

Flexnet Publisher V11.16及其以下版本加密种子点恢复方法

By: YangMyron

Flexnet Publisher V11.16已经做了改进，隐藏了原来恢复加密种子点的标志**3D4DA1D6**，无法实现快速定位。现提供两种方法：通用法和标志法。针对**V11.16**版，未使用**Imgr_trl**库链接的守护神可以通过标志**0F42400h**定位直接得到加密种子点。

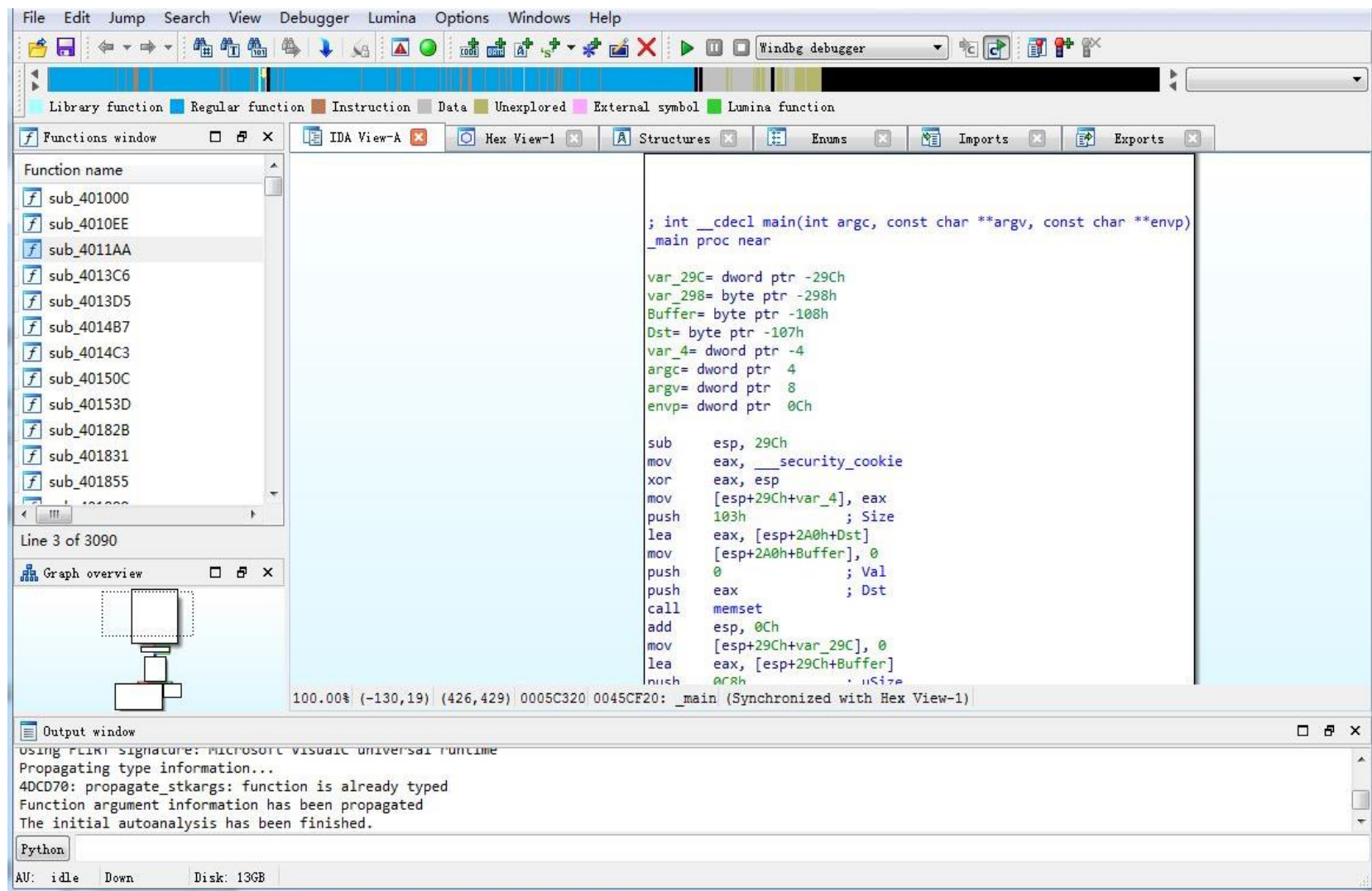
Flexnet Publisher V11.16及其以下版本加密种子点恢复方法

通用法

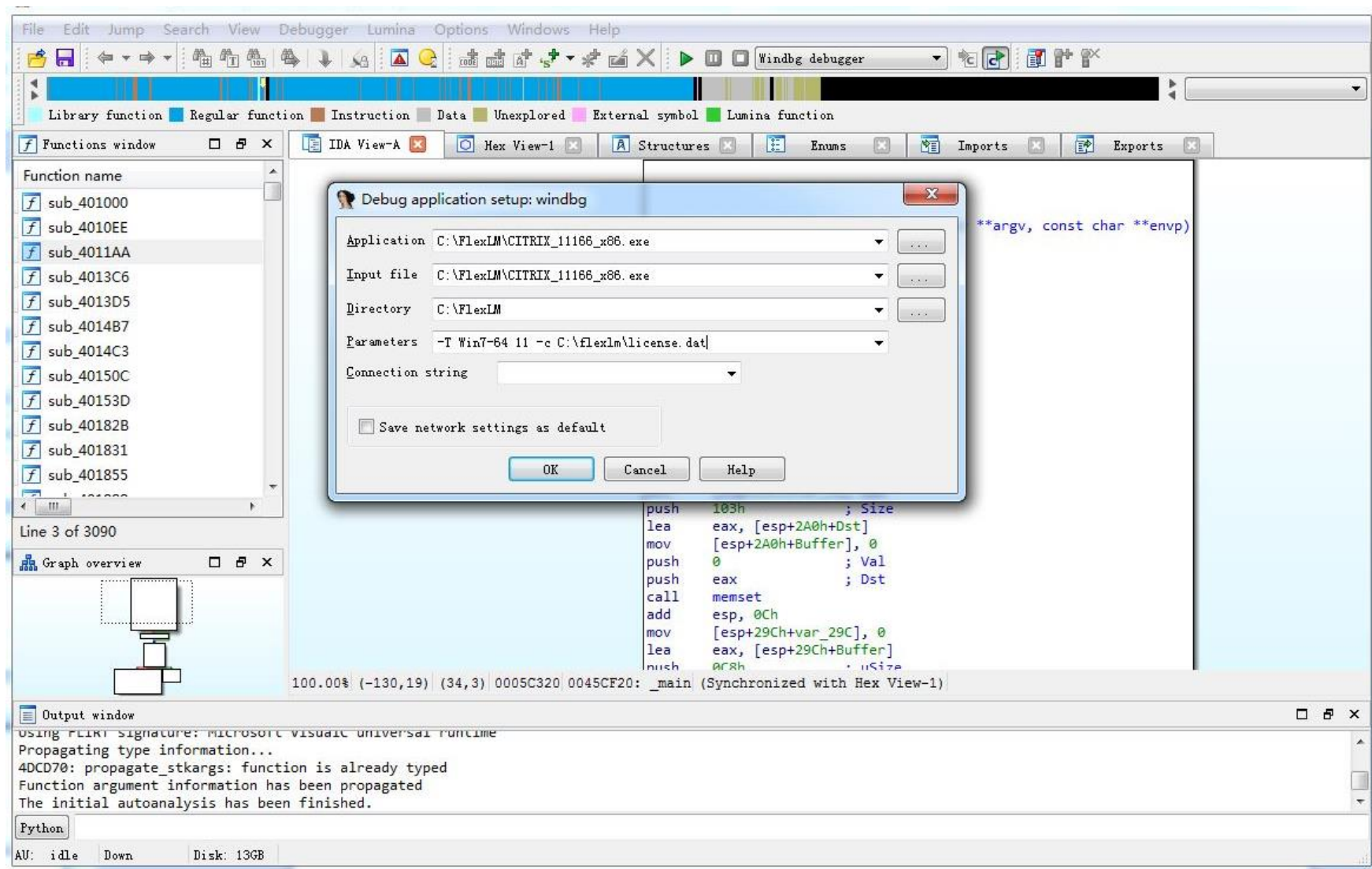
1、准备好格式正确的license.dat文件，一般放在c:\flexlm目录下。

```
SERVER WIN7-64 [REDACTED]
DAEMON [REDACTED] c:\flexlm\[REDACTED]_11166_x86.exe
INCREMENT session [REDACTED] 8.8 31-dec-2037 9998 D4D693AE978F HOSTID=ANY \
    user_info=<aalbuquerque@agillity.com.br> ISSUER="GraphOn \
    Corporation" ISSUED=21-Nov-14 NOTICE="Copyright (C) GraphOn \
    Corporation. All Rights Reserved" ck=152 \
    SN=FXQ3BJ_PRODUCTCODE=141001162524934M53E622qXwAVT TS_OK \
    SIGN=81BE7DE89164
INCREMENT any_app [REDACTED] 8.8 31-dec-2137 9999 2A56248BCDD5 HOSTID=ANY \
    user_info=<aalbuquerque@agillity.com.br> ISSUER="GraphOn \
    Corporation" ISSUED=21-Nov-14 NOTICE="Copyright (C) GraphOn \
    Corporation. All Rights Reserved" ck=179 \
    SN=FXQ3BJ_PRODUCTCODE=141001162524934M53E622qXwAVT TS_OK \
    SIGN=C676EAB876E6
```

