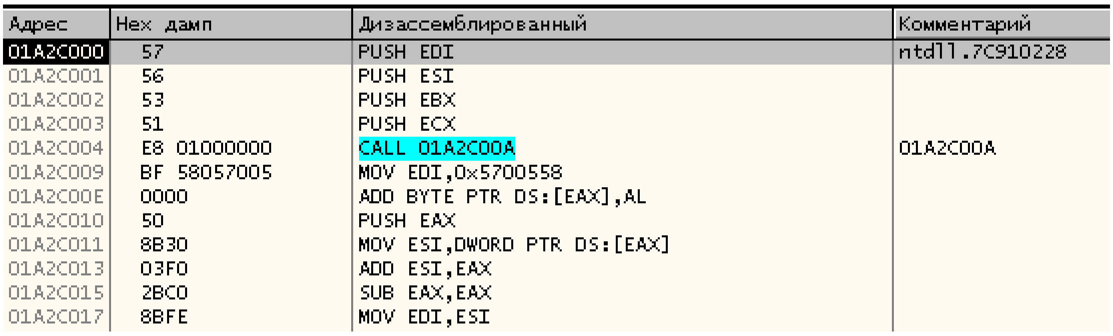
Introduction

Somehow I needed to hack a program that was protected by the HASP electronic key SRM . This key is provided by the program developer for only one machine. If the customer has several machines, then it is necessary to purchase several keys, which is very expensive. So I was asked to untie this program from the key so that it could be used on several machines. But, preliminary, it was necessary to unpack the file, which is covered with an envelope HASP SRM .

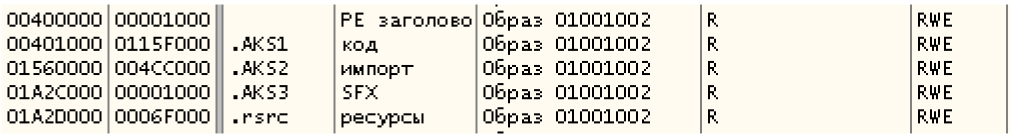
To remove the HASP envelope SRM It is necessary to have either the key itself or its emulator based on ***MultiKey*** . In this case, I had access to the key that was at the customer's.

Download our victim to the debugger OllyDbg , and the program stops on the ***Entry Point*** (***EP***) :



```
Адрес     Hex дамп         Дизассемблированный            Комментарий
01A2C000   57              PUSH EDI                       ntdll.7C910228
01A2C001   56              PUSH ESI
01A2C002   53              PUSH EBX
01A2C003   51              PUSH ECX
01A2C004   E8 01000000     CALL 01A2C00A                  01A2C00A
01A2C009   BF 58057005     MOV EDI,0x5700558
01A2C00E   0000            ADD BYTE PTR DS:[EAX],AL
01A2C010   50              PUSH EAX
01A2C011   8B30            MOV ESI,DWORD PTR DS:[EAX]
01A2C013   03F0            ADD ESI,EAX
01A2C015   2BC0            SUB EAX,EAX
01A2C017   8BFE            MOV EDI,ESI
```

And if we look at the memory card, we'll see the following:



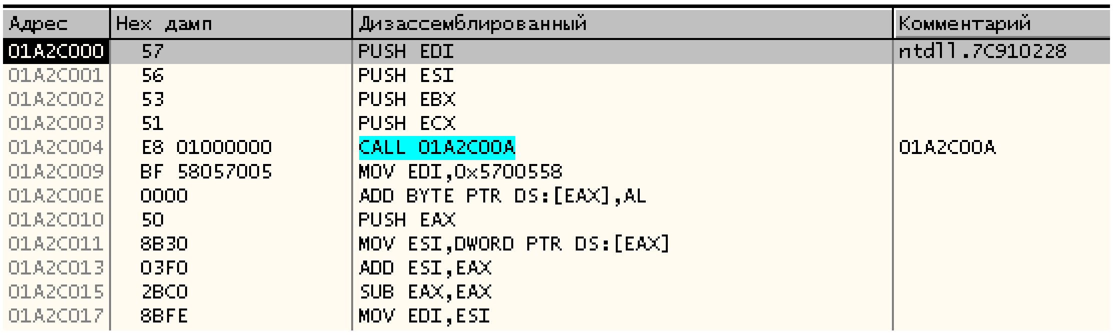
```
00400000 00001000        PE заголово Образ 01001002 R    RWE
00401000 0115F000  .AKS1 код         Образ 01001002 R    RWE
01560000 004CC000  .AKS2 импорт      Образ 01001002 R    RWE
01A2C000 00001000  .AKS3 SFX         Образ 01001002 R    RWE
01A2D000 0006F000  .rsrc ресурсы     Образ 01001002 R    RWE
```

Here we see sections of an unpacked file with names ***. AKS 1. AKS 2*** and ***. AKS 3*** , where the last section is the resource section ***. rsrc*** .

We are trying to start the program, and the program starts normally :

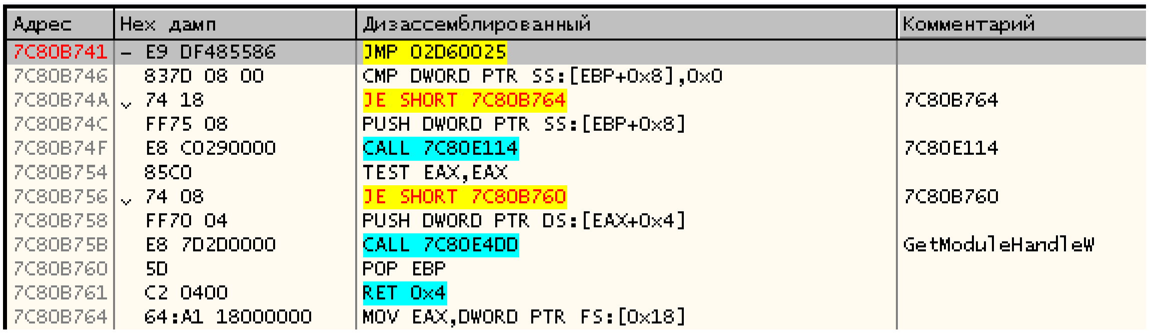
Search OEP

So, we reboot the program in the debugger OllyDbg , and it stops on the **EP** :



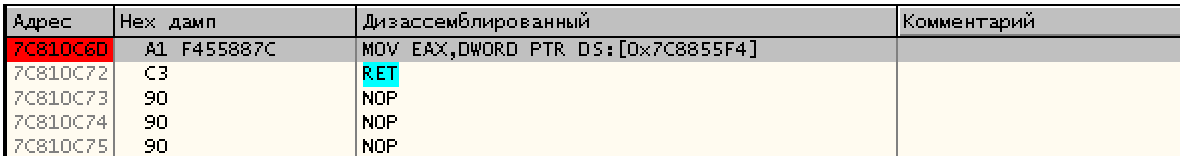
Адрес	Hex дамп	Дизассемблированный	Комментарий
01A2C000	57	PUSH EDI	ntdll.7C910228
01A2C001	56	PUSH ESI	
01A2C002	53	PUSH EBX	
01A2C003	51	PUSH ECX	
01A2C004	E8 01000000	CALL 01A2C00A	01A2C00A
01A2C009	BF 58057005	MOV EDI,0x5700558	
01A2C00E	0000	ADD BYTE PTR DS:[EAX],AL	
01A2C010	50	PUSH EAX	
01A2C011	8B30	MOV ESI,DWORD PTR DS:[EAX]	
01A2C013	03F0	ADD ESI,EAX	
01A2C015	2BC0	SUB EAX,EAX	
01A2C017	8BFE	MOV EDI,ESI	

To search for **OEP** (or addresses near **OEP**) it is recommended to use the function ***GetModuleHandleA*** , however this function is emulated by the protector:



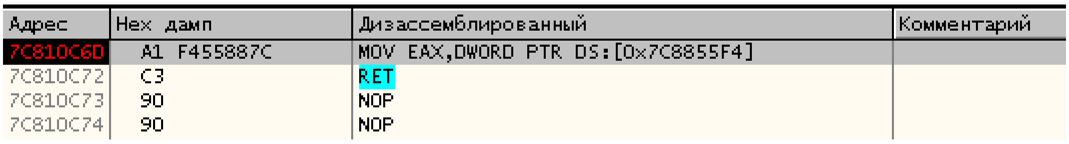
Адрес	Hex дамп	Дизассемблированный	Комментарий
7C80B741	- E9 DF485586	JMP 02D60025	
7C80B746	837D 08 00	CMP DWORD PTR SS:[EBP+0x8],0x0	
7C80B74A	⌄ 74 18	JE SHORT 7C80B764	7C80B764
7C80B74C	FF75 08	PUSH DWORD PTR SS:[EBP+0x8]	
7C80B74F	E8 C0290000	CALL 7C80E114	7C80E114
7C80B754	85C0	TEST EAX,EAX	
7C80B756	⌄ 74 08	JE SHORT 7C80B760	7C80B760
7C80B758	FF70 04	PUSH DWORD PTR DS:[EAX+0x4]	
7C80B75B	E8 7D2D0000	CALL 7C80E4DD	GetModuleHandleW
7C80B760	5D	POP EBP	
7C80B761	C2 0400	RET 0x4	
7C80B764	64:A1 18000000	MOV EAX,DWORD PTR FS:[0x18]	

Therefore, we will use the function ***GetCommandLineA*** , which is not emulated by the protector:



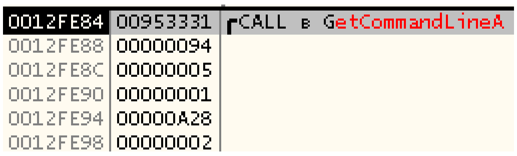
Адрес	Hex дамп	Дизассемблированный	Комментарий
7C810C6D	A1 F455887C	MOV EAX,DWORD PTR DS:[0x7C8855F4]	
7C810C72	C3	RET	
7C810C73	90	NOP	
7C810C74	90	NOP	
7C810C75	90	NOP	

Set it to ***breakpoint*** , run the program, and after two presses on the **F 9** key , it turns out here (the first time this function is called from the protector code):



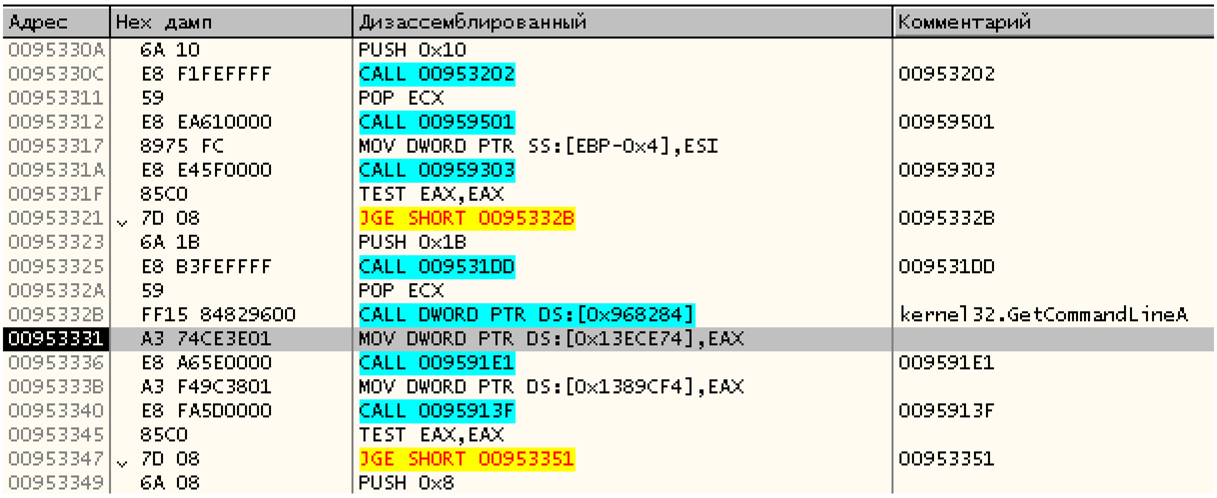
Адрес	Hex дамп	Дизассемблированный	Комментарий
7C810C6D	A1 F455887C	MOV EAX,DWORD PTR DS:[0x7C8855F4]	
7C810C72	C3	RET	
7C810C73	90	NOP	
7C810C74	90	NOP	

