Unpacking ASProtect v of 2.xx (cutting sections, the restoration of the [skramblernogo] code, [dekompiljatsija] VM, the restoration of import, [inlajn] of [patch])

Victim: FontExpert of 2005 Version: 7.0 Release 1.
To take is possible here: the http://www.proximasoftware.com/
If on the site there will be new version, and by you will be necessary precisely this – you will report to me, and I will lay out somewhere for the running off.

Author:PE_Kill
e the mail:PE_Kill@mail.ru

Introduction

As occasion for writing of this article served the complete absence of information, about the restoration of the [skramblernogo] code and [dekompiljatsii] “virtual machine”. I began to write article, simultaneously becoming acquainted with work and equipment Of aSProtect'a. Therefore after writing of article I much rethought, saw many errors and learned considerably more than he wrote in this article. This article more greatly resembles not the management on the removal Of aSProtect'[a], but the diary of the cracker, in which in the order it is said, as I broke concrete program.

Here there are no some, very necessary scripts. For example script for the passage to OEP. This is done intended, since most likely this is last article with more or less complex [iskhodnikami]. To whom will be wanted to work unknown how much at [dekompiljatorom], to and then lay out him to [pablik], so that in following [bilde] all this would cover? It is necessary to be investigated with all itself. I having been, for example, steamed, twentieth time hands to reach OEP, became to study OllyScript and to search for into ASProtect'e of regularity. Already in a day other day I wrote OEP Of finder. So that there is nothing complex of – dare, investigate!

Note: Most likely the addresses of the chosen sections of memory coincide will not be! Even in me on the middle of article was changed the address of the [skramblernogo] code. In this program there is no table of initialization; therefore in this article nothing it it is discussed. The table of initialization exists only in the programs, written to **Delphi**, although it can be, also, to Borland Of c++. Information on the restoration of the table of initialization you can take from the article **Of sergSh** “unpacking Of aSProtect 2.13 based on the example Of icolover.exe”**,** which lies at the division **of rar- article** on the site **www.cracklab.ru.**

Necessary tools:

OllyDbg 1.10, with [plaginami], for the concealment of the presence of diagnostic routine.

OllyScript by Of sHaG v 0.92 or **ODbgScript by Of epsylon3 v 1.41** (better it)

PETools by Of nEOx of v1.5 RC6 of – is compulsory of this version!

WinHex or another hexadecimal editor.

ResFixer v of 1.0 beta 1 by seeQ of – or another utility for [rebilda] of the resources

PEiD of v0.93 and v0.94

ImpREC 1.6 for restoring the import.

Necessary knowledge:

Knowledge PE Of format of – is compulsory

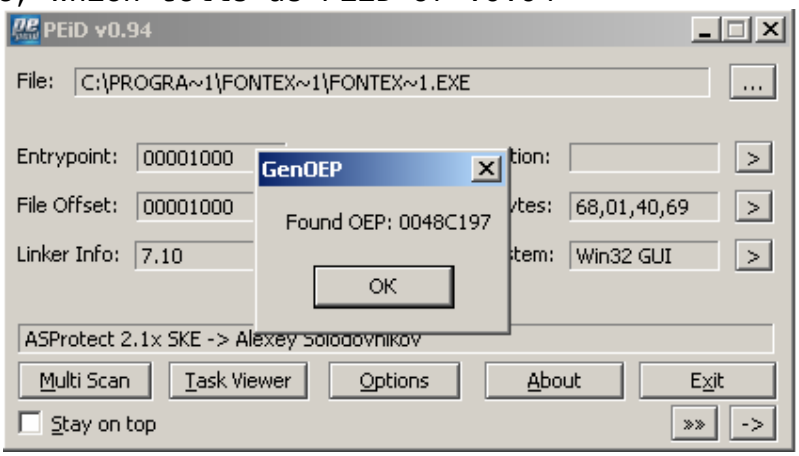
Knowledge of the assembler of – superficially

Necessary habits:

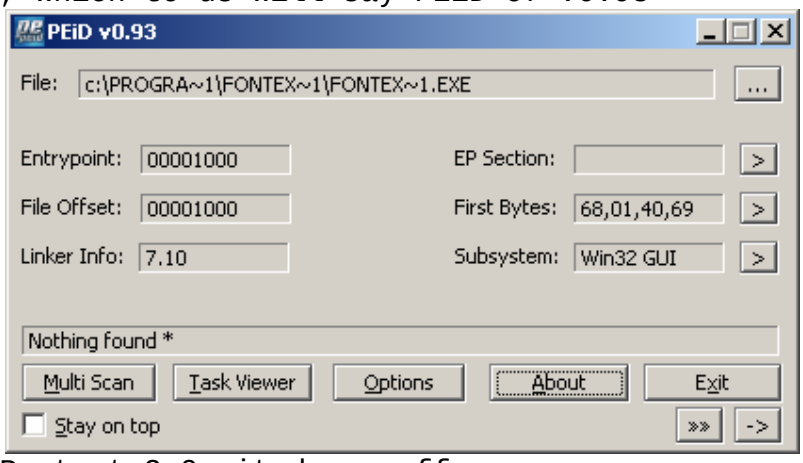
It is necessary to have at least initial habits of unpacking.

Determination of version and the search for original entrance point.

And so, here, which tells us PEiD of v0.94



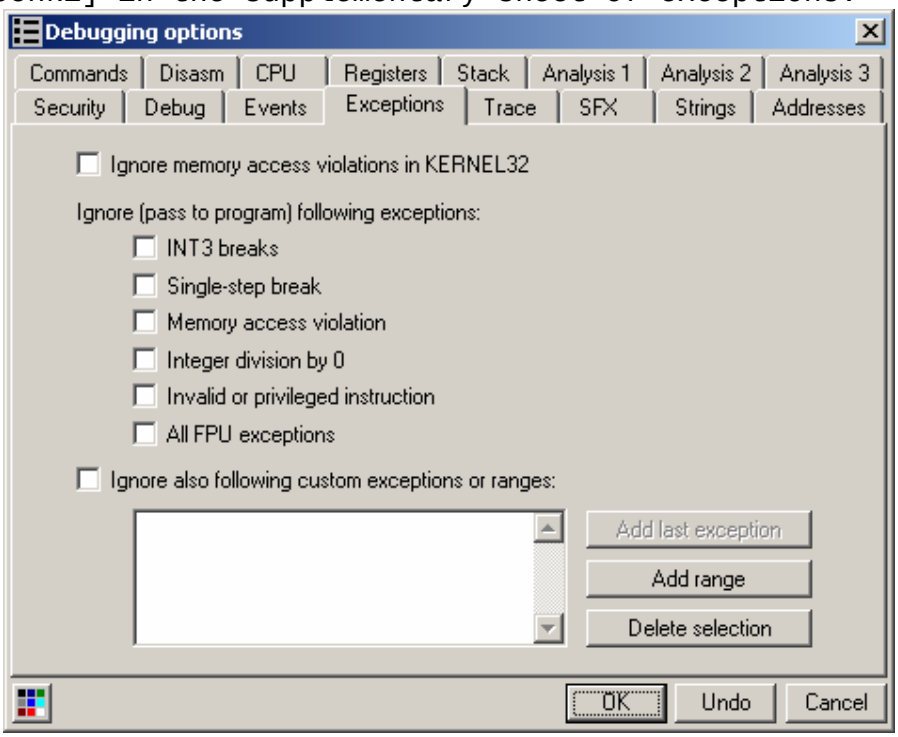
I will say immediately according to the experience that version 0.94 equally defines AsProtect 2.0x, AsProtect of 2.1xxx and AsProtect 2.2 as AsProtect 2.1x SKE.
But here version 0.93 of these 3[kh] of versions can recognize only AsProtect 2.0x, about the remaining versions he speaks Nothing of found *.
Let us look, which to us will say PEiD of v0.93



It means AsProtect 2.0x it drops off.
How to determine, with which of the remained two versions we do deal? This we will explain only further, and we thus far load program into OllyDbg and we see the standard beginning Of asProtect'a of any version (except the early).

```
00401000   68 01406900     PUSH FONTEX~1.00694001
00401005   E8 01000000     CALL FONTEX~1.0040100B
0040100A   C3               RETN
0040100B   C3               RETN
0040100C   93               XCHG EAX,EBX
0040100D   336A 28          XOR EBP,DWORD PTR DS:[EDX+28]
00401010   9E               SAHF
00401011   5D               POP EBP
00401012   93               XCHG EAX,EBX
00401013 - 7F E5            JG SHORT FONTEX~1.00400FFA
00401015   69BC17 07F2080(  IMUL EDI,DWORD PTR DS:[EDI+EDX+908F207]
00401020   129F DC982A65     ADC BL,BYTE PTR DS:[EDI+652A98DC]
00401026   E7 86            OUT 86,EAX                          I/O command
00401028   B8 76346193      MOV EAX,93613476
0040102D   7B 49            JPO SHORT FONTEX~1.00401078
0040102F v 7D 79            JGE SHORT FONTEX~1.004010AA
00401031   6D               INS DWORD PTR ES:[EDI],DX           I/O command
00401032   7B 0B            JPO SHORT FONTEX~1.0040103F
```

Now let us open the options of diagnostic routine and will remove all [galochki] in the supplementary sheet Of exceptions.

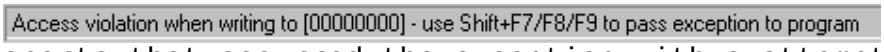


"Why?" – you will ask. Protector with the work generates exceptions, for the difficulty of its fixing. I do not know, can, when it only thought, this it interfere withd coma that fixing its code, but today such Of antiTracing can frighten perhaps that entirely green novice. Well that zh, who harms us, that us will help. Usually the number of exceptions from the moment of the load of program into the diagnostic routine and before complete unpacking one and also (although there are the exceptions (in the sense there is the unequal number of exceptions)). After last exception occurs the even more insignificant correction of data, and then leap to the original entrance point, i.e., in the place, from which the program began to be performed to the protection by protector. This point is our purpose. Costing on the address of original entrance point into the program we let us be able to throw out the unpacked dump of program from the memory to the disk.

And so we start our program (F9). They stopped in this place:

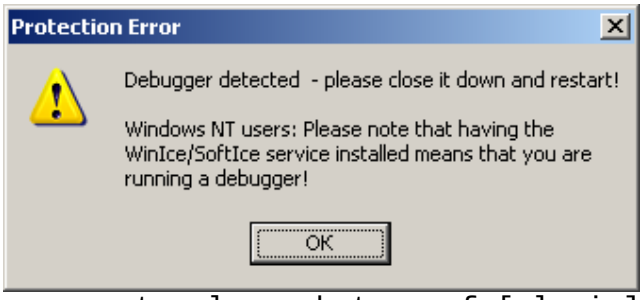
```
00C4B530   C700 7FECC512   MOV DWORD PTR DS:[EAX],12C5EC7F
00C4B536   18A4B5 A31F426( SBB BYTE PTR SS:[EBP+ESI*4+65421FA3],AH
00C4B53D   20C6            AND DH,AL
00C4B53F   5F              POP EDI
00C4B540   67:64:8F06 000( POP DWORD PTR FS:[0]
00C4B546   83C4 04         ADD ESP,4
00C4B549   334424 28       XOR EAX,DWORD PTR SS:[ESP+28]
00C4B54D   C1D8 79         RCR EAX,79                    Shift constant out of range 1..31
00C4B550   58              POP EAX
```

Below in the line of state we see:



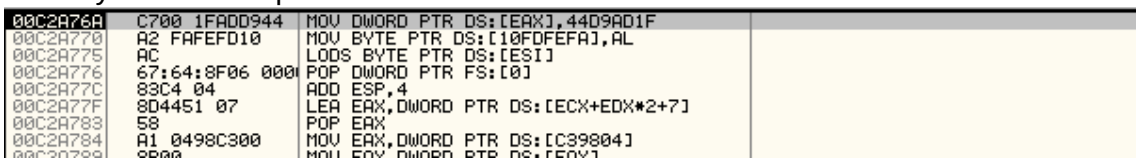
OllyDbg reports that occurred the exception with a attempt at the writing into the memory with address 00000000 and proposes to harvest Shift+F7/F8/F9 in order to transmit this exception into

the program for the subsequent working. This is those exceptions themselves, which are necessary to us. Now it is possible to more precisely determine the version of protector. In the version Of asProtect 2.2, as far as to me known, generally there are no exceptions. Apparently the author introduced large changes in charger. Therefore AsProtect 2.2 drops off. In order to reach the original entrance point necessary to press Shift+F9 to that moment, when program is neglected. If after sequential exception the window appears:



, then it is necessary to place what or of [plagin] for OllyDbg the hiding diagnostic routine from the detection (IsDebugPresent, Hide Of debugger).

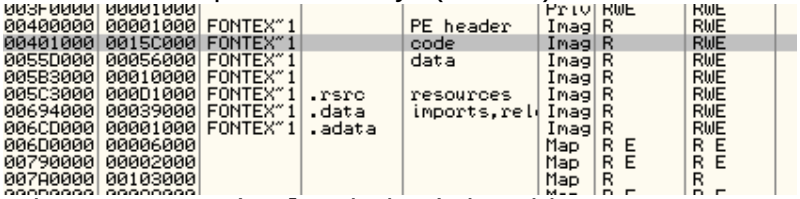
We memorize the number of occurred exceptions and it is reloaded program (Ctrl+F2). We start [pogrammu] (F9), we are interrupted on the exception and harvest Shift+F9 as many times, as they counted exceptions past time minus one (this and understandably, otherwise program again it will be neglected). In me this number is equal to 35. They must stop where that here:



But do not entangle! These places: 2. Us are necessary the second.

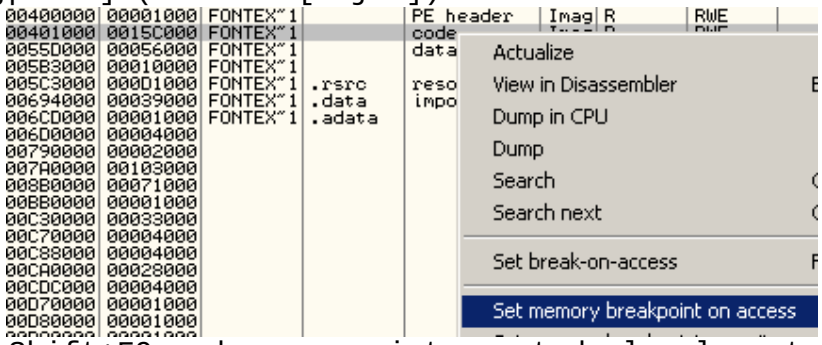
But how to us to now find original entrance point? Simplest way of – this to place the point of stop in the section of the code. Since after last exception in the section of the code of more than anything it is written, the following turning to it will be when the code, located in this section it will begin be carried out. But it will be carried out it will begin certainly from our original entrance point (further OEP) into the program.

It is discovered the map of memory (Alt+M) and we see:

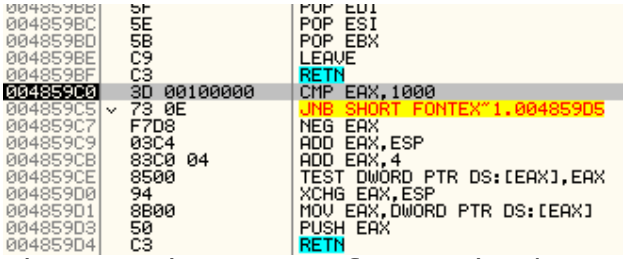


We see that the program is loaded with address 00400000. The first region with address 00400000 with size of 1000 (PE of header) is PE by the title of our victim (read the description PE Of format'a). The following region with address 00401000 with size of 15C000 is the section of the code, it is here to it to us and it

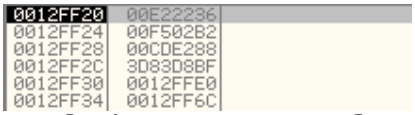
is necessary to place [brejkpoint] on the access to the memory. We place [brejpoint] (further [brjak]):



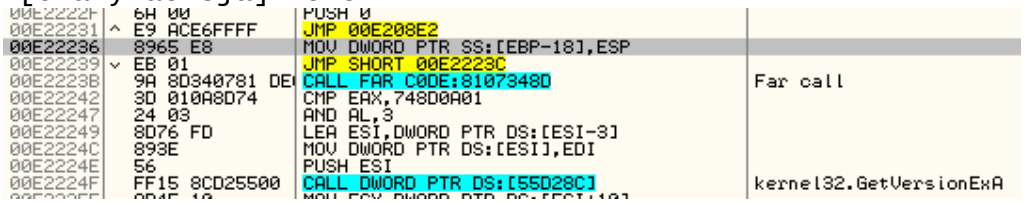
We harvest Shift+F9 and… we are interrupted clearly not on OEP. Well not terribly, simply AsProtect stole in the program the first several bytes of the code and carried out their itself, but in their place it entered zero. It means above that address, where we now are located they must be zero or the command ADD BYTE PTR DS:[EAX], AL of – so [dizassembliruet] zero disassembler. We do look above and… that after features?