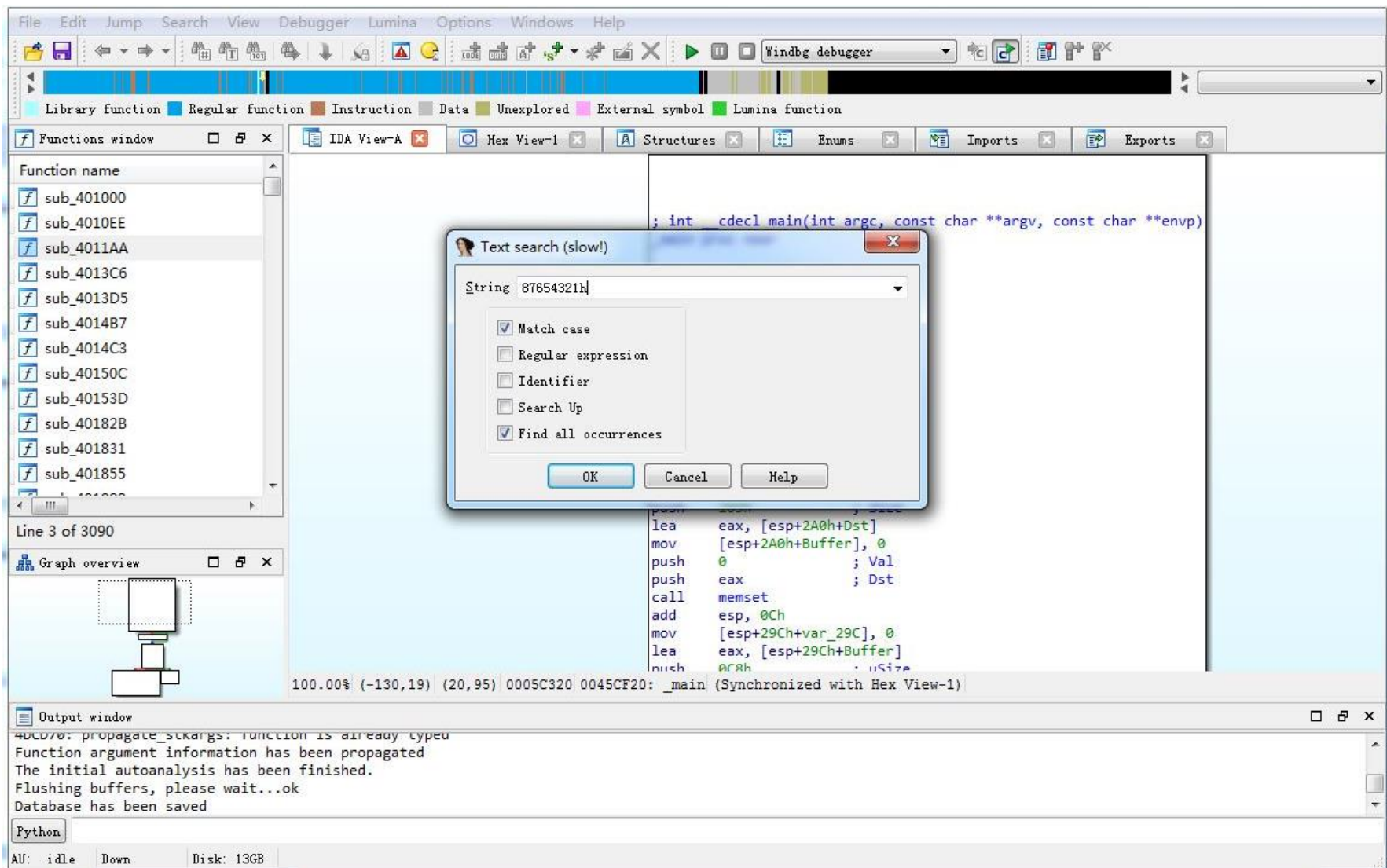
2、用IDA Pro打开守护神文件，等待分析完成。



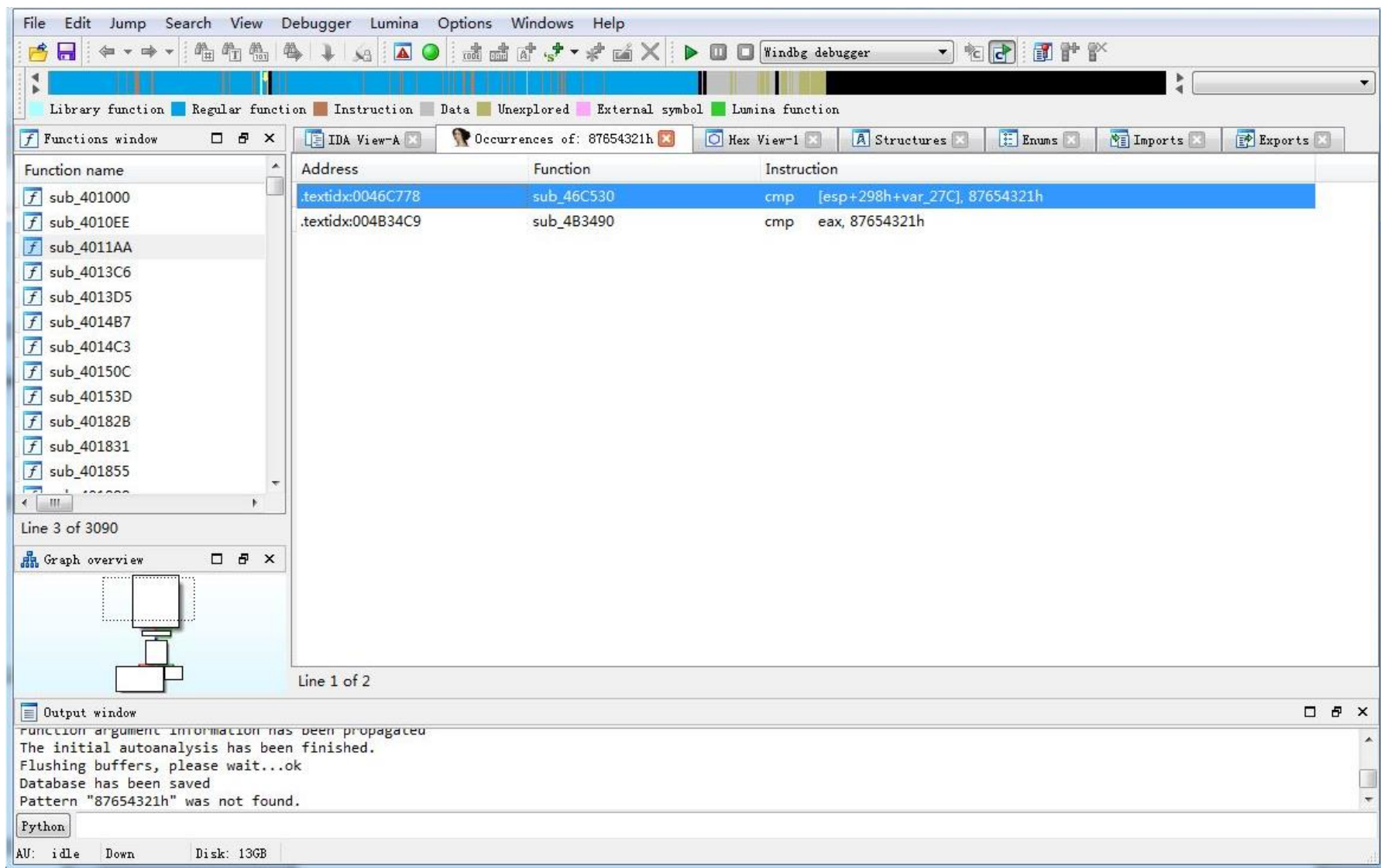
3、设置调试参数: -T Win7-64 11 -c C:\flexlm\license.dat



