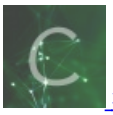
《漏洞战争》学习笔记·2 CVE-2010-3333

原创

useror 于 2019-05-27 13:58:18 发布 495 收藏

分类专栏： 漏洞

本文链接：https://blog.csdn.net/useror/article/details/90600818



知识补充

RTF格式是为文本和图像信息格式的交换制定的一种文件格式，使用与不同设备，操作环境和系统。RTF文件基本元素：正文、控制字、控制符号、群组

控制字： RTF用来标记打印控制字符和管理文档信息的一种特殊格式的命令，RTF用它做正文格式的控制代码，每个控制字均以一个 **反斜杠\开头** ,由**a~z小写字母组成**，通常不应该包含任何大写字母，而每个**分隔符标志着控制字名称的结束**。使用格式：**\字母序列<分隔符>**。

控制符号由反斜杠后跟一个单独的、非字母的字符，表示一个特定的符号。

群组由**包含在大括号中的文本、控制字或控制符组成。左扩符 ｛ 表示组的开始，有扩符 ｝表示组的结束**。每个组包括文本和文本的不同属性。RTF文件也能同时包括字体、格式、屏幕颜色、图形、脚注、注释、文件头、文件尾、摘要信息、域和书签的组合、以及文档区段、段落和字符的格式属性。

一个**完整的RTF文件包括文件头<header>和文档区<document>** 两大部分，可用以下语法表示:

<File>'{'<header> <document>'}'

漏洞发生原因

样本数据分析如下：

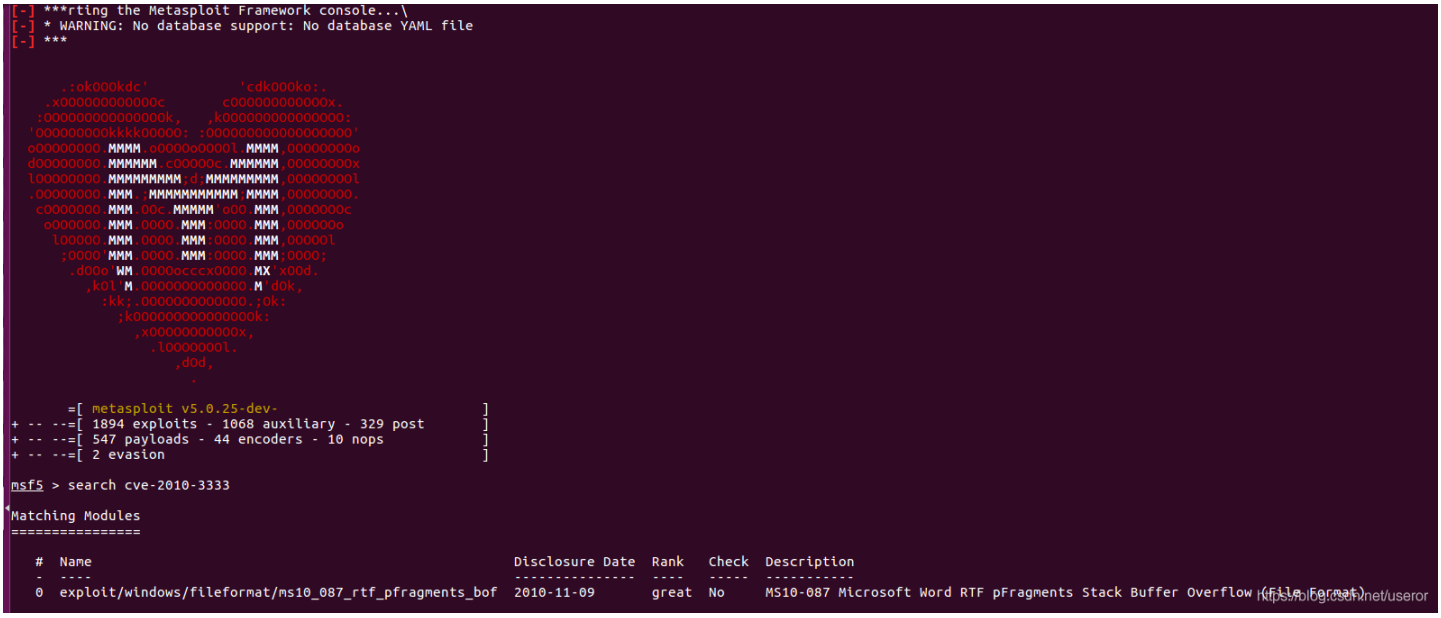
\rtf1 —— RTF 版本
\ansi —— 支持 ANSI 字符集
\shp —— 绘图对象
*\shpinst —— 图片引用
\sp —— 绘图对象属性定义
\sn pFragments—— 定义属性名称，pFragments 段是图形的附加部分，属于数组结构。它允许图形包含多个路径和分段，该属性列出图形各个碎片
\sv —— 定义属性值