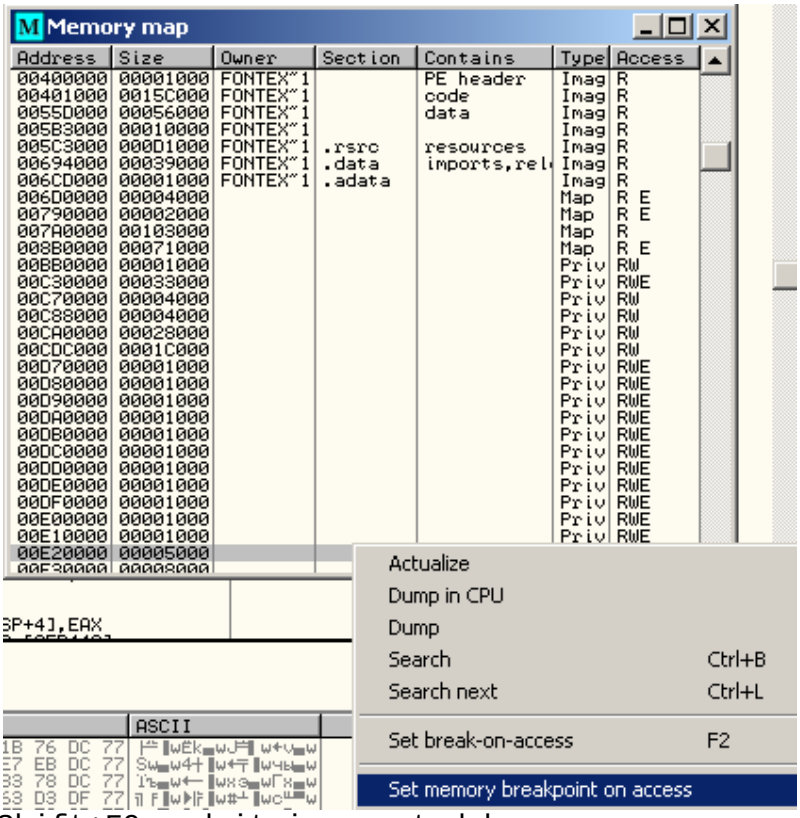
Above is located instruction RET, of zero it does not smell. I will say on the secret that this not OEP, but altogether only one of the numerous functions of program. How I did learn, that this not OEP? This will arrive with the experience. Try to look as it appears BY OEP in the programs, written in the different languages of programming and you will understand everything. Then another question asserts itself: “If this not OEP, this first turning to the section of the code and this is one of the numerous functions of program, then from where was caused this function?”. This question to answer is not complicated. Once this is function, then with its call the address of recovery will be brought in into the stack. We look, what do we have in the stack:



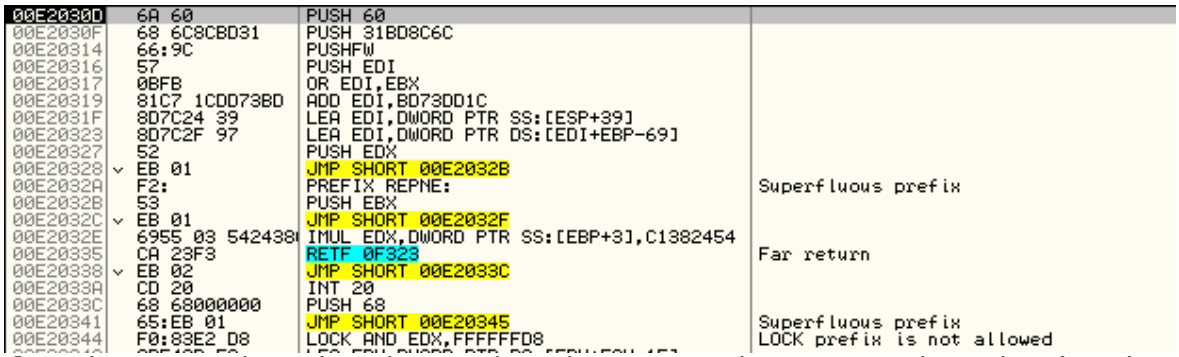
We see that the address of the recovery of 00E22236. Council. Never mind, which is located in the third column of the window of stack. Here – nothing, but is - complete absurdity. Now let us pass to this address, we harvest Ctrl+G, enter 00E22236, Enter.[Okazyvaemsja] here:



We see that to 00E2224F is caused BY API the function Of getVersionExA. We recall, that this function one of the first is caused in the programs, written to Visual C. if this program is written on, then we are located almost on OEP. The result is that program begins they will be carried out not in the section of the code, but in this region to memory and [brjak] to the access to the memory on the last exception must be placed precisely on this region, and not in the section of the code. Let us consider this. In order to determine the beginning of this region we press the key For home and it is exerted on the address of 00E20000, this is a beginning of region. Let us memorize this number and it is reloaded program. We reach the last exception, it is discovered the map of memory and we place [brjak] on the access to the memory to that region itself:



We harvest Shift+F9 and it is exerted here:



If I do not make mistakes, then by command PUSH 60 does begin the program, compiled by what that (I do not remember) however, with version s. I that it is obtained? But is obtained this is what – Of asProtect now it does not steal several bytes with OEP (to the

first call of call), but takes away completely all beginning of program into itself and memories are carried out in region chosen under this code. Well this is not terrible! What does prevent us from [sdampit] program and separately this region memory, and then tightening this region of in the form new section to the dump? Let us look only below and we see:

```
00E2039C  EB 02          JMP SHORT 00E203A0
00E2039E  CD 20          INT 20
00E203A0  8D7F 06        LEA EDI,DWORD PTR DS:[EDI+6]
00E203A3  F2:            PREFIX REPNE:                    Superfluous prefix
00E203A4  EB 01          JMP SHORT 00E203A7
00E203A6  E8 3EEB02CD    CALL CDE4EEE9
00E203AB  2052 57        AND BYTE PTR DS:[EDX+57],DL
00E203AE  2BFB           SUB EDI,EBX
00E203B0  8BFC           MOV EDI,ESP
00E203B2  83C7 04        ADD EDI,4
00E203B5  EB 02          JMP SHORT 00E203B9
00E203B7  CD 20          INT 20
00E203B9  C707 588A5700  MOV DWORD PTR DS:[EDI],578A58
00E203BF  5F             POP EDI
00E203C0  8F07           POP DWORD PTR DS:[EDI]
00E203C2  5F             POP EDI
00E203C3  66:9D          POPFW
00E203C5  68 B008E200    PUSH 0E208B0
00E203CA  E8 31FC0600    CALL 00E90000
00E203CF  8BD8           MOV EBX,EAX
00E203D1  E9 9A190000    JMP 00E21D70
00E203D6  50             PUSH EAX
00E203D7  FF36           PUSH DWORD PTR DS:[ESI]
00E203D9  E9 9D190000    JMP 00E21D7B
00E203DE  8D45 80        LEA EAX,DWORD PTR SS:[EBP-80]
00E203E1  E9 3C170000    JMP 00E21B22
00E203E6  C3             RETN
00E203E7  55             PUSH EBP
00E203E8  E9 F80C0000    JMP 00E210E5
00E203ED  8B00           MOV EAX,DWORD PTR DS:[EAX]
00E203EF  85C0           TEST EAX,EAX
00E203F1  E8 0AFC0600    CALL 00E90000
00E203F6  85C0           TEST EAX,EAX
00E203F8  E9 34210000    JMP 00E22531
00E203FD  56             PUSH ESI
00E203FE  66:9C          PUSHFW
00E20400  51             PUSH ECX
00E20401  2BCD           SUB ECX,EBP
00E20403  8D4C35 68      LEA ECX,DWORD PTR SS:[EBP+ESI+68]
```

We see that the function of 00E90000 it is caused to of 00E203CA and 00E203F1. Moreover in the first case into the stack is placed 00E208B0 (similarly to the address of recovery), but the secondly into the stack generally nothing it is placed. Really this is is one additional trick Of asProtect'a? Let us verify. We harvest by the right button of mouse and we select Search of for->All of commands. In the appeared window we write
call of 00E90000 we harvest Enter.
The imposing list was obtained:

```
00E203CA  CALL 00E90000  (Initial CPU selection)
00E203F1  CALL 00E90000
00E205B8  CALL 00E90000
00E205C2  CALL 00E90000
00E205ED  CALL 00E90000
00E205F4  CALL 00E90000
00E20604  CALL 00E90000
00E20614  CALL 00E90000
00E20826  CALL 00E90000
00E208AB  CALL 00E90000
00E208DD  CALL 00E90000
00E208EF  CALL 00E90000
00E208F9  CALL 00E90000
00E2092A  CALL 00E90000
00E209EA  CALL 00E90000
00E20A27  CALL 00E90000
00E20A32  CALL 00E90000
00E20A9F  CALL 00E90000
00E20ABD  CALL 00E90000
00E20ACF  CALL 00E90000
00E20B8D  CALL 00E90000
```

In all in me it was located 112 calls of these functions. It means, to [sdampit] this region will not come out ☹. “but why!” – you say: “It is possible to [sdampit] and this function!”. But here this will not come out, since this is that “virtual machine itself” (further VM), about which everything they heard, but that it from itself represents and as to fight they do not know with it. To [sdampit] it is possible, but [gimorno] is not-pretty. [Gimorno], because it is already disposed to the current addresses

yes and the size in it impressing, and is not-pretty because not-pretty… And one additional problem consists in the fact that checking registration also occurs in VM.

[Dekompiljatsija] of virtual machine.

However, what such “virtual machine”? AsProtect with the packing of program moves away from it some instructions and they will replace with their call VM. In this context of understanding – this is VM the region memory, in which are carried out the specific actions. The result of these actions will be the same as with the fulfillment of original instructions. But, in contrast to the original instructions, in VM all is carried out more tangled. Maltsters it tried as it is possible to more strongly hide the mechanism of work VM. As a result with the dump of program remain many calls VM, which now is not. Therefore dump is obtained nonworking. To fight with this is possible by several methods. Best - to restore all stolen instructions, to thus remove dependence on VM. For restoring the instructions there are two methods:
1) to completely study work VM, to understand the operating principles with its tables, to understand the size of utilized data and to as a result write analog, but not for fulfilling the stolen instructions, but for their restoration.
2) to find such places in the nucleus VM, where, after stopping at [brjake], it is possible to accurately determine the type of the emulated instruction and directly its operands. For example, after stopping in similar you [mete], on what that to signs to understand, that this is the instruction of cmp, to neglect program further, to again stop and to understand that the register of eax is compared with which the, and after stopping where the further - to understand with which it is compared.

The first method is completely long, is labor-consuming and in principle no one not necessary. The second more promising and the spent time depends only on the power of observation and on the level of meditation. Certainly, I selected the second method ☺. as a result I it wrote script for restoring the stolen instructions.

I will not describe in detail as I it searched for each control point for the script, since, although this and more rapid method, not on so many in order for two passages to understand the logic of that, to what Of [soldovnikov] it dedicated so much time. Script is applied to this article (**RebuildVM.osc**) and I tried as it is possible to in more detail describe each line of the code. To whom is not interesting the theory of – that it can simply neglect script and it on the automaton will restore the stolen instructions, well and all rest I please to pass with me into the peace of the captivating step-by-step laying out of the code ☺.

Let us approach! Let us place indicator on the first call VM and will press Ctrl+ * or by the right button of [myshi]->New of origin of here, thus, after establishing location counter (eip) to

the necessary address. Now we harvest F7 (Step of into) and fall into the function of 00E90000. We see the heap of rubbish and instructions of jmp. How I did understand, where the rubbish? Beginning of the function:

```
00E90000 JMP SHORT of 00E90005
00E90003 INT 20
00E90005 PUSH EDI
00E90006 PUSHFD
00E90007 JMP SHORT of 00E9000C
00E9000A INT 20
00E9000C LEA EDI, DWORD PTR SS:[EBP+ESI+405AC8]
00E90013 SUB EDI, ESI
00E90015 SUB ESP, 20
00E90018 LEA EDI, DWORD PTR DS:[ECX+EDX * 2+5F]
```

We see to 00E90000 that is carried out the leap through the command INT 20 to 00E90005. Then into the stack is placed register EDI. Remains register EFL (state of all flags). Again the leap through the command INT 20 to 00E9000[S]. in EDI is placed dword with the address EBP+ESI+405AC8. From EDI is read ESI. ESP decreases by 20h. In EDI is placed dword with the address ECX+EDX * 2+5F. Note that for the recording of dword'[a] onto the register EDI to 00E90018 is used not one value of the previous calculations. Result simply is rerecorded! I.e. all this can be replaced with the following code:

```
PUSH EDI
PUSHFD
SUB ESP, 20
LEA EDI, DWORD PTR DS:[ECX+EDX * 2+5F]
```

Even then, if you look further, then you will see, that EDI again is rerecorded, so that the command `LEA EDI, DWORD PTR DS:[ECX+EDX * 2+5F]` is not also necessary. Now, if we want, then it is possible to completely clean rubbish from this function and to ascertain that the useful code in it entirely a little, in essence of – this calculation of the address of the passage into the following region of memory. Passage into this region is accomplished with the aid of the command CALL EXX, where instead of EXX there can be any register.

Interesting us passage:

```
00E9012F   26:EB 02        JMP SHORT 00E90134                  Superfluous prefix
00E90132   CD 20           INT 20
00E90134   8DBC0B 3EF2440( LEA EDI,DWORD PTR DS:[EBX+ECX+44F23E]
00E9013B   8DBC21 FC84C50( LEA EDI,DWORD PTR DS:[ECX+C584FC]
00E90142   2BF9            SUB EDI,ECX
00E90144   FFD7            CALL EDI
00E90146   68 848C4EF9     PUSH F94E8C84
00E9014B   337C24 28       XOR EDI,DWORD PTR SS:[ESP+28]
00E9014F   337C24 08       XOR EDI,DWORD PTR SS:[ESP+8]
00E90153   5F              POP EDI
00E90154   2078 CC         AND BYTE PTR DS:[EAX-34],BH
00E90157   00C3            ADD BL,AL
00E90159   0000            ADD BYTE PTR DS:[EAX],AL
00E9015B   0000            ADD BYTE PTR DS:[EAX],AL
00E9015D   0000            ADD BYTE PTR DS:[EAX],AL
00E9015F   0000            ADD BYTE PTR DS:[EAX],AL
00E90161   0000            ADD BYTE PTR DS:[EAX],AL
00E90163   0000            ADD BYTE PTR DS:[EAX],AL
00E90165   0000            ADD BYTE PTR DS:[EAX],AL
00E90167   0000            ADD BYTE PTR DS:[EAX],AL
```

Upon transfer into the following region of memory we fall into the function, which achieves identification of the stolen instruction, and it is more concrete specific, function sorts out the coded values in its table of hash and compares with hash of the current function, if they coincided, then the primary initialization of function is produced and it passes directly for emulation. This function also not large – of the order of 0ACh (172) of bytes.

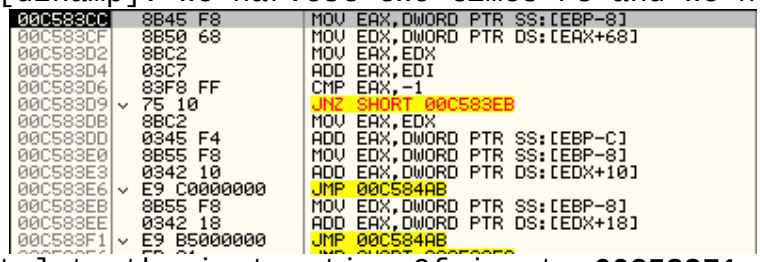
```
00C58559   8945 F8          MOV DWORD PTR SS:[EBP-8],EAX
00C5855C   85FF             TEST EDI,EDI
00C5855E   76 38            JBE SHORT 00C58598
00C58560   EB 01            JMP SHORT 00C58563
00C58562   C7               ???                          Unknown command
00C58563   8B45 F8          MOV EAX,DWORD PTR SS:[EBP-8]
00C58566   0FB600           MOVZX EAX,BYTE PTR DS:[EAX]
00C58569   8B5483 40        MOV EDX,DWORD PTR DS:[EBX+EAX*4+40]
00C5856D   8BC6             MOV EAX,ESI
00C5856F   FFD2             CALL EDX
00C58571   3B45 FC          CMP EAX,DWORD PTR SS:[EBP-4]
00C58574   75 1A            JNZ SHORT 00C58590
00C58576   8B45 10          MOV EAX,DWORD PTR SS:[EBP+10]
00C58579   50               PUSH EAX
00C5857A   8B45 14          MOV EAX,DWORD PTR SS:[EBP+14]
00C5857D   50               PUSH EAX
00C5857E   E8 19FAFFFF      CALL 00C57F9C
00C58583   50               PUSH EAX
00C58584   8BCE             MOV ECX,ESI
00C58586   8B55 18          MOV EDX,DWORD PTR SS:[EBP+18]
00C58589   8BC3             MOV EAX,EBX
00C5858B   E8 D4FDFFFF      CALL 00C58364
00C58590   4F               DEC EDI
00C58591   0373 6C          ADD ESI,DWORD PTR DS:[EBX+6C]
00C58594   85FF             TEST EDI,EDI
00C58596   77 CB            JA SHORT 00C58563
00C58598   68 B485C500      PUSH 0C585B4                 ASCII "111♪◙"
00C5859D   E8 7ACAFEFF      CALL 00C4501C
00C585A2   5F               POP EDI
```

CALL EDX to 00C5856F is occupied by sample and decoding of hash from the table of hash. Immediately the comparison of obtained hash with hash of the current function is produced after it. If they are not equal, then is checked not last this hash in the table. If the latter, then reveals “Error: 111”. But here function to 00C5858B will be carried out only if hash it coincided. Here this function is a emulator of the stolen instructions! Let us place [brjak] on this function and will neglect program. They interrupted. We harvest F7 and fall directly into the emulator. This function most tangled, since many different actions here are produced. Only we see below such instructions

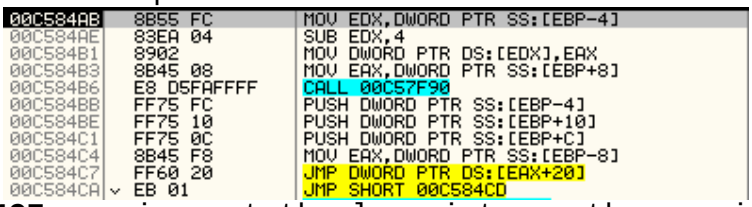
```
00C583A3   EB 01            JMP SHORT 00C583A6
00C583A5   E9 33C08A43      JMP 445043DD
00C583AA   018B 55F88B54    ADD DWORD PTR DS:[EBX+548BF855],ECX
00C583B0   8240 8B C6       ADD BYTE PTR DS:[EAX-75],-3A
00C583B4   FFD2             CALL EDX
00C583B6   2C 02            SUB AL,2
00C583B8   72 12            JB SHORT 00C583CC
00C583BA   74 3D            JE SHORT 00C583F9
00C583BC   FEC8             DEC AL
00C583BE   0F84 82000000    JE 00C58446
00C583C4   E9 DA000000      JMP 00C584A3
00C583C9   EB 01            JMP SHORT 00C583CC
00C583CB   9A 8B45F88B 50   CALL FAR 6850:8BF8458B
```

CALL EDX obtains the type of the stolen instruction:
AL= 0 (it is stolen call)
AL= 1 (it is stolen jmp)
AL= 2 (it is stolen jcc (one of 16 [dzhampov]))
AL= e (they are stolen cmp+jcc)
Yes, ASProtect can emulate immediately several instructions. Respectively now we see that here VM branches out also depending on what type emulation, will be produced passage to the appropriate branch of emulation.
Let us place [brjak] immediately after call edx.[Zapuskaem] program. They interrupted, we see that in AL 0, means it is emulated call (you remember into the stack it was placed the

address of recovery?). Means to us it is necessary first conditional [dzhamp]. We harvest two times F8 and we here.