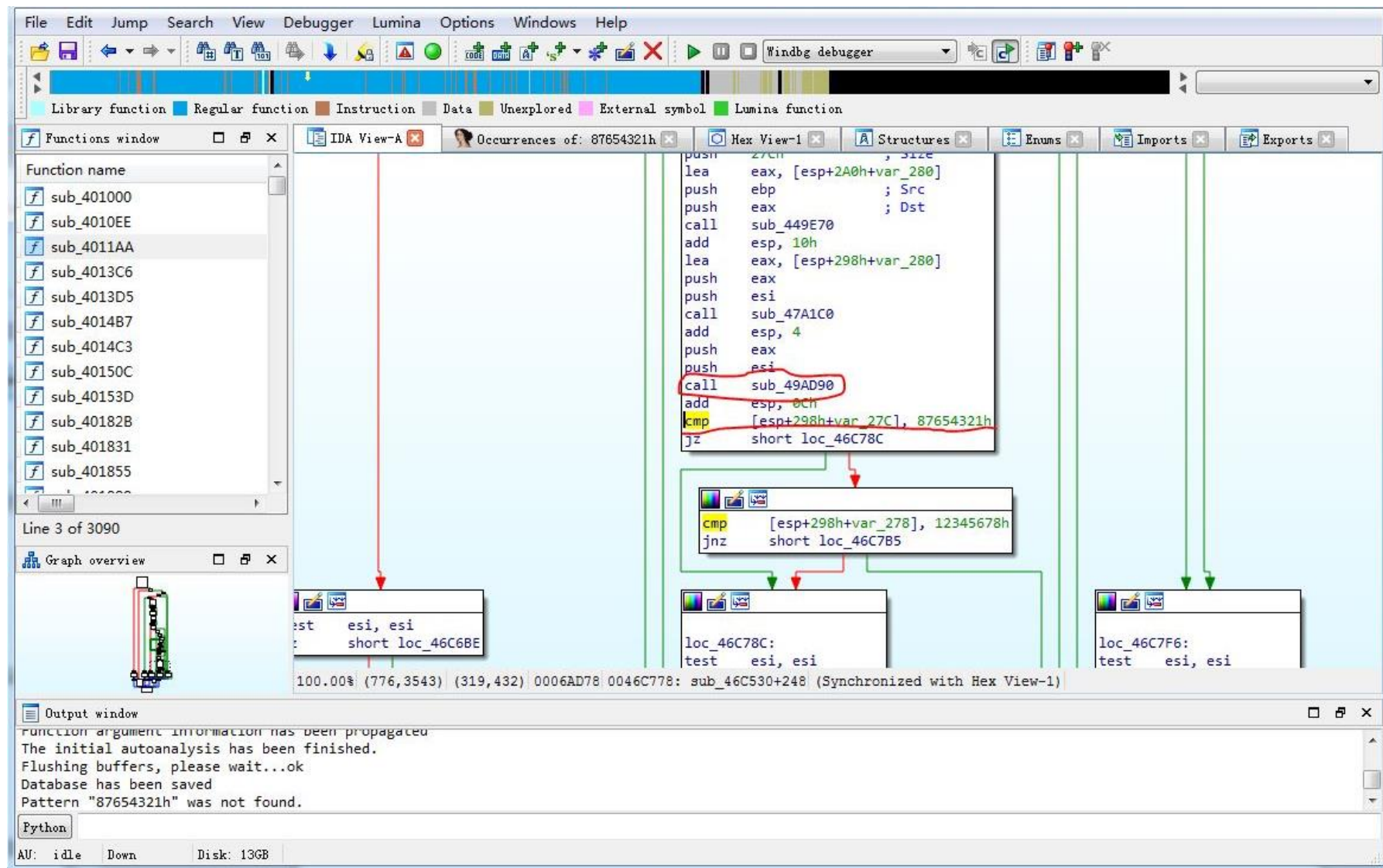
4、在汇编窗口查找“87654321h”



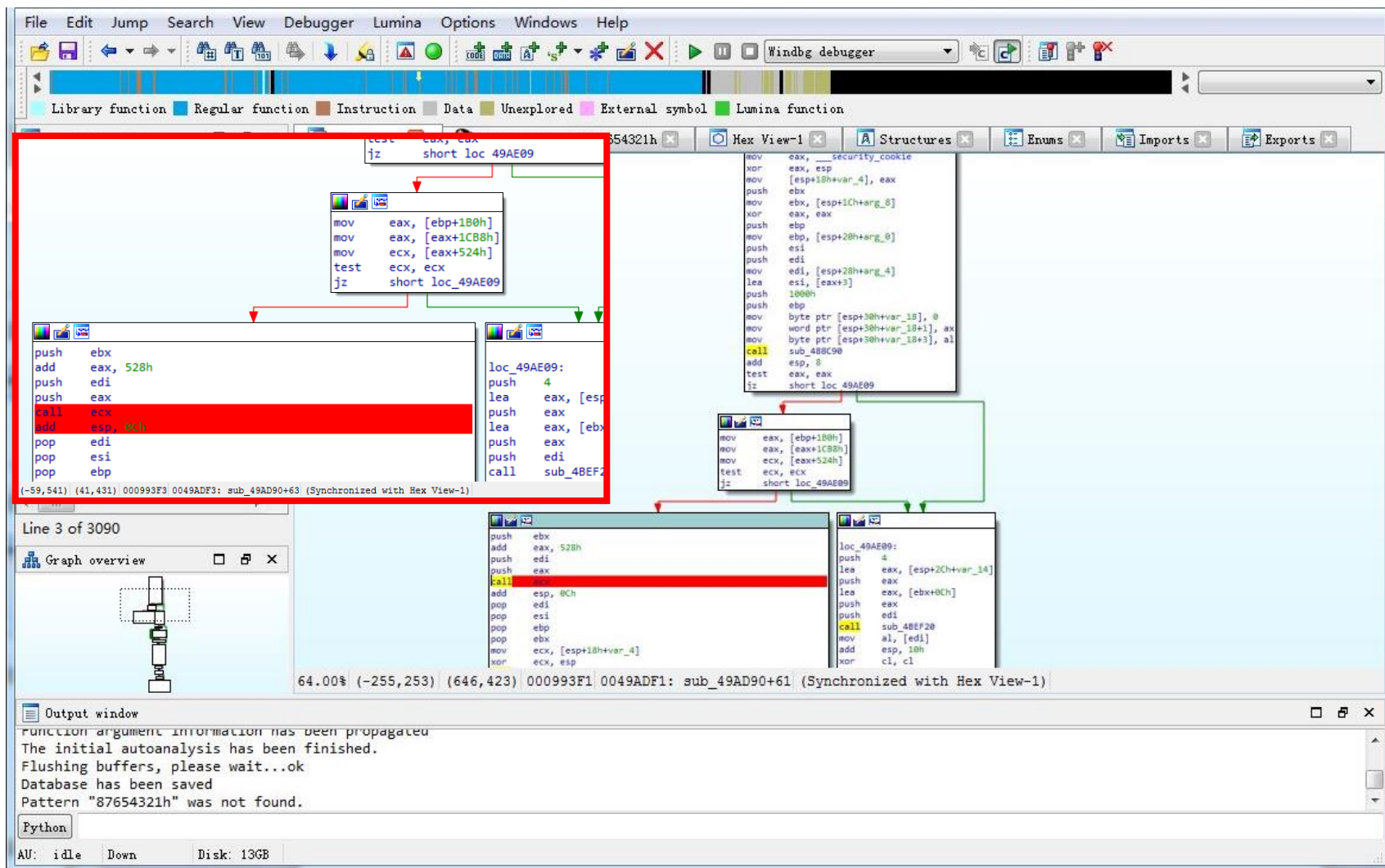
5、在搜索结果窗口双击第一个找到的“87654321h”



6、在汇编窗口找到“87654321h”上面紧挨着的那个call



7、双击上步查找的call后面的函数（上图示例为sub_49AD90），在流程图显示第二个“jz”语句下的第一call（一般为寄存器或地址表达式形式），按F2下断点，紧挨其后的语句也按F2下断点。



8、按F9开始调试守护神程序，直到在第一个call断点处停下，按F7步进入子函数

The screenshot displays the IDA Pro interface with the following components:

- Main Window (IDA View-EIP):** Shows assembly code for `sub_1131C10`. The code includes variable declarations (e.g., `var_2C= dword ptr -2Ch`) and a `push ebp` instruction. The status bar indicates the current instruction is `sub_1131C10` at address `00001010`.
- Right-Hand Pane (IDA View-EIP):** Shows the disassembly of the function `sub_11CAD90+63`. The code includes a `call` instruction to `@_security_check_cookie@4`. The status bar indicates the current instruction is `sub_11CAD90+63` at address `000993F1`.
- Bottom Pane (Hex View-1):** Shows the hex view of the code. The status bar indicates the current instruction is `sub_1131000+40` at address `00000440`.
- Output Window:** Displays the PDB file path and a message about the PDB file not being found. The message is: `PDB: using PDBIDA provider`, `PDB: downloading http://msdl.microsoft.com/download/symbols/CITRIX.pdb/B8DCCF349D7C47F089AB11621971DE922/CITRIX.pdb => C:\Users\Myron\AppData\Local\Temp\ida\CITRIX.pdb\B8DCCF349D7C47F089AB11621971DE922\CITRIX.pdb`, `Could not find PDB file 'CITRIX.pdb'.`, `Please check PDBSYM_SYMPATH in pdb.cfgExpected data back.`

9、用Ctrl+鼠标滚轮缩放流程图，找到一长串有“time”注释后面的第二个JMP语句，F2下断点

The screenshot displays the IDA Pro interface with the following components:

- File Edit Jump Search View Debugger Lumina Options Windows Help** (Menu bar)
- Library function Regular function Instruction Data Unexplored External symbol Lumina function** (Color-coded legend)
- Debug View Structures Enums** (Tabbed interface)
- IDA View-EIP** (Main disassembly window)
- Hex View-1** (Hexadecimal view of the code)
- Output window** (Bottom panel showing PDB status and disk space)

The assembly code in the disassembly window is as follows:

```
.text:0113202E mov [ecx+4], esi
.text:01132031 push 0
.text:01132033 call sub_11318F0
.text:01132038 add esp, 4
.text:0113203B and eax, 0FFh
.text:01132040 and edx, 0
.text:01132043 xor eax, 0A9h
.text:01132048 xor edx, 0
.text:0113204B mov ecx, 1
.text:01132050 imul ecx, 3
.text:01132053 mov edx, [ebp+var_1C]
.text:01132056 mov [edx+ecx+8], al
.text:0113205A jmp short loc_1132061
```

The flowchart shows the execution path leading to the highlighted block of code. The highlighted block contains the following instructions:

```
.text:01132061 loc_1132061:
.text:01132061 mov [ebp+var_24], 0
.text:01132068 jmp short loc_1132073
```

The flowchart also shows the execution path leading to the next block of code:

```
.text:01132073 loc_1132073:
.text:01132073 cmp [ebp+var_24], 0Ah
.text:01132077 jge short loc_11320DD
```

The output window shows the following message:

```
PDB: using PDBIDA provider
PDB: downloading http://msdl.microsoft.com/download/symbols/CITRIX.pdb/B8DCCF349D7C47F089AB11621971DE922/CITRIX.pdb => C:\Users\Myron\AppData\Local\Temp\ida\CITRIX.pdb\B8DCCF349D7C47F089AB11621971DE922\CITRIX.pdb
Could not find PDB file 'CITRIX.pdb'.
Please check PDBSYM_SYMPATH in pdb.cfgExpected data back.
```

10、按F9继续调试，程序会停在上步设置的断点处。在输出窗口的WinDBG命令输入框中输入命令：**dd ebp** (dd后面具体要输入什么要根据JMP语句的上一条来确定)

The screenshot displays the WinDBG interface with three main panes: Assembly, Disassembly, and Memory Dump.

Assembly Pane:

- .text:01132050 imul ecx, 3
- .text:01132053 mov edx, [ebp+var_1C]
- .text:01132056 mov [edx+ecx+8], al
- .text:0113205A jmp short loc_1132061

Disassembly Pane:

- .text:01132061
- .text:01132061 loc_1132061:
- .text:01132061 mov [ebp+var_24], 0
- .text:01132068 jmp short loc_1132073

Memory Dump Pane:

WINDBG>dd ebp

0042C840	0089C9D0	0042C800	00000004	00000000
0042C850	0042C83C	00000004	537FF8B0	6F07E453
0042C7E0	008A6630	011CADF3	008A5CF8	008A5CCC
0042C7F0	0042C854	00902EF0	008A6630	00000000
0042C800	008A6398	00000000	00525400	0042C854
0042C810	52544943	00005849	B2668A4B	011C9003
0042C820	008A6630	008A5CCC	0042C854	00902EF0
0042C830	00000000	008A6398	008A6630	72985058
0042C840	0089C9D0	0042C800	00000004	00000000
0042C850	0042C83C	00000004	537FF8B0	6F07E453
008A5CF8	00000000	0000003C	1620EE62	9A917009
008A5D08	5B6B50ED	00000000	00000000	008E6548
008A5D18	00000000	011322A0	00904E78	00000000
008A5D28	00000000	00000000	00000000	00000000
008A5D38	0119D490	00001880	00000000	00000000
008A5D48	00000000	00000000	00000000	00000000
008A5D58	00000000	00000000	00000000	00000000
008A5D68	00000000	00000000	00000000	00000000

WINDBG

11、现在注意看“dd ebp”后的输出结果，（结合图示）第三个地址EBP+8（008a5cf8）里是存着job结构的起始地址。（结合图示）在输出窗口的WinDBG命令输入框中输入命令：“dd 008a5cf8”后就看到了job结构的值，（按顺序从job, job+4, job+8, job+c, job+10h ...）。此时我们暂不修改任何值。

The screenshot displays the WinDBG interface with three main panes. The top pane shows assembly code with instructions at addresses 01132053, 01132056, and 0113205A. The middle pane shows the disassembly of the code at address 01132061, including a jump instruction to loc_1132073. The bottom pane shows the disassembly of the code at address 01132073, including a compare and jump instruction. The right pane shows the memory dump for the command 'dd ebp', displaying a list of memory addresses and their corresponding values. The third address, 008a5cf8, is highlighted, indicating the start of the 'job' structure. The bottom status bar shows the current register values: ecx, [ebp+var_24] and the address 011320DD.

```
.text:01132053 mov     edx, [ebp+var_1C]
.text:01132056 mov     [edx+ecx+8], al
.text:0113205A jmp     short loc_1132061

.text:01132061
.text:01132061 loc_1132061:
.text:01132061 mov     [ebp+var_24], 0
.text:01132068 jmp     short loc_1132073

.text:01132073
.text:01132073 loc_1132073:
.text:01132073 cmp     [ebp+var_24], 0Ah
.text:01132077 jge     short loc_11320DD

v     ecx, [ebp+var_24]
.text:011320DD
```

WINDBG>dd ebp

0042c7e0	008a6630	011cadf3	008a5cf8	008a5ccc
0042c7f0	0042c854	00902ef0	008a6630	00000000
0042c800	008a6398	00000000	00525400	0042c854
0042c810	52544943	00005849	b2668a4b	011c9003
0042c820	008a6630	008a5ccc	0042c854	00902ef0
0042c830	00000000	008a6398	008a6630	72985058
0042c840	0089c9d0	0042c800	00000004	00000000
0042c850	0042c83c	00000004	537ff8b0	6f07e453

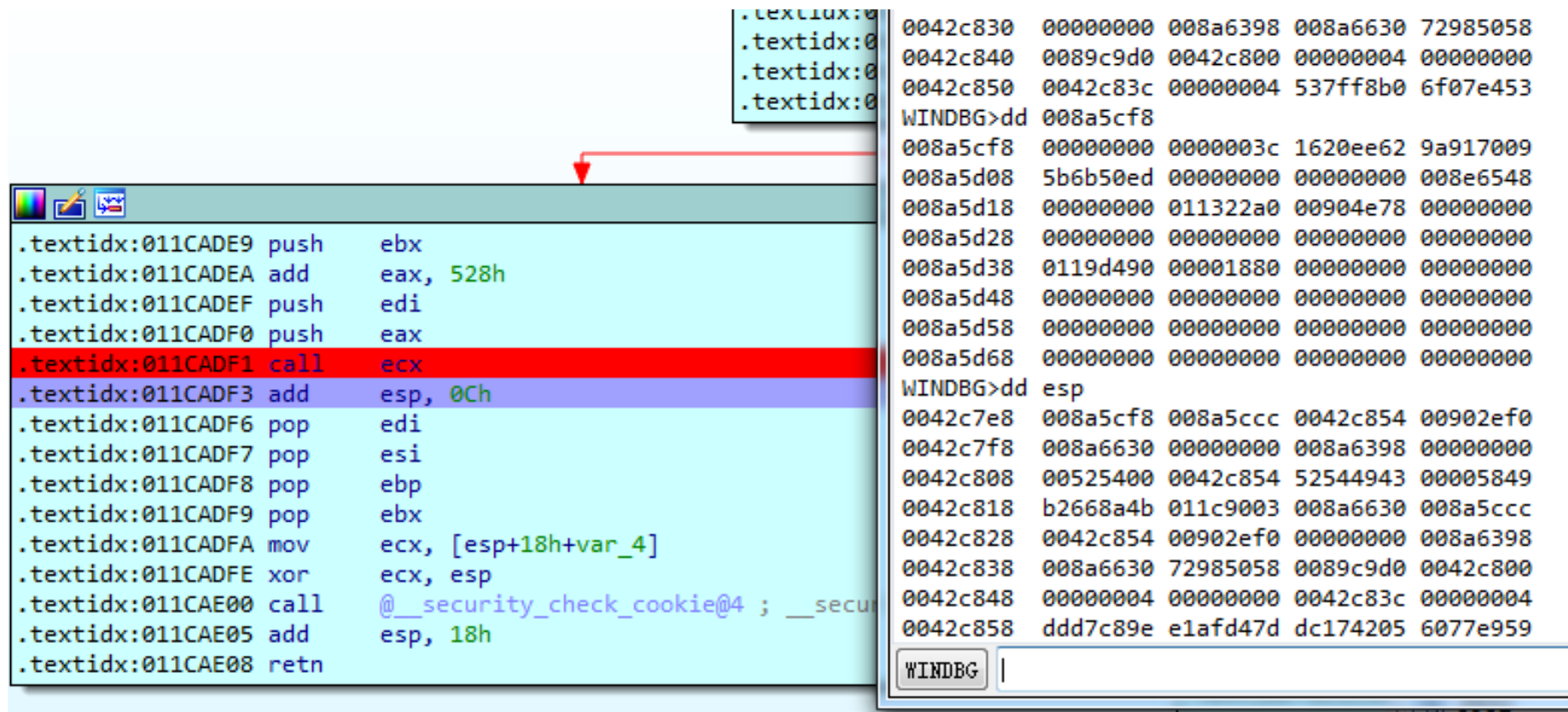
WINDBG>dd 008a5cf8

008a5cf8	00000000	0000003c	1620ee62	9a917009
008a5d08	5b6b50ed	00000000	00000000	008e6548
008a5d18	00000000	011322a0	00904e78	00000000
008a5d28	00000000	00000000	00000000	00000000
008a5d38	0119d490	00001880	00000000	00000000
008a5d48	00000000	00000000	00000000	00000000
008a5d58	00000000	00000000	00000000	00000000
008a5d68	00000000	00000000	00000000	00000000