And in the stack window we see:



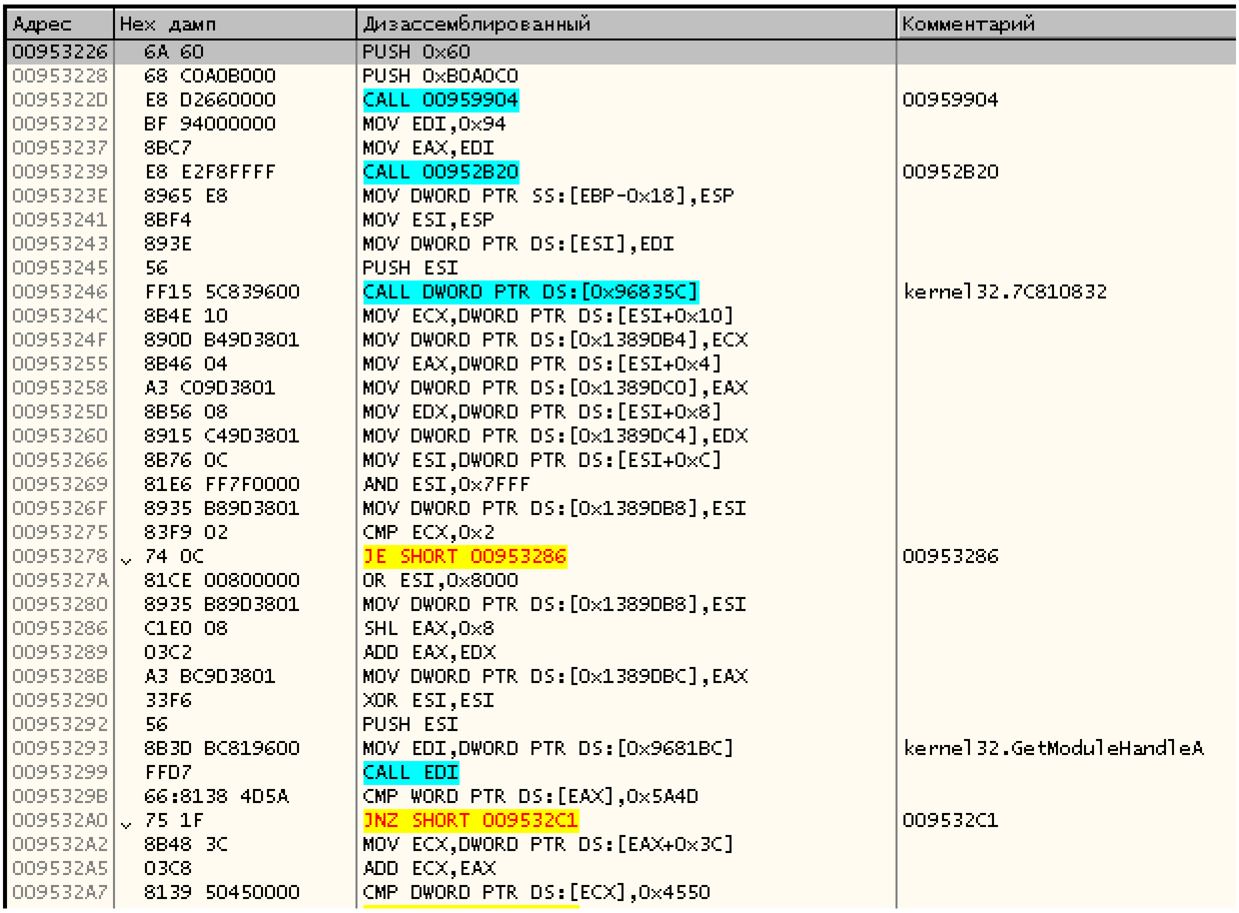
0012FE84	00953331	┌CALL в GetCommandLineA
0012FE88	00000094	
0012FE8C	00000005	
0012FE90	00000001	
0012FE94	00000A28	
0012FE98	00000002	

We perform this function, and it turns out here:



```
Адрес     Hex дамп       Дизассемблированный                    Комментарий
0095330A   6A 10         PUSH 0x10
0095330C   E8 F1FEFFFF   CALL 00953202                          00953202
00953311   59            POP ECX
00953312   E8 EA610000   CALL 00959501                          00959501
00953317   8975 FC       MOV DWORD PTR SS:[EBP-0x4],ESI
0095331A   E8 E45F0000   CALL 00959303                          00959303
0095331F   85C0          TEST EAX,EAX
00953321 ⌄ 7D 08         JGE SHORT 0095332B                     0095332B
00953323   6A 1B         PUSH 0x1B
00953325   E8 B3FEFFFF   CALL 009531DD                          009531DD
0095332A   59            POP ECX
0095332B   FF15 84829600 CALL DWORD PTR DS:[0x968284]           kernel32.GetCommandLineA
00953331   A3 74CE3E01   MOV DWORD PTR DS:[0x13ECE74],EAX
00953336   E8 A65E0000   CALL 009591E1                          009591E1
0095333B   A3 F49C3801   MOV DWORD PTR DS:[0x1389CF4],EAX
00953340   E8 FA5D0000   CALL 0095913F                          0095913F
00953345   85C0          TEST EAX,EAX
00953347 ⌄ 7D 08         JGE SHORT 00953351                     00953351
00953349   6A 08         PUSH 0x8
```

And if we scroll the code up a bit, we'll see the following:

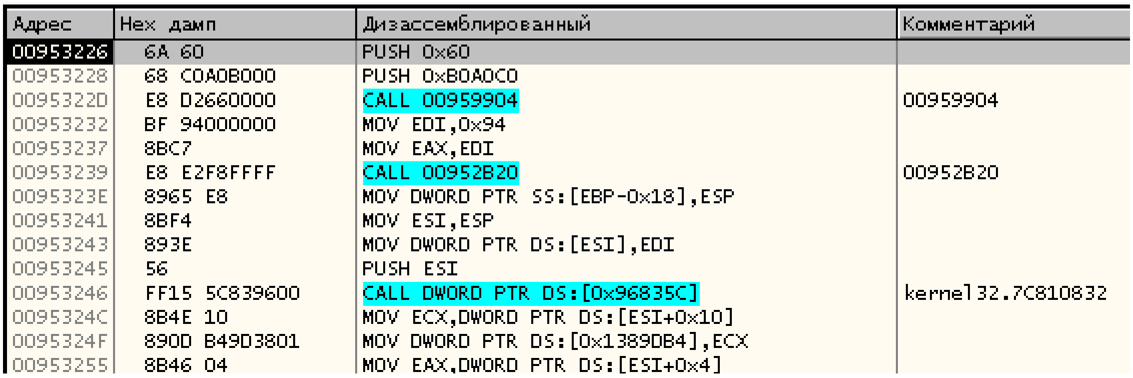


```
Адрес     Hex дамп       Дизассемблированный                    Комментарий
00953226   6A 60         PUSH 0x60
00953228   68 C0A0B000   PUSH 0xB0A0C0
0095322D   E8 D2660000   CALL 00959904                          00959904
00953232   BF 94000000   MOV EDI,0x94
00953237   8BC7          MOV EAX,EDI
00953239   E8 E2F8FFFF   CALL 00952B20                          00952B20
0095323E   8965 E8       MOV DWORD PTR SS:[EBP-0x18],ESP
00953241   8BF4          MOV ESI,ESP
00953243   893E          MOV DWORD PTR DS:[ESI],EDI
00953245   56            PUSH ESI
00953246   FF15 5C839600 CALL DWORD PTR DS:[0x96835C]           kernel32.7C810832
0095324C   8B4E 10       MOV ECX,DWORD PTR DS:[ESI+0x10]
0095324F   890D B49D3801 MOV DWORD PTR DS:[0x1389DB4],ECX
00953255   8B46 04       MOV EAX,DWORD PTR DS:[ESI+0x4]
00953258   A3 C09D3801   MOV DWORD PTR DS:[0x1389DC0],EAX
0095325D   8B56 08       MOV EDX,DWORD PTR DS:[ESI+0x8]
00953260   8915 C49D3801 MOV DWORD PTR DS:[0x1389DC4],EDX
00953266   8B76 0C       MOV ESI,DWORD PTR DS:[ESI+0xC]
00953269   81E6 FF7F0000 AND ESI,0x7FFF
0095326F   8935 B89D3801 MOV DWORD PTR DS:[0x1389DB8],ESI
00953275   83F9 02       CMP ECX,0x2
00953278 ⌄ 74 0C         JE SHORT 00953286                      00953286
0095327A   81CE 00800000 OR ESI,0x8000
00953280   8935 B89D3801 MOV DWORD PTR DS:[0x1389DB8],ESI
00953286   C1E0 08       SHL EAX,0x8
00953289   03C2          ADD EAX,EDX
0095328B   A3 BC9D3801   MOV DWORD PTR DS:[0x1389DBC],EAX
00953290   33F6          XOR ESI,ESI
00953292   56            PUSH ESI
00953293   8B3D BC819600 MOV EDI,DWORD PTR DS:[0x9681BC]        kernel32.GetModuleHandleA
00953299   FFD7          CALL EDI
0095329B   66:8138 4D5A  CMP WORD PTR DS:[EAX],0x5A4D
009532A0 ⌄ 75 1F         JNZ SHORT 009532C1                     009532C1
009532A2   8B48 3C       MOV ECX,DWORD PTR DS:[EAX+0x3C]
009532A5   03C8          ADD ECX,EAX
009532A7   8139 50450000 CMP DWORD PTR DS:[ECX],0x4550
```

This is the classic **OEP** programs written in ***Microsoft Visual C ++*** . So, the address of **OEP** is **00953226** . Install on this address ***Hardware breakpoint on execute*** , because we have to stop the program on **OEP** , to dump its memory.