[Dotrassirujte] to the instruction Of jmp to **00C583F1** and will carry out it. Let us prove to be here.



Jmp to **00C584C7** carries out the leap into another region of memory, but there it occurs nothing interesting, only is normalized stack, the address of recovery is placed into the stack and leap to the function of program is carried out. I.e. if in the program to the protection it was as follows:

Call of xxxxxxxx

The **ASProtect** substitutes this by the call VM, where, if we strongly strongly simplify it is carried out:

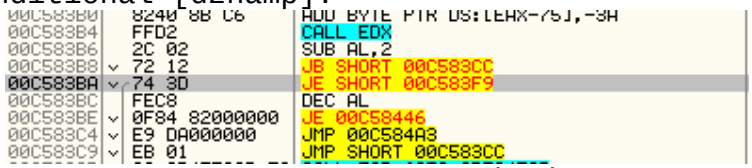
Push the address of the recovery

Jmp of xxxxxxxx

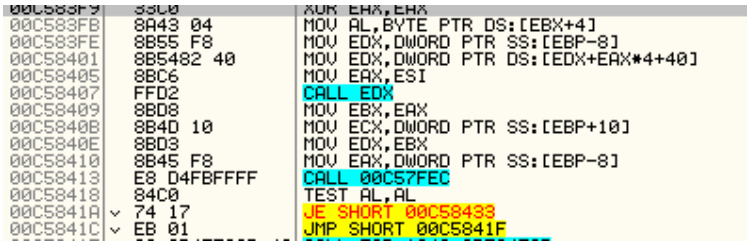
Where instead of xxxxxxxx there can be both the address in the section of the code and the address in the chosen region of memory.

In my script I substitute call VM by precisely such instructions. The address, which the instruction **of push** places into the stack necessary to change by the address, with which will be placed the stolen code after dump. With emulation **of call** they were dismantled. That I can say apropos emulation **of jmp**, and nothing. This one and the same, with one reservation alone, that, if it is caused BY VM for emulation **of call**, then before **call VM** goes **push** of xxxxxxxx. But if **jmp**, then nothing it is placed into the stack.

Let us return to the function, which determines the type of the stolen instruction, for this sufficient several times to harvest key “-” on the digital keyboard. Let us move indicator for following conditional [dzhamp].



And let us press Enter. Let us prove to be here:



Call of edx to 00C58407 determines the type of the stifling of jcc. After its fulfillment of eax contains the type of the stifling of jcc.

Number of [dzhampa]	[Dzhamp]	[Opkod]
0	Jo	70
1	Jno	71
2	Jb	72
e	Jnb	73
4	Je	74
shch	Jnz	75
'	Jpe	7A
"	Jpo	7B
8	Js	78
9	Jns	79
A	Jbe	76
B	Ja	77
C	Jl	7C
D	Jge	7D
E	Jle	7E
F	Jg	7F

As you see entire **jcc** they go practically on the growth, with exception of two.
But **call** to **00C58413** performs complex logical operations, leading the values of conditional flags to one – 0 or 1, and is placed it in **ZF**. I.e., converts any (one of 16) of [dzhampov] in **jz**.
Depending on the state of flag **ZF** is carried out one of two passages. Both they lead to the functions of the formation of the address of passage, the first - if condition is satisfied, the second - if no. But passage is further accomplished there, where during emulation **of call**.

I.e. if in the program to the protection it was as follows:

Jcc of xxxxxxxx

That ASPR $$RTASPR - automated system of planned calculations substitutes this by the call VM, where, if we simplify, it is carried out:

Call Of getStolenJmp
Call Of convertFlags
Jz xxxxxxxx
Jmp of xxxxxxxx

Well here, with emulation **of jcc** also they were dismantled. Remained most complex – emulation **of cmp+jcc**.

Again let us return to the function, which determines the type of the stolen instruction and let us place indicator on following conditional [dzhamp].

```
00C583B0   8240 8B C6      ADD BYTE PTR DS:[EAX-75],-3A
00C583B4   FFD2            CALL EDX
00C583B6   2C 02           SUB AL,2
00C583B8   72 12           JB SHORT 00C583CC
00C583BA   74 3D           JE SHORT 00C583F9
00C583BC   FEC8            DEC AL
00C583BE   0F84 82000000   JE 00C58446
00C583C4   E9 DA000000     JMP 00C584A3
00C583C9   EB 01           JMP SHORT 00C583CC
```

Let us press Enter. Let us prove to be here:

```
00C58446   8BCE            MOV ECX,ESI
00C58448   8B55 0C         MOV EDX,DWORD PTR SS:[EBP+C]
00C5844B   8B45 F8         MOV EAX,DWORD PTR SS:[EBP-8]
00C5844E   E8 D5FDFFFF     CALL 00C58228
00C58453   8945 10         MOV DWORD PTR SS:[EBP+10],EAX
00C58456   EB 01           JMP SHORT 00C58459
```

You will memorize this address, since for us it is necessary to here still return. **Call** to **00C5844E** emulates the instruction **of cmp**. We place on it indicator and harvest Enter. Now we proved to be directly in this function. We see below this code only:

```
00C58260   FFD2            CALL EDX
00C58262   8BD0            MOV EDX,EAX
00C58264   80EA 08         SUB DL,8
00C58267   0F92C2          SETB DL
00C5826A   80FA 01         CMP DL,1
00C5826D   75 11           JNZ SHORT 00C58280
```

Call of edx to **00C58260** determines the first operand for the comparison. If into **eax** after the fulfillment of function the number from 0 to ", then the first operand of – one of eight registers:

Number of the register	Register
0	Eax
1	Ecx
2	Edx
e	Ebx
4	Esp
shch	Ebp
'	Esi
"	Edi

Otherwise this is constant and its value is calculated only below. We see below the same code:

```
00C5829C   FFD2            CALL EDX
00C5829E   8BD0            MOV EDX,EAX
00C582A0   80EA 08         SUB DL,8
00C582A3   0F92C2          SETB DL
00C582A6   80FA 01         CMP DL,1
00C582A9   75 13           JNZ SHORT 00C582BE
```

Entire the same, only for the second operand. Even we see below this:

```
00C582DC   FFD2            CALL EDX
00C582DE   83E0 7F         AND EAX,7F
00C582E1   83F8 04         CMP EAX,4
00C582E4   77 4E           JA SHORT 00C58334
00C582E6   FF2485 ED82C50( JMP DWORD PTR DS:[EAX*4+C582ED]
```

Call of edx to **00C582DC** determines the type of that emulated **of cmp**. In **eax**, after the fulfillment of this [kella] and command **and eax, 7F**, is located the number, which designates the type of emulation:

Number of	Mask of the comparison

the comparison	
0	**Cmp of dword of ptr [????????]????????**
1	**Cmp????????, dword of ptr [????????]**
2	**Cmp of byte of ptr [????????]??**
e	**Cmp??, byte of ptr [????????]**
4	**Cmp????????????????**

Where instead of questions can stand either register or constant. **Jmp** to **00C582E6** accomplishes a passage to the instructions, which initialize one or another type emulations.
For example, during emulation **of cmp of dword of ptr [of eax], ebx** it is necessary to first obtain **dword** to **eax**, to and then compare it from **ebx**, that also make these instructions.

We look still below

```
00C58339   E8 DECCFEFF    CALL 00C4501C
00C5833E   8B5424 10      MOV EDX,DWORD PTR SS:[ESP+10]
00C58342   8BC5           MOV EAX,EBP
00C58344   E8 CBFEFFFF    CALL 00C58214
00C58349   83C4 14        ADD ESP,14
00C5834C   5D             POP EBP
00C5834D   5F             POP EDI
00C5834E   5E             POP ESI
00C5834F   5B             POP EBX
00C58350   C3             RETN
```

Call to **00C58344** answers directly for the comparison of operands. Before its fulfillment into **eax** lies the value of the first operand, into **edx** of – of the second. All, with emulation **of cmp** were dismantled, now let us look how is emulated **jcc** after it. You do remember, I did request to memorize address, before the entrance into this function? Let us return to it.

```
00C58446   8BCE           MOV ECX,ESI
00C58448   8B55 0C        MOV EDX,DWORD PTR SS:[EBP+C]
00C5844B   8B45 F8        MOV EAX,DWORD PTR SS:[EBP-8]
00C5844E   E8 D5FDFFFF    CALL 00C58228
00C58453   8945 10        MOV DWORD PTR SS:[EBP+10],EAX
00C58456   EB 01          JMP SHORT 00C58459
```

Jmp to **00C58456** is altogether only debris instruction for the trapping of disassembler. We place on it indicator, harvest Enter even we see that the instructions recognized correctly.

```
00C58459   33C0           XOR EAX,EAX
00C5845B   8A43 04        MOV AL,BYTE PTR DS:[EBX+4]
00C5845E   8B55 F8        MOV EDX,DWORD PTR SS:[EBP-8]
00C58461   8B5482 40      MOV EDX,DWORD PTR DS:[EDX+EAX*4+40]
00C58465   8BC6           MOV EAX,ESI
00C58467   FFD2           CALL EDX
00C58469   8BD8           MOV EBX,EAX
00C5846B   8B4D 10        MOV ECX,DWORD PTR SS:[EBP+10]
00C5846E   8BD3           MOV EDX,EBX
00C58470   8B45 F8        MOV EAX,DWORD PTR SS:[EBP-8]
00C58473   E8 74FBFFFF    CALL 00C57FEC
00C58478   84C0           TEST AL,AL
00C5847A   74 17          JE SHORT 00C58493
00C5847C   EB 01          JMP SHORT 00C5847F
00C5847E   E9 8B45F88B    JMP 8CBDCA0E
```

However, and that we do see? Yes nothing else but previous emulation **of jcc**, only with another address. I was shocked! Why two times to write one and the same? I think that you will be dismantled themselves.

Well here we dismantled cursor VM. As you see nothing complex. To there remained only write script, that I already made. But! Script is not a little stable. In it there are several [bagov], which I will not correct. This script was my first script for OllyScript, I studied his possibilities and commands. Therefore he is not very optimized and it is terribly realized.
Known [bagi]:

Incorrect algorithm of the restoration of that emulated **of call**. Necessary to correct **push** only if with obtaining of the type of the stolen instruction it will be accurately known that this **call**. But script restores push always at the given moment, if it costs before **call VM**, and this not correctly and there were already errors.

Script itself searches for the empty place for the at the end current region of memory under the restored instructions and it is not always correct. Sometimes simply does not be sufficient vacant place and the part of instructions is not restored. I treated this by the fact that with the load of program intercepted **VirtualAllocExA** and looked in the stack the size of the inquired memory, if it coincided with the size of necessary to me region memory, then increased it by 1000h.

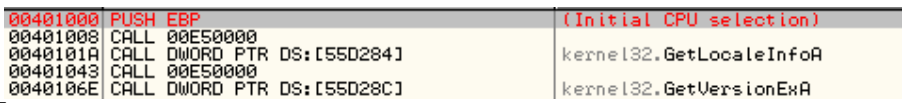
For the fastening of knowledge, you [perezapustite] program and place [brjaki] on the obtained control points. It will be useful to with its own eyes see work VM.

Now it is possible to restore all stolen instructions and to [sdampit] this region memory and to hitch it as the new section to the dump. But for the fact that to restore the stolen instructions, to script it is necessary to indicate the base address of section, which will be added to the dump. But for this by first [sdampim] our program. And here here again problem. ASProtect, besides the theft of instructions, is occupied even and by the theft of the calls API of functions, substituting call API, by the call of its function, which is located in the chosen region memory. Therefore, if we to [sdampim] program, these calls will indicate in anywhere. It means it is necessary to first restore all which is connected with the import.

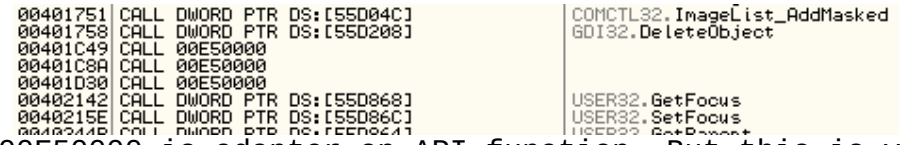
Restoration of the import

In this version Of aSProtect'[a] the import is protected considerably stronger. Protection appears just as in the previous versions, and it here works differently.

For those, who are not familiar with the old protection. Pass into the section of the code. For this we harvest Ctrl+G, we write with 00401000 and harvest Enter. Now we harvest by the right button of [myshi]-> **Of search of for->All of intermodular of calls** (to find all calls of functions).
We see:



It is below:

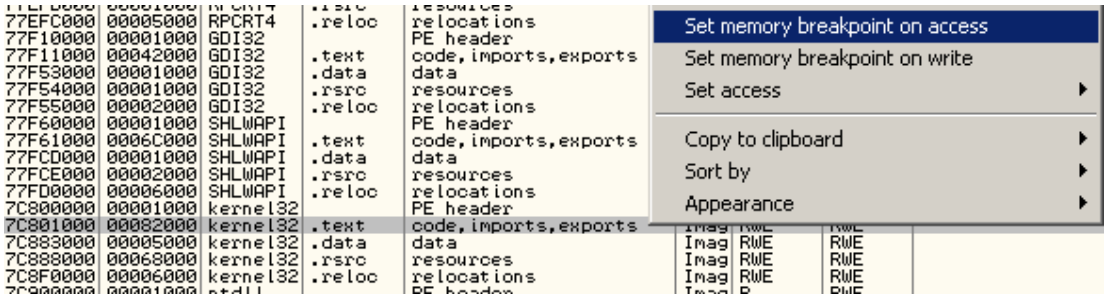


Call of 00E50000 is adapter on API function. But this is very sly adapter. It not as small as in the old versions (entirely old) yes

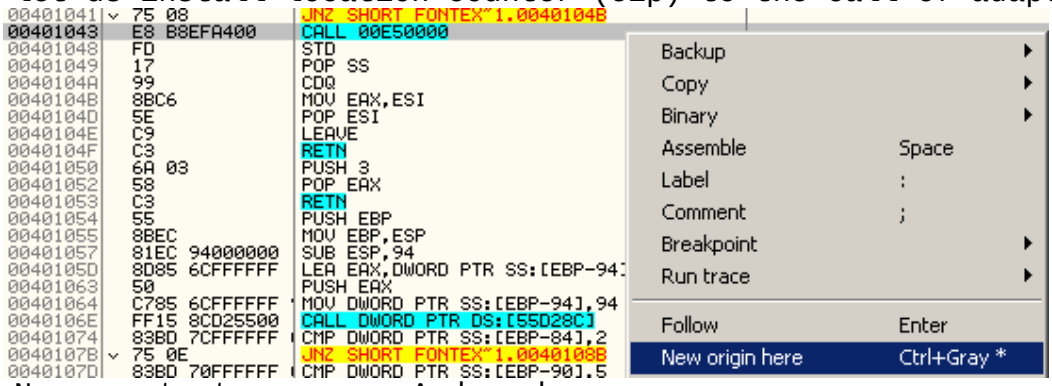
even works according to another principle. Earlier than ASProtect instead of the address API in IAT substituted the address of its adapter and for API of function was caused adapter on API function. Now ASProtect takes function from IAT, searches for all its calls and substitutes them by the calls of its universal function, and then is moved away the address API of function from IAT. Thus! Now in order to restore this function, it is necessary to learn, which for API function causes this adapter and to restore the address of this API in IAT, to and then restore its call. It is here with this exactly of problem in this version. Theory is one and the same. It is necessary to learn beginning and end of the table of import. Then as that to learn that for API causes the function of – adapter. To find this function in IAT (if there is), if no, then write it into the end IAT. To change the call of adapter to the call API of function. In the old versions Of aSProtect'[a] (from 1.33 to 2.00) the adapter worked as follows:

Generally the this was not the adapter, but the function, which, it formed. Those, who are familiar with this concept, as Delay Of import of – know, this is it and there is. With turning of program to this function, ASProtect calculated the address of function necessary TO API and formed new adapter to it, then money-changers the address of the call of its function to the address of adapter. In more detail read the appropriate articles. I will say only that for the formation of the adapter Of aSProtect used API the function Of virtualAlloc. After placing on it [brjak] and, after neglecting program, we jingled to VirtualAlloc and in the stack, on the specific displacement was visible the address of stolen API.