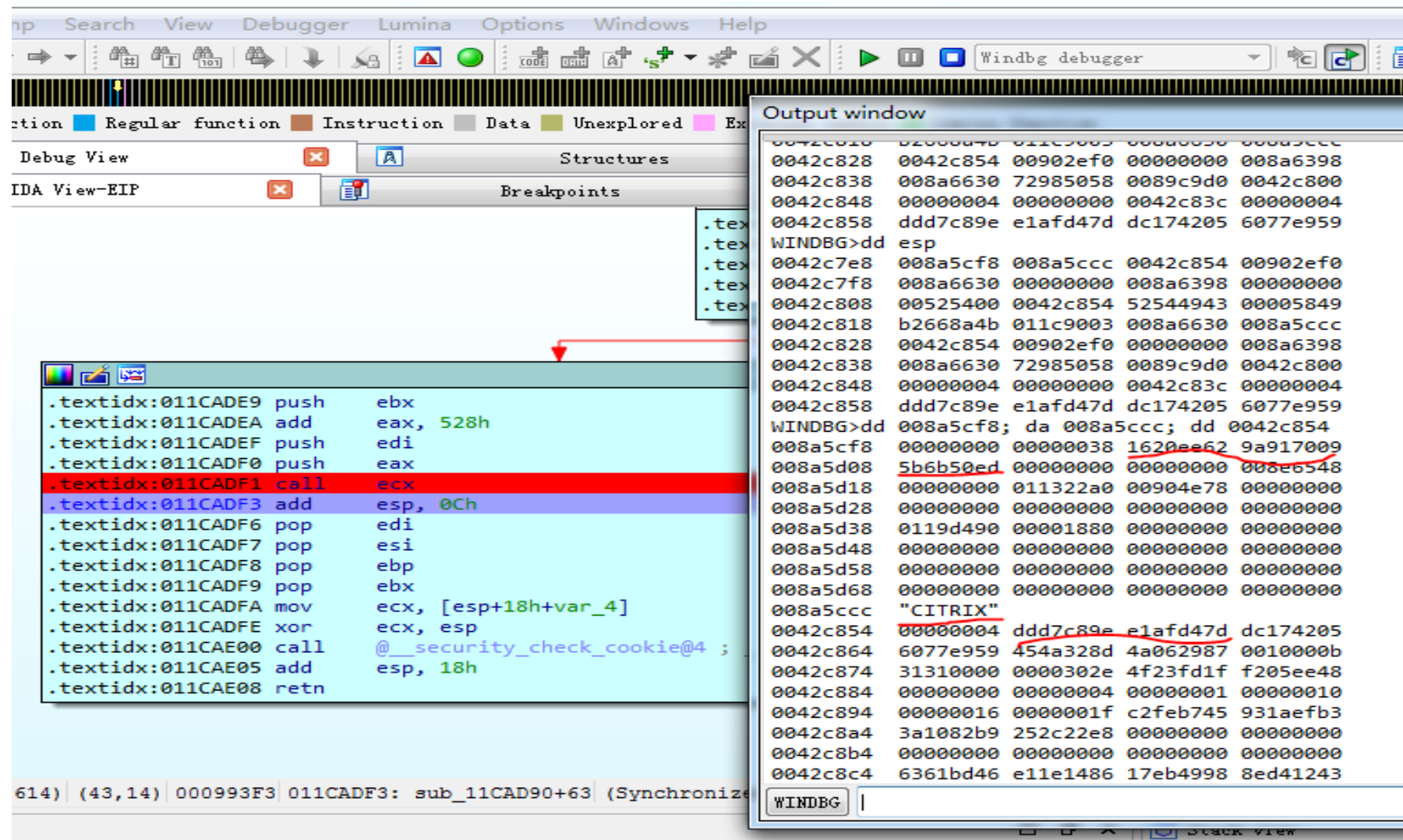
WINDBG |

4728

12、按F9继续调试，程序将停在call语句下面的语句，因为我们事先设置了断点。现在注意看WinDBG输入“dd esp”后的输出结果（注意不是“dd ebp”了，也是在假设没有上一步修改job值的情况下），（结合图示）此时第一个地址ESP（008a5cf8）里是存着Job结构的起始地址，第二个地址ESP+4（008a5ccc）里是存着Vendor名字的地址，第三个地址ESP+8里（0042c854）是存着Data结构的起始地址。



13、现在我们来看看其结构。现在WinDBG输入“dd 008a5cf8; da 008a5ccc; dd 0042c854”后的输出结果。（结合图示）此时第一个地址ESP（008a5cf8）里是存着Job结构的起始地址，第二个地址ESP+4（008a5ccc）里是存着Vendor名字的地址，第三个地址ESP+8里（0042c854）是存着Data结构的地址。



14、有了这些值，我们就可以利用工具calcseed.exe来计算加密种子点的值了。

dx:011CADE9 push ebx
dx:011CADEA add eax, 528h
dx:011CADEF push edi

calcseed - Calculate FLEXlm Encryption Seeds

Values from vendorcode		Derived encryption	
data[0]	ddd7c89e	encseed[0]	d48726fc
data[1]	e1afd47d	encseed[1]	e8ff3a1f

vendorname: C:\[redacted] Selector: a[0], a[1], a[9], a[4]

Values from job		XOR value
job+0x00	1620ee62	0950ee62
job+0x04	9a917009	
job+0x10	5b6b50ed	

WINDBG>dd 008a5cf8; da 008a5ccc; dd 0042c854

008a5cf8 00000000 00000038 1620ee62 9a917009
008a5d08 5b6b50ed 00000000 00000000 008e6548
008a5d18 00000000 011322a0 00904e78 00000000
008a5d28 00000000 00000000 00000000 00000000
008a5d38 0119d490 00001880 00000000 00000000
008a5d48 00000000 00000000 00000000 00000000
008a5d58 00000000 00000000 00000000 00000000
008a5d68 00000000 00000000 00000000 00000000
008a5ccc "C:\[redacted]
0042c854 00000004 ddd7c89e e1afd47d dc174205
0042c864 6077e959 454a328d 4a062987 0010000b
0042c874 31310000 0000302e 4f23fd1f f205ee48
0042c884 00000000 00000004 00000001 00000010
0042c894 00000016 0000001f c2feb745 931aefb3
0042c8a4 3a1082b9 252c22e8 00000000 00000000
0042c8b4 00000000 00000000 00000000 00000000
0042c8c4 6361bd46 e11e1486 17eb4998 8ed41243

WINDBG

15、按F9两次继续调试，程序会停在JMP断点处。在输出窗口的WinDBG命令输入框中输入命令：`dd ebp` (`dd`后面具体要输入什么要根据JMP语句的上一条来确定)。其中`ebp+8`中的地址对应的应该的是与系统时间有关的随机数，也可能是job结构的值。在输出窗口的WinDBG命令输入框中输入命令“`dd 008a5cf8`”就看到其值。