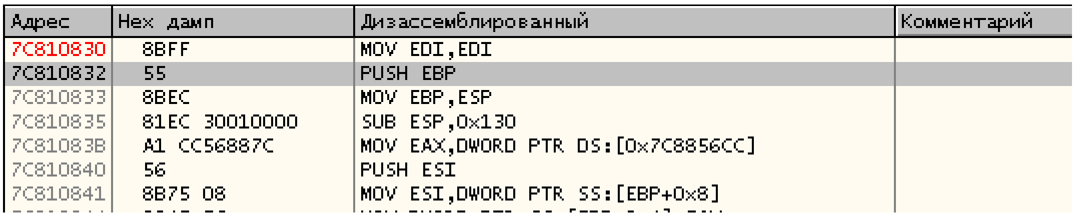
IAT Table Lookup and Validation

First look at the first **CALL** after **OEP** :



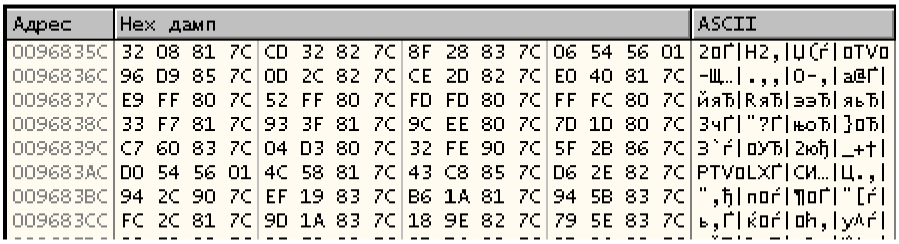
```
Адрес     Hex дамп          Дизассемблированный                     Комментарий
00953226   6A 60            PUSH 0x60
00953228   68 C0A0B000      PUSH 0xB0A0C0
0095322D   E8 D2660000      CALL 00959904                           00959904
00953232   BF 94000000      MOV EDI,0x94
00953237   8BC7             MOV EAX,EDI
00953239   E8 E2F8FFFF      CALL 00952B20                           00952B20
0095323E   8965 E8          MOV DWORD PTR SS:[EBP-0x18],ESP
00953241   8BF4             MOV ESI,ESP
00953243   893E             MOV DWORD PTR DS:[ESI],EDI
00953245   56               PUSH ESI
00953246   FF15 5C839600    CALL DWORD PTR DS:[0x96835C]            kernel32.7C810832
0095324C   8B4E 10          MOV ECX,DWORD PTR DS:[ESI+0x10]
0095324F   890D B49D3801    MOV DWORD PTR DS:[0x1389DB4],ECX
00953255   8B46 04          MOV EAX,DWORD PTR DS:[ESI+0x4]
```

We put on it mouse cursor, press the *Enter* key , and below we see the following :



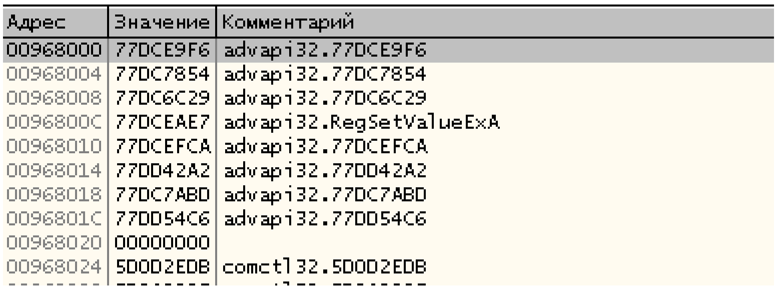
```
Адрес     Hex дамп          Дизассемблированный                     Комментарий
7C810830   8BFF             MOV EDI,EDI
7C810832   55               PUSH EBP
7C810833   8BEC             MOV EBP,ESP
7C810835   81EC 30010000    SUB ESP,0x130
7C81083B   A1 CC56887C      MOV EAX,DWORD PTR DS:[0x7C8856CC]
7C810840   56               PUSH ESI
7C810841   8B75 08          MOV ESI,DWORD PTR SS:[EBP+0x8]
```

At the address **00953246** we see the call of some function ***kernel32.7C810832*** . We pass to the address **00953246** in the dump of the debugger, and see here the ***IAT table*** :



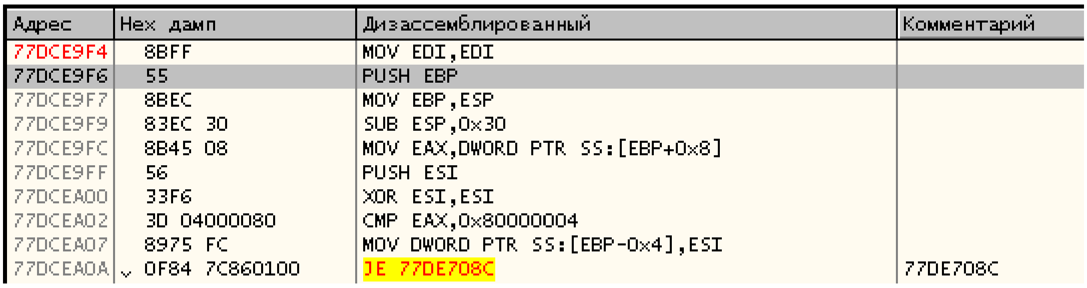
```
Адрес     Hex дамп                                         ASCII
0096835C  32 08 81 7C CD 32 82 7C 8F 28 83 7C 06 54 56 01  2□Ѓ|Н2‚|Џ(ѓ|□TV□
0096836C  96 D9 85 7C 0D 2C 82 7C CE 2D 82 7C E0 40 81 7C  –Щ…|.,‚|О-‚|а@Ѓ|
0096837C  E9 FF 80 7C 52 FF 80 7C FD FD 80 7C FF FC 80 7C  йяЂ|RяЂ|ээЂ|яьЂ|
0096838C  33 F7 81 7C 93 3F 81 7C 9C EE 80 7C 7D 1D 80 7C  3чЃ|“?Ѓ|њоЂ|}□Ђ|
0096839C  C7 60 83 7C 04 D3 80 7C 32 FE 90 7C 5F 2B 86 7C  З`ѓ|□УЂ|2юђ|_+†|
009683AC  D0 54 56 01 4C 58 81 7C 43 C8 85 7C D6 2E 82 7C  РTV□LXЃ|CИ…|Ц.‚|
009683BC  94 2C 90 7C EF 19 83 7C B6 1A 81 7C 94 5B 83 7C  ”,ђ|п□ѓ|¶□Ѓ|”[ѓ|
009683CC  FC 2C 81 7C 9D 1A 83 7C 18 9E 82 7C 79 5E 83 7C  ь,Ѓ|ќ□ѓ|□ћ‚|y^ѓ|
```

Change the display mode to ***Long - Address*** , scroll up a bit , and find the beginning of the IAT table :



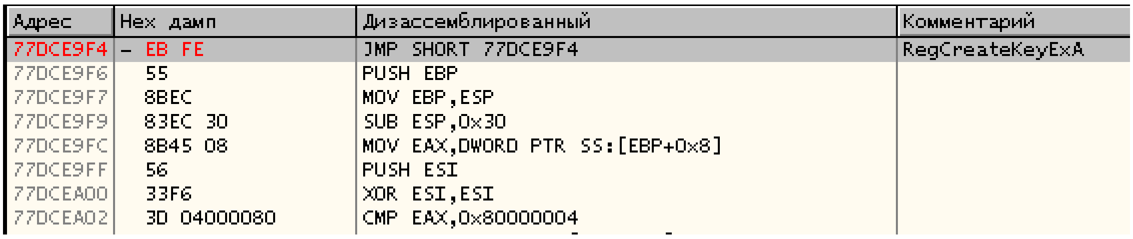
```
Адрес     Значение  Комментарий
00968000  77DCE9F6  advapi32.77DCE9F6
00968004  77DC7854  advapi32.77DC7854
00968008  77DC6C29  advapi32.77DC6C29
0096800C  77DCEAE7  advapi32.RegSetValueExA
00968010  77DCEFCA  advapi32.77DCEFCA
00968014  77DD42A2  advapi32.77DD42A2
00968018  77DC7ABD  advapi32.77DC7ABD
0096801C  77DD54C6  advapi32.77DD54C6
00968020  00000000
00968024  5D0D2EDB  comctl32.5D0D2EDB
```

Here we see one correct value of the ***advapi32.RegSetValueExA*** function , and some function addresses from the ***advapi32*** library ***. dll*** . Let's go through the disassembler window to the address **77DCE9F6** , and scroll the code up a bit:



```
Адрес     Hex дамп           Дизассемблированный                    Комментарий
77DCE9F4   8BFF              MOV EDI,EDI
77DCE9F6   55                PUSH EBP
77DCE9F7   8BEC              MOV EBP,ESP
77DCE9F9   83EC 30           SUB ESP,0x30
77DCE9FC   8B45 08           MOV EAX,DWORD PTR SS:[EBP+0x8]
77DCE9FF   56                PUSH ESI
77DCEA00   33F6              XOR ESI,ESI
77DCEA02   3D 04000080       CMP EAX,0x80000004
77DCEA07   8975 FC           MOV DWORD PTR SS:[EBP-0x4],ESI
77DCEA0A v 0F84 7C860100     JE 77DE708C                            77DE708C
```

And change the instruction to **MOV EDI , EDI** on the jump:

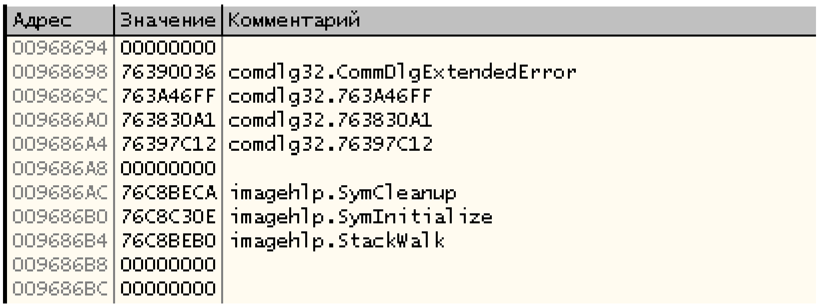


```
Адрес     Hex дамп           Дизассемблированный                    Комментарий
77DCE9F4 - EB FE             JMP SHORT 77DCE9F4                     RegCreateKeyExA
77DCE9F6   55                PUSH EBP
77DCE9F7   8BEC              MOV EBP,ESP
77DCE9F9   83EC 30           SUB ESP,0x30
77DCE9FC   8B45 08           MOV EAX,DWORD PTR SS:[EBP+0x8]
77DCE9FF   56                PUSH ESI
77DCEA00   33F6              XOR ESI,ESI
77DCEA02   3D 04000080       CMP EAX,0x80000004
```

We see that this address is the address of the ***advapi32*** function **.** ***RegCreateKeyExA*** . That is, the protector skips the instruction **MOV EDI , EDI** , and immediately performs a jump to the next instruction !!!

So, the beginning of the ***IAT Table*** is located at **00968000** .

We scroll down the debugger dump and look for the end of the ***IAT Table*** :



```
Адрес     Значение Комментарий
00968694  00000000
00968698  76390036 comdlg32.CommDlgExtendedError
0096869C  763A46FF comdlg32.763A46FF
009686A0  763830A1 comdlg32.763830A1
009686A4  76397C12 comdlg32.76397C12
009686A8  00000000
009686AC  76C8BECA imagehlp.SymCleanup
009686B0  76C8C30E imagehlp.SymInitialize
009686B4  76C8BEB0 imagehlp.StackWalk
009686B8  00000000
009686BC  00000000
```

So, the end of the ***IAT Table*** is located at **009686BC** .