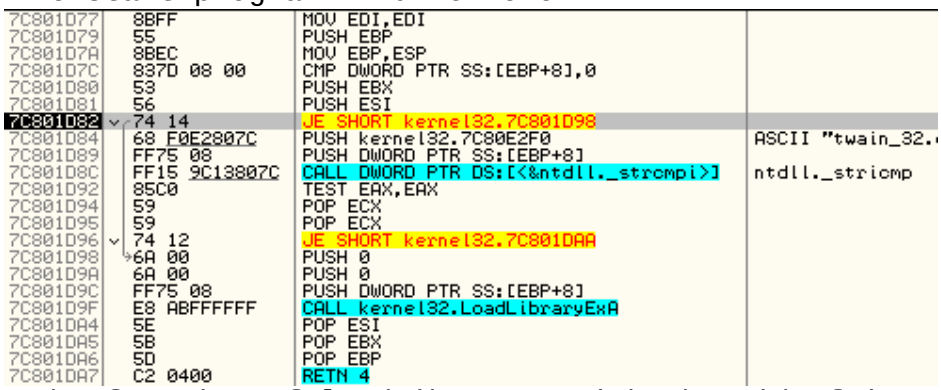
Now everything otherwise. Now ASProtect uses immediately two forms of adapters. They appear equally – the call of one and the same function in the chosen memory. But, in one case Of aSProtect continues to as before cause VirtualAlloc, to form adapter and to [propatchivat] the place of call, and in other of – it connects its new VM for emulation of the call API of functions. For this he does not use VirualAlloc and not [propatchivat] the call of adapter. But how to us to now learn the address of [ukradenoj] API of function? Nothing complex. It suffices to a little [potrassirovat] and we will see in the stack address API. But there is a universal method, with the installation of [brjaka] on API the function, which uses ASProtect for determining the address API. I thought that ASProtect cannot but use not one API of function for the work its VM. Let us place [brjak] on the access to the memory in the section of the code in the library of kernel32.dll.



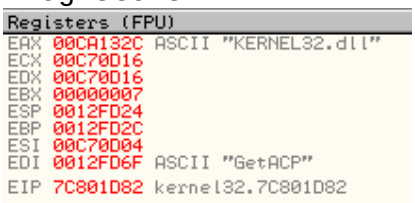
Then let us install location counter (eip) to the call of adapter.



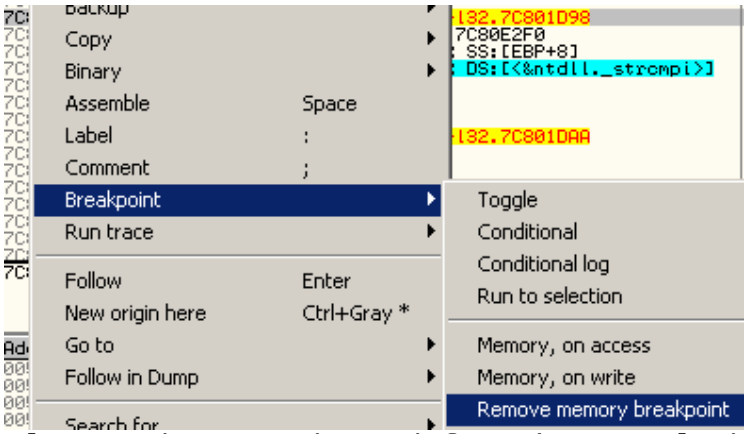
All. Now we start program. And we here:



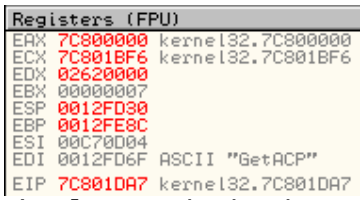
This API the function Of loadLibraryA, with the aid of it ASProtect [podgruzhaet] or, if it is already loaded, is obtained the base DLL, in which is located stolen API the function. We look into the window of the registers:



To [uuu]… This is already very interesting. Give [dotrassiruem] to the instruction **RETN 4**, that would load necessary DLL, and then let us look, where indicator to the name of function will move. And so, we remove [brjak] from the section of the code:



we harvest Ctrl+F9 and we at the end function. We look into the window of the registers:

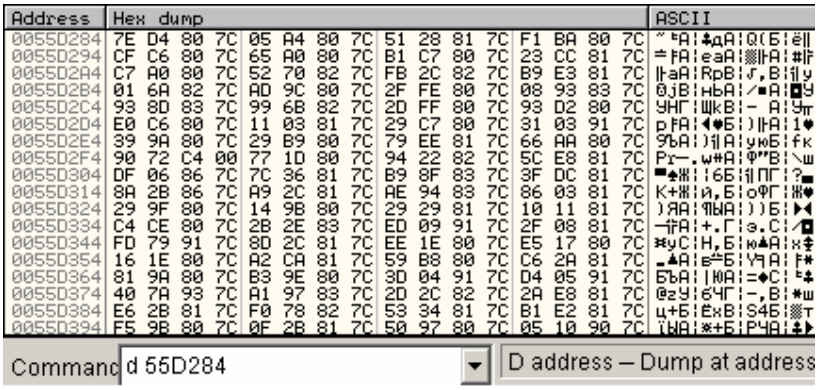


It is excellent! In eax is located the base of the necessary of dll, while into edi indicator to the name of the stolen function. Who does not know – this two parameters, the necessary for API functions Of getProcAddress, which returns the address API of function. The experiments can be ended on this, since more than ASProtect API functions does not cause (only stolen). Yes to us in the principle of more than anything and it is must. Now it is possible to write script on the restoration of adapters. But there are difficulties. To us it is not possible to allow the fulfillment of the stolen function, otherwise we will lose control over the program. If we interrupt work VM, then the balance of stack will be disrupted and during the restoration of the large number API of functions it simply will fall through and program will collapse. Therefore before the fulfillment of adapter we will preserve the register of esp, and after the restoration of adapter - respectively restore the register of esp. To calculate the address API of function we will be with the aid of the function Of getProcAddress. Cause it we will be with the aid of the injection of the code. Everything else, as in the previous versions Of aSProtect'[a]. by the way, the it turned out that this method works also for the previous versions. Therefore we will restore all adapters by one script. Script, as you surmised, I already wrote (**IAT_Recover.osc**) it itself finds the adapters Of aSProtect'[a] it restores them, but it should indicate beginning and end IAT.

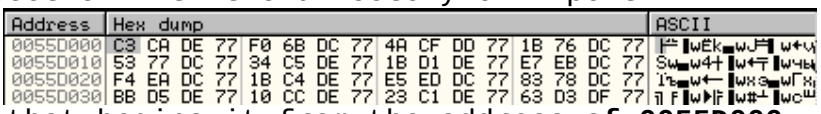
I fairly often hear, that in many appear the problems with the determination of beginning and end IAT, and also about the fact that in IAT much rubbish is, that among it also are encountered API of function. Give let us try to solve this problem. For this let us pass into the section of the code and will find any call API of function.

```
00401000   55              PUSH EBP
00401001   8BEC            MOV EBP,ESP
00401003   51              PUSH ECX
00401004   51              PUSH ECX
00401005   56              PUSH ESI
00401006   33F6            XOR ESI,ESI
00401008   E8 F3EFA400    CALL 00E50000
0040100D   C46A 07         LES EBP,FWORD PTR DS:[EDX+7]  Modification of segment r
00401010   8D4D F8         LEA ECX,DWORD PTR SS:[EBP-8]
00401013   51              PUSH ECX
00401014   68 04100000    PUSH 1004
00401019   50              PUSH EAX
0040101A   FF15 84D25500   CALL DWORD PTR DS:[55D284]   kernel32.GetLocaleInfoA
00401020   85C0            TEST EAX,EAX
00401022   74 1F           JE SHORT FUNTEX~1.00401043
```

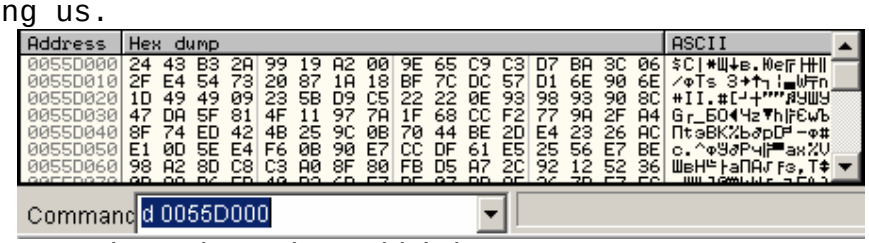
We see that is caused BY API the function **Of getLocaleInfoA**, whose address lies to **0055D284**. This address (**0055D284**) is one of the addresses IAT. Let us determine the beginning of the section, in which is located this address. For this in the command line we write **with d of 0055D284** and harvest Enter. We see:



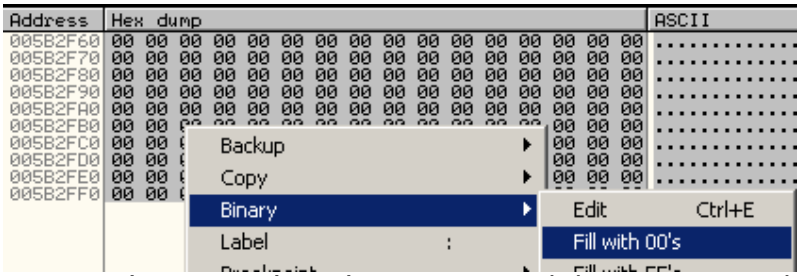
This is IAT. To novice certainly this window not about which will say, this understanding will arrive with the time (experience). Now let us make a window of dump active (sufficient to call to it by mouse) and let us press the key **For home**. Now we in the beginning section with the directory of import.



It leaves, that begins it from the address **of 0055D000**. Let us memorize and [perezagruzim] program. Thus far ASProtect it did not begin its work, let us look, which is located in th section e interesting us.

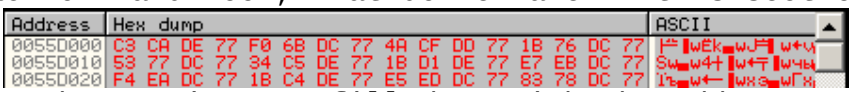


This is none other than the rubbish, not necessary not To aSProtect'[u], not to us. Let us place indicator on number 24 to **0055D000** and let us twist the window of dump downward to the end itself by mouse for [skrol]. Now let us stop up the key **For shift** and will call by mouse to quite last [chiso] of this section. Well here was isolated entire section. Now we harvest by the right button of mouse we select:

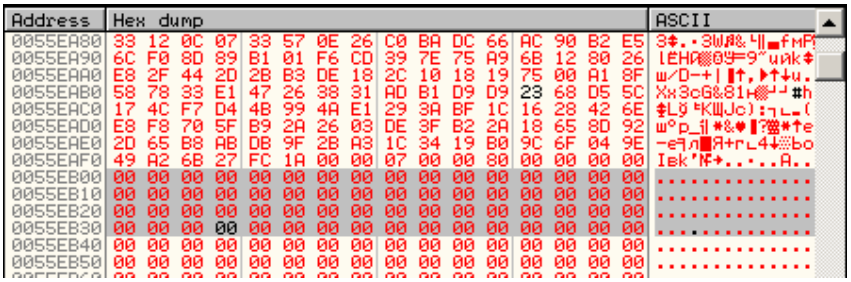


Well here. Now entire section is oppressed by zero, and it means, there is no rubbish greater:

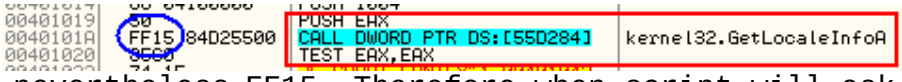
Now we reach OEP and look, what do we have in this section.



We see that the section was filled up with the addresses of the imported functions. Let us look end IAT. It is here here easy to be mistaken. Therefore I always search for the empty place at the end for section and for itself I consider that this is a end IAT.



We will consider that began IAT **of 0055D000**, and the end **of 0055EB00**. If we did not drive in this section by zero, then she would be entire filled with rubbish (except the addresses API of functions) and we did not find vacant place.
Apropos adapters. As you know programs written to **Delphi** they have adapters on API of the function of the form
jmp of dword of ptr [of xxxxxxxx**]**,
and on SI of – **of call of dword of ptr [**of xxxxxxxx**]**. It is natural that also [opkody] in these instructions are different: in the first case **FF25**, in the second of – **FF15**. Therefore to script necessary to indicate what form passages we restore. You do remember the first obtained call API?

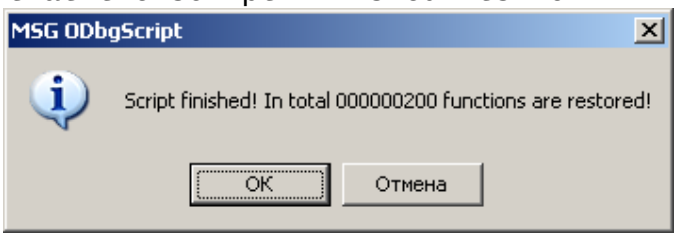


It means nevertheless FF15. Therefore when script will ask to use [obkod] “**of call**” (**FF15**) necessary to harvest **yes**. Well that zh, let us neglect script and will wait for the end of work. I recommend the decreasing of the window Of ollyDbg to the minimum,

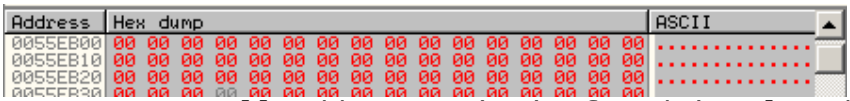
since with the work of script, [Olli] in the line of state he writes:

Too long (recursive?) SEH chain

And in this case very slowly it works, and here if window was minimized, then everything is normal and script very rapidly works.
We start script and on the first a question answer **0055D000**, on the second **of 0055EB00** well and to third **yes**. Through several minutes we see that the script finished its work.



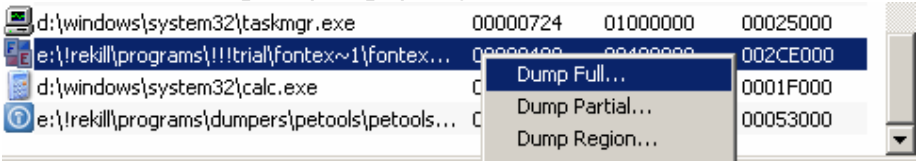
200h = 512 functions are restored! Yes, I tried to optimize script, that he so rapidly worked. Let us look, did add the script of the address of functions in IAT.



We see that no. Means all addresses it it found in already existing IAT. If it added function, then respectively would increase size IAT.

Here now we can [sdampit] program to the disk.

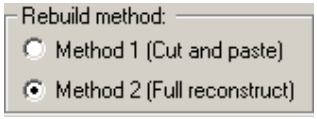
We start PETools and [dampim] program.