RTF分析器正是在解析pFragments属性值时，没有正确计算属性值所占用的空间大小，导致栈溢出漏洞的发生。

来自tools loversorry: 分析一个漏洞的成因，不是只是再现。我觉得真正的成因其实你没有找到，应该是在**处理什么功能的时候，调用了哪个函数，然后又调用了哪个函数，再调用哪个函数……最后在哪里产生溢出，这个溢出是否特殊，如何可以触发，触发的条件是什么，shellcode有没有特殊要求**。2、一般找溢出点我自己以前是这样弄的：**触发溢出，在处理seh的那个handler的api函数上下断点，中断后根据seh链逐级往上走，最终走如代码空间就找到了**。当然有时也会**对堆栈的一个地址下写中断，中断后直接就找到点了**。3、其实找触发，再现触发这些都不难。**难的是处理各种特殊环境的情况，如何让你的shellcode避开这些特殊环境，如何避开操作系统的各类安全机制**。

基于栈回溯的漏洞分析方法

首先，**通过Metasploit生成可触发漏洞的Poc样本**，实际的病毒样本对分析Exploit技术或Shellcode技术更有研究和学习价值（但分析环境也更复杂）。Metasploit安装方法，linux下安装吧，windows杀软报毒爆炸。



命令：

root@ubuntu:~# msfconsole

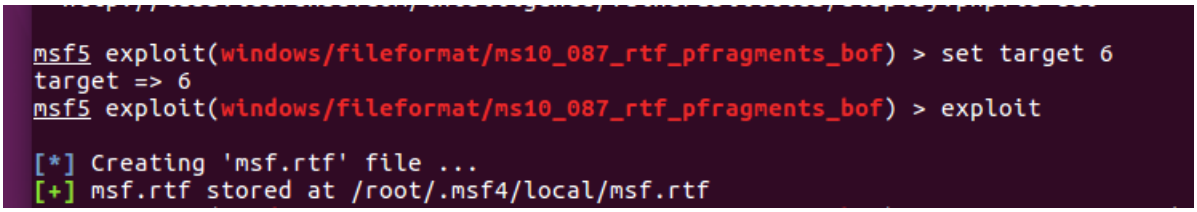
msf5 > search cve-2010-3333

msf5 > use exploit/windows/fileformat/ms10_087_rtf_pfragments_bof

msf5 exploit(windows/fileformat/ms10_087_rtf_pfragments_bof) > info

msf5 exploit(windows/fileformat/ms10_087_rtf_pfragments_bof) > set target 6

msf5 exploit(windows/fileformat/ms10_087_rtf_pfragments_bof) > exploit



环境配置

操作系统：xp pro sp3 来源msdn itellyou

调试器：Immunity Debugger　Windbg 合集 key:0zy6

反汇编器：IDA Pro 7 key:wjey

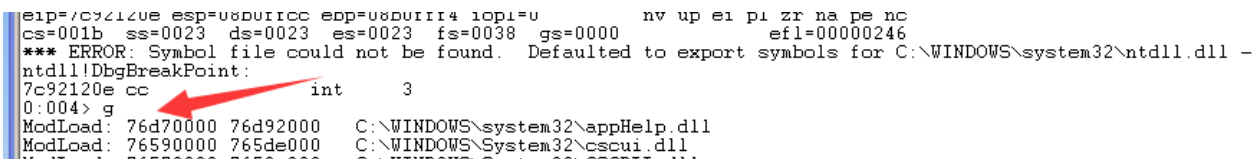
漏洞软件：Microsoft Word 2003 sp3

注意事项

使用windbg必须配置调试符号 （配置符号真的很？所以我觉得还是用Immunity Debugger来追溯吧）

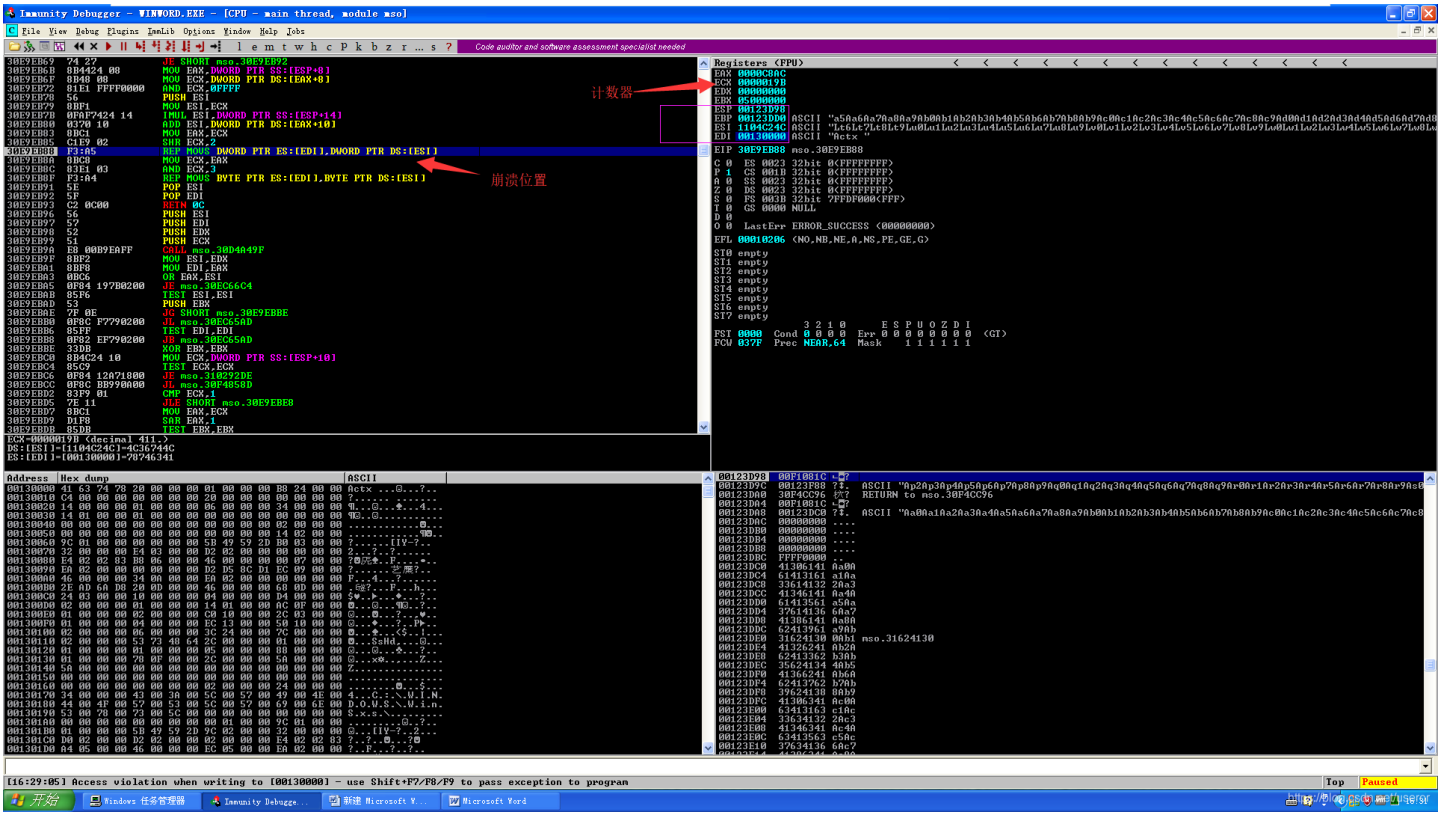
调试过程

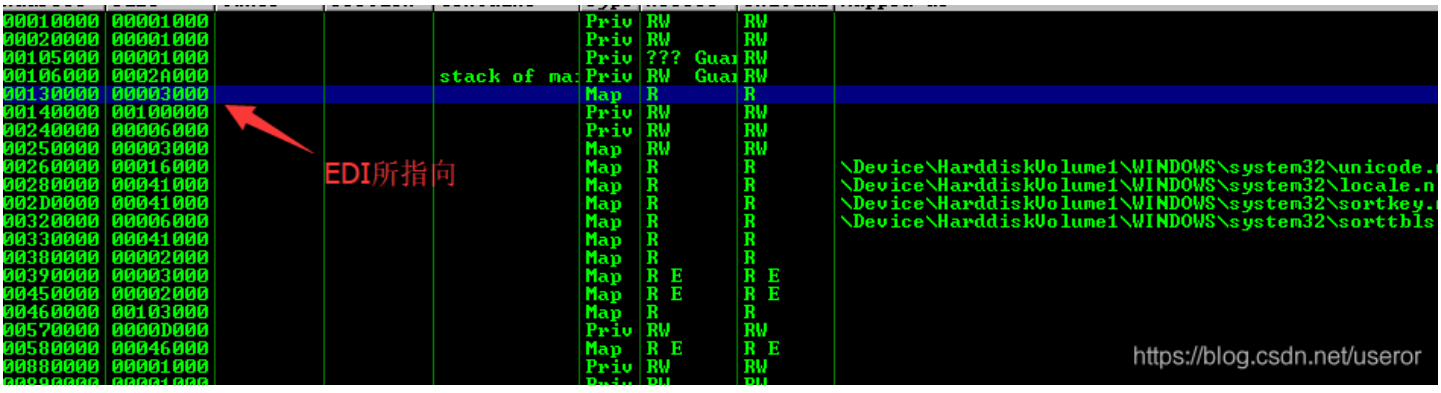
1·附加调试 g 命令之后打开msf文件即断到异常处



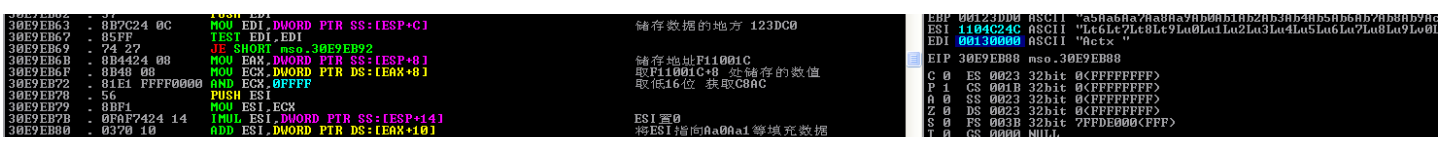
```
ModLoad: 76570000 7658c000   C:\WINDOWS\System32\CSCDLL.dll
ModLoad: 75ef0000 75fed000   C:\WINDOWS\system32\browseui.dll
ModLoad: 7e550000 7e6c1000   C:\WINDOWS\system32\shdocvw.dll
ModLoad: 765e0000 76673000   C:\WINDOWS\system32\CRYPT32.dll
ModLoad: 76db0000 76dc2000   C:\WINDOWS\system32\MSASN1.dll
ModLoad: 75430000 754a1000   C:\WINDOWS\system32\CRYPTUI.dll
ModLoad: 76680000 76726000   C:\WINDOWS\system32\WININET.dll
ModLoad: 76c00000 76c2e000   C:\WINDOWS\system32\WINTRUST.dll
ModLoad: 76c60000 76c88000   C:\WINDOWS\system32\IMAGEHLP.dll
ModLoad: 76f30000 76f5c000   C:\WINDOWS\system32\WLDAP32.dll
ModLoad: 39800000 399b3000   C:\Program Files\Microsoft Office\OFFICE11\GdiPlus.DLL
ModLoad: 76f20000 76f28000   C:\WINDOWS\system32\WTSAPI32.DLL
ModLoad: 762d0000 762e0000   C:\WINDOWS\system32\WINSTA.dll
(3e8.2f0): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=0000c8ac ebx=05000000 ecx=0000019b edx=00000000 esi=1104c24c edi=00130000
eip=30e9eb88 esp=00123d98 ebp=00123dd0 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010206
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for C:\Program Files\Common Files\Microsoft Shared\
mso!Ordinal6426+0x64d:
30e9eb88 f3a5            rep movs dword ptr es:[edi],dword ptr [esi]
0:000>
```

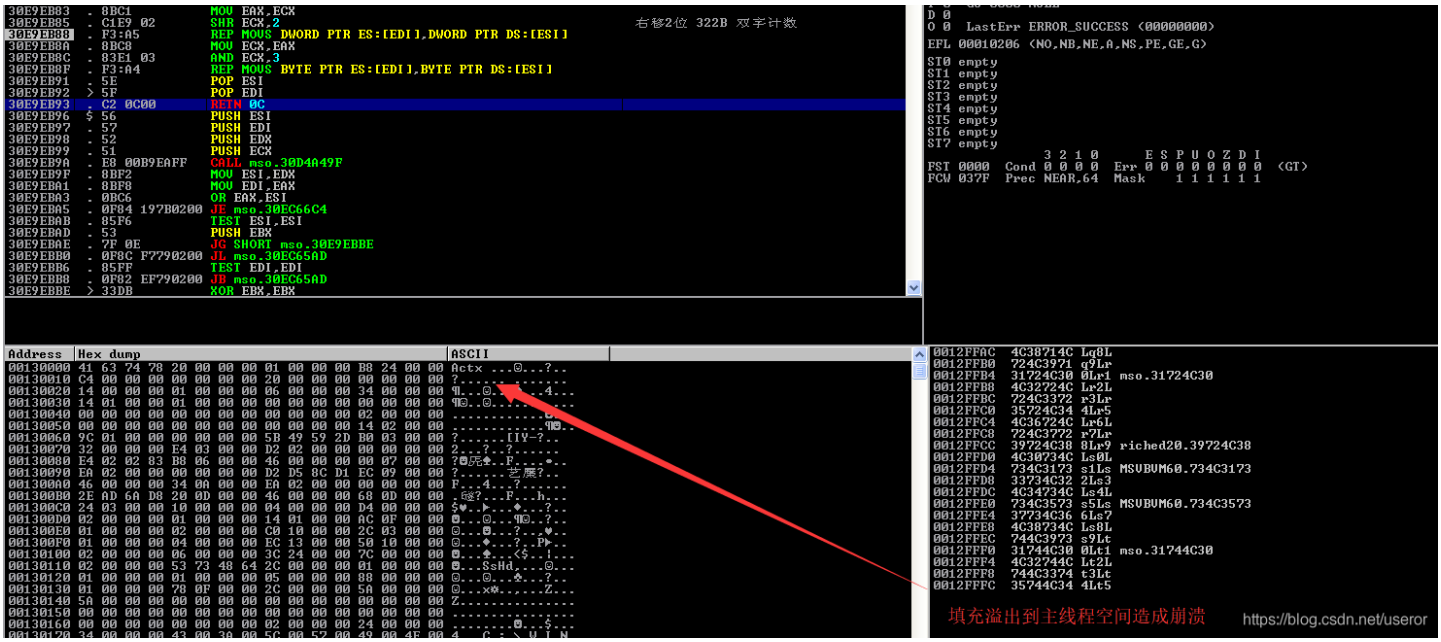
异常所在处



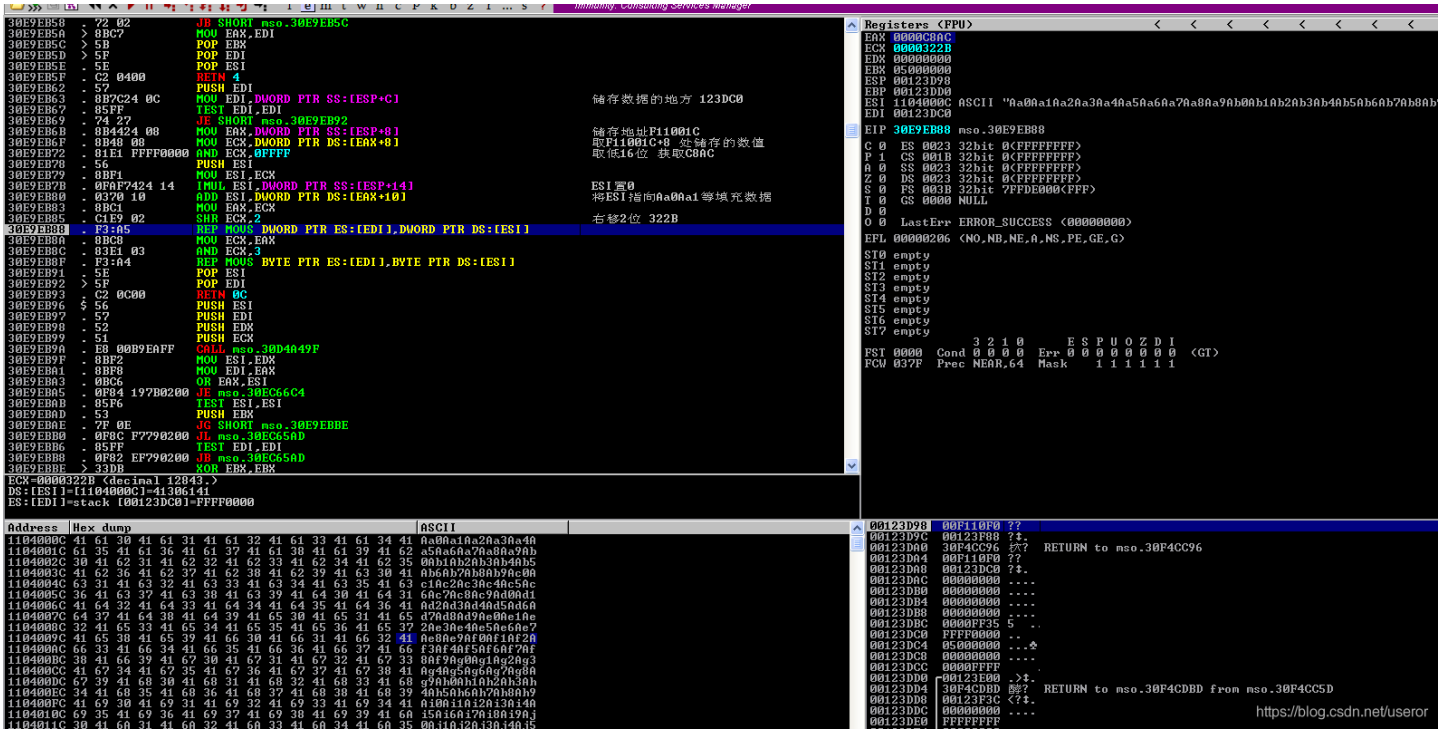


主线程溢出所在点





崩溃点分析



开始前填充寄存器状态

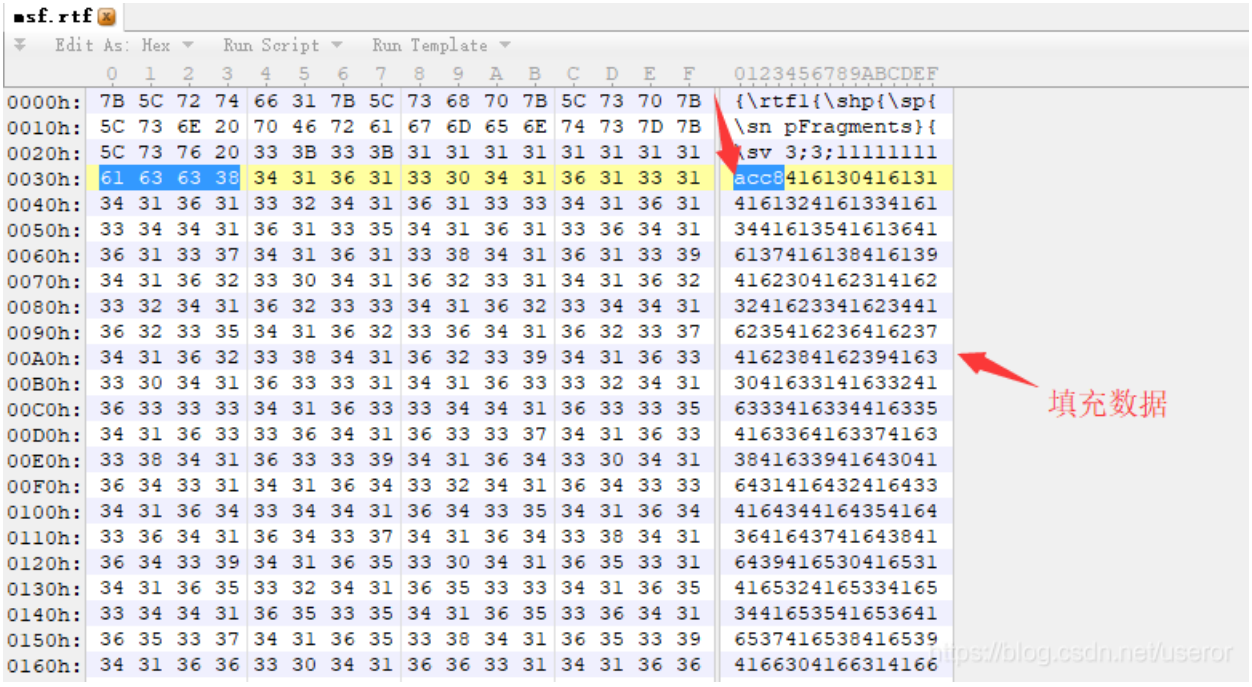
EAX= 0xC8AC 这是写在RTF文档中的（目前理解是拷贝大小 pFragements

属性值的第3个字段，偏移8个字符后即为复制数据的大小 **希望大牛可以指点下如何查找这种粒度的文档　查到的资料只有描述没有这种字节程度的定义** ）

ECX=0x322B C8AC右移2位 实现双字计数

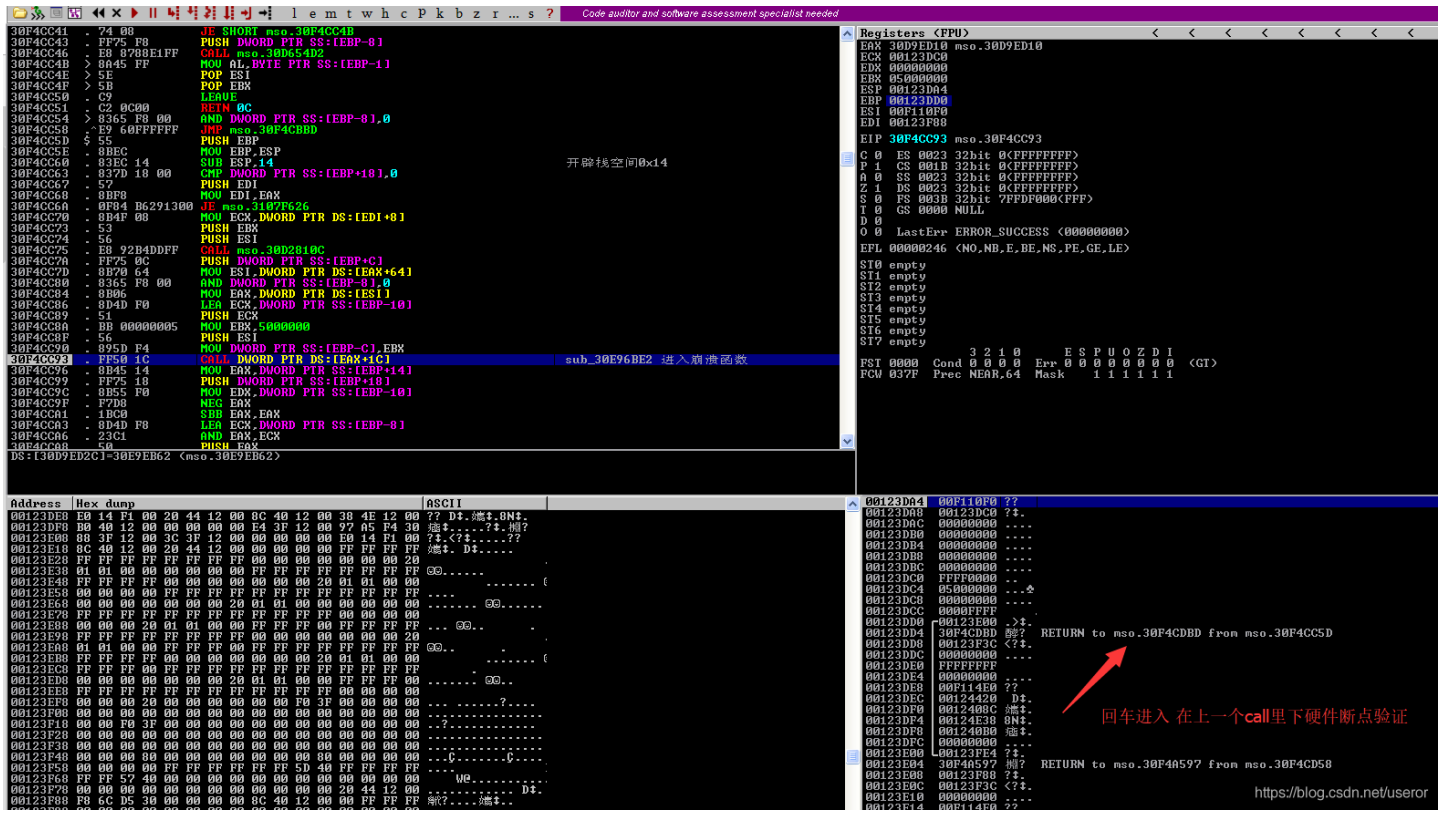
ESI=1104000C 指向被复制的数据储存起始 （此处数据应该可以改写为shllecode）

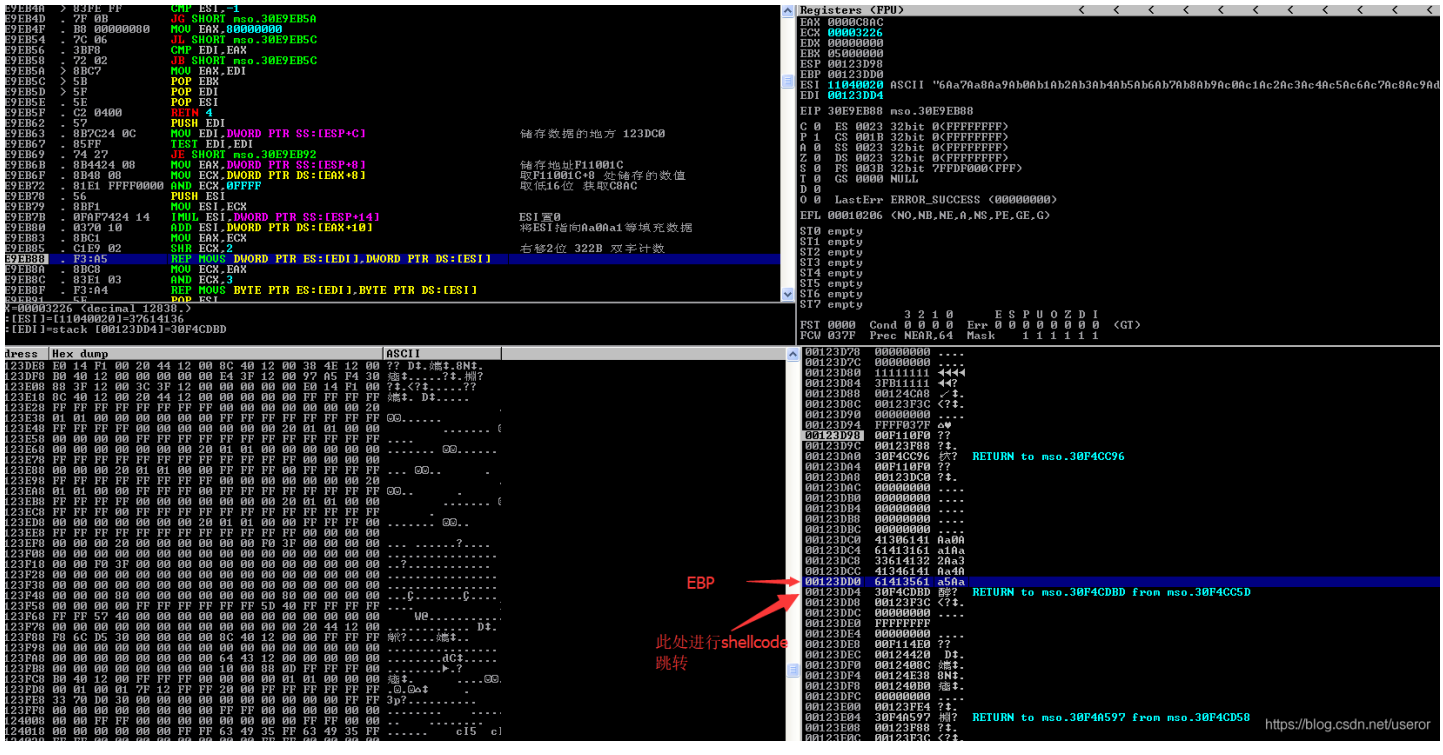
EDI=00123DC0 数据填充起始地址



崩溃过程利用分析

主要操作：硬件断点追踪再重新附加





结合这两张截图可以看到 在调用sub_30E96BE2崩溃函数的上一层函数sub_30F4CC5D中通过sub ESP,14指令开辟的栈空间被填充数据覆盖，由于样本msf.rtf中复制的内存数据较大，导致
REP MOVS 指令在复制的过程中覆盖到不可写的内存地址（主线程的0013 0000）从而触发异常，因此没有执行到被覆盖的返回地址或者SEH异常处理函数。~~所以有以下两种思路来实现GetPC：~~

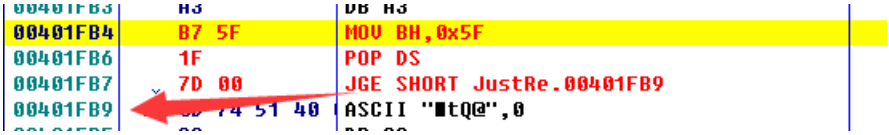
~~1.复制数据大小超过0x14覆盖sub_30F4CC5D等调用函数的返回地址，通过RETN返回GetP~~

~~2.覆盖到SEH结构 通过修改SEH处理异常流程GetPC~~

利用过程

将返回地址用Jmp Esp指令地址覆盖，具体操作为 复制数据大小的值之后再偏移0x14字节，覆盖到返回地址；然后将shellcode放置在后面，由于在漏洞函数的结尾出，在返回时会弹出0x14大小的栈空间，因此在Jmp Esp中填充一些垃圾字节以填充空缺。

5f c9 c2 14 00



b7 5f 1f 7d 00

关于GetPC技术在看雪的几种神仙操作