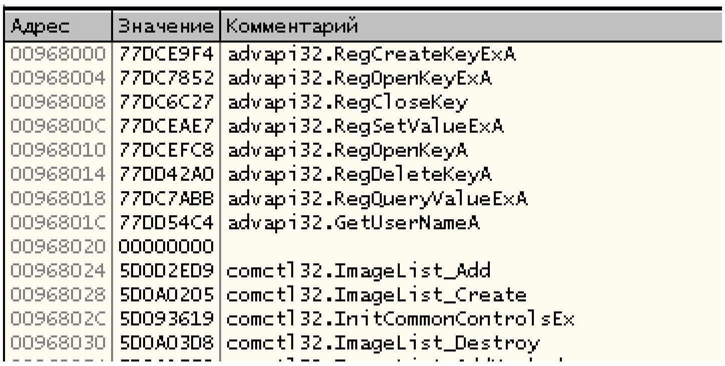
The size ***IAT tables*** = **009686BC** - **00968000** = **00000 6 BC**

Of course, all these functions can be restored manually, but this is a long and tedious process. Therefore, we will write a small script that will automatically do all the work:

```
var Start_IAT
var End_IAT
var Address_API
var Address_IAT

var temp_1
var temp_2
var temp_3
var temp_4
```

```
// Enter the parameters of the IAT Table (start and end)
MOV Start_IAT,00968000
MOV Address_IAT,00968000
MOV End_IAT,009686BC

@L_4:
CMP Address_IAT,End_IAT
JE @L_3
MOV temp_1,[Address_IAT]
CMP temp_1,0
JE @L_1

SUB temp_1,2
CMP [temp_1],0FF8B,2
JNZ  @L_2

MOV [Address_IAT],temp_1

@L_1:
ADD Address_IAT,4
JMP @L_4

@L_2:
ADD Address_IAT,4
JMP @L_4

@L_3:
ret
```

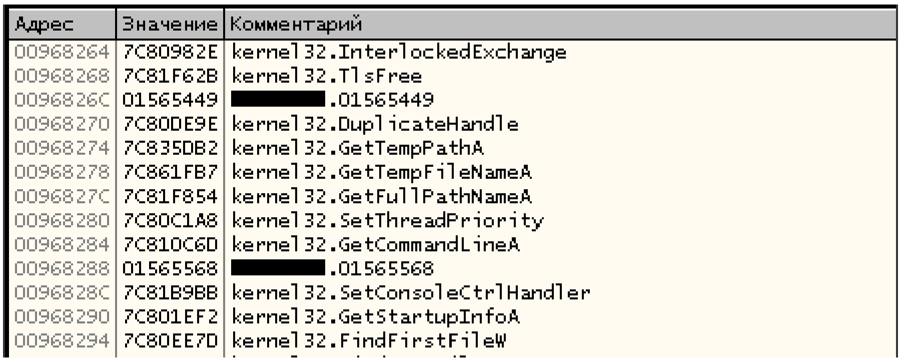
This script does not need any explanation. Run this script, and see:



Адрес	Значение	Комментарий
00968000	77DCE9F4	advapi32.RegCreateKeyExA
00968004	77DC7852	advapi32.RegOpenKeyExA
00968008	77DC6C27	advapi32.RegCloseKey
0096800C	77DCEAE7	advapi32.RegSetValueExA
00968010	77DCEFC8	advapi32.RegOpenKeyA
00968014	77DD42A0	advapi32.RegDeleteKeyA
00968018	77DC7ABB	advapi32.RegQueryValueExA
0096801C	77DD54C4	advapi32.GetUserNameA
00968020	00000000	
00968024	5D0D2ED9	comctl32.ImageList_Add
00968028	5D0A0205	comctl32.ImageList_Create
0096802C	5D093619	comctl32.InitCommonControlsEx
00968030	5D0A03D8	comctl32.ImageList_Destroy

Real function addresses in the **IAT** table - restored ...

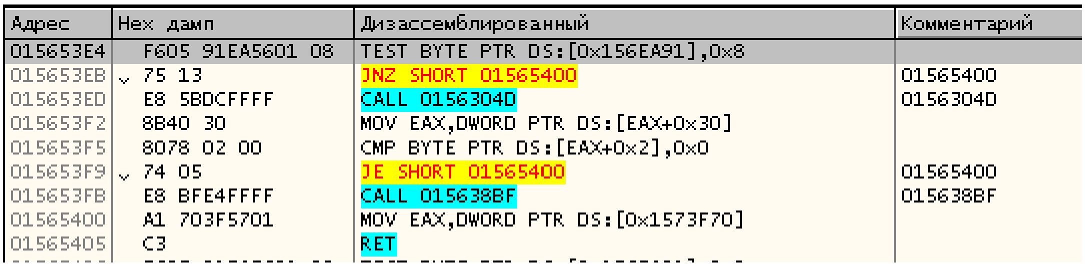
However, if we look closely at the restored function addresses in the **IAT** table , we see that some of the functions from the ***kernel 32. dll*** library are not restored:



Адрес	Значение	Комментарий
00968264	7C80982E	kernel32.InterlockedExchange
00968268	7C81F62B	kernel32.TlsFree
0096826C	01565449	[redacted].01565449
00968270	7C80DE9E	kernel32.DuplicateHandle
00968274	7C835DB2	kernel32.GetTempPathA
00968278	7C861FB7	kernel32.GetTempFileNameA
0096827C	7C81F854	kernel32.GetFullPathNameA
00968280	7C80C1A8	kernel32.SetThreadPriority
00968284	7C810C6D	kernel32.GetCommandLineA
00968288	01565568	[redacted].01565568
0096828C	7C81B9BB	kernel32.SetConsoleCtrlHandler
00968290	7C801EF2	kernel32.GetStartupInfoA
00968294	7C80EE7D	kernel32.FindFirstFileW

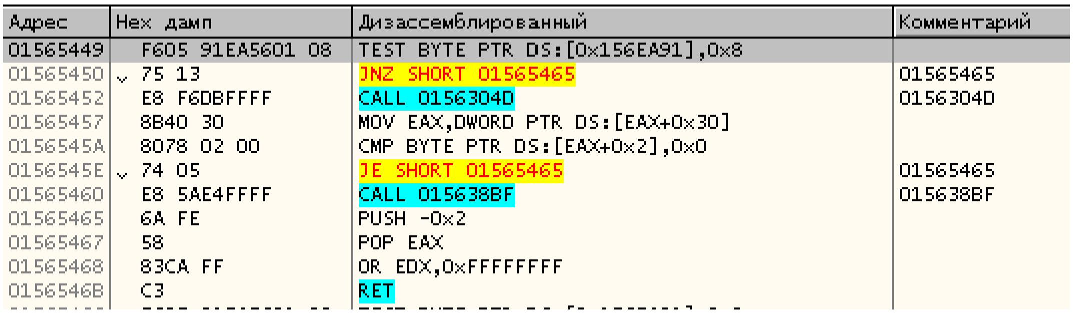
And there are **11** such undefined functions in the file . These functions are performed in the tread section, and we need to restore them manually. I'll give here the code of the emulated functions, and their correspondence to the functions from the ***kernel 32. dll*** library .

1. The function ***kernel32.GetCurrentProcessId*** :



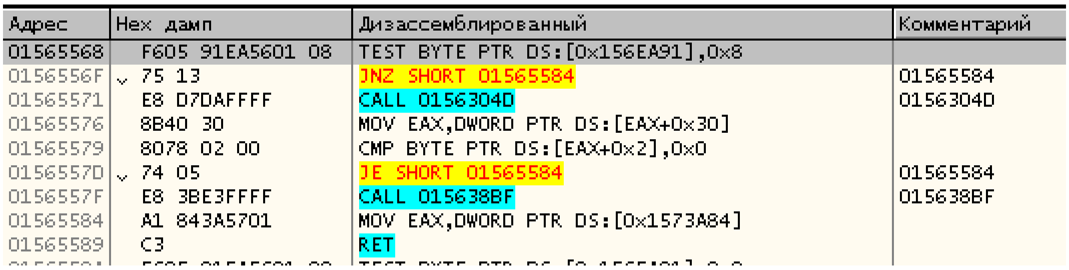
Адрес	Hex дамп	Дизассемблированный	Комментарий
015653E4	F605 91EA5601 08	TEST BYTE PTR DS:[0x156EA91],0x8	
015653EB	⌄ 75 13	JNZ SHORT 01565400	01565400
015653ED	E8 5BDCFFFF	CALL 0156304D	0156304D
015653F2	8B40 30	MOV EAX,DWORD PTR DS:[EAX+0x30]	
015653F5	8078 02 00	CMP BYTE PTR DS:[EAX+0x2],0x0	
015653F9	⌄ 74 05	JE SHORT 01565400	01565400
015653FB	E8 BFE4FFFF	CALL 015638BF	015638BF
01565400	A1 703F5701	MOV EAX,DWORD PTR DS:[0x1573F70]	
01565405	C3	RET	

2. The function ***kernel32.GetCurrentThread*** :



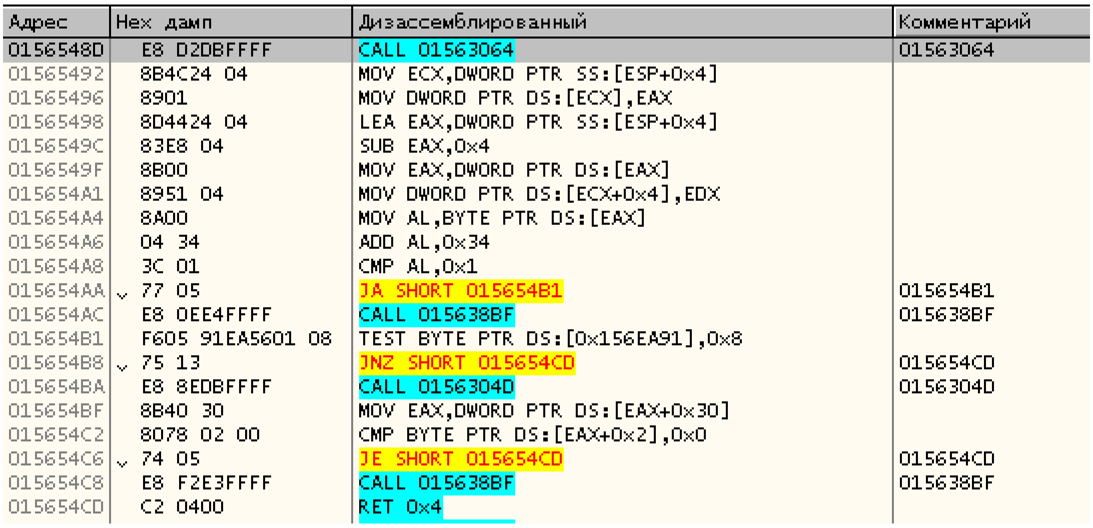
Адрес	Hex дамп	Дизассемблированный	Комментарий
01565449	F605 91EA5601 08	TEST BYTE PTR DS:[0x156EA91],0x8	
01565450	⌄ 75 13	JNZ SHORT 01565465	01565465
01565452	E8 F6DBFFFF	CALL 0156304D	0156304D
01565457	8B40 30	MOV EAX,DWORD PTR DS:[EAX+0x30]	
0156545A	8078 02 00	CMP BYTE PTR DS:[EAX+0x2],0x0	
0156545E	⌄ 74 05	JE SHORT 01565465	01565465
01565460	E8 5AE4FFFF	CALL 015638BF	015638BF
01565465	6A FE	PUSH -0x2	
01565467	58	POP EAX	
01565468	83CA FF	OR EDX,0xFFFFFFFF	
0156546B	C3	RET	