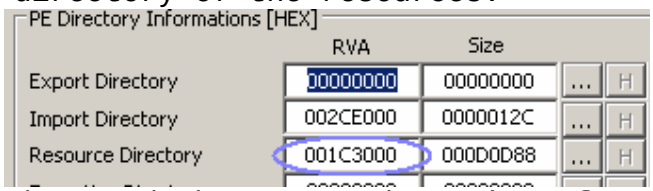
To before restore import, give to [otrezhim] in dump excess sections. Indeed now file is completely unpacked, import let us soon restore, with VM will be finished – why to us now the section Of aSProtect'[a]? But it cannot be thus far cut off, since ASProtect steals even and resources. If we will cut off its section, then let us be deprived of the part of resources. Give them let us restore.
I use **ResFixer by of seeQ**. We start this remarkable utility and we select our dump. We see that the utility counted all resources of our dump. Now let us twist window downward. We see:

Res type	File Offset	RVA 2 Data	Size	Sec №	Sec name
BITMAP	001C6A18	0026AF98	00000144	04	.rsrc
ICON	001C6A28	0029DDBC	000002E8	05	.data
ICON	001C6A38	0029DC94	00000128	05	.data
ICON	001C6A48	0029D3EC	000008A8	05	.data
ICON	001C6A58	0029CE84	00000568	05	.data
ICON	001C6A68	0029C1DC	00000CA8	05	.data

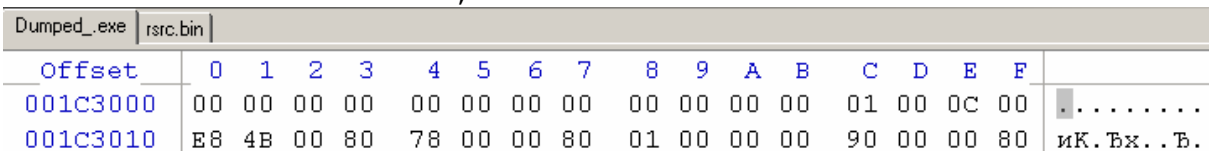
Everything which is marked by red, ASProtect of [sper] to its section and we should return this in the place. We select as Rebuild Of method:



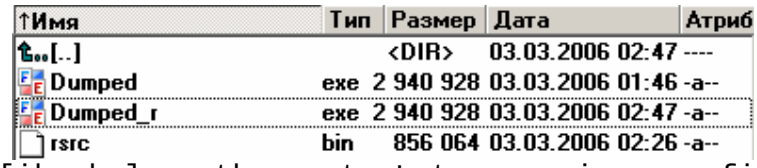
And we harvest Rebuild. We prescribe any, pleasing itself name and preserve the restored section to the disk.
Now let us open our dump with the aid of **PETools** and will open Directory Of editor (we harvest to the button Of directories). We look RVA of the directory of the resources:



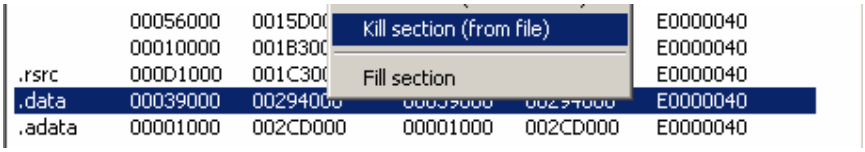
Now let us open dump and the restored section of resources into **WinHex'[e].** in the dump let us pass to the address of 001C3000, for this we harvest Alt+G, enter **001C3000** and harvest Enter.



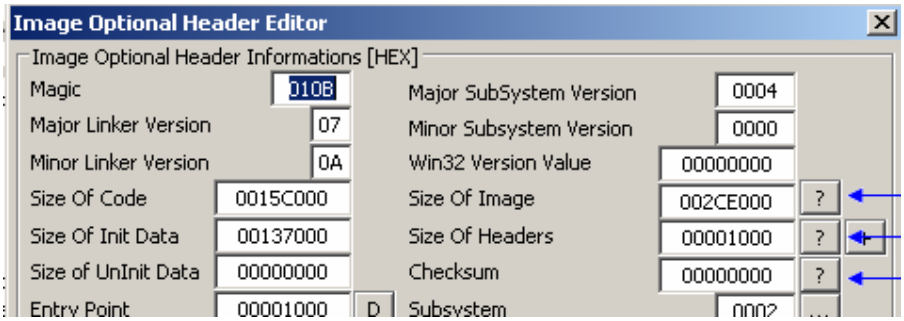
It is now discovered supplementary sheet with the directory of resources, we harvest Ctrl+A (to isolate everything), Ctrl+Shift+C (to copy as the sequence of bytes) and we pass to the supplementary sheet with our dump. We place cursor on **00** to **001C3000** and harvest Ctrl+B (to put the sequence of bytes), we agree with all questions and preserve our dump. Let us look to [ikonku] of our dump, if it did not disappear, then everything is normal.



We see that [ikonka] on the spot. Let us again open file into PETools and will open Section Of editor (we harvest to the button Of sections) and let us remove both sections after the section of rsrc.

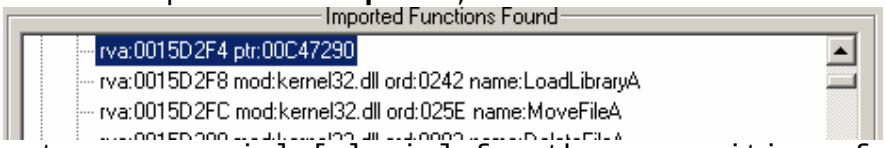


After this, we should correct some parameters in PE title, for this let us shut the current window and will open Image Of optional Of header Of editor (we harvest to the button Of optional Of header). We harvest to all [voprosiki] in this window:

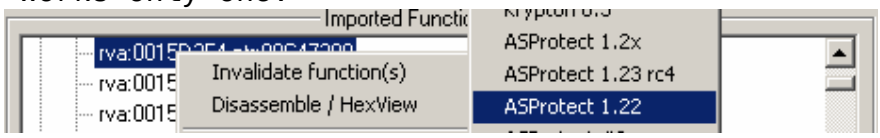


We shut **PETools**.

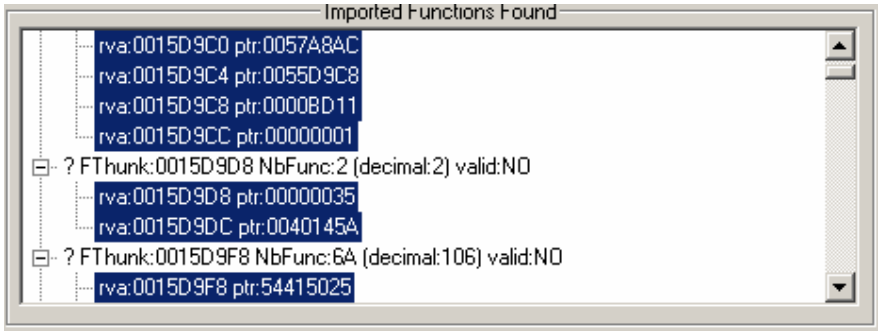
Now we start **ImpREC** for restoring the import and we select our process. In the field **OEP** it is possible to introduce although that (main thing in the limits of file) I entered 1000, since present OEP in the file we as yet do not have. In the field **RVA** we introduce the address of the table of import minus **Of imageBase** (address of load) of our program, i.e., **of 0055D000-00400000=0015D000**. In the field **Of size** we introduce **0055EB00-0055D000=1b00**. We press **GetImports**, then **ShowInvalid**.



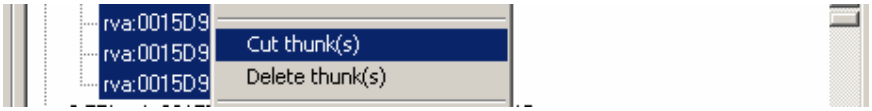
Let us try to use special [plagin] for the recognition of the old adapters Of aSProtect'[a]. in me of such [plaginov] much, but correctly works only one.



We see that the function recognized – this **GetProcAddress**. We again harvest **ShowInvalid**.

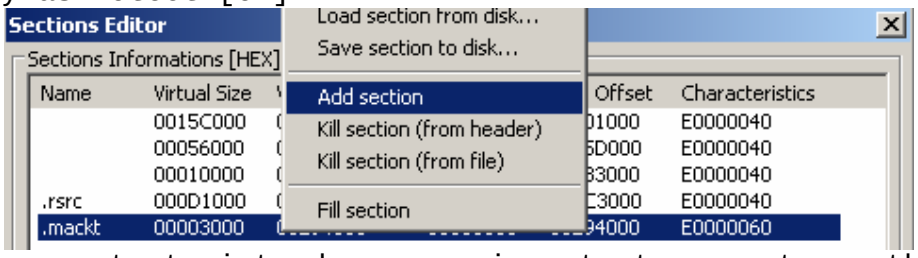


Now, if we twist window to the bottom itself, we will see, that there is not one identified address API of function. Therefore boldly we harvest:

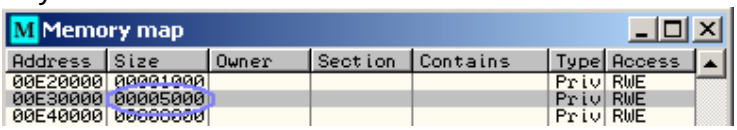


All unrecognized addresses were removed. This it indicates only that all [vosstanovlenye] with the aid of the script adapters already have a address in IAT. But here if one address was added in IAT, then in the very to the bottom of window we would see identified API of function.

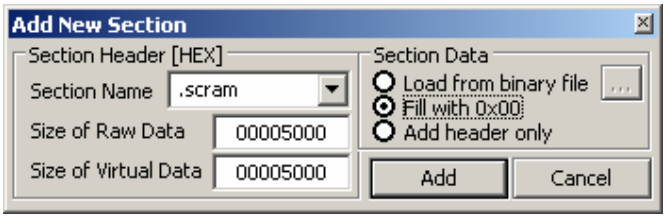
Now we harvest FixDump and we select our dump. It is finished with the import.
Is discovered dump with the aid of PETools and is discovered Section Of editor (we harvest to the button Of sections). We should add the new section, into which we will place th code e stolen By aSProtect'[om].



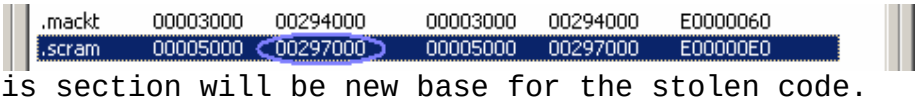
PETools requests to introduce some important parameters: the name of section, the virtual and real size of section. But so he asks, we do want to put section from the file, to fill with its zero or to put the description of section only into the title, but not to put it in the file. As the name it is possible to introduce everything, anything. In order to determine the size of section, let us open into OllyDbg the map of memory and will look the size of region memory with the stolen code.



It means we introduce into PETools as the virtual and real size of the section of – **00005000** we select the flag **Of fill of with of 0x00** so that the editor would create both the description of section in PE title and section itself in the file.



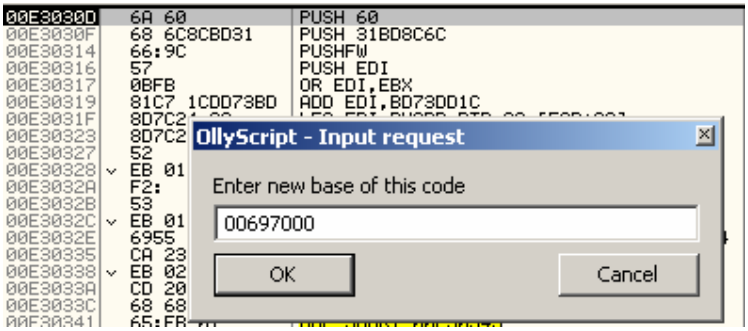
We harvest Add and look VA of new section.



VA of this section will be new base for the stolen code. Specifically, this value must be introduced into the script, which restores the stolen instructions.

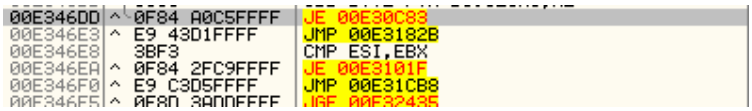
[Perezapuskaem] program we reach OEP. All, it is possible to start script on the restoration of the stolen instructions. We start script, await thus far it it [samonastroitsja] and it will inquire the new base of the code. We after which introduce VA of new

section in the dump plus **Of imageBase**, i.e., **of 00297000+00400000=00697000**.

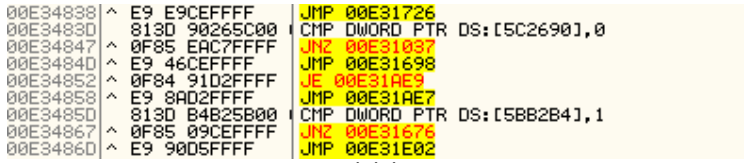


We harvest Enter and await, until script restores instructions. All, script worked out. We harvest ***** on the digital keyboard in order to move for the instantaneous value **of eip**. But now let us press the key **For end** and let us prove to be the at the end chosen memory. We harvest **Pages Up**, until zero, are changed into the code. Beginning from the address **of 00E346ED** it begins the imposing list of conditional [dzhampov] and commands of comparison. This is the restored instructions. Here are some fragments.

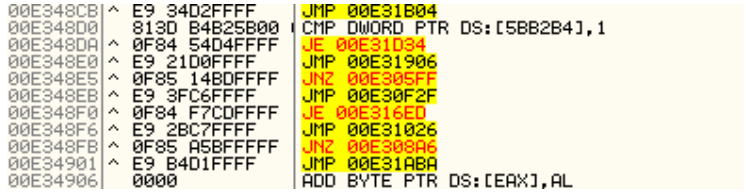
* * *



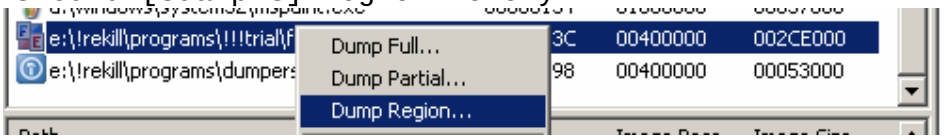
* * *



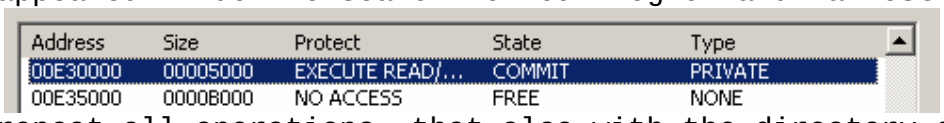
* * *



Now it is possible to [sdampit] region memory with the stolen code. We start PETools, we select our process even we indicate that we should [sdampit] region memory.



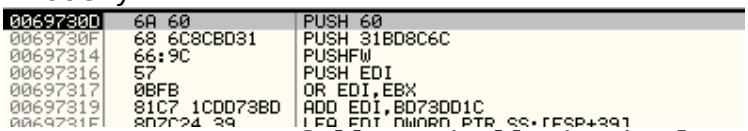
In the appeared window we search for our region and harvest Dump.



Now we repeat all operations, that also with the directory of resources. Is discovered dump and [sdamplenyj] region into WinHex'[e]. in the dump we pass to **00297000** (beginning of new section). We pass to the supplementary sheet with the [sdamplenym]

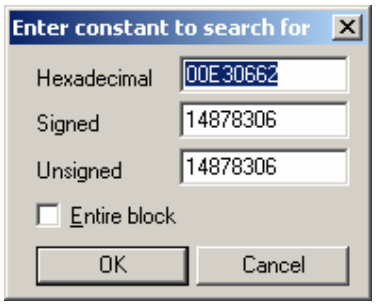
region Of ctrl+A, Ctrl+Shift+C. We pass to the supplementary sheet with the dump Of ctrl+B. We preserve dump. It is finished with the scrambler and VM (I I hope). Is now reparable **OEP**. **OEP** will be equal to address **OEP** in the chosen memory minus the old base of the stifling code plus the new base of the stifling code. **OEP = of 00E3030D-00E30000+00297000=0029730D**. Is discovered our dump with the aid of **PETools** and is discovered Image Of optional Of header Of editor (we harvest to the button Of optional Of header). In the field Of entry Of point we change **00001000** by **0029730D** and preserve changes. In principle the program is unpacked. But always there are its “but!”.

Let us open our dump into OllyDbg and will wait for, until the analysis of the code ends. We see that the now stolen code is located through another address. It turns out that we did everything correctly.

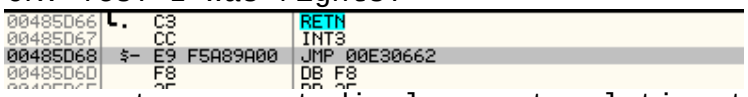


We start program F9. Program fell, and OllyDbg in [storoke] of state it writes:

To [ugu]. Error, with the starting of the code to **00E30662**. Interesting address. Indeed the same the address of the memory, where there was the stifling code! Similarly program where that still causes it. By [perezapustim] program let us pass into the section of the code (Ctrl+G, 00401000, Enter). Let us try to look the address **of 00E30662** as the constant. For this we harvest by the right button of [myshi]->Search **of for->Constant**. In the appeared window we write:



And [zhmjom] OK. Yes! I was rights!



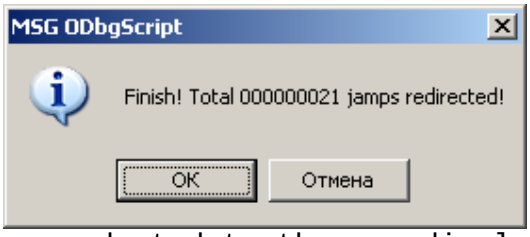
It is now necessary to recount displacement relative to new section and to change the address of [dzhampa]. New address **= of 00E30662-00E30000+00297000 = 00297662**. Now it is possible to change **jmp of 00E30662** for **jmp 00297662**, but you do not hurry. I will say immediately that such [dzhampov] in this program THERE ARE VERY many. Therefore I wrote script on the restoration of such [dzhampov]. He is called **JMP_VM_REDIRECT.osc**. This script must be disposed to your program.
Variables:

RegionVM_Start of – the beginning of the chosen region memory, in which there was the stolen code. In me is equal **00E30000**.