The screenshot displays the WinDBG interface with three main panels. The top-left panel shows assembly code with instructions: `imul ecx, 3`, `mov edx, [ebp+var_1C]`, `mov [edx+ecx+8], al`, and `jmp short loc_1132061`. A blue arrow points from the `jmp` instruction to the disassembly panel. The middle-left panel shows the disassembly of `loc_1132061`, including `mov [ebp+var_24], 0` and `jmp short loc_1132073`. Another blue arrow points from this `jmp` to the bottom-left panel. The bottom-left panel shows the disassembly of `loc_1132073`, including `cmp [ebp+var_24], 0Ah` and `jge short loc_11320DD`. The right panel shows the memory dump. The command `WINDBG>dd ebp` has been entered, resulting in a list of memory addresses and their values. Two values are highlighted with red boxes: `1523ed61 9992730a` at address `00595e38` and `586853ee` at address `00595e48`. The command `WINDBG>dd 00595e38` has also been entered, showing the same two highlighted values. The bottom of the right panel shows the `WINDBG` command input box.

```
.text:01132050 imul    ecx, 3
.text:01132053 mov     edx, [ebp+var_1C]
.text:01132056 mov     [edx+ecx+8], al
.text:0113205A jmp     short loc_1132061

.text:01132061
.text:01132061 loc_1132061:
.text:01132061 mov     [ebp+var_24], 0
.text:01132068 jmp     short loc_1132073

.text:01132073
.text:01132073 loc_1132073:
.text:01132073 cmp     [ebp+var_24], 0Ah
.text:01132077 jge     short loc_11320DD

WINDBG>dd ebp
0017c5b8 00596770 011cadf3 00595e38 00595e0c
0017c5c8 0017c62c 005f5428 00596770 00000000
0017c5d8 005964d8 00000000 00525400 0017c62c
0017c5e8 52544943 00005849 1f23bd7e 011c9003
0017c5f8 00596770 00595e0c 0017c62c 005f5428
0017c608 00000000 005964d8 00596770 0017c70c
0017c618 00580000 7753a969 0060cfc0 00000020
0017c628 774e300d 00000004 537ff8b0 6f07e453
WINDBG>dd 00595e38
00595e38 00000000 00c400e1 1523ed61 9992730a
00595e48 586853ee 00000000 00000000 005d6688
00595e58 00000000 011322a0 005f4fb8 00000000
00595e68 00000000 00000000 00000000 00000000
00595e78 0119d490 00001880 00000000 00000000
00595e88 00000000 00000000 00000000 00000000
00595e98 00000000 00000000 00000000 00000000
00595ea8 00000000 00000000 00000000 00000000

WINDBG
```

16、按照神人老外的研究成果，其中ebp+8中的地址对应的应该是与系统时间有关的随机数，也可能是job结构的值（本人不确定）。在此直接修改job+8，job+c，job+10h的值为0，在后面一步就直接得到加密种子点的值。根据上步说明的地址，在输出窗口的WinDBG命令输入框中（结合图示）输入命令“ed 00595e38+8 0; ed 00595e38+c 0; ed 00595e38+10h 0”修改其值。然后可以确认修改结果：dd 008a5cf8

The screenshot displays the WinDBG interface with the assembly window and the dump window.

Assembly Window:

```
.text:01132053 mov     eax, [ebp+var_1c]
.text:01132056 mov     [edx+ecx+8], al
.text:0113205A jmp      short loc_1132061

.text:01132061 loc_1132061:
.text:01132061 mov     [ebp+var_24], 0
.text:01132068 jmp      short loc_1132073

.text:01132073 loc_1132073:
.text:01132073 cmp     [ebp+var_24], 0Ah
.text:01132077 jge     short loc_11320DD
```

Dump Window:

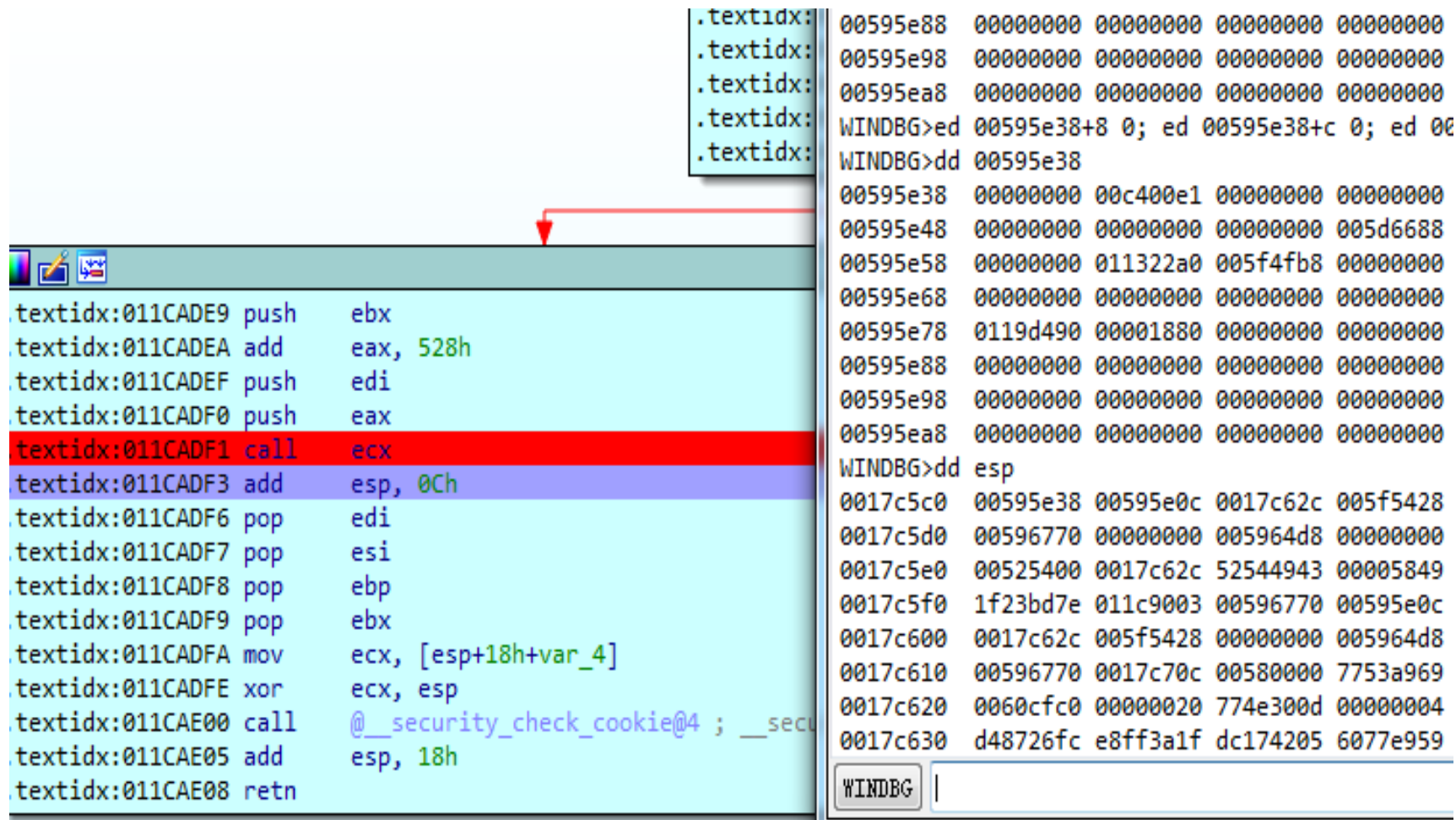
Initial state (before modification):

```
WINDBG>dd 00595e38
00595e38  00000000 00c400e1 1523ed61 9992730a
00595e48  586853ee 00000000 00000000 005d6688
00595e58  00000000 011322a0 005f4fb8 00000000
00595e68  00000000 00000000 00000000 00000000
00595e78  0119d490 00001880 00000000 00000000
00595e88  00000000 00000000 00000000 00000000
00595e98  00000000 00000000 00000000 00000000
00595ea8  00000000 00000000 00000000 00000000
```

Modified state (after modification):

```
WINDBG>ed 00595e38+8 0; ed 00595e38+c 0; ed 00595e38+10h 0
WINDBG>dd 00595e38
00595e38  00000000 00c400e1 00000000 00000000
00595e48  00000000 00000000 00000000 005d6688
00595e58  00000000 011322a0 005f4fb8 00000000
00595e68  00000000 00000000 00000000 00000000
00595e78  0119d490 00001880 00000000 00000000
00595e88  00000000 00000000 00000000 00000000
00595e98  00000000 00000000 00000000 00000000
00595ea8  00000000 00000000 00000000 00000000
```

17、确认修改正确后按F9继续调试，程序将停在call语句下面的语句。现在注意看WinDBG输入“dd esp”后的输出结果（注意不是“dd ebp”），（结合图示）根据在第一次没有修改的情况下，理论上此时第一个地址ESP（00595e38）里是存着Job结构的起始地址，第二个地址ESP+4（00595e0c）里是存着Vendor名字的地址，第三个地址ESP+8里（0017c62c）是存着Data结构的起始地址。

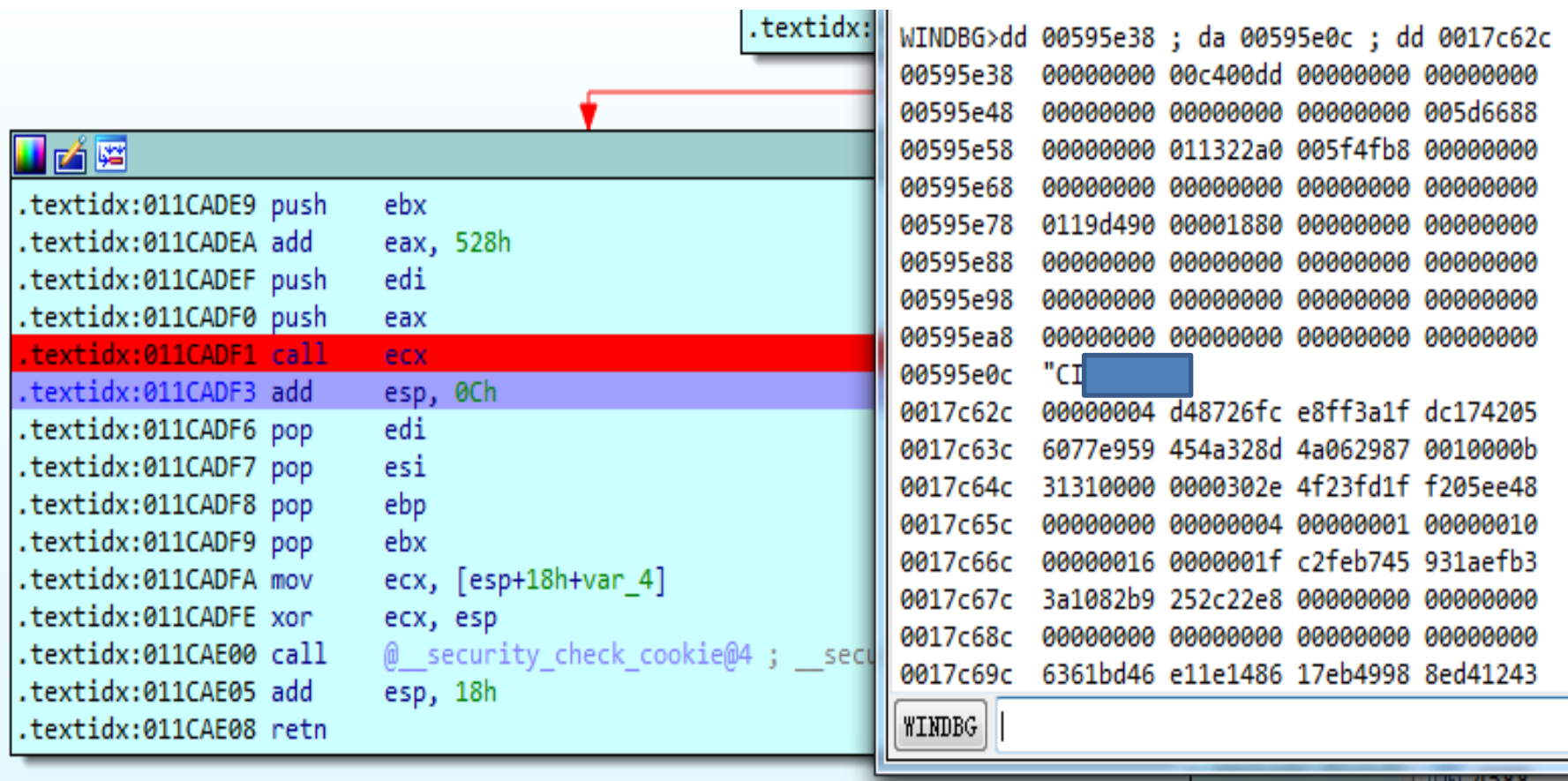


```
.textidx:011CADE9 push    ebx
.textidx:011CADEA add     eax, 528h
.textidx:011CADEF push    edi
.textidx:011CADF0 push    eax
.textidx:011CADF1 call    ecx
.textidx:011CADF3 add     esp, 0Ch
.textidx:011CADF6 pop     edi
.textidx:011CADF7 pop     esi
.textidx:011CADF8 pop     ebp
.textidx:011CADF9 pop     ebx
.textidx:011CADFA mov     ecx, [esp+18h+var_4]
.textidx:011CADFE xor     ecx, esp
.textidx:011CAE00 call    @__security_check_cookie@4 ; __secu
.textidx:011CAE05 add     esp, 18h
.textidx:011CAE08 retn
```

```
00595e38 00000000 00c400e1 00000000 00000000
00595e48 00000000 00000000 00000000 005d6688
00595e58 00000000 011322a0 005f4fb8 00000000
00595e68 00000000 00000000 00000000 00000000
00595e78 0119d490 00001880 00000000 00000000
00595e88 00000000 00000000 00000000 00000000
00595e98 00000000 00000000 00000000 00000000
00595ea8 00000000 00000000 00000000 00000000
0017c5c0 00595e38 00595e0c 0017c62c 005f5428
0017c5d0 00596770 00000000 005964d8 00000000
0017c5e0 00525400 0017c62c 52544943 00005849
0017c5f0 1f23bd7e 011c9003 00596770 00595e0c
0017c600 0017c62c 005f5428 00000000 005964d8
0017c610 00596770 0017c70c 00580000 7753a969
0017c620 0060cfc0 00000020 774e300d 00000004
0017c630 d48726fc e8ff3a1f dc174205 6077e959
```

WINDBG

18、实际上由于我们的修改，再次用命令“dd 00595e38 ; da 00595e0c ; dd 0017c62c”查看，它们的值已经发生了改变（尽管其结构没变，这也是其要求的基本条件）。



The screenshot shows the WinDBG interface. On the left, the assembly window displays code from .textidx:011CADE9 to .textidx:011CAE08. The instruction at .textidx:011CADF1, 'call ecx', is highlighted in red. A red arrow points from this instruction to the memory dump on the right. The memory dump shows the state of memory after the 'call ecx' instruction. The address 00595e0c contains the string 'CI' followed by a blue box. The address 0017c62c contains a sequence of hexadecimal values.

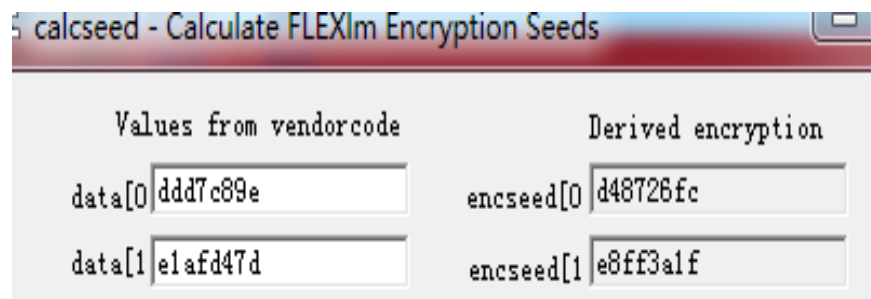
```
.textidx:011CADE9 push    ebx
.textidx:011CADEA add     eax, 528h
.textidx:011CADEF push    edi
.textidx:011CADF0 push    eax
.textidx:011CADF1 call     ecx
.textidx:011CADF3 add     esp, 0Ch
.textidx:011CADF6 pop     edi
.textidx:011CADF7 pop     esi
.textidx:011CADF8 pop     ebp
.textidx:011CADF9 pop     ebx
.textidx:011CADFA mov     ecx, [esp+18h+var_4]
.textidx:011CADFE xor     ecx, esp
.textidx:011CAE00 call     @__security_check_cookie@4 ; __sec
.textidx:011CAE05 add     esp, 18h
.textidx:011CAE08 retn
```

WINDBG>dd 00595e38 ; da 00595e0c ; dd 0017c62c

00595e38	00000000	00c400dd	00000000	00000000
00595e48	00000000	00000000	00000000	005d6688
00595e58	00000000	011322a0	005f4fb8	00000000
00595e68	00000000	00000000	00000000	00000000
00595e78	0119d490	00001880	00000000	00000000
00595e88	00000000	00000000	00000000	00000000
00595e98	00000000	00000000	00000000	00000000
00595ea8	00000000	00000000	00000000	00000000
00595e0c	"CI [redacted]"			
0017c62c	00000004	d48726fc	e8ff3a1f	dc174205
0017c63c	6077e959	454a328d	4a062987	0010000b
0017c64c	31310000	0000302e	4f23fd1f	f205ee48
0017c65c	00000000	00000004	00000001	00000010
0017c66c	00000016	0000001f	c2feb745	931aefb3
0017c67c	3a1082b9	252c22e8	00000000	00000000
0017c68c	00000000	00000000	00000000	00000000
0017c69c	6361bd46	e11e1486	17eb4998	8ed41243

WINDBG |

19、现在WinDBG输入“dd 0017c62c”后（ESP+8），其data[0]和data[1]位置的结果即为两个加密种子点值。与工具计算出的值一致



```
.textidx:011CADE9 push    ebx
.textidx:011CADEA add     eax, 528h
.textidx:011CADEF push    edi
.textidx:011CADF0 push    eax
.textidx:011CADF1 call    ecx
.textidx:011CADF3 add     esp, 0Ch
.textidx:011CADF6 pop     edi
.textidx:011CADF7 pop     esi
.textidx:011CADF8 pop     ebp
.textidx:011CADF9 pop     ebx
.textidx:011CADFA mov     ecx, [esp+18h+var_4]
.textidx:011CADFE xor     ecx, esp
.textidx:011CAE00 call    @__security_check_cookie
.textidx:011CAE05 add     esp, 18h
.textidx:011CAE08 retn
```

```
0017c5c0 00595e38 00595e0c 0017c62c 005f5428
0017c5d0 00596770 00000000 005964d8 00000000
0017c5e0 00525400 0017c62c 52544943 00005849
0017c5f0 1f23bd7e 011c9003 00596770 00595e0c
0017c600 0017c62c 005f5428 00000000 005964d8
0017c610 00596770 0017c70c 00580000 7753a969
0017c620 0060cfc0 00000020 774e300d 00000004
0017c630 d48726fc e8ff3a1f dc174205 6077e959
0017c640 00000000 00000000 00000000 00000000
0017c650 00000000 00000004 00000001 00000010
0017c660 00000016 0000001f c2feb745 931aefb3
0017c670 3a1082b9 252c22e8 00000000 00000000
0017c680 00000000 00000000 00000000 00000000
0017c690 6361bd46 e11e1486 17eb4998 8ed41243
```

