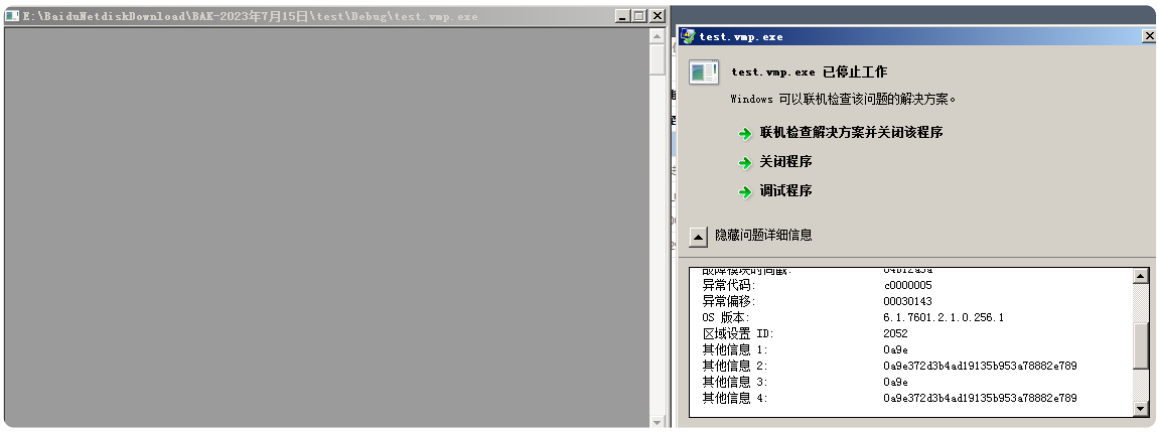


在 VMP 中添加 main 函数。



选项设置：超级

点击加壳就好了，运行程序测试，异常了，漂亮。

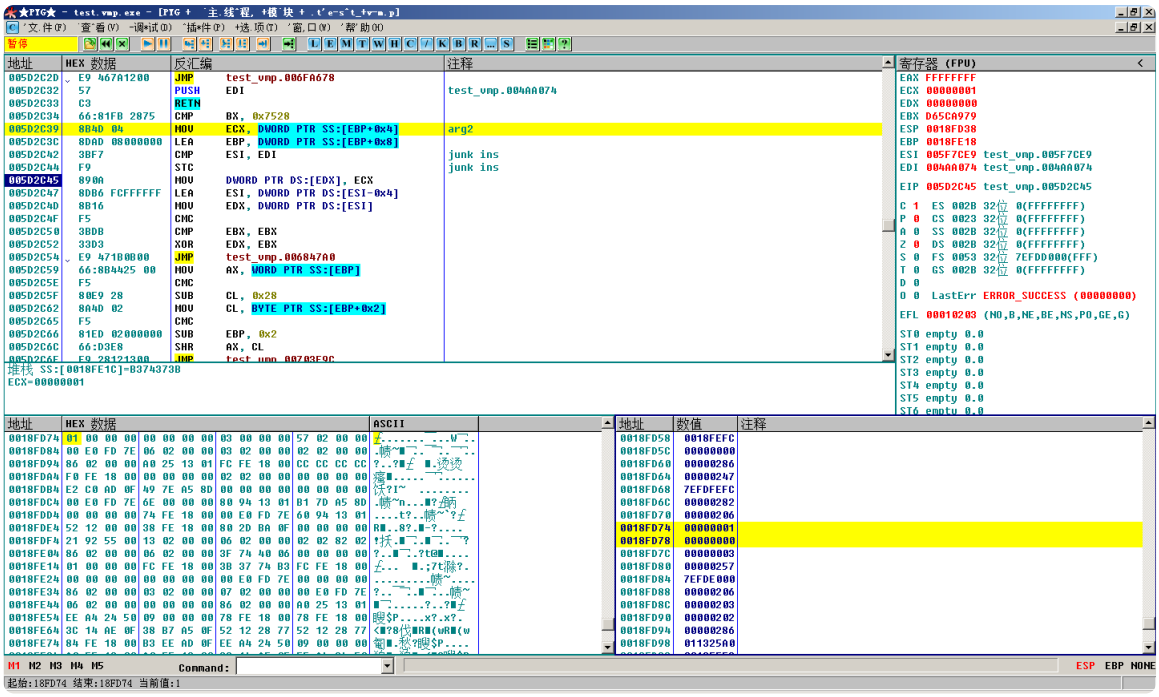


3. 调试

将 test.vmp.exe 扔到 OllyDbg_P.Y.G 里，然后跑起来。

mov32 handler

程序出现异常断下来了，可以看到 arg2 ，即我们的 *p = arg2 ，看到右下角的堆栈窗口，我们看到了 p 这个变量的指针（地址）



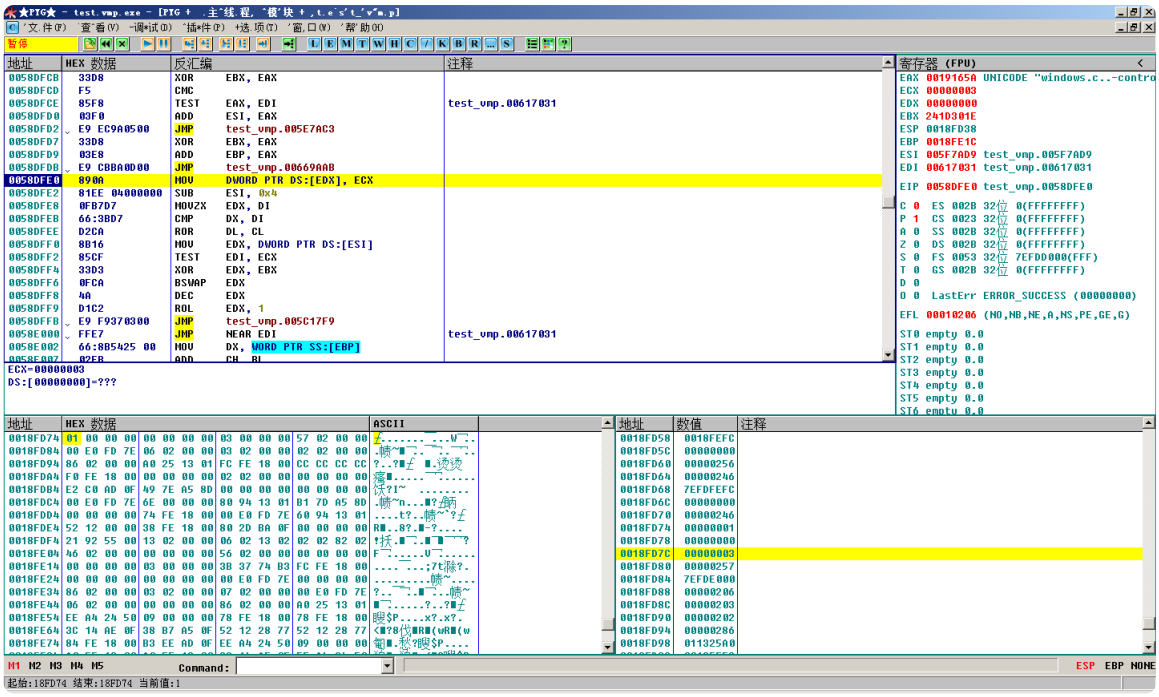
所以我们能够写出如下匹配指令模板：

```
MOV     EDX, DWORD PTR SS:[EBP]          ; arg1
MOV     ECX, DWORD PTR SS:[EBP+0x4]          ; arg2
MOV     DWORD PTR DS:[EDX], ECX
进一步概括为
MOV     REG, DWORD PTR SS:[EBP]          ; arg1
MOV     REG2, DWORD PTR SS:[EBP+0x4]          ; arg2
MOV     DWORD PTR DS:[REG], REG2
```

好了这个 handler 就这么结束了，下一站，addHandler。让我们 F9 起来。

add32 handler

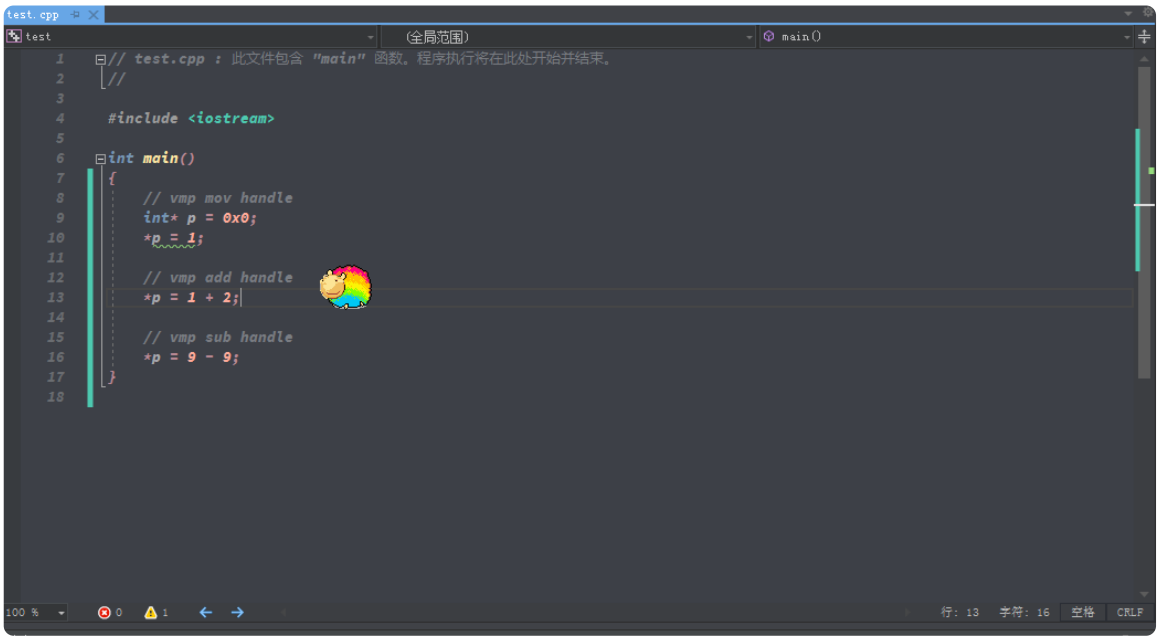
有没有搞错啊，这次连一个参数都不给看。真不给面子。



目前我们有如下信息，根据上一步的经验，我们可以发现，我们的代码写错了，哈哈，这依旧是条 mov32 handler。

```
MOV     DWORD PTR DS:[EDX], ECX
```

让我们好好想想：*p = 1 + 2 这行代码应该执行什么样的 handler ？



公布答案~ 大概就是这样了吧，有问题的话，以后再修改。

```
push32 arg1 // 1
push32 arg2 // 2
add32()
pop32 eflag // 206
pop32 arg2  // 3

push32 arg2 // 1+2 = 3
push32 arg3 // p
mov32()
pop eflag // 520
xxxx
```