3. The function ***kernel32.GetACP*** :



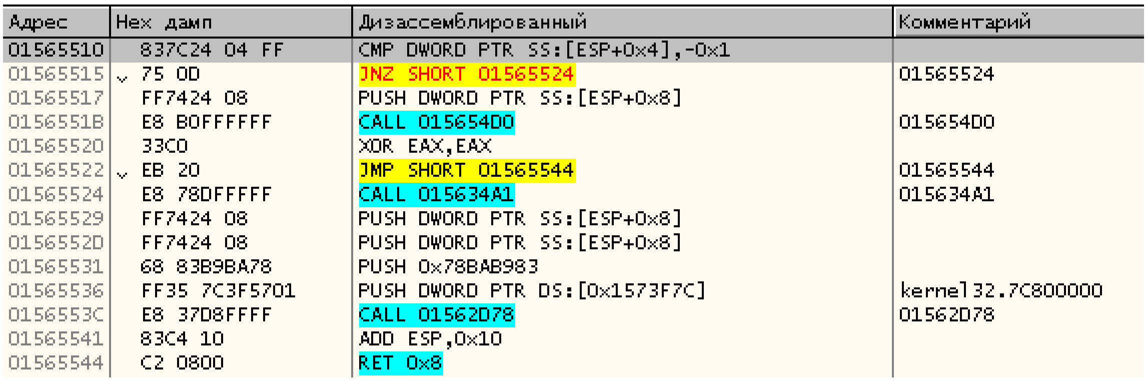
Адрес	Hex дамп	Дизассемблированный	Комментарий
01565568	F605 91EA5601 08	TEST BYTE PTR DS:[0x156EA91],0x8	
0156556F	⌄ 75 13	JNZ SHORT 01565584	01565584
01565571	E8 D7DAFFFF	CALL 0156304D	0156304D
01565576	8B40 30	MOV EAX,DWORD PTR DS:[EAX+0x30]	
01565579	8078 02 00	CMP BYTE PTR DS:[EAX+0x2],0x0	
0156557D	⌄ 74 05	JE SHORT 01565584	01565584
0156557F	E8 3BE3FFFF	CALL 015638BF	015638BF
01565584	A1 843A5701	MOV EAX,DWORD PTR DS:[0x1573A84]	
01565589	C3	RET	

4. The function ***kernel32.GetSystemTimeAsFileTime*** :



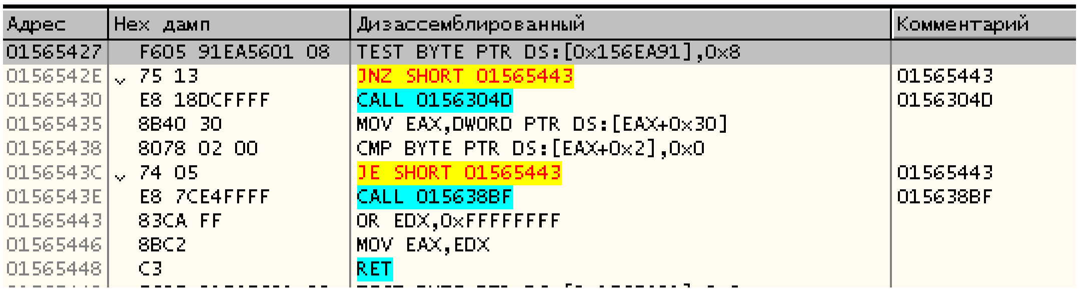
```
Адрес     Hex дамп              Дизассемблированный                      Комментарий
0156548D   E8 D2DBFFFF          CALL 01563064                            01563064
01565492   8B4C24 04            MOV ECX,DWORD PTR SS:[ESP+0x4]
01565496   8901                 MOV DWORD PTR DS:[ECX],EAX
01565498   8D4424 04            LEA EAX,DWORD PTR SS:[ESP+0x4]
0156549C   83E8 04              SUB EAX,0x4
0156549F   8B00                 MOV EAX,DWORD PTR DS:[EAX]
015654A1   8951 04              MOV DWORD PTR DS:[ECX+0x4],EDX
015654A4   8A00                 MOV AL,BYTE PTR DS:[EAX]
015654A6   04 34                ADD AL,0x34
015654A8   3C 01                CMP AL,0x1
015654AA ⌄ 77 05                JA SHORT 015654B1                        015654B1
015654AC   E8 0EE4FFFF          CALL 015638BF                            015638BF
015654B1   F605 91EA5601 08     TEST BYTE PTR DS:[0x156EA91],0x8
015654B8 ⌄ 75 13                JNZ SHORT 015654CD                       015654CD
015654BA   E8 8EDBFFFF          CALL 0156304D                            0156304D
015654BF   8B40 30              MOV EAX,DWORD PTR DS:[EAX+0x30]
015654C2   8078 02 00           CMP BYTE PTR DS:[EAX+0x2],0x0
015654C6 ⌄ 74 05                JE SHORT 015654CD                        015654CD
015654C8   E8 F2E3FFFF          CALL 015638BF                            015638BF
015654CD   C2 0400              RET 0x4
```

5. The function **_kernel32.TerminateProcess_** :



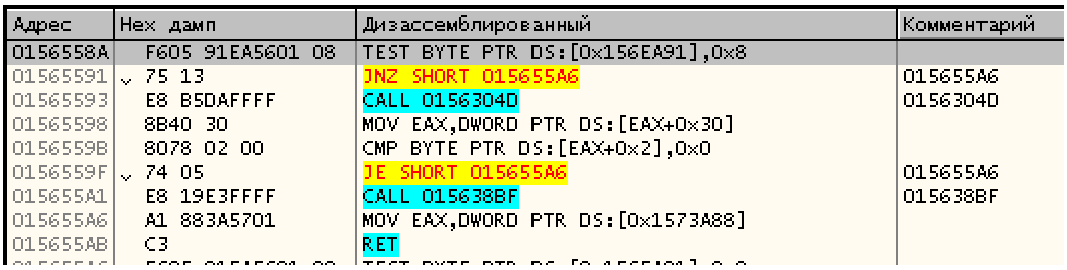
```
Адрес     Hex дамп         Дизассемблированный                    Комментарий
01565510   837C24 04 FF    CMP DWORD PTR SS:[ESP+0x4],-0x1
01565515 ⌄ 75 0D           JNZ SHORT 01565524                     01565524
01565517   FF7424 08       PUSH DWORD PTR SS:[ESP+0x8]
0156551B   E8 B0FFFFFF     CALL 015654D0                          015654D0
01565520   33C0            XOR EAX,EAX
01565522 ⌄ EB 20           JMP SHORT 01565544                     01565544
01565524   E8 78DFFFFF     CALL 015634A1                          015634A1
01565529   FF7424 08       PUSH DWORD PTR SS:[ESP+0x8]
0156552D   FF7424 08       PUSH DWORD PTR SS:[ESP+0x8]
01565531   68 83B9BA78     PUSH 0x78BAB983
01565536   FF35 7C3F5701   PUSH DWORD PTR DS:[0x1573F7C]          kernel32.7C800000
0156553C   E8 37D8FFFF     CALL 01562D78                          01562D78
01565541   83C4 10         ADD ESP,0x10
01565544   C2 0800         RET 0x8
```

6. The function **_kernel32.Get CurrentProcess_** :



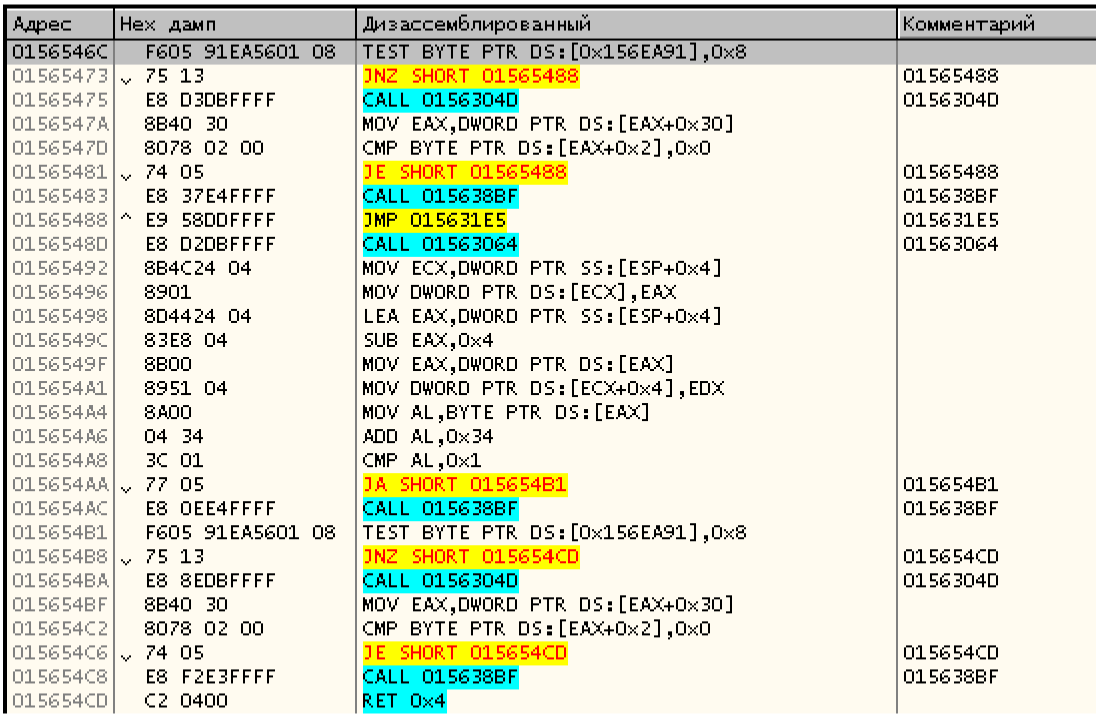
```
Адрес     Hex дамп              Дизассемблированный                 Комментарий
01565427   F605 91EA5601 08     TEST BYTE PTR DS:[0x156EA91],0x8
0156542E ⌄ 75 13                JNZ SHORT 01565443                  01565443
01565430   E8 18DCFFFF          CALL 0156304D                       0156304D
01565435   8B40 30              MOV EAX,DWORD PTR DS:[EAX+0x30]
01565438   8078 02 00           CMP BYTE PTR DS:[EAX+0x2],0x0
0156543C ⌄ 74 05                JE SHORT 01565443                   01565443
0156543E   E8 7CE4FFFF          CALL 015638BF                       015638BF
01565443   83CA FF              OR EDX,0xFFFFFFFF
01565446   8BC2                 MOV EAX,EDX
01565448   C3                   RET
```

7. Function **_kernel32.GetOEMCP_** :



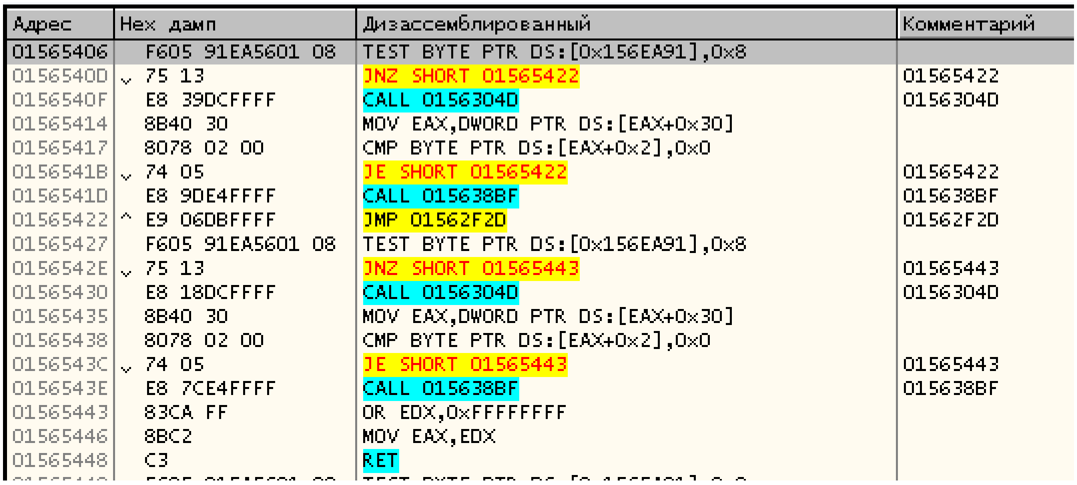
```
Адрес     Hex дамп              Дизассемблированный                 Комментарий
0156558A   F605 91EA5601 08     TEST BYTE PTR DS:[0x156EA91],0x8
01565591 ⌄ 75 13                JNZ SHORT 015655A6                  015655A6
01565593   E8 B5DAFFFF          CALL 0156304D                       0156304D
01565598   8B40 30              MOV EAX,DWORD PTR DS:[EAX+0x30]
0156559B   8078 02 00           CMP BYTE PTR DS:[EAX+0x2],0x0
0156559F ⌄ 74 05                JE SHORT 015655A6                   015655A6
015655A1   E8 19E3FFFF          CALL 015638BF                       015638BF
015655A6   A1 883A5701          MOV EAX,DWORD PTR DS:[0x1573A88]
015655AB   C3                   RET
```

8. Function **_kernel32.GetTickCount_** :



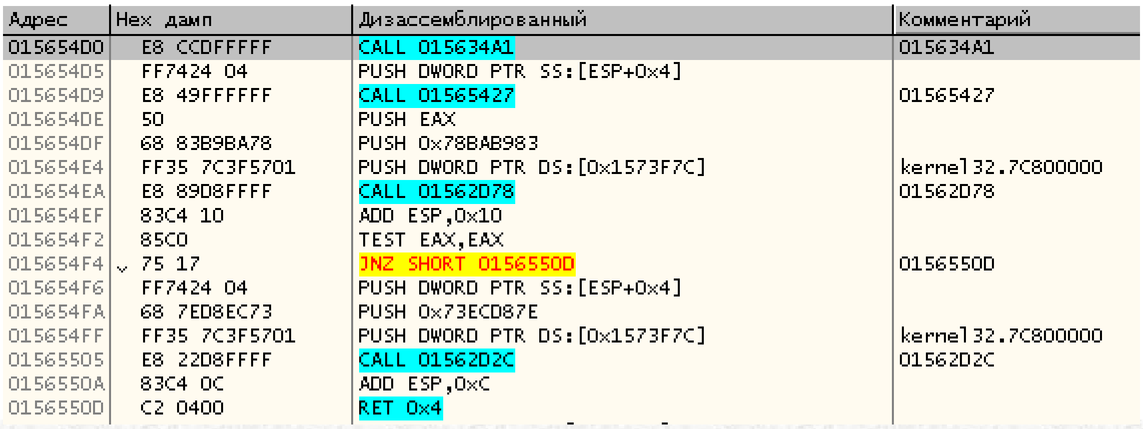
Адрес	Hex дамп	Дизассемблированный	Комментарий
0156546C	F605 91EA5601 08	TEST BYTE PTR DS:[0x156EA91],0x8	
01565473	⌄ 75 13	JNZ SHORT 01565488	01565488
01565475	E8 D3DBFFFF	CALL 0156304D	0156304D
0156547A	8B40 30	MOV EAX,DWORD PTR DS:[EAX+0x30]	
0156547D	8078 02 00	CMP BYTE PTR DS:[EAX+0x2],0x0	
01565481	⌄ 74 05	JE SHORT 01565488	01565488
01565483	E8 37E4FFFF	CALL 015638BF	015638BF
01565488	^ E9 58DDFFFF	JMP 015631E5	015631E5
0156548D	E8 D2DBFFFF	CALL 01563064	01563064
01565492	8B4C24 04	MOV ECX,DWORD PTR SS:[ESP+0x4]	
01565496	8901	MOV DWORD PTR DS:[ECX],EAX	
01565498	8D4424 04	LEA EAX,DWORD PTR SS:[ESP+0x4]	
0156549C	83E8 04	SUB EAX,0x4	
0156549F	8B00	MOV EAX,DWORD PTR DS:[EAX]	
015654A1	8951 04	MOV DWORD PTR DS:[ECX+0x4],EDX	
015654A4	8A00	MOV AL,BYTE PTR DS:[EAX]	
015654A6	04 34	ADD AL,0x34	
015654A8	3C 01	CMP AL,0x1	
015654AA	⌄ 77 05	JA SHORT 015654B1	015654B1
015654AC	E8 0EE4FFFF	CALL 015638BF	015638BF
015654B1	F605 91EA5601 08	TEST BYTE PTR DS:[0x156EA91],0x8	
015654B8	⌄ 75 13	JNZ SHORT 015654CD	015654CD
015654BA	E8 8EDBFFFF	CALL 0156304D	0156304D
015654BF	8B40 30	MOV EAX,DWORD PTR DS:[EAX+0x30]	
015654C2	8078 02 00	CMP BYTE PTR DS:[EAX+0x2],0x0	
015654C6	⌄ 74 05	JE SHORT 015654CD	015654CD
015654C8	E8 F2E3FFFF	CALL 015638BF	015638BF
015654CD	C2 0400	RET 0x4	

9. Function ***kernel32.GetCurrentThreadId*** :



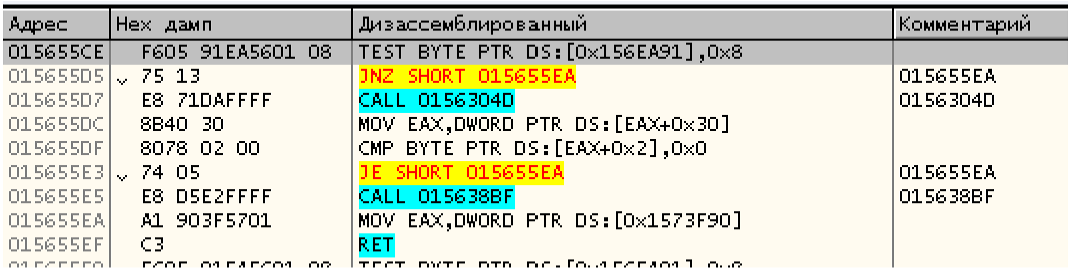
Адрес	Hex дамп	Дизассемблированный	Комментарий
01565406	F605 91EA5601 08	TEST BYTE PTR DS:[0x156EA91],0x8	
0156540D	⌄ 75 13	JNZ SHORT 01565422	01565422
0156540F	E8 39DCFFFF	CALL 0156304D	0156304D
01565414	8B40 30	MOV EAX,DWORD PTR DS:[EAX+0x30]	
01565417	8078 02 00	CMP BYTE PTR DS:[EAX+0x2],0x0	
0156541B	⌄ 74 05	JE SHORT 01565422	01565422
0156541D	E8 9DE4FFFF	CALL 015638BF	015638BF
01565422	^ E9 06DBFFFF	JMP 01562F2D	01562F2D
01565427	F605 91EA5601 08	TEST BYTE PTR DS:[0x156EA91],0x8	
0156542E	⌄ 75 13	JNZ SHORT 01565443	01565443
01565430	E8 18DCFFFF	CALL 0156304D	0156304D
01565435	8B40 30	MOV EAX,DWORD PTR DS:[EAX+0x30]	
01565438	8078 02 00	CMP BYTE PTR DS:[EAX+0x2],0x0	
0156543C	⌄ 74 05	JE SHORT 01565443	01565443
0156543E	E8 7CE4FFFF	CALL 015638BF	015638BF
01565443	83CA FF	OR EDX,0xFFFFFFFF	
01565446	8BC2	MOV EAX,EDX	
01565448	C3	RET	

10. Function ***kernel32.ExitProcess*** :



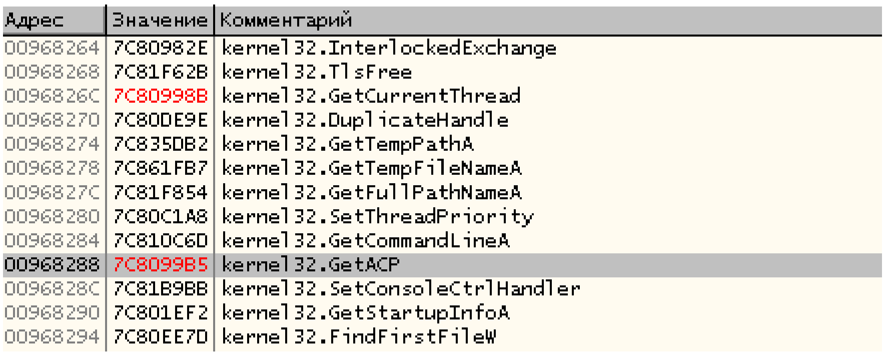
Адрес	Hex дамп	Дизассемблированный	Комментарий
015654D0	E8 CCDFFFFF	CALL 015634A1	015634A1
015654D5	FF7424 04	PUSH DWORD PTR SS:[ESP+0x4]	
015654D9	E8 49FFFFFF	CALL 01565427	01565427
015654DE	50	PUSH EAX	
015654DF	68 83B9BA78	PUSH 0x78BAB983	
015654E4	FF35 7C3F5701	PUSH DWORD PTR DS:[0x1573F7C]	kernel32.7C800000
015654EA	E8 89D8FFFF	CALL 01562D78	01562D78
015654EF	83C4 10	ADD ESP,0x10	
015654F2	85C0	TEST EAX,EAX	
015654F4	⌄ 75 17	JNZ SHORT 0156550D	0156550D
015654F6	FF7424 04	PUSH DWORD PTR SS:[ESP+0x4]	
015654FA	68 7ED8EC73	PUSH 0x73ECD87E	
015654FF	FF35 7C3F5701	PUSH DWORD PTR DS:[0x1573F7C]	kernel32.7C800000
01565505	E8 22D8FFFF	CALL 01562D2C	01562D2C
0156550A	83C4 0C	ADD ESP,0xC	
0156550D	C2 0400	RET 0x4	

11. Function ***kernel32.GetVersion*** :



Адрес	Hex дамп	Дизассемблированный	Комментарий
015655CE	F605 91EA5601 08	TEST BYTE PTR DS:[0x156EA91],0x8	
015655D5	75 13	JNZ SHORT 015655EA	015655EA
015655D7	E8 71DAFFFF	CALL 0156304D	0156304D
015655DC	8B40 30	MOV EAX,DWORD PTR DS:[EAX+0x30]	
015655DF	8078 02 00	CMP BYTE PTR DS:[EAX+0x2],0x0	
015655E3	74 05	JE SHORT 015655EA	015655EA
015655E5	E8 D5E2FFFF	CALL 015638BF	015638BF
015655EA	A1 903F5701	MOV EAX,DWORD PTR DS:[0x1573F90]	
015655EF	C3	RET	

Manually correct in the **IAT** table addresses of emulated functions:



Адрес	Значение	Комментарий
00968264	7C80982E	kernel32.InterlockedExchange
00968268	7C81F62B	kernel32.TlsFree
0096826C	7C80998B	kernel32.GetCurrentThread
00968270	7C80DE9E	kernel32.DuplicateHandle
00968274	7C835DB2	kernel32.GetTempPathA
00968278	7C861FB7	kernel32.GetTempFileNameA
0096827C	7C81F854	kernel32.GetFullPathNameA
00968280	7C80C1A8	kernel32.SetThreadPriority
00968284	7C810C6D	kernel32.GetCommandLineA
00968288	7C8099B5	kernel32.GetACP
0096828C	7C81B9BB	kernel32.SetConsoleCtrlHandler
00968290	7C801EF2	kernel32.GetStartupInfoA
00968294	7C80EE7D	kernel32.FindFirstFileW

T EPER we are ready for Dumping memory unpacked file.

Getting the unpacked file dump

I love that the unpacked file looks like the original file before it is processed by the protector. HASP envelope SRM combines all sections of the file before the resource section into one section with the name ***. ASK 1*** , so we need to divide this section into several sections that had the original file. We already know that the program is written in ***Microsoft Visual C ++*** . Therefore, we need to find a similar unpacked file, and in its sections determine the sections that the unpacked file should.

Comparing files shows that the unpacked file should have 4 sections:

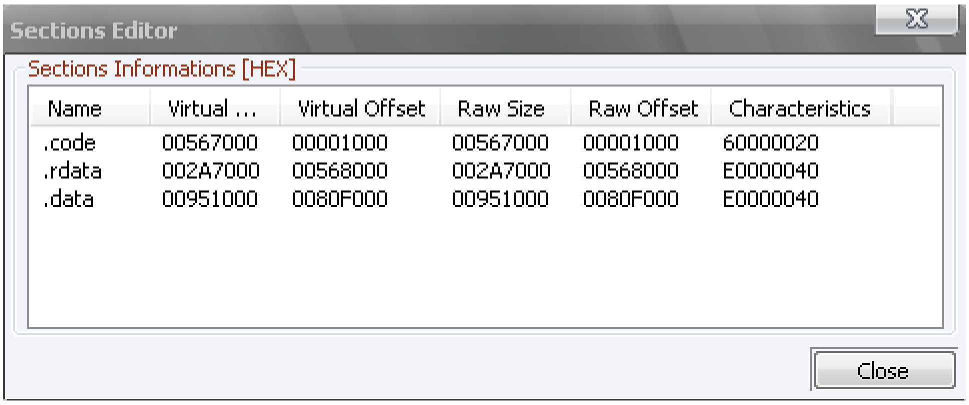
```
VirtualAddress_1 - 00401000 .code
VirtualAddress_2 - 00968000 .rdata
VirtualAddress_3 - 00C0F000 .data
VirtualAddress_4 - 01560000 .rsrc
VirtualAddress_5 - 0
VirtualAddress_6 - 0
VirtualAddress_7 - 0
VirtualAddress_8 - 0
VirtualAddress_9 - 0
VirtualAddress_10 - 0
Число секций - 4

* Начало Таблицы IAT: 00968000
* Конец  Таблицы IAT: 009686BC
* Размер Таблицы IAT: 000006BC
* Начало таблицы импорта: 00C0C000 (0080C000)
==========================================================================
* Адрес OEP: 00953226
* Адрес OEP для ImpREC: 00553226
* Адрес Таблицы IAT: 00968000
* Адрес Таблицы IAT для ImpREC: 00568000
* Размер Таблицы IAT: 000006BC
```

To get a dump of the unpacked file's memory, I use the universal script " *Themida - PE correction -head and dumping the unpacked file by vnekrilov . osc* " , which is attached to this article. With the help of this script, you can get a dump of the unpacked program's memory with the required sections for all protectors, not just for ***Themida*** .

So, run the script, enter the required data (from the table above), and we get a dump of the unpacked program with the name ***dumped.exe*** .

And we need to get a dump of the memory of our unpacked file using the plugin ***OllyDumpEx v 0.90*** . We need this to get the resource section of the unpacked file. Run the plugin , in the drop-down box select our victim . Click on the button " ***Get*** " ***EIP as OEP*** " , then click on the " ***Dump*** " button , and get a file named ***xxxxxxxx _dump. exe*** .

Open in ***PE Tools v 1.5 RC 7*** file ***dumped.exe*** , correct the section names, and delete the last section (resource section) from the file:



Name	Virtual ...	Virtual Offset	Raw Size	Raw Offset	Characteristics
.code	00567000	00001000	00567000	00001000	60000020
.rdata	002A7000	00568000	002A7000	00568000	E0000040
.data	00951000	0080F000	00951000	0080F000	E0000040

Go to the ***Optional*** tab ***Header*** , and click on the buttons with a " **?"** " :

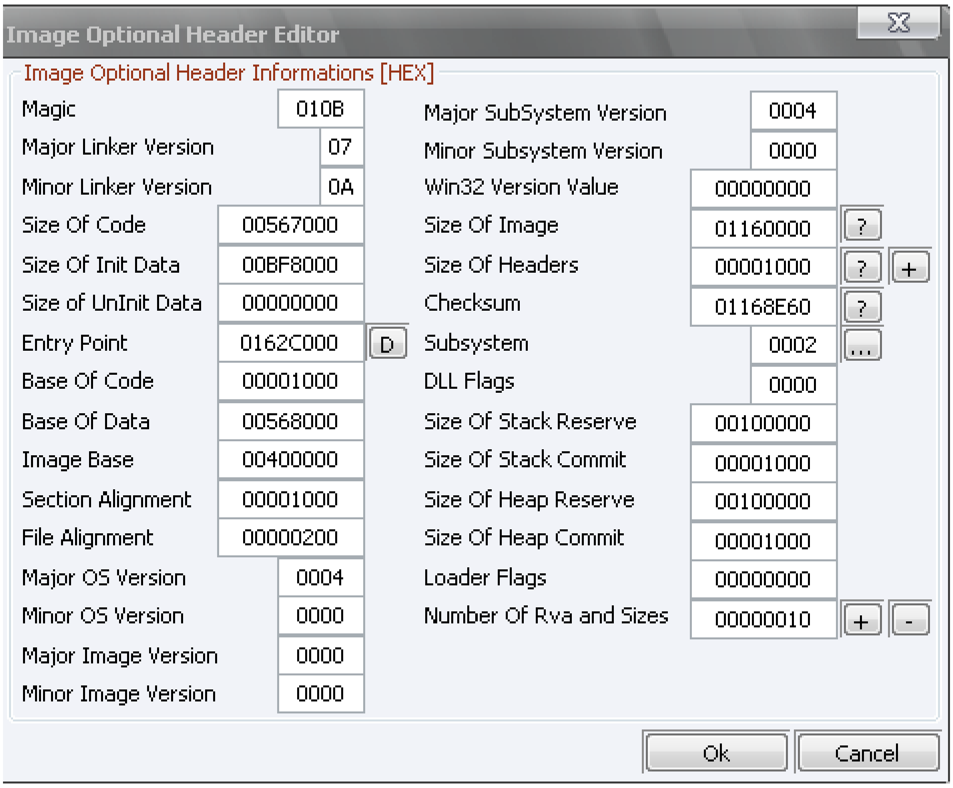
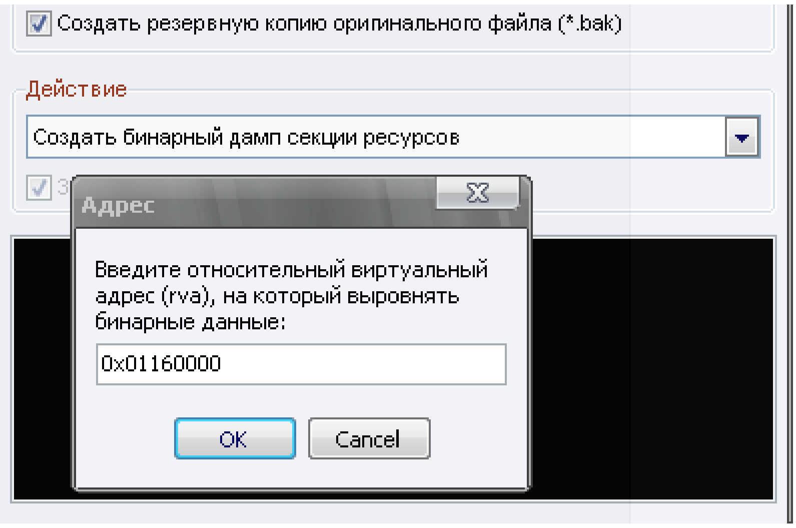


Image Optional Header Editor

Image Optional Header Informations [HEX]

Magic	010B	Major SubSystem Version	0004
Major Linker Version	07	Minor Subsystem Version	0000
Minor Linker Version	0A	Win32 Version Value	00000000
Size Of Code	00567000	Size Of Image	01160000
Size Of Init Data	00BF8000	Size Of Headers	00001000
Size of UnInit Data	00000000	Checksum	01168E60
Entry Point	0162C000	Subsystem	0002
Base Of Code	00001000	DLL Flags	0000
Base Of Data	00568000	Size Of Stack Reserve	00100000
Image Base	00400000	Size Of Stack Commit	00001000
Section Alignment	00001000	Size Of Heap Reserve	00100000
File Alignment	00000200	Size Of Heap Commit	00001000
Major OS Version	0004	Loader Flags	00000000
Minor OS Version	0000	Number Of Rva and Sizes	00000010
Major Image Version	0000		
Minor Image Version	0000		

Now run the utility " ***Resource Binder 3.1.5*** " , open the file *xxxxxxxx _dump* in it ***. exe*** , select the action " ***Create a binary file of the resource section*** " , the copied image size from ***PE Tools v 1.5 RC 7*** :



☑ Создать резервную копию оригинального файла (*.bak)

Действие

Создать бинарный дамп секции ресурсов

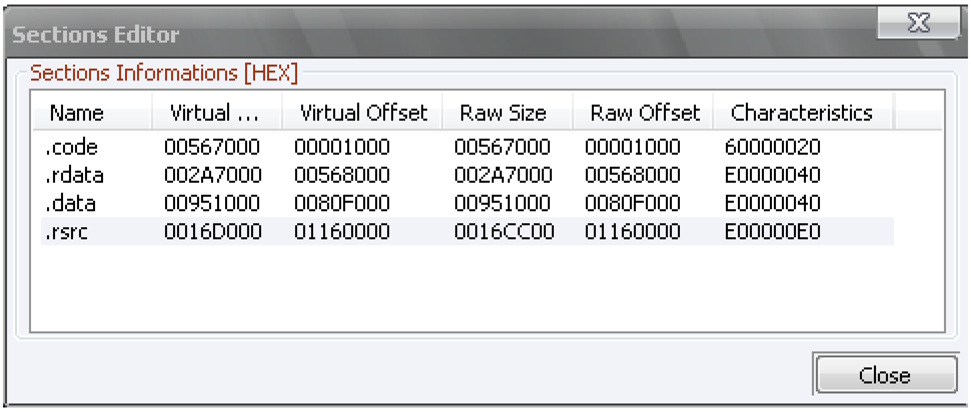
Адрес

Введите относительный виртуальный адрес (rva), на который выровнять бинарные данные:

0x01160000

We click the " ***OK*** " button , and we get a dump of the resource section, aligned to the address **0x01160000** .

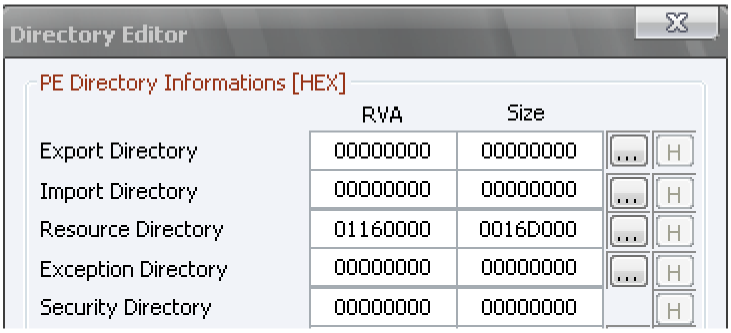
And with the help of the utility ***PE Tools v 1.5 RC 7*** , we dump the resource section dump to the ***dumped.exe*** file :



Sections Informations [HEX]

Name	Virtual ...	Virtual Offset	Raw Size	Raw Offset	Characteristics
.code	00567000	00001000	00567000	00001000	60000020
.rdata	002A7000	00568000	002A7000	00568000	E0000040
.data	00951000	0080F000	00951000	0080F000	E0000040
.rsrc	0016D000	01160000	0016CC00	01160000	E00000E0

Go to the " ***Directories*** " tab , and enter the values in the **RVA** fields and **Size** :



PE Directory Informations [HEX]

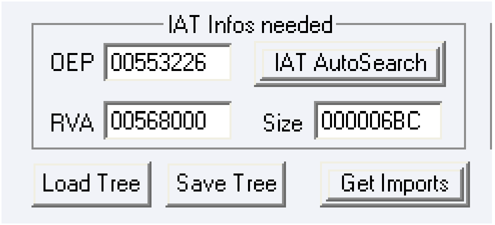
	RVA	Size
Export Directory	00000000	00000000
Import Directory	00000000	00000000
Resource Directory	01160000	0016D000
Exception Directory	00000000	00000000
Security Directory	00000000	00000000

Save the received changes.

Recovery Program import tables

Now we need to restore ***The*** program ***import table is*** restored to its native place. If we scroll through the code in the section **.** ***rdata*** down, then we will see the free space for restoring the import table at address **00C0C000** .

Now run the ***ImportRE C*** utility , and enter the corresponding values in the windows:

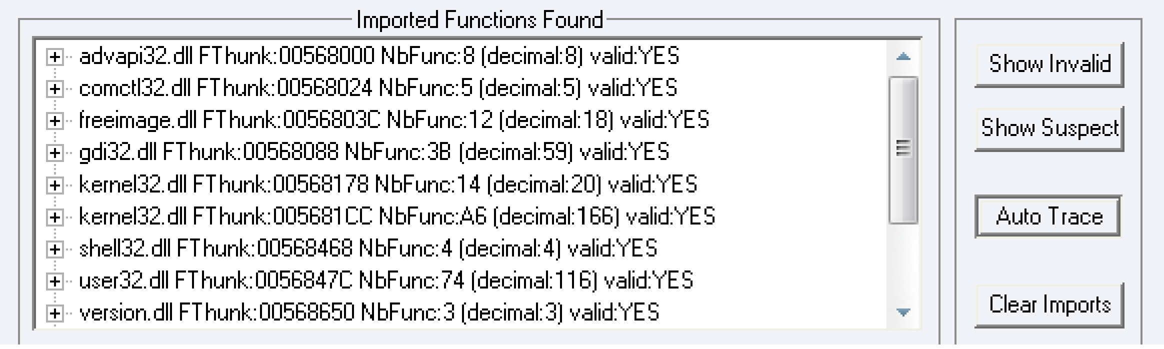


IAT Infos needed

OEP 00553226

RVA 00568000 Size 000006BC

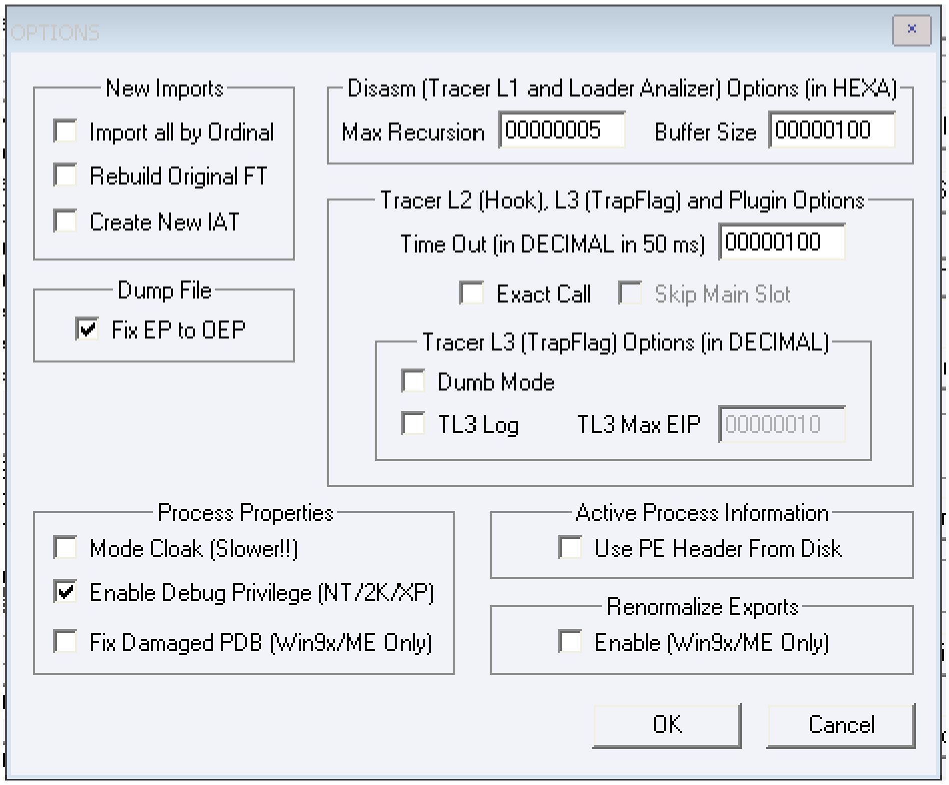
We press the button " ***Get*** " ***Imports*** " , and get all the recognized functions of the ***IAT Table*** :



Imported Functions Found

```
advapi32.dll FThunk:00568000 NbFunc:8 (decimal:8) valid:YES
comctl32.dll FThunk:00568024 NbFunc:5 (decimal:5) valid:YES
freeimage.dll FThunk:0056803C NbFunc:12 (decimal:18) valid:YES
gdi32.dll FThunk:00568088 NbFunc:3B (decimal:59) valid:YES
kernel32.dll FThunk:00568178 NbFunc:14 (decimal:20) valid:YES
kernel32.dll FThunk:005681CC NbFunc:A6 (decimal:166) valid:YES
shell32.dll FThunk:00568468 NbFunc:4 (decimal:4) valid:YES
user32.dll FThunk:0056847C NbFunc:74 (decimal:116) valid:YES
version.dll FThunk:00568650 NbFunc:3 (decimal:3) valid:YES
```

Now we need to configure the options of ***ImportREC*** :



OPTIONS

New Imports
□ Import all by Ordinal
□ Rebuild Original FT
□ Create New IAT

Dump File
☑ Fix EP to OEP

Disasm (Tracer L1 and Loader Analizer) Options (in HEXA)
Max Recursion 00000005 Buffer Size 00000100

Tracer L2 (Hook), L3 (TrapFlag) and Plugin Options
Time Out (in DECIMAL in 50 ms) 00000100
□ Exact Call □ Skip Main Slot

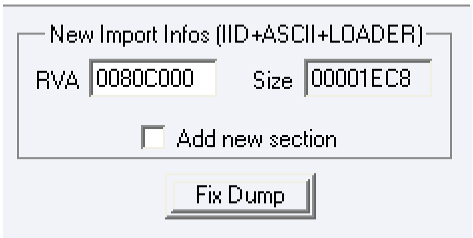
Tracer L3 (TrapFlag) Options (in DECIMAL)
□ Dumb Mode
□ TL3 Log TL3 Max EIP 00000010

Process Properties
□ Mode Cloak (Slower!!)
☑ Enable Debug Privilege (NT/2K/XP)
□ Fix Damaged PDB (Win9x/ME Only)

Active Process Information
□ Use PE Header From Disk

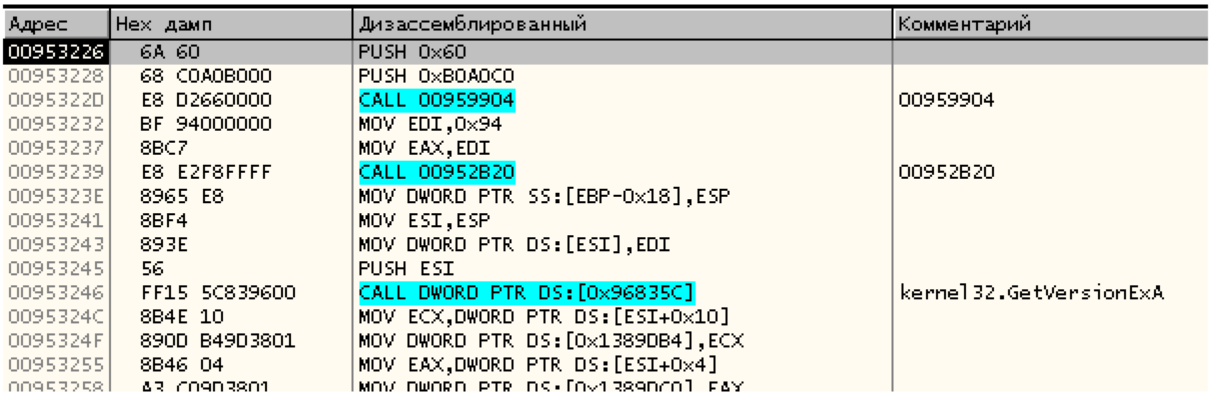
Renormalize Exports
□ Enable (Win9x/ME Only)

Close this window, remove the checkbox on the option " ***Add new section*** " , enter **RVA** start the ***import tables*** - **0080C000** , and press the " ***Fix Dump*** " :



New Import Infos (IID+ASCII+LOADER)
RVA 0080C000 Size 00001EC8
□ Add new section

And we get a file called ***dumped _ .exe*** . We are trying to load it into the debugger:



```
Адрес     Hex дамп          Дизассемблированный                  Комментарий
00953226  6A 60             PUSH 0x60
00953228  68 C0A0B000       PUSH 0xB0A0C0
0095322D  E8 D2660000       CALL 00959904                        00959904
00953232  BF 94000000       MOV EDI,0x94
00953237  8BC7              MOV EAX,EDI
00953239  E8 E2F8FFFF       CALL 00952B20                        00952B20
0095323E  8965 E8           MOV DWORD PTR SS:[EBP-0x18],ESP
00953241  8BF4              MOV ESI,ESP
00953243  893E              MOV DWORD PTR DS:[ESI],EDI
00953245  56                PUSH ESI
00953246  FF15 5C839600     CALL DWORD PTR DS:[0x96835C]         kernel32.GetVersionExA
0095324C  8B4E 10           MOV ECX,DWORD PTR DS:[ESI+0x10]
0095324F  890D B49D3801     MOV DWORD PTR DS:[0x1389DB4],ECX
00953255  8B46 04           MOV EAX,DWORD PTR DS:[ESI+0x4]
00953258  A3 C09D3801       MOV DWORD PTR DS:[0x1389DC0],EAX
```

And everything is loaded perfectly. This ends the removal of the HASP envelope SRM .

Application:

Скрипт “***Themida - Корректировка PE-заголовка и дампирование распакованного файла by vnekrilov.osc***”.

vnekrilov

24 июня 2017