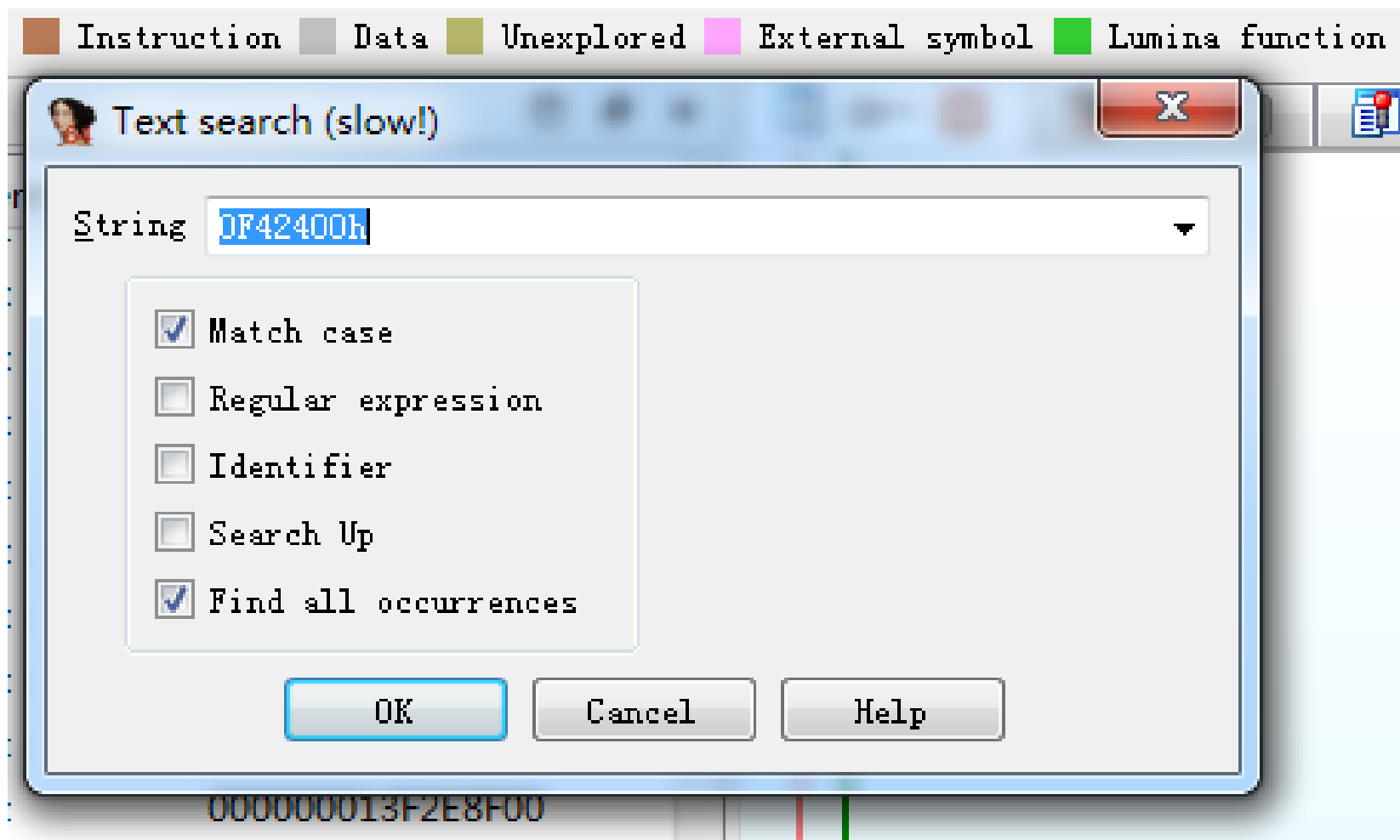
WINDBG

Flexnet Publisher V11.16及其以下版本加密种子点恢复方法

By: YangMyron

标志0F42400h定位法

1、在汇编窗口查找“0F42400h”



2、在搜索结果窗口双击找到的第一和第二两个“0F42400h”，并分别下断点。

The screenshot displays a debugger's instruction list and a search results window. The instruction list at the top shows various memory load instructions (leal) using the constant 0F42400h. Below it, a search results window is open, showing two absolute addresses where the search term was found.

Address	Function	Instruction
.textidx:000000013F21A880	sub_13F21A030	leal ecx, [rdx+0F42400h]
.textidx:000000013F21A91C	sub_13F21A030	leal ecx, [r8+0F42400h]
.textidx:000000013F21C4F3	sub_13F21C4C0	leal eax, [rcx+0F42400h]
.textidx:000000013F21D680	sub_13F21D610	leal eax, [rcx+0F42400h]
.textidx:000000013F21E0C5	sub_13F21D610	leal eax, [rcx+0F42400h]
.textidx:000000013F21E285	sub_13F21E180	leal eax, [rdx+0F42400h]
.textidx:000000013F21E2D0	sub_13F21E180	leal eax, [rdx+0F42400h]
.textidx:000000013F21F224	sub_13F21EE20	leal eax, [rcx+0F42400h]
.textidx:000000013F21F270	sub_13F21EE20	leal eax, [rcx+0F42400h]
.textidx:000000013F21F2BC	sub_13F21EE20	leal eax, [rsi+0F42400h]
.textidx:000000013F21F4B6	sub_13F21EE20	leal eax, [rdx+0F42400h]

Type	Location	Pa
Abs	0x13F21A880	
Abs	0x13F21A91C	

Line 1 of 52

3、设置好调试参数，不断F9后，如果能在断点处停下，第一处停下的rdx（x86的edx）值就是加密种子点1；第二处停下的r8d（x86的edx）值就是加密种子点2。（具体情况看本行命令里的寄存器是谁）。如果停不住就是该守护神不支持Non-TRL加密模式。

The screenshot displays a debugger interface with assembly code and general registers. The assembly code is organized into three sections, each with a red box around a specific instruction. The first section shows the instruction `lea ecx, [rdx+0F42400h]`. The second section shows the instruction `lea ecx, [r8+0F42400h]`. The third section shows the instruction `sar r8d, 18h`. The general registers window on the right shows the values of RAX, RBX, RCX, RDX, RSI, RDI, RBP, RSP, and RIP. The value of RDX is highlighted in red in the first section, and the value of R8 is highlighted in red in the second section.

Assembly code sections:

```
.textidx:000000013FDCA880  
.textidx:000000013FDCA880 loc_13FDCA880:  
.textidx:000000013FDCA880 lea ecx, [rdx+0F42400h]  
.textidx:000000013FDCA886 cmp ecx, 1E84800h  
.textidx:000000013FDCA88C jbe short loc_13FDCA896
```

```
.textidx:000000013FDCA911 mov ecx, r8d  
.textidx:000000013FDCA914 sar ecx, 10h  
.textidx:000000013FDCA917 xor [rax], cl  
.textidx:000000013FDCA919 inc rax
```

```
.textidx:000000013FDCA91C  
.textidx:000000013FDCA91C loc_13FDCA91C:  
.textidx:000000013FDCA91C lea ecx, [r8+0F42400h]  
.textidx:000000013FDCA923 cmp ecx, 1E84800h  
.textidx:000000013FDCA929 jbe loc_13FDCAA31
```

```
.textidx:000000013FDCA92F sar r8d, 18h  
.textidx:000000013FDCA933 xor [rax], r8b  
.textidx:000000013FDCA936 jmp loc_13FDCAA31
```

General registers window:

Register	Value	Comment
RAX	000000013FEE943B	.data:byte_13FEE943B
RBX	000000000001EC840	debug005:000000000001EC840
RCX	00000000000001964	
RDX	00000000000019640328	
RSI	00000000000000000	
RDI	000000000000000001	
RBP	000000000001EB400	debug005:000000000001EB400
RSP	000000000001EB300	debug005:000000000001EB300
RIP	000000013FDCA880	sub_13FDCA030:loc_13FDCA880

General registers window (second section):

Register	Value	Comment
RAX	000000013FEE943F	.data:word_13FEE943E+1
RBX	000000000001EC840	debug005:000000000001EC840
RCX	00000000000001996	
RDX	000000000003F73D0	debug021:000000000003F73D0
RSI	00000000000000000	
RDI	000000000000000001	
RBP	000000000001EB400	debug005:000000000001EB400
RSP	000000000001EB300	debug005:000000000001EB300
RIP	000000013FDCA91C	sub_13FDCA030:loc_13FDCA91C
R8	00000000000019960716	
R9	00000000000000000	
R10	00000000000000000B	
R11	000000000000000001	
R12	00000000000491872	debug021:00000000000491872
R13	00000000000000017	