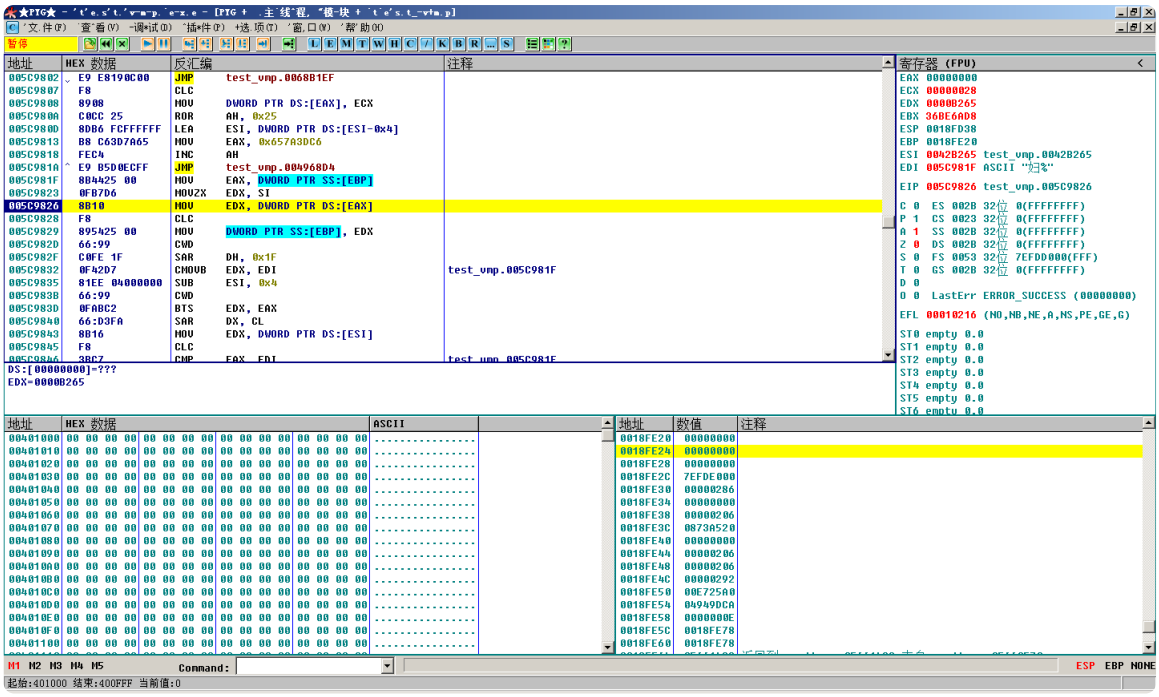
既然现在知道了 handler 的执行流程。 我们就可以知道哪里有问题了，应该改成如下代码：

```
*p = 1 + *p;
```

既然 这行写错了，那么 vmp sub handle 这一块代码就也写错了，将其更改成这样的形式：

```
*p = 9 - *p;
```

再编译一次，重新调试一下。可以看到，程序断下了，是取 p 的操作。



```
MOV     EAX, DWORD PTR SS:[EBP]
MOV     EDX, DWORD PTR DS:[EAX]
MOV     DWORD PTR SS:[EBP], EDX
进一步概括
MOV     REG, DWORD PTR SS:[EBP]
MOV     REG2, DWORD PTR DS:[REG]
MOV     DWORD PTR SS:[EBP], REG2
```

跳过出异常的指令，同时我们给 ebp 指向的地址下个硬件断点。接下来给出单步时，发现的信息。

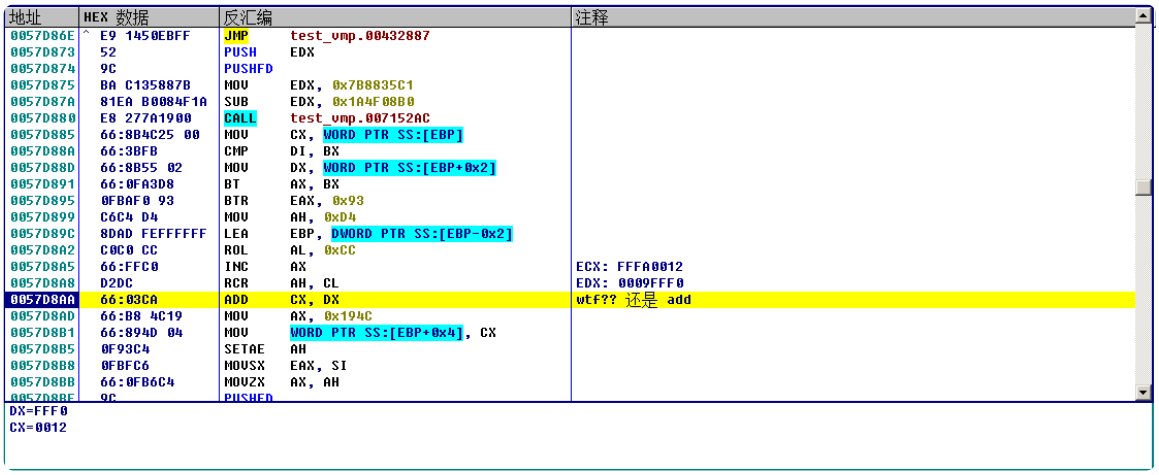
```
MOV     EDX, DWORD PTR SS:[EBP]
MOV     EAX, DWORD PTR SS:[EBP]
MOVZX   EDX, SI
MOV     EDX, DWORD PTR DS:[EAX]
CLC
MOV     DWORD PTR SS:[EBP], EDX

发现了 add32 handler 。
MOV     ECX, DWORD PTR SS:[EBP]
MOV     EDX, DWORD PTR SS:[EBP+0x4]
ADD     ECX, EDX
MOV     DWORD PTR SS:[EBP+0x4], ECX
PUSHFD
POP     DWORD PTR SS:[EBP]
进一步概括
MOV     REG, DWORD PTR SS:[EBP]
```

```
MOV     REG2, DWORD PTR SS:[EBP+0x4]
ADD     REG, REG2
MOV     DWORD PTR SS:[EBP+0x4], REG
PUSHFD
POP     DWORD PTR SS:[EBP]
```

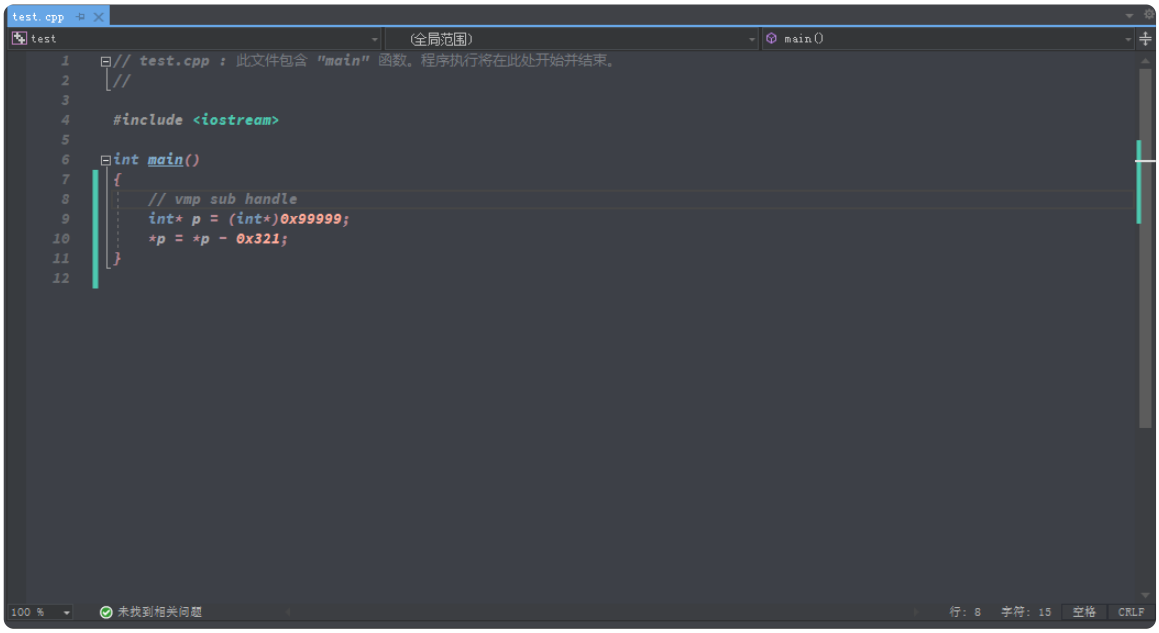
好啦，现在我们定位到了 vmp add32 handler，下一站 subHandler。

sub16 handler

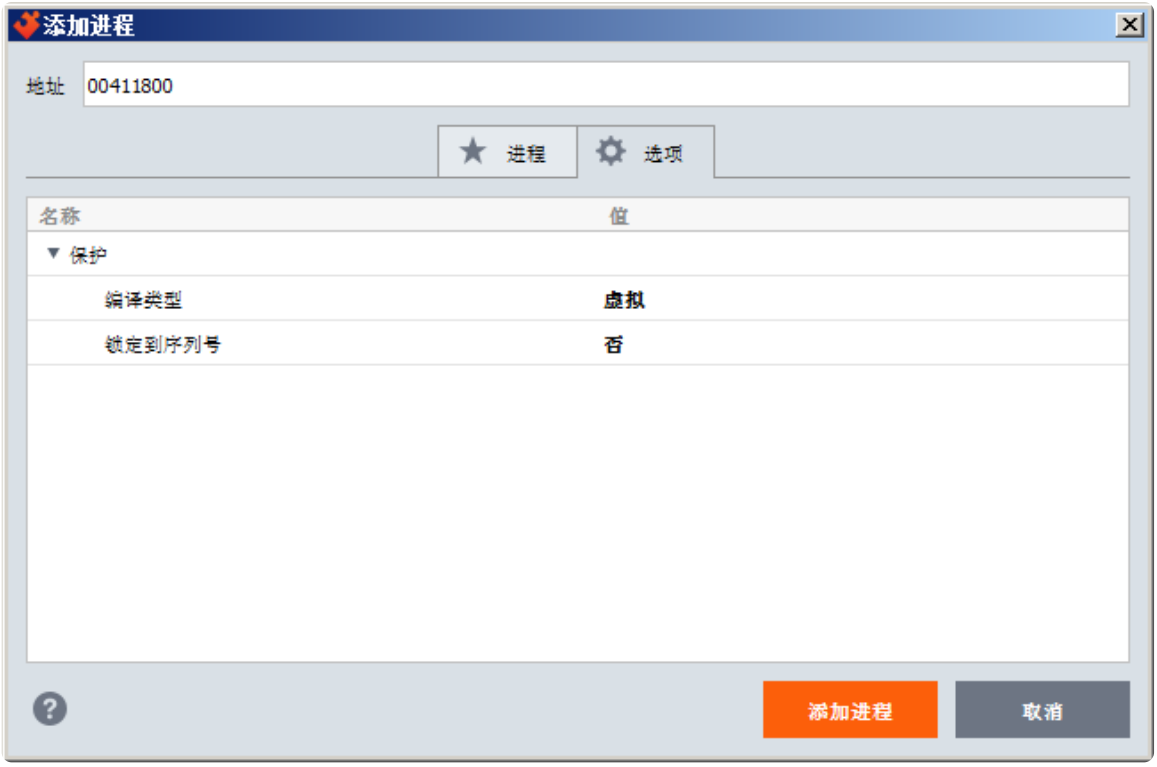
此时，我的硬件断点仍没有取消，直接 F9 就好了。 让我们看看，不得了不得了，怎么会是加法运算呢，让我们再次修改下代码，进行调试看看。



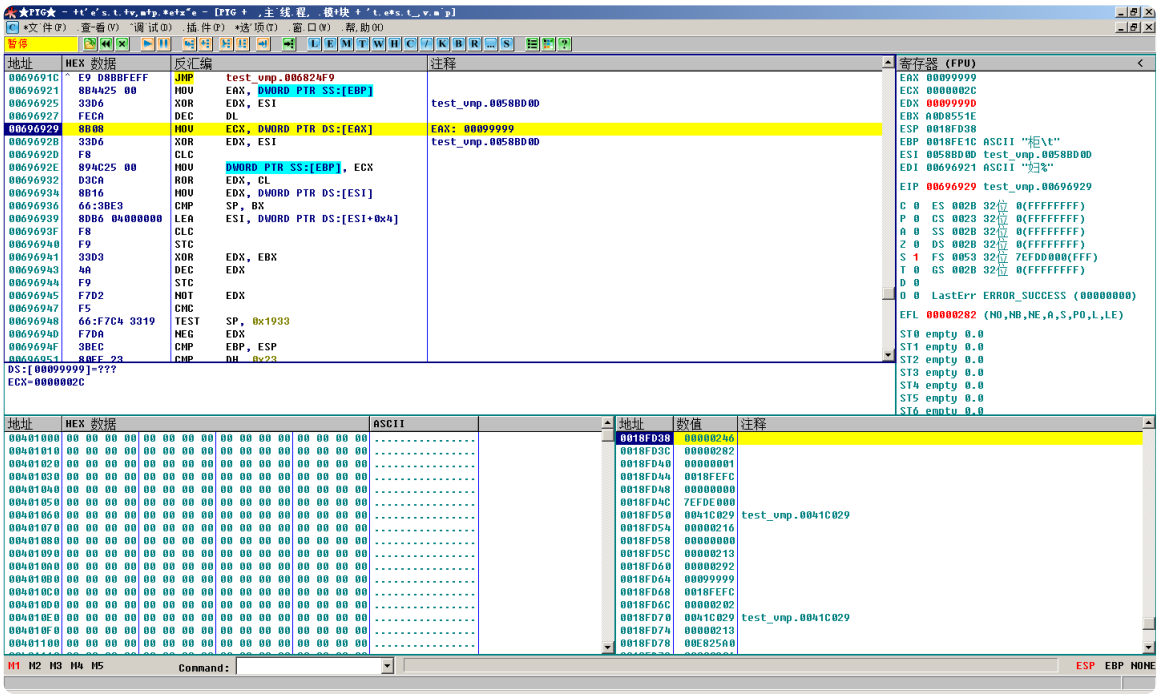
代码修改如下，只留了一个减法指令。



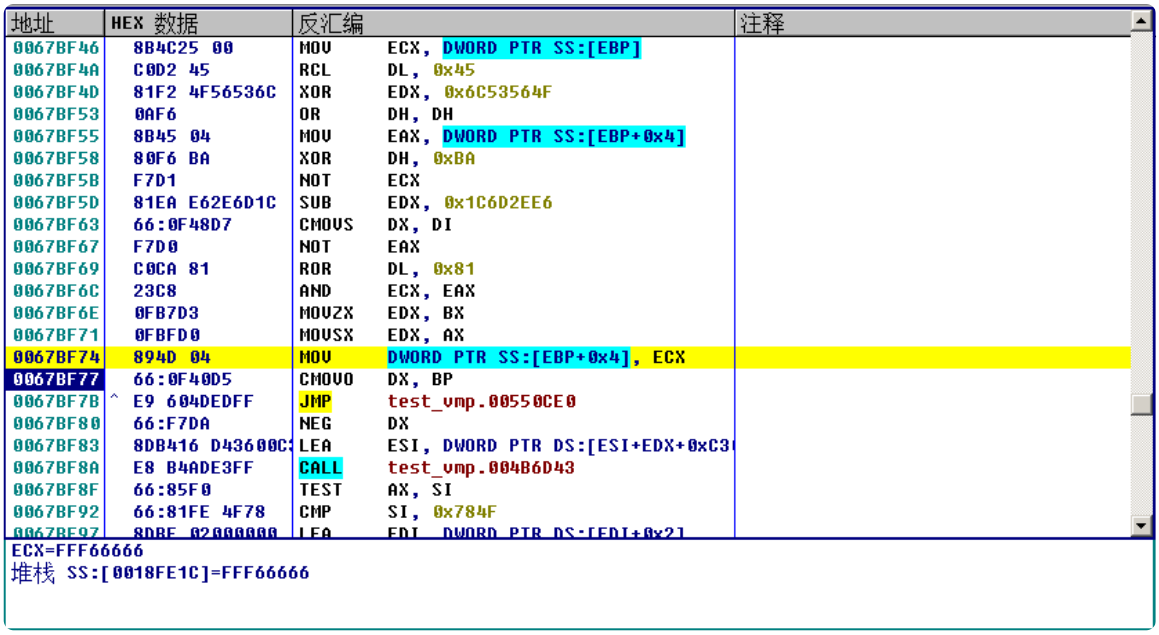
加壳的时候，选项改为虚拟，我倒要看看，vmp 把 sub 指令变成什么样了。



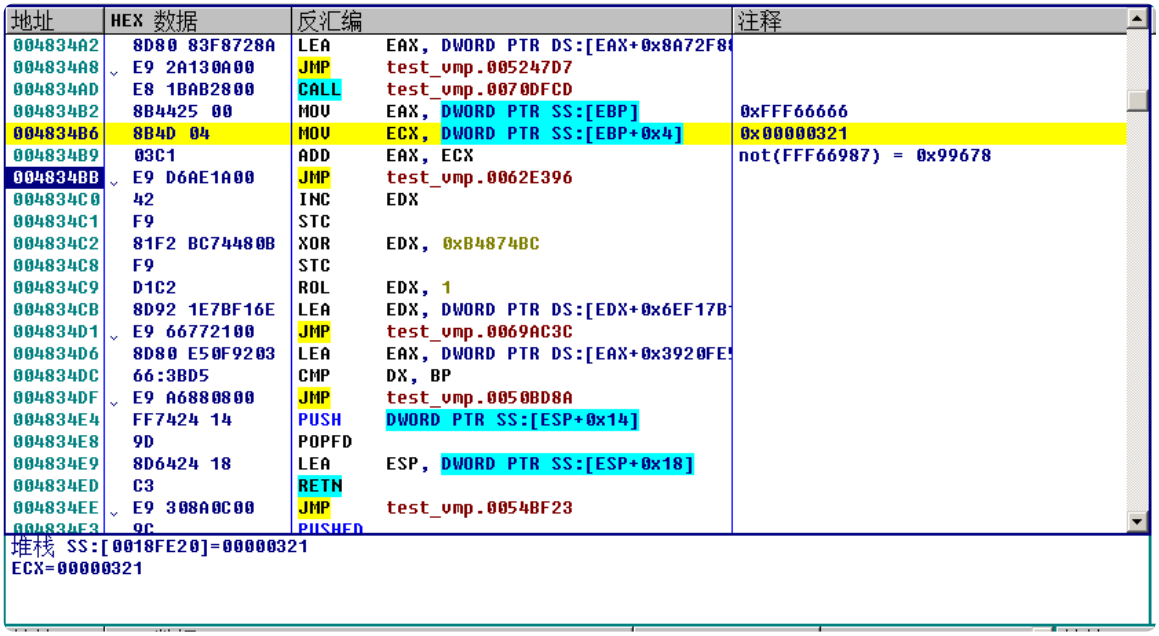
让我们重整旗鼓，再来一次。此时程序读取了 *p 的内容，看样子是要做减法运算了。



现在我要传授最基本技术点：单步 F8。 这里可以看到 vmp 对 0x99999 进行了 not 操作产生了 FFF66666 。



此时，我们对 [ebp + 4] 下硬件访问断点，我们成功断在了正确的位置上。



让我们来好好想想 vmp 的 sub 指令是如何运算的。(不用想了，刚才的调试已经告诉我们答案了)

```
sub(a,b) = not(add(not(a),b))
```

4. 结尾

希望大家多多支持、参与这个开源项目。(〃'▽'〃) https://github.com/FSMargoo/one-day-one-point

测试文件下载地址：

https://easyx-1314999863.cos.ap-nanjing.myqcloud.com/(2023.7.15)%E8%BD%AF%E4%BB%B6%E5%8A%A0%E5%AF%86%E4%B8%8E%E8%A7%A3%E5%AF%86-%E7%95%AA%E5%A4%961%5BXDbg%5D-%E6%B5%8B%E8%AF%95%E6%96%87%E4%BB%B6test.rar