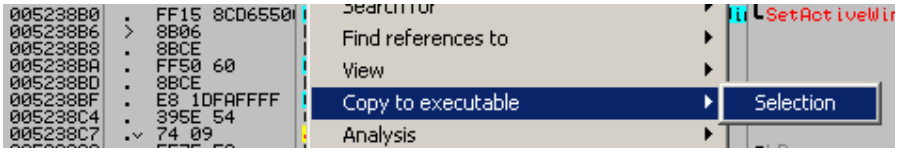
RegionVM_End of – respectively the end of this region.
In me is equal **00E30000+00005000 = of 00E35000**

RegionMain_Start of – the beginning of new section with the [sdamlenym] stolen code plus **Of imageBase**.
I have – **of 00297000+00400000=00697000**.

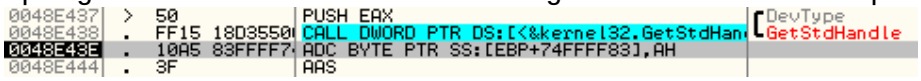
We enter values in the script and start it in our dump. We see in second:



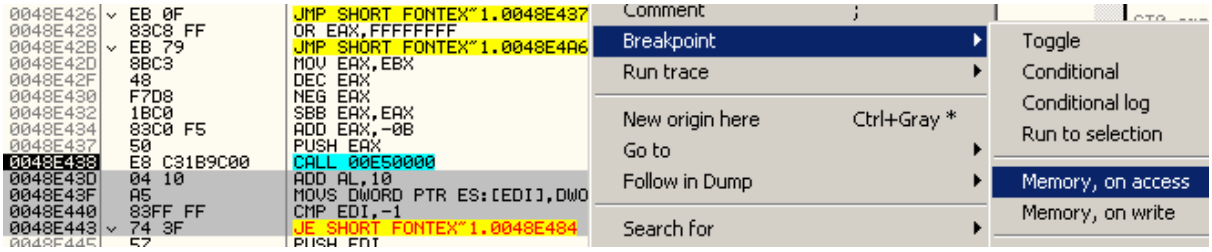
21h = 33 [dzhampa] are adapted to the new displacement. Now we pass into the section of the code, we place indicator on the very first command and it is moved by mouse for the sulky the window of the code into the bottom itself. We press Shift and will call by mouse on the last command of this window. We see that entire section of the code was isolated. Let us call by the right button of mouse and we select:



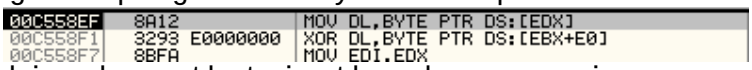
We shut the appeared window, OllyDbg it asks, we do want to preserve changes. We harvest yes and OllyDbg proposes to introduce the name of file. It is possible to preserve into the same dump, but I always preserve into the new file, so more easily to make a recoil, in the case of error. Is discovered into OllyDbg the preserved dump even we await, until the analysis of the code ends. We start program even we see that it again fell in this place:



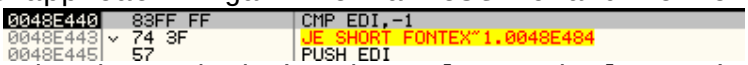
This still that after command is such after the call API of function? For the first time similar I see. Let us roll up for the moment OllyDbg and will neglect one additional copy Of ollyDbg, in which let us open the protected program. Let us reach OEP and will pass to the address **of 0048E438** and will install location counter (eip) to this address (Ctrl+ *). Now let us isolate several bytes after the call API of function establish [brjak] to the access to the memory.



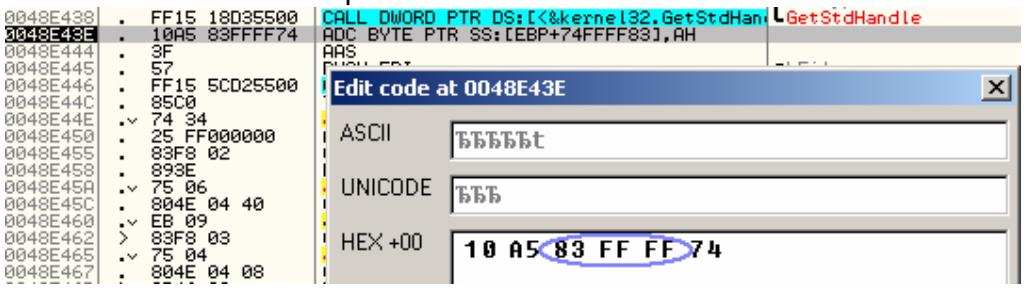
Now let us neglect program. They interrupted here:



We see that this where that in the chosen region memory and to us this does not approach. Again we harvest F9 and we here:

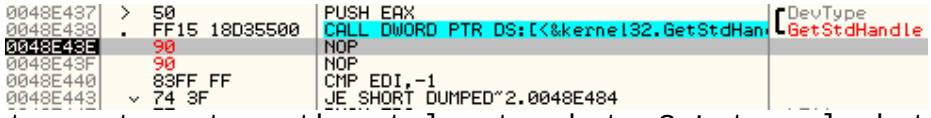


But this already the code being been located almost immediately after the call API of function, you will memorize its [opkody] (**83FFFF**). Open diagnostic routine with the dump. Now let us place indicator on the address **of 0048E43E** and will press Ctrl+E. Hexadecimal editor was opened. We see:



We see that the bytes, on which we dwelled, be present, but here to them is even what that bytes and here their ASProtect stole and carried out somewhere earlier, after fulfillment API of function, also, to the return to the section of the code. Now at their place the debris bytes, which do not give to disassembler it is normal to recognize commands. Give we change them for 90 90 ([opkody] **of nop**).

We harvest Enter even we see that now the instructions recognized normally.



But how to us to return the stolen two bytes? Let us look to the logic of the work of program.

First is caused the function Of getStdHandle, which returns to eax of [khendl]. Then they go ([jot]) unknowns ([aja]) to us the command (a) then of edi it is compared with -1. By the way the number - 1 (0xFFFFFFFF) is the constant (INVALID_HANDLE_VALUE), which returns with the functions, which work with [khendlami] as the result of the unsuccessful attempt at the discovery [khenda]. It is interesting that after the call [API] of [khendl] it will be into eax, and to the validity is checked edi. Not about which he does not speak? So that finally it will be convinced of my theory

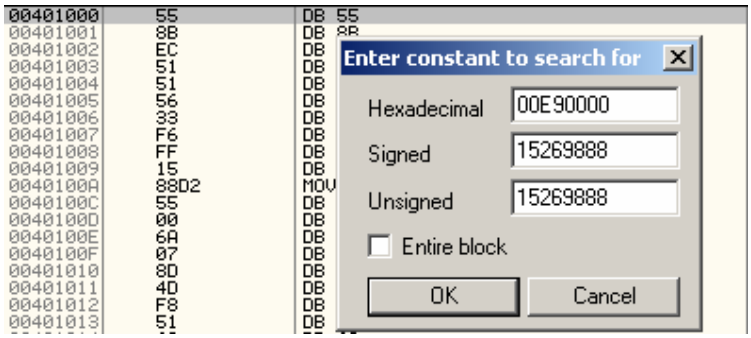
let us look that it lies at the register of eax and edi in our dump, and in the protected program.

In the dump: In that protected:

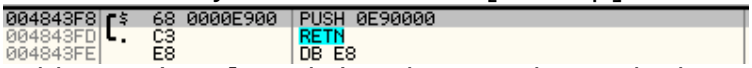
As we see in the protected program the register of edi so it contains the result of the work API of function as eax. I know only one command, capable of copying value from eax into edi, and which occupies the size of – of 2 bytes. This is **mov of edi, eax**. We enter it instead of two **nop'[ov]**, for this we harvest gap, we introduce necessary command and we harvest Enter.

We preserve and is discovered the preserved dump. We start program even we see that it again fell. In the window of the code nothing it is reflected, but it is here in the line of the state:

It is familiar, not so whether? Error, with a attempt at the fulfillment of the code to **00E90000**, only now this is already accurate not the address of our restored code. Means ASProtect where that still it stole the piece of the code and for us one must find it. For the beginning let us find the call of this piece. We search for just as past time as the constant in the section of the code.



And we again find ☺. only now this not [dzhamp]:



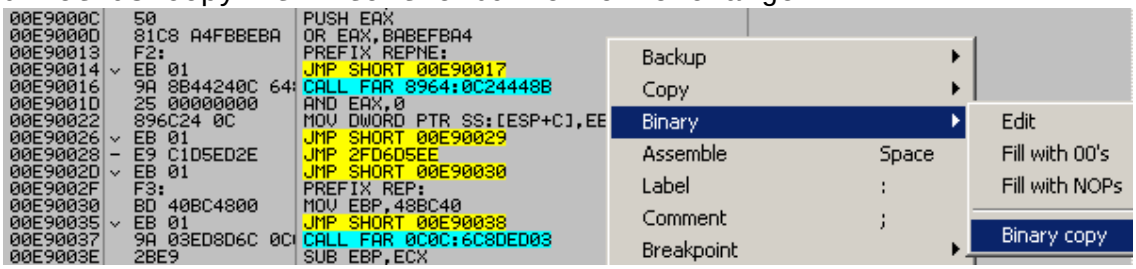
The required address is placed in the stack, and then the command **of retn** extracts it it from there and accomplishes to it a passage. Let us look, what do we have here in the protected program. The same. Therefore in the protected program we place location counter (eip) on this address and harvest two times F8. Now we in the chosen region memory, to **00E90000**. We see that this is the entirely small piece of the code and we can decrease it still more, if we clean from the rubbish and even let us be able to return it in the place.

```
00E90000   68 FFFFFFFF      PUSH -1
00E90005   50               PUSH EAX
00E90006   6A 00            PUSH 0
00E90008   58               POP EAX
00E90009   64:8B00          MOV EAX,DWORD PTR FS:[EAX]
00E9000C   50               PUSH EAX
00E9000D   81C8 A4FBBEBA    OR EAX,BABEFBA4
00E90013   F2:              PREFIX REPNE:
00E90014 v EB 01            JMP SHORT 00E90017
00E90016   9A 8B44240C 64:  CALL FAR 8964:0C24448B
00E9001D   25 00000000      AND EAX,0
00E90022   896C24 0C        MOV DWORD PTR SS:[ESP+C],EBP
00E90026 v EB 01            JMP SHORT 00E90029
00E90028 - E9 C1D5ED2E      JMP 2FD6D5EE
00E9002D v EB 01            JMP SHORT 00E90030
00E9002F   F3:              PREFIX REP:
00E90030   BD 40BC4800      MOV EBP,48BC40
00E90035 v EB 01            JMP SHORT 00E90038
00E90037   9A 03ED8D6C 0C(  CALL FAR 0C0C:6C8DED03
00E9003E   2BE9             SUB EBP,ECX
00E90040   F2:              PREFIX REPNE:
00E90041 v EB 01            JMP SHORT 00E90044
00E90043   F0:50            LOCK PUSH EAX
00E90045   C3               RETN
00E90046   0000             ADD BYTE PTR DS:[EAX],AL
```

But that not to fan even without that large article we will not this make. Let us look, where it is possible to insert this code. The first, that occurs of – this the section of the code (indeed there it and it was earlier). In the diagnostic routine with the dump in the window of the code let us press the key **For end**. Now we at the end the section of the code, see continuous zero. Let us twist window upward, until we meet the first nontrivial bytes.

```
0055C78B   .  85C0          TEST EAX,EAX
0055C78D   .v 7D 06         JGE SHORT DUMPED~3.0055C795
0055C78F   .  50            PUSH EAX
0055C790   .  E8 DBA3FBFF   CALL DUMPED~3.00516B70
0055C795   >  C3            RETN
0055C796      00            DB 00
0055C797      00            DB 00
0055C798      00            DB 00
0055C799      00            DB 00
```

It means insert the code we will be to **0055C796**.
In the window with the protected program let us isolate the code and let us copy it into the buffer of exchange.



Let us pass into the diagnostic routine with the dump let us press Ctrl+E, Shift+Insert, Enter. Now this code will be placed to **0055C796**. It is naturally necessary to change reference to this code. We pass to the address **of 004843F8** and instead of **00E90000** let us enter **0055C796**:

```
004843F8      68 96C75500   PUSH DUMPED~3.0055C796
004843FD L.   C3            RETN
```

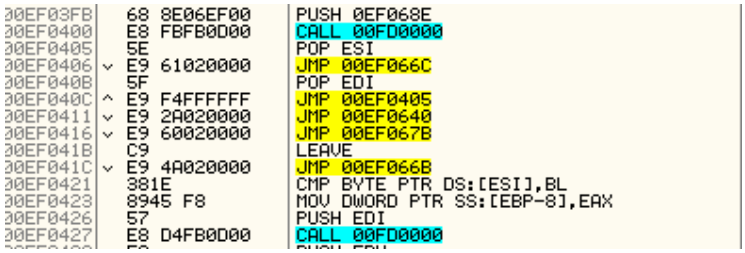
We again separate entire section of the code and we preserve changes. Is discovered dump. F9. In the line of the state:

```
Access violation when executing [00EF0000] - use Shift+F7/F8/F9 to pass exception to program
```

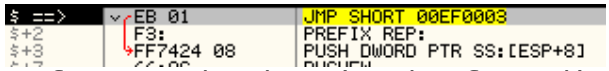
Search for the constant **of 00EF0000**.

```
0044CEEB   >-E9 1031AA00    JMP 00EF0000
0044CEF0      BA            DB BA
```

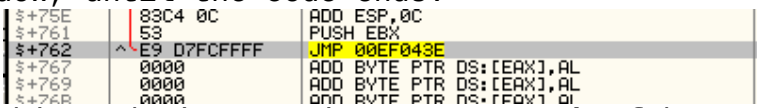
In the protected program we pass to the address **of 0044CEEB** and it is passed into the chosen memory. This time the code of completely solid size, and, after twisting window downward, we see:



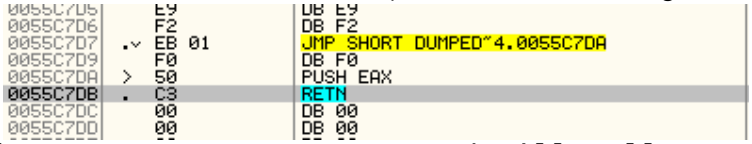
Yes ☹ this again VM. To this code it is first necessary to apply script, on [rebildu] VM, and already then to copy into its dump. But, as we remember, to script it is necessary to indicate the new base of the restorable code. [Petomu] first let us be determined, where it will be placed. Let us arise to the beginning of the code and let us call two times by mouse at the intersection of the first column and current line.



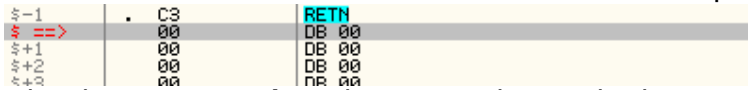
Now the addresses of commands show in the form displacement. Let us twist window, until the code ends.



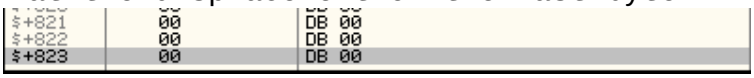
Leaves that this code it occupies **0x00000766h** of bytes. Let us pass into the diagnostic routine with the dump. Do remember where we they did put the last stolen code? We pass to the address **of 0055C796** and let us twist downward, until zero begin:



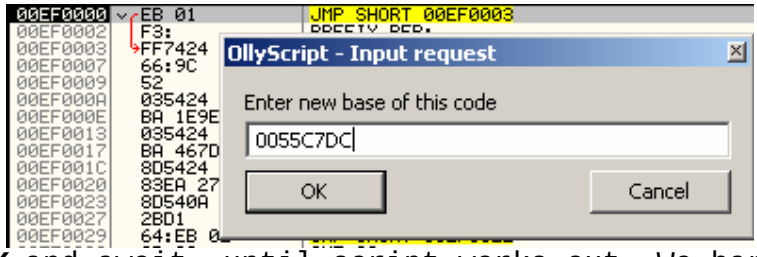
Now let us place cursor on **0055C7DC** and will call two times by mouse at the intersection of the first column and the display line. We see that also here the addresses became displacement.



Let us press the key **For end** and we at the end the section of the code. We look at the displacement of the last byte:

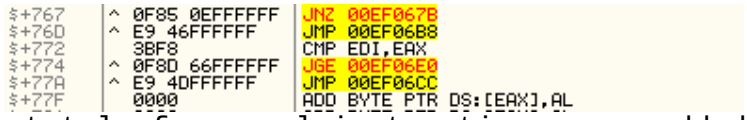


[Khekh]. Place is sufficient, still and it remains. It means, this code we will place to **0055C7DC**. In the diagnostic routine with the protected program we start script for [rebilda] VM, and we write to a question about the new base of the code:

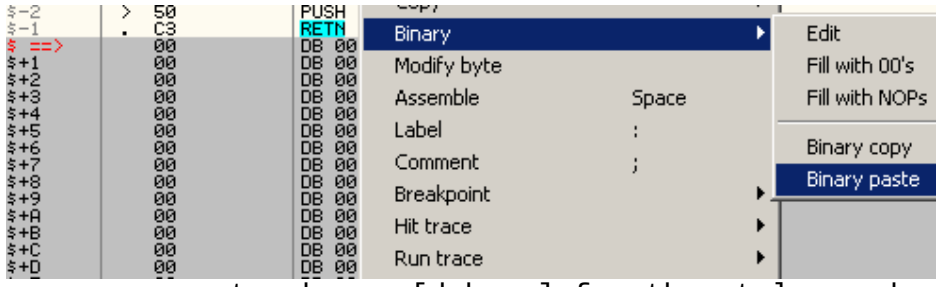


We harvest **OK** and await, until script works out. We harvest * and we again on the beginning of the stifling code, as you already know script it throws down the restored instructions at the end of

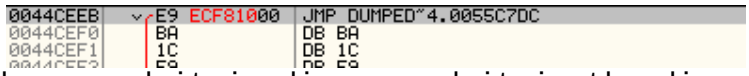
the code. Let us look, they did not exceed the permissible size of the code.



