任务一 实现简单的跳转壳

第一步：在PE文件中添加新节

要在PE中添加一个新节需要修改的字段有PE头中的NumberOfSections、PE扩展头中的SizeOfImage和SizeOfHeader，另外还需要新增一个节表项。

1. 修改NumberOfSections

原PE文件中有3个节，分别为.text、.rdata和.data，所以NumberOfSections的值为3，如图1所示，在010editor中直接将其修改为4并刷新即可。

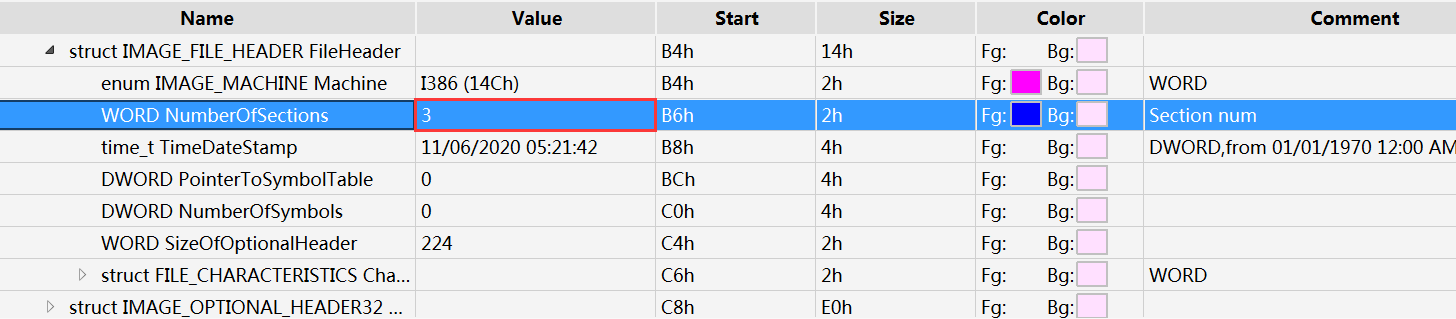


图 1 修改NumberOfSections

1. 编辑新增节的节表项

将新增节表项的Name字段设置为.newsec。考虑到该实验中的壳程序代码量很小，所以新增节设置为一个扇区的大小（200h Byte）即可，因此将VirtualSize和SizeOfRawData都设置为200h。因为前一节即.data节的VirtualAddress为3000h且.data节只会占据一个内存页，所以新增节的VirtualAddress为4000h。同理.data节的PointerToRawData为800h且.data节只占用了一个扇区大小，所以新增节的PointerToRawData为A00h。最后将Characteristics字段设置为 60000020h即可，即说明该新增节是可读、可执行且包含代码的节。其余字段默认为0即可，在010editor中编辑如图2所示。

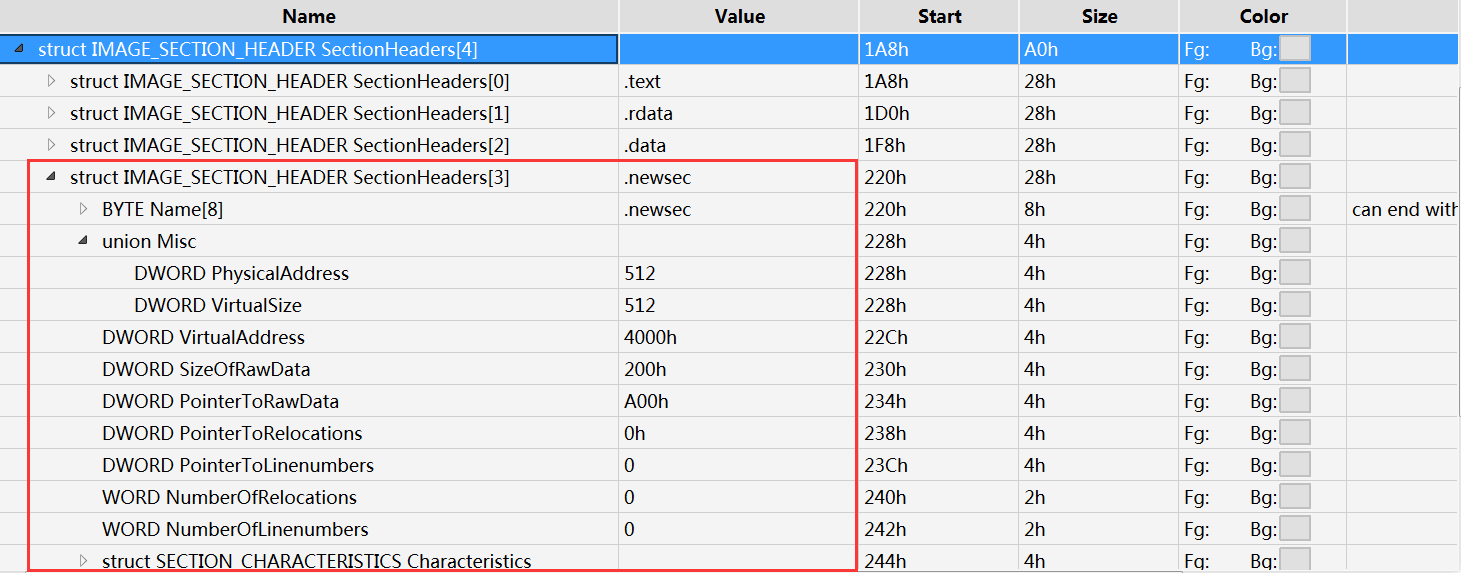


图 2 编辑节表项

1. 添加新增节的内容

前两步只是增加了一个节表项，文件中还没有该节的数据，所以需要增加该节的数据。在010editor中选择 Edit→Insert/Overwrite→Insert Bytes即可往PE文件中插入字节数据，如图3所示。

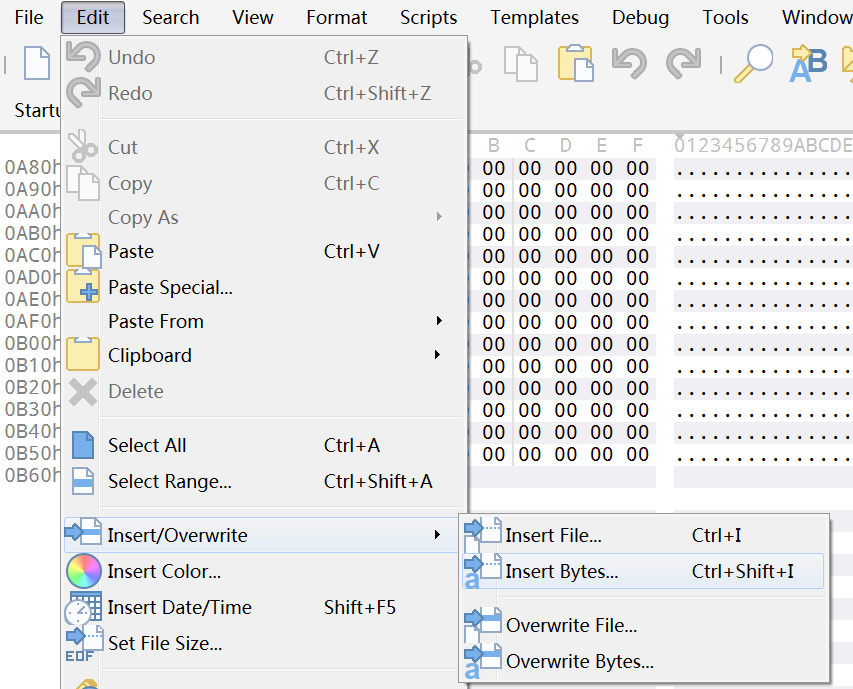


图 3 010editor插入数据

因为之前将新增节的大小设置为了200h个字节，所以这里直接在文件末尾插入200h字节全0数据即可，如图4所示。

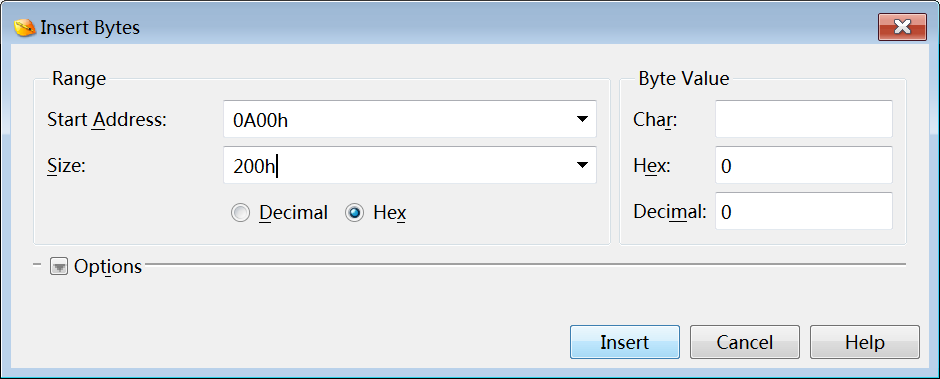


图 4 用0填充新增节

1. 修改SizeOfImage

新增节显然会使PE加载到内存时所需要的空间增大，由于之前设置新增节的大小只有200h个字节，所以加载到内存只需要1页的空间即1000h个字节，所以将SizeOfImage增加1000h。原来的PE文件有文件头、.text、.rdata、.data四个部分，分别占用1页内存，SizeOfImage大小为4000h，如图5所示，直接将其修改为5000h即可。

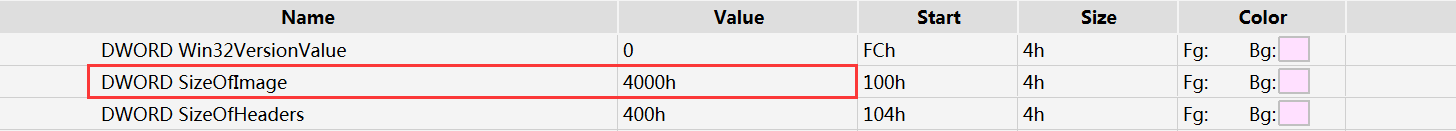


图 5 修改SizeOfImage

由于此次新增节并没有使文件头变得更大，因为新增节表项只是占用了之前为了文件对齐而填充的0，所以不需要修改SizeOfHeader。

此时新增节已经完成，保存所有修改后EXE文件可以正常运行，新增节操做成功。

第二步：写入跳转壳

跳转壳需要写入到前一步新增的节中，要让程序启动时首先执行壳中的代码需要修改PE头中的AddressOfEnterPoint字段为新增节的起始RVA地址，即同之前设置节表项时的VirtualAddress（4000h）值。于此同时将原来的AddressOfEntryPoint值保存下来，写跳转壳代码时使用。

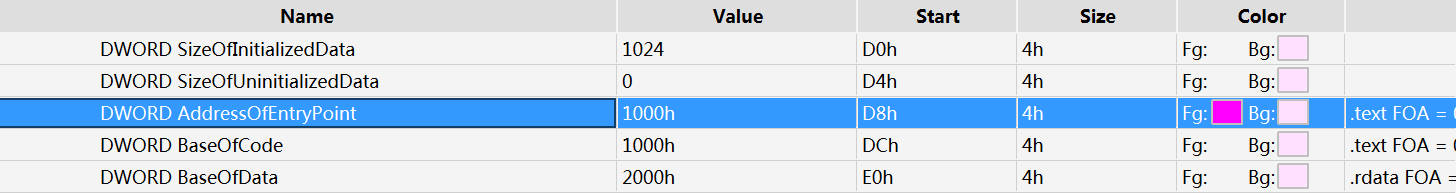


图 6 修改AddressOfEntryPoint

修改保存后使用Ollydbg打开该文件可以发现开始执行的地址为00404000h，即ImageBase（400000h）+AddressOfEntryPoint（4000h）。说明程序已经可以在启动时进入到新增节的起始位置开始执行。

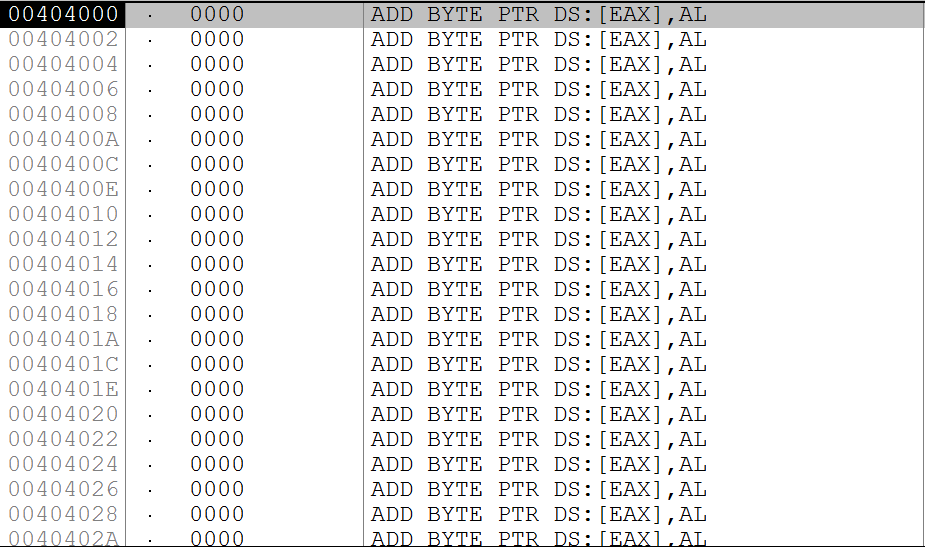


图 7 Ollydbg观察入口点

现在就需要在新增节的起始位置编入一条JMP指令使其能够跳转到原来的程序入口点执行。可以算出原来的程序入口点为00401000，即ImageBase（400000h）+AddressOfEntryPoint（1000h），直接利用OllyDbg编入JMP 0x00401000指令，如图8所示。

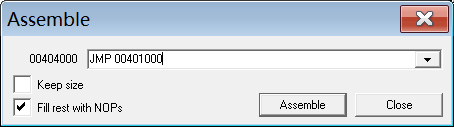


图 8 编入JMP指令

编入的结果如图9所示。

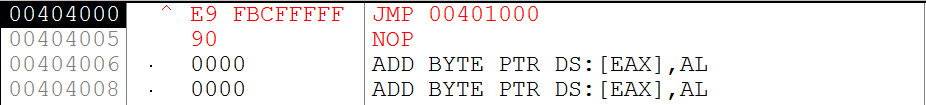


图 9 跳转壳的代码

此时在Ollydbg中执行Next可以看到其成功跳转到了原程序入口处，如图10所示。

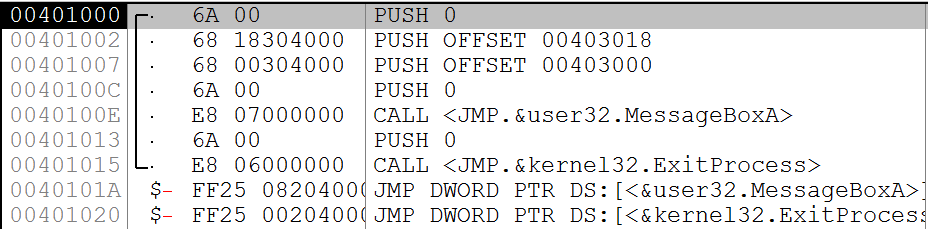


图 10 跳转回原入口点

将修改保存到EXE文件后直接启动该程序可以看到程序正常运行，跳转壳实现完成。

任务二 实现PE文件加密壳

因为只需要加密.text节的数据，所以不需要考虑导入表加密后无法使用的问题。首先需要在010editor中对.text节进行异或加密，操作方法为选中.text节的所有数据并点击Tools→Hex Operations→Binary Xor...，如图11所示。

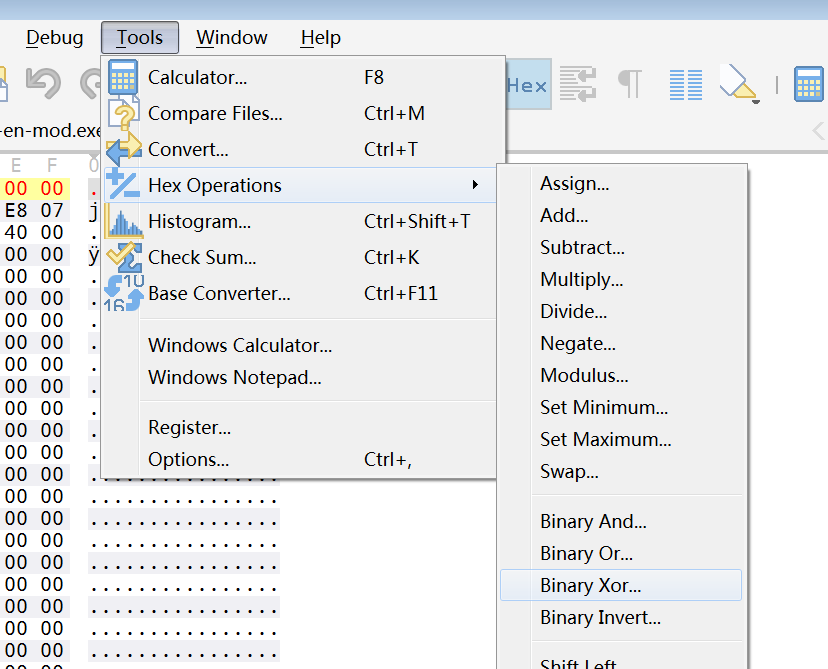


图 11 异或加密操作

选择0x12进行异或运算，如图12所示。

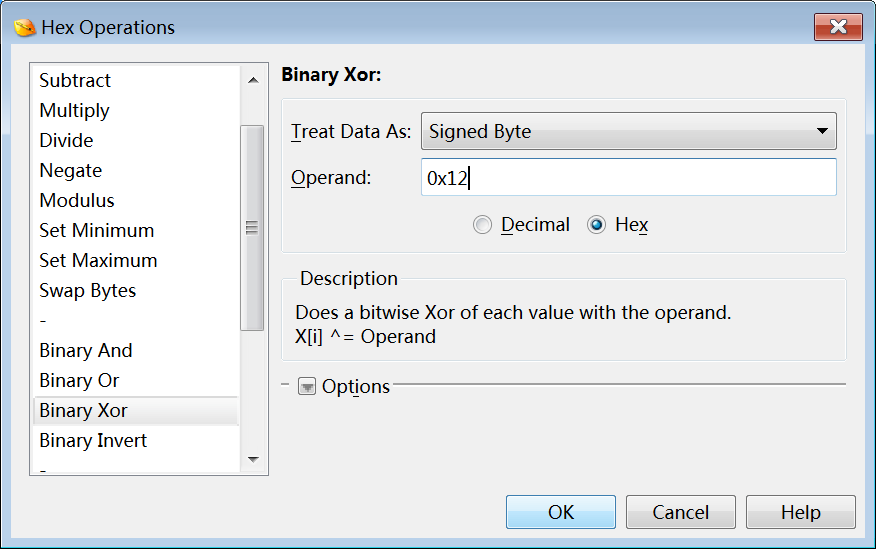
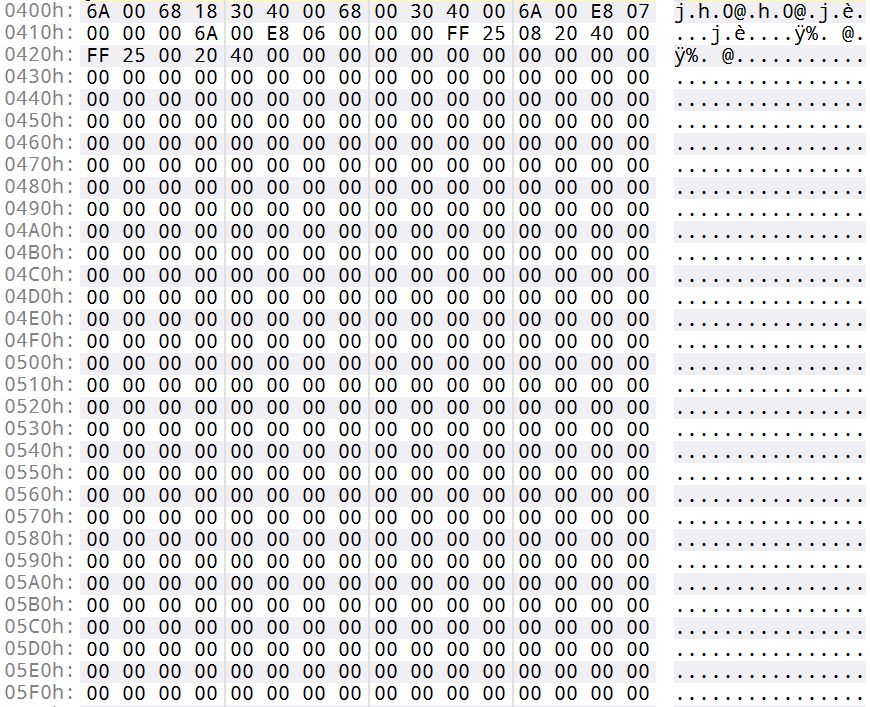


图 12 异或加密

加密前后.text节的数据对比如图13所示。

加密前



加密后

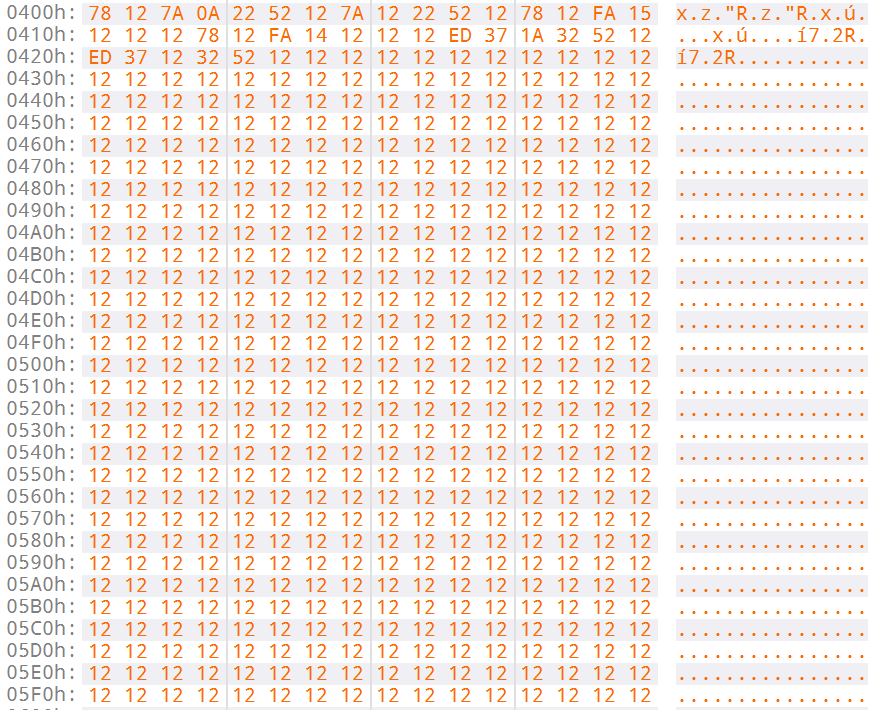


图 13 加密前后.text对比（省略了部分0填充）

同任务一进行新增节操作并修改AddressOfEntryPoint为新增节的起始RVA，并在Ollydbg中写入异或解密的程序，如图14所示。

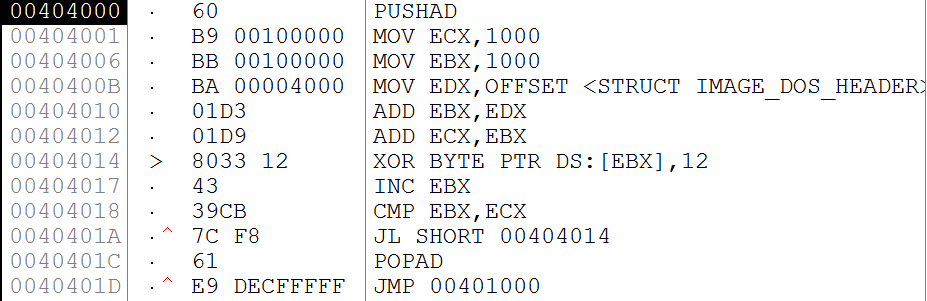


图 14 解密壳代码

该程序将加载到内存的.text节的每个字节分别与0x12进行异或从而完成解密，解密后使用JMP 00401000跳转到原入口点开始执行。但是目前这段代码并不能正常执行，因为.text节没有是不可写的，所以需要在010editor中将.text的可写属性设置为1，如图15所示。

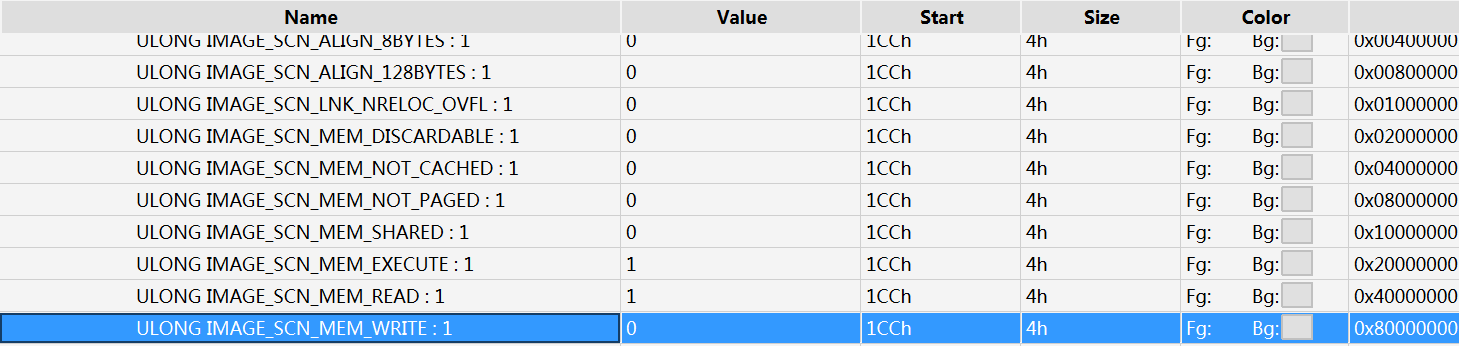


图 15 将.text节设置为可写

将修改全部保存后直接启动该程序可以看到程序正常运行，加密壳实现完成。