测试环境：

windows xp sp3

adobe reader 9.0

首先在kali下生成poc pdf，

payload设置为messagebox： hello from msf

说明：这个pdf的poc无法在这个环境下稳定运行，多次运行后才有可能会出现meesagebox的窗口。

————————

复制出来CoolType.dll

alt+T搜索sing就能来到漏洞函数附近



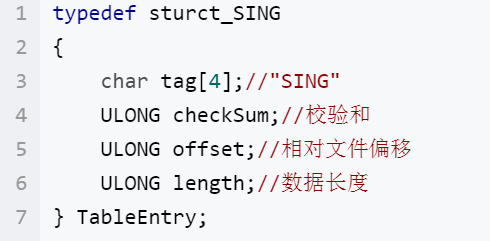
WIN7

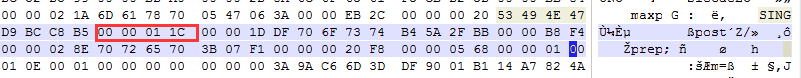
73563A18

关于提取sing表的object：

直接用winhex打开poc，搜索sing字符串是找不到sing的，只有在用PdfStreamDumper.exe，找到sing表的object，然后save decompressed stream然后用winhex打开dump出来的二进制文件，搜索sing才能找到sing表结构：

官方的sing表结构：





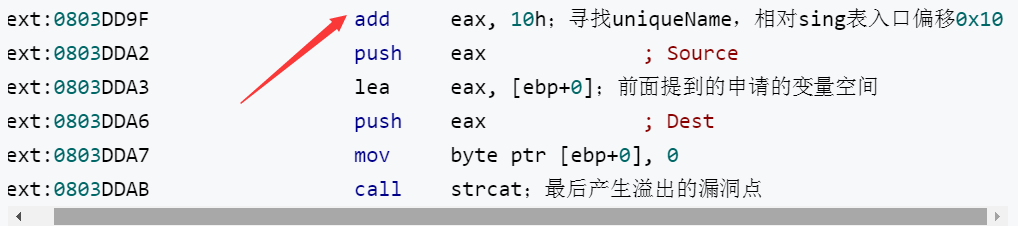
看相对文件偏移11c

来到11c



sing表的数据结构：

因为逆向过程中发现这个：



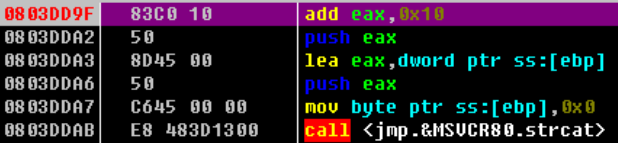
在sing表偏移16的地方是uniquename，

（其实这里定位漏洞的逻辑可能有点问题：因为首先是你不知道漏洞到底是在哪个dll里，而泉哥的书中就是直接定位cooltype.dll，然后进行分析。我个人感觉在定位的时候应该是：1.首先你应该了解pdf的结构，知道sing表，知道sing表偏移0x10的地方有个uniquename域，然后用OD动态调试：对这个uniquename域的数据下内存断点，然后F9，看看会断在哪里，这个时候应该是断在复制这个域的指令处，然后你查看这个指令的地址是在哪个dll里，然后再进行具体分析。）

执行strcat后会将uniquename起始的部分复制，直到遇到null。

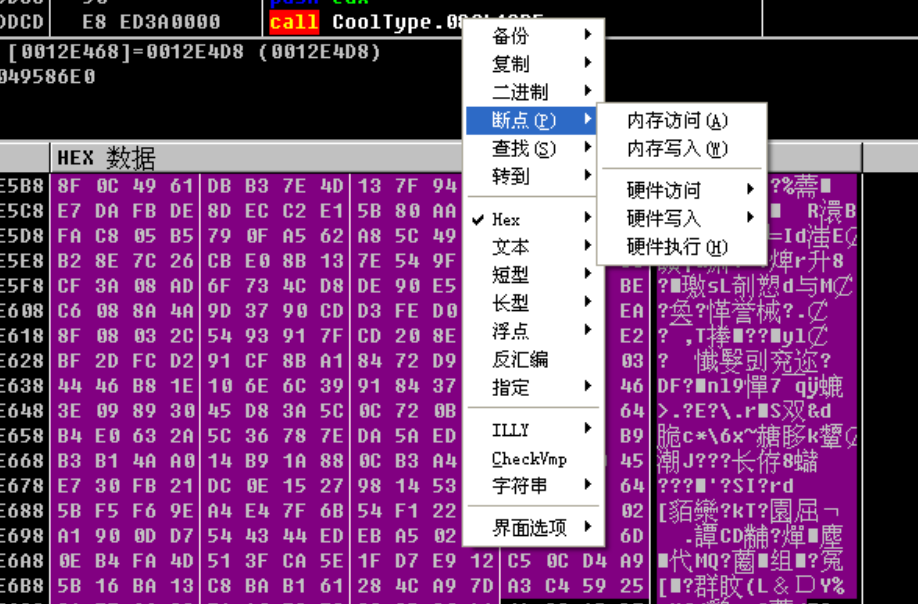
动态调试：

对去sing表偏移0x10的地方下断



溢出修改了一个虚表的指针使得指指向你精心构造的ROP链

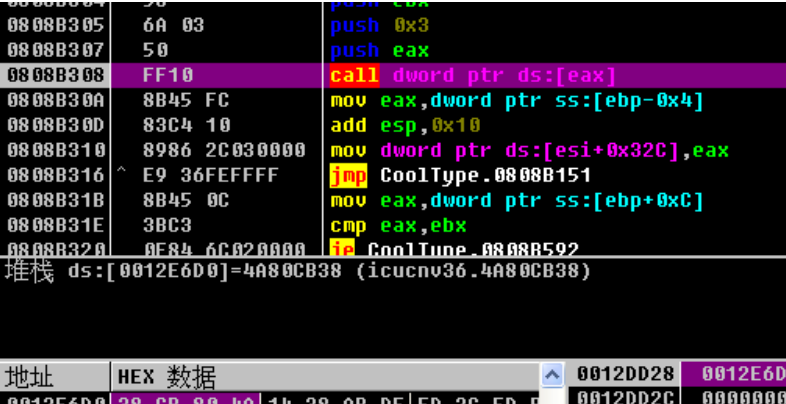
这里为了方便调试将刚刚写入占中的地址全部设上访问中断



复制数据：



这里一个调用虚表的指令：



这里就用到了一种方法：即使程序开启GS保护，但是绕过GS的一种很简单的方法就是，在函数返回前劫持函数，方法之一就是利用虚表，这是GS无法阻止的，而且很有效。

然后就是如何编写shellcode的问题了：需要将这个虚表的地址覆盖，

我觉得的可能的思路就是：  
第一种:直接将shellcode写在内存中，覆盖虚表函数地址为一个ROP链，调用virtualProtect或者virtualalloc或者zwsetInformationprocess来使shellcode的DEP保护失效，

第二种：因为PDF可以内嵌JS代码，所以可以采用堆喷的方法：覆盖地址为0c0c0c0c，然后申请内存=nop slide+shellcode，然后跳回去执行shellcode，这里可能也会碰到dep的问题，因为申请的内存可能也具有数据执行保护，所以还要采用第一种方式来绕过DEP。

剩下的问题就是如何构造ROP链了：mona插件自动生成rop链