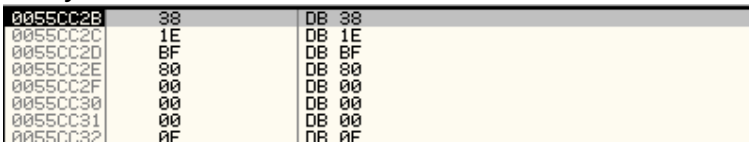
We see that a total of several instructions were added and the code still gets in into the section of the code. We separate entire code and we copy into the buffer of exchange. We pass into the diagnostic routine with the dump, we separate from **0055C7DC** the bytes of the long**of 77F**bytes and we harvest by the right button of the mouse:



It is now necessary to change [dzhamp] for the stolen code. We pass to **0044CEEB** and we correct **00EF0000** to **0055C7DC**

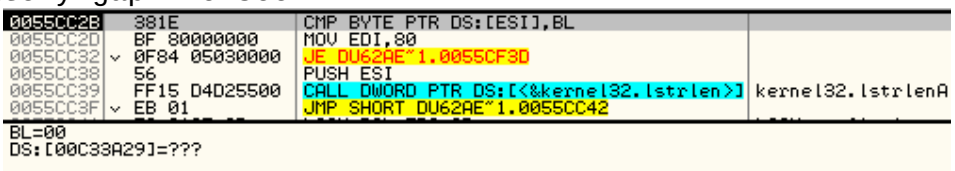


We preserve dump, and it is discovered it in the diagnostic routine. F9. They fell here:

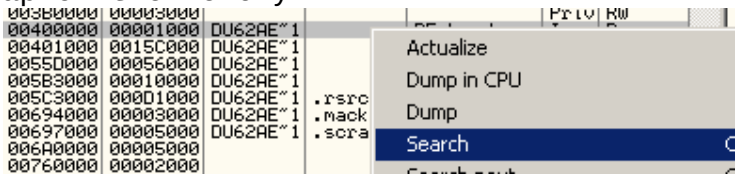


Line of the state

What after…? Judging by the line of state in this place it occurs a attempt at reading to **00C33A29**. We harvest Ctrl+A and immediately gap. We see:

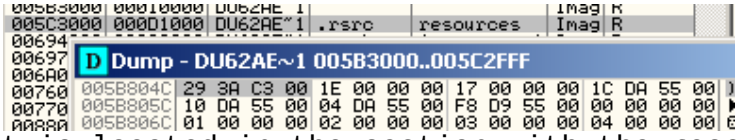


It turns out that program compares from [nuljom] something, which is stale to **00C33A29**, when there was ASProtect. And you [zamete], that if this something is not equal to zero, then program calculates the length of line, which is been located to **00C33A29**. Nothing it does resemble? Greatly it reminds me of testing the registration code. Give let us try to substitute the address **of 00C33A29** to the address, which it will indicate what or line. Let us open the map of the memory:

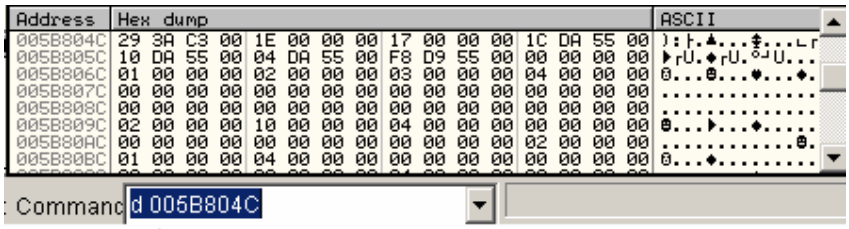


In the appeared window we write:

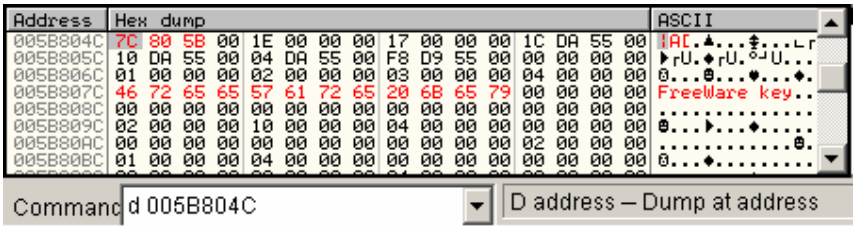
And we harvest Enter. Was opened the window of dump even it shows that the address is found.



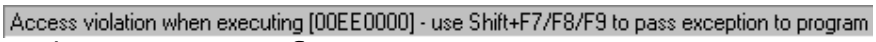
We see that it is located in the section with the resources to **005B804C**. In the command line we write **with d of 005B804C** and we see.



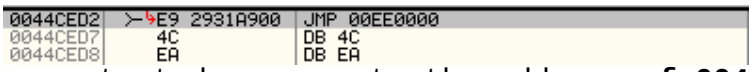
From the address **of 005B807C** go zero. Let us there and enter our key ☺.A to **005B804C** let us correct indicator from **00C33A29** to **005B807C**.



Now we separate all this matter and we preserve changes. Is discovered the preserved dump. F9. They fell. In the line of the state:

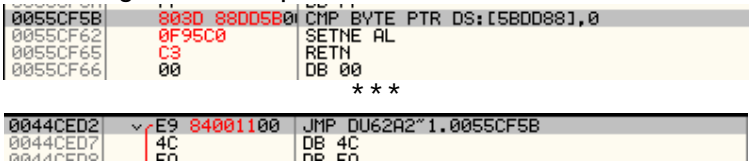


Search for the constant **of 00EE0000**:

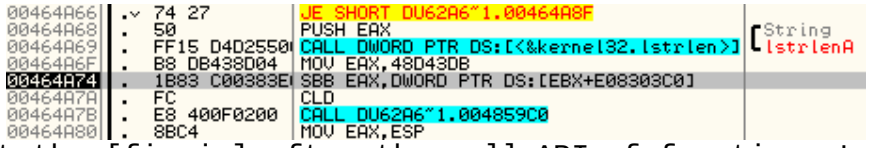


We pass in the protected program to the address **of 0044CED2** and it is passed to **00EE0000**:

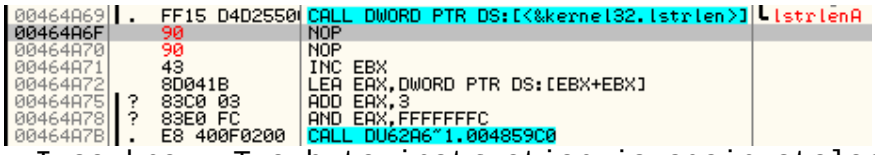
We see that this entirely [kazjavka]. We insert into the section of the code and we guide adapter.



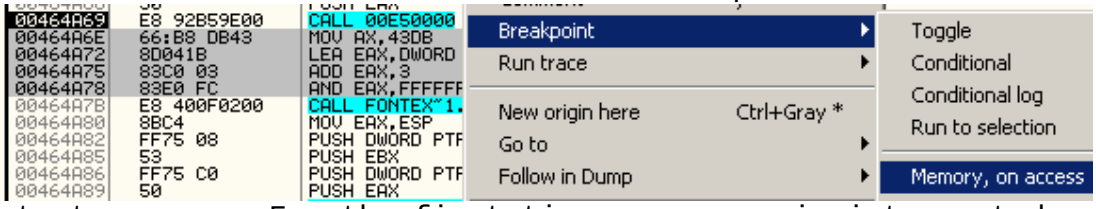
We preserve changes. Is discovered the preserved dump. F9. Program is loaded…, is loaded…, is loaded. It appeared in [taskbare] and bang… it again fell here:



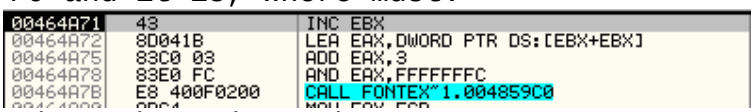
Again what the [fignja] after the call API of function. Let us try to use the same measures. Instead of the first two bytes after call API let us enter 90 90.



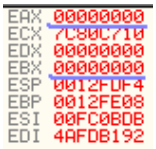
Well here, I so knew. Two-byte instruction is again stolen. But judging from the fact, that after **nop'[ov]** go inc of ebx, the most likely this **mov of ebx, eax**. Let us verify. [Perezapustim] the protected program. Let us pass to the address **of 00464A69** and will install location counter (eip) to this address. We place [brjak] on the instruction after the call API of adapter to the access:



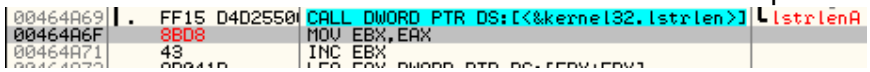
We start program. For the first time we are again interrupted not there. Again F9 and it is, where must.



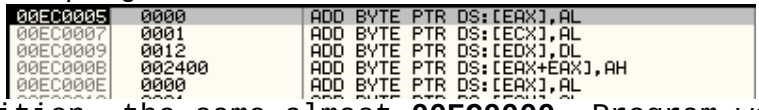
We look, what do we have into ebx:



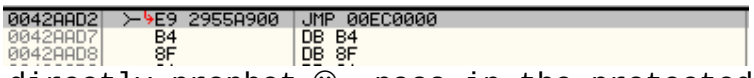
As we see – the same as into eax. We correct our dump:



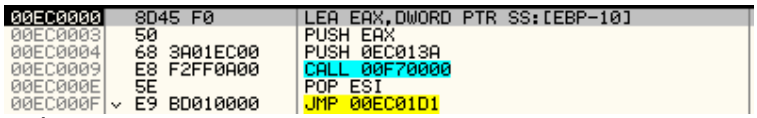
We preserve changes. Is discovered the preserved dump. F9. And… On the miracle!!! Program was neglected! We go in **Help->About Of fontExpert…** and program falls here here:



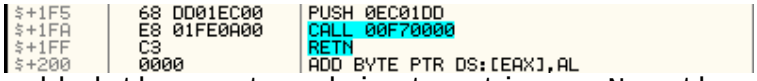
No-load condition, the same almost **00EC0000**. Program would fall as earlier, but this time to **00EC0000** randomly proved to be the code, and several instructions even were carried out. We search for **00EC0000** as the constant:



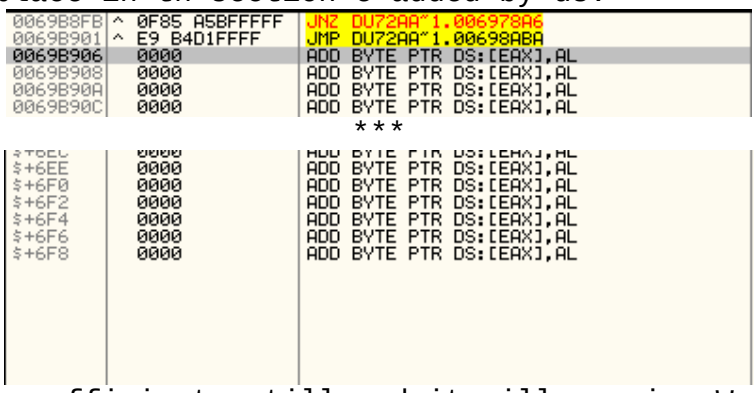
Well here, I directly prophet ☺. pass in the protected program to the address **of 0042AAD2** and it is passed into the chosen memory. We see that the code not small and is present VM.



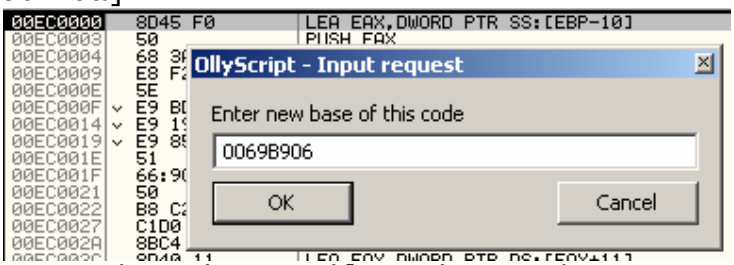
Size of the code:



Plus will be added the restored instructions. Now the code we this matter will not clearly insert into the section. Let us look, how much vacant place in th section e added by us.

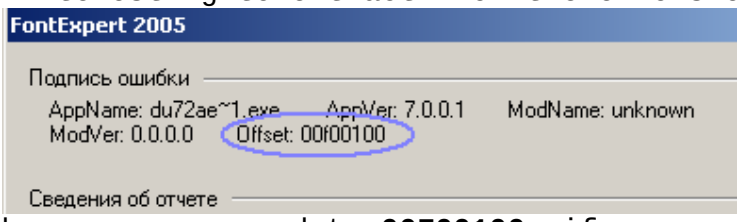


Place will be sufficient, still and it will remain. We start script for [rebilda] VM:



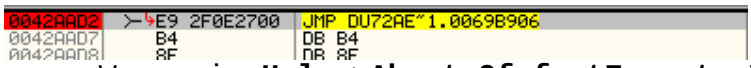
We harvest Enter and await, until script works out. We harvest * and we on the beginning of region with the stolen code. We separate the code, we copy into the buffer of exchange, troubleshooter with the dump is discovered and we put the code to **0069B906**. Now we change [dzhamp] to **0042AAD2** from **00EC0000** to **0069B906**.

We preserve changes. Is discovered the preserved dump. F9. Program was neglected! We go in **Help->About Of fontExpert…** This time generally system error was thrown out. Well nothing. Also it is possible much interesting to extract from the errors of system:

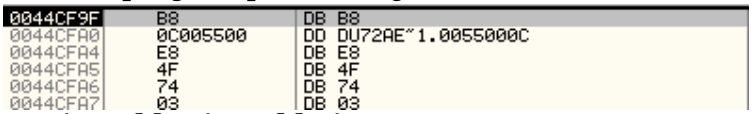


We see that the error occurred to **00F00100**, if we are relied on entire obtained by us experience, then possible to assume that program turned to **00F00000**, and by it transported so, that it it could reach [azh] the hundredth displacement. Let us verify theory. We search for **00F00000** as constant. No, unfortunately this time I was mistaken. Then we enter on – to other. To

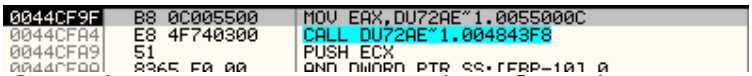
[perezagruzim] program let us place [brjak] on last restored [dzhamp].



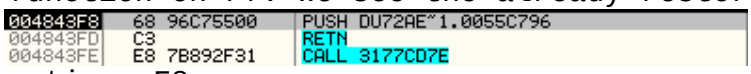
We start program. We go in **Help->About Of fontExpert…** They interrupted on our [brjake]. We begin to locate and we fall here:



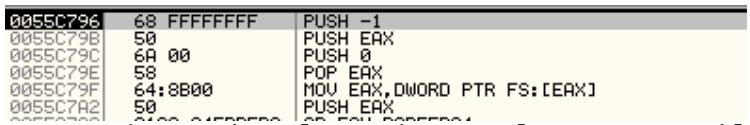
It is strange, why OllyDbg all in no way wants to recognize the restorable code. We harvest Ctrl+A and immediately gap. All recognized:



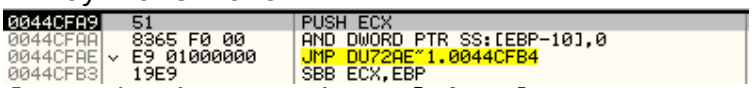
We go in the function on F7. We see the already restored adapter:



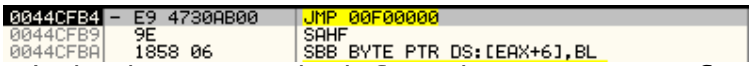
We harvest two times F8.



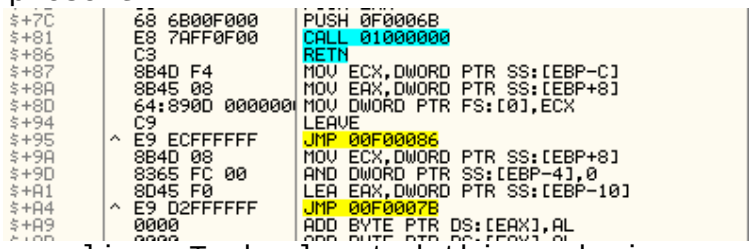
This is the restored previously code. We locate, until we leave the function. They left here:



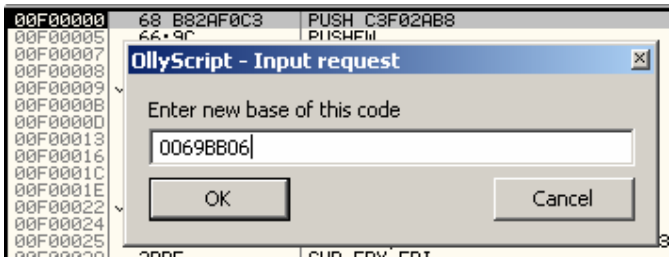
[Dotrassiruem] to the instruction **of jmp** let us carry out it. Burn here:



[Oppaaa]… But indeed we searched for the constant **of 00F00000**! Yes, I and itself was strongly astonished. It leaves, it does not be worthwhile to separately entrust to the mechanism of search into OllyDbg. Let us pass to this address in the protected program and will visit into the chosen region memory. The code of small and again is present VM.

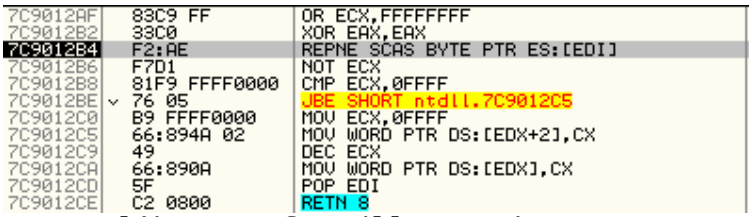


All, as it is earlier. To be located this code in me will be to **0069BB06**, it means:

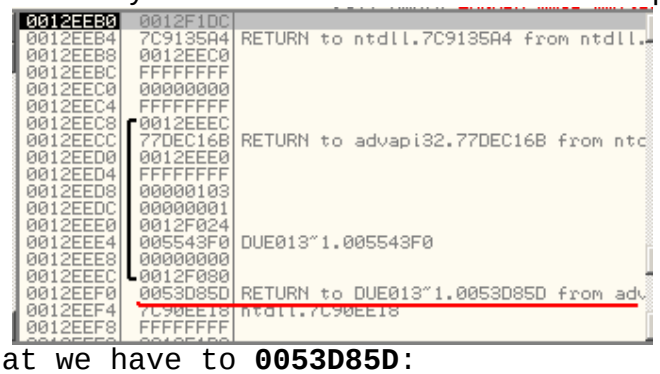


Script worked out. We copy the code in the buffer of exchange and put to **0069BB06**. We change [dzhamp] to **0044CFB4** from **00F00000** to **0069BB06**. We preserve changes. Is discovered the preserved dump. F9. Program was neglected! We go in **Help->About Of fontExpert…**

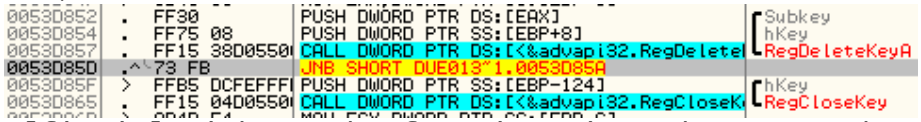
[UraAAAaa]! Program earned. First test of – closing program. I noted that with this ASProtect'[om] in essence two problems. These are the window Of about and the correct completion of the work of program. We shut program. Well, I so thought – program it fell here:



This is the system library of ntdll. But how to us to find where does fall? Let us look, into the window of stack, there must be the address of recovery from the function into the program.



Let us look, that we have to **0053D85D**:



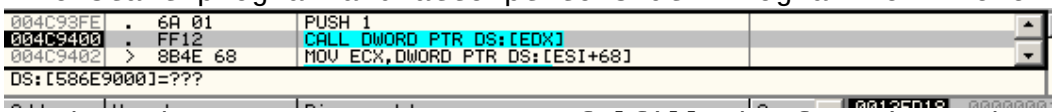
What for [fignja] with API by functions is today created? I see for the first time so that after the call API of function would be conditional jump back to the call API! This smells a little by already infinite cycle! I make the assumption that two bytes,

after the call API of function are stolen By aSProtect'[om], and instead of them is inserted this insidious [dzhamp]. Here here simple so to surmise, that after command it is stifling it will not succeed. Let us try to [pomeditirovat]. Let us open diagnostic routine with the protected program, let us reach OEP and will pass to the address **of 0053D857**, let us install to it location counter (eip) and let us place [brjak] on several bytes after call API to the access to the memory. We harvest F9. For the first time not there, let us pass, but they landed by the here second in the section of the code.

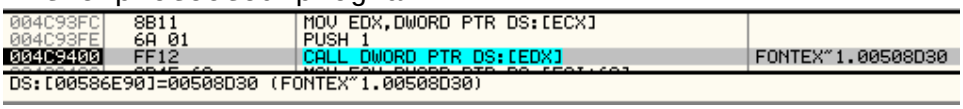
Thus, now let us open diagnostic routine with our dump and also will pass to the address **of 0053D857**. Let us carry out the call API of function (F8). But now let us compare registers in the protected program and in our dump.

Protected program	Our dump
EAX 00000006 ECX 7C91056D EDX 00030000 EBX 3D83D8BF ESP 0012FF6C EBP 0012FF64 ESI 00000006 EDI 00E3030D	EAX 00000000 ECX 0012FFB0 EDX 7C90EB94 EBX 7FFD6000 ESP 0012FFC4 EBP 0012FFF0 ESI FFFFFFFF EDI 7C910738

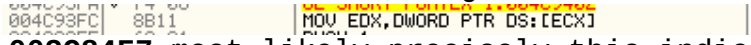
The function Of regDeleteKeyA returns result to the register of eax, and in the protected program the values of eax and esi are equal. But the in the manner that stolen command has already been carried out the, it can be assumed that stolen command it is **mov of esi, eax**. We substitute in the dump conditional [dzhamp] by **mov of esi, eax**, we preserve changes and is discovered the preserved dump. We start program and attempt to shut. Program fell here:



We see that the program attempts to fulfill the function, whose address is located into edx. But this address indicates generally beyond the limits of memory. Let us look, which occurs in this place in the protected program.

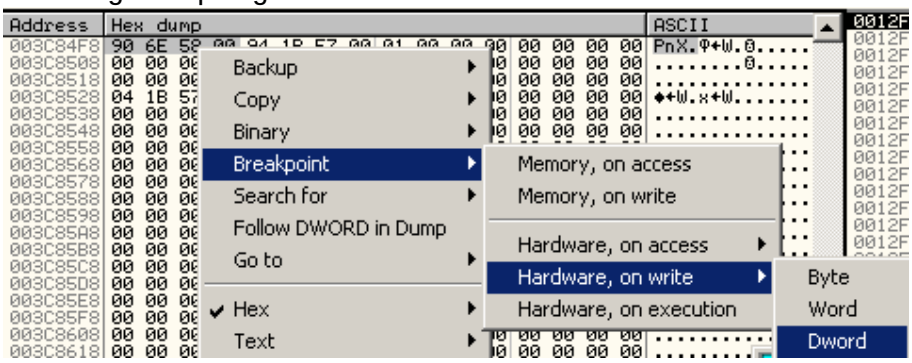


It cannot be! The number into edx is also the very, but is shifted by two discharges to the right! I searched for here this error very for long. And it was already it solved, that this is mysticism and maltsters were accepted to itself to the service of sorcerer. But I solidly understood that never he is worthwhile to despair. How I did localize this [bag]? If you [potrassiruete] a little program, then you will understand because of what Of [ares] it is shifted by several discharges. In reality it is not shifted. The address falls as follows in the register of edx:

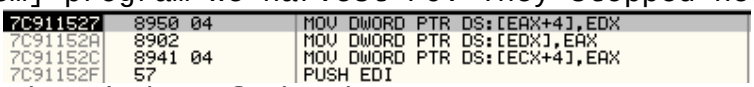


Ecx is equal **003C84F7** most likely precisely this indicator to what that by means it decreases by one, indicating no longer the beginning of address. I decided to search for the place, where to

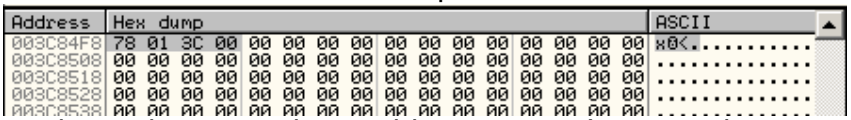
003C84F8 will be brought in th address e necessary for the work of program. In the command line we write **with d of 003C84F8**, harvest Enter and we see th address e interesting us. Let us place on it hardware of [brjak] to the record, with the size of dword (4 bytes). To place is necessary precisely hardware of [brjak], since after reloading of program it will still act.



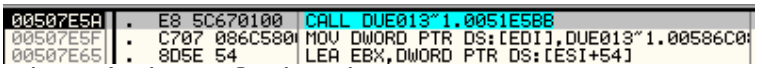
[Perezapuskaem] program we harvest F9. They stopped here:



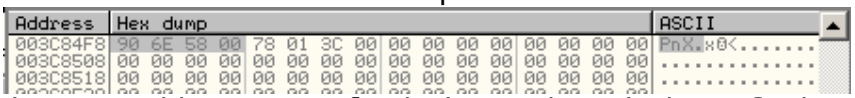
We look into the window of the dump:



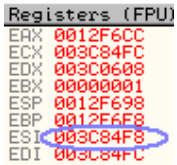
No, this is what that another address. Again F9, they stopped here:



We look into the window of the dump:

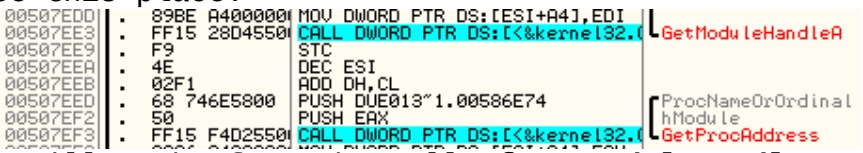


Yes, this is our address. We look into the window of the registers:



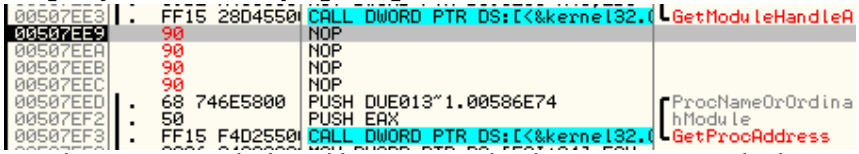
We see that into esi lies the normal unspoiled indicator. Let us remove hardware of [brjak] from the indicator so that it would not interfere.

But now we will locate the code on F8 and follow the register of esi. We see this place:



This that still such after the call **Of getModuleHandleA**? Here, that tells me my scant knowledge of [asma]: STC advances flag [S]F in one, then DEC ESI decreases the indicator by th address by one e interesting us!!! But are further generally no one not necessary

operations. Is similar this the again stolen instructions. As we earlier substitute two bytes after the call API of function by 9090. To [khe]! But indeed the following instructions did not recognize! It leaves, that ASProtect of [sper] of here 4 bytes. Now let us look to the logic of the work of program. The function **Of getModuleHandleA** calculates [khendel] of necessary module, unknown 4 bytes further go, into the stack is placed the indicator to the name of function, then into the stack is placed [khendel] obtained from **GetModuleHandleA**. However, that they could make these 4 bytes, indeed in principle they here were not necessary! Give [zabjom] by their **nop'[ami]** and the case is closed ☺.



We preserve changes and is discovered the preserved dump. We start program and attempt to shut.

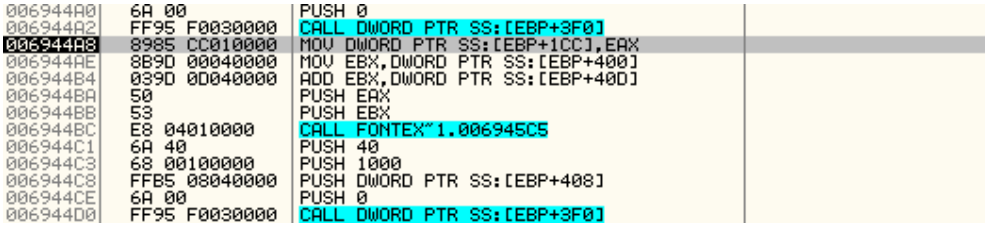
All! We completely unpacked program! But there is one [bag]. I revealed him after [inlajn] of [patcha]. About it I will describe in chapter about [inlajn] [patche].

As you understood, in this program I did not in vain give so much attention to the stolen code after the calls API of functions. However, what this? But this nothing else but second VM Of aSProtect'[a]. I call its VM API. You do not relate to it disdainfully, since it emulates many commands. It can emulate even call of function after call API!
You do remember the place, where we did replace 4 bytes **with nop'[ami]**? It is so here there stolen:
Cmp of eax, edi
je 00507EFF
It is strong? In this VM is much more powerful the mechanism of checking the integrity of the code. I began to write script on its restoration, but thus far it far to [reliza]. Most frequently ASProtect do not use this VM with the protection; therefore you can never not meet with it. So in the program can be encountered checkings to [raspakovannost] with the aid of the special macros. Therefore I consider that ASProtect it is necessary to [patchit], but not to unpack. So it is much safer. I never made [inlajn] of [patch] (in me it was another technology, but not [loader]), but in this version I it decided nevertheless to try.

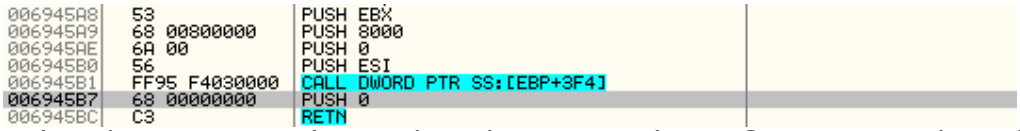
We write [inlajn] of [patch].

After studying, a article Of alex'[a] about [inlajn] of [patch] Of aSProtect 2.0, I decided to make a little differently. Theory is the same: ASProtect has multilayer structure, i.e., with the starting it it unpacks its body in parts into chosen for this regions memory, which hampers its [propatchivanie]. In order to [propatchit] program, it is necessary to follow ASProtect'[om] into these regions of memory before the complete unpacking of program, to and then [propatchit] and program. But complexity

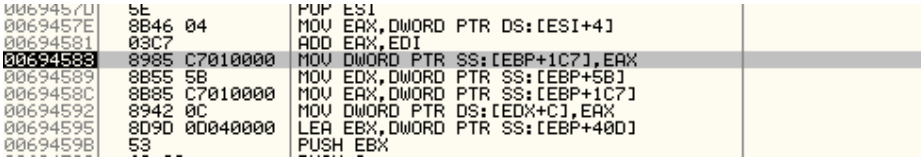
consists also in the fact that the passages into the following region of memory are encoded and are deciphered dynamically, in proportion to the fulfillment of the code.
Let us begin from the fact that we will not search for place for the arrangement of our [patcha]. Alex wrote that ASProtect cleans the code On the Border of sections, moreover several times. Let us consider this and will place the code between the end PE of title and the beginning of the first section. Let us attempt to trace, where occurs the first passage Of aSProtect'[a] into the chosen memory. We place [brjak] on VirtualAlloc of [zhmjom] F9, we are interrupted and we pass to the address of recovery.



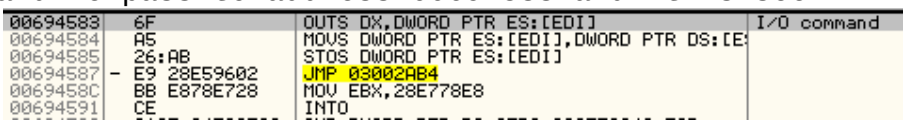
Let us twist window we a little downward and see:



Push 0 is the passage into the chosen region of memory. Thus far there 0, but if we a little [potrassirovat], then it is possible to find the place, where the address of passage is written instead of zero. Here is it:

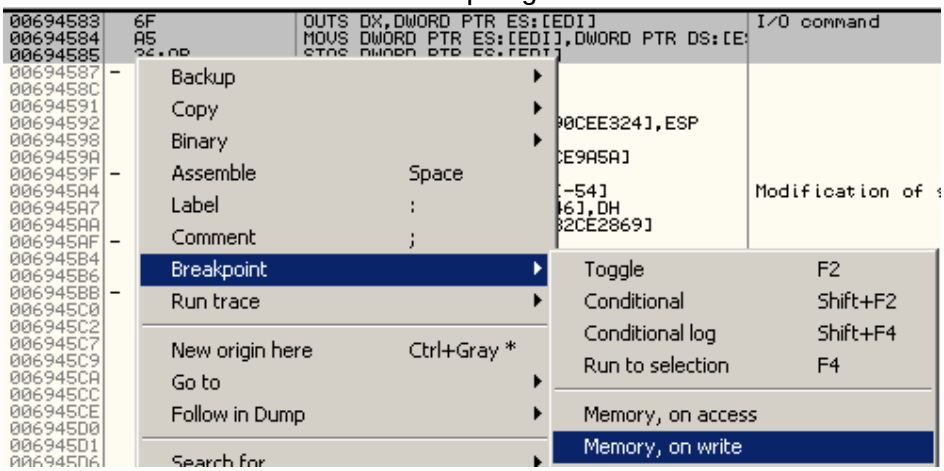


It is here necessary to enter [dzhamp] to our [patch]. But in the manner that this [dzhamp] will rub over the original code, then in [patche] it is necessary to first carry out the rubbed over code, to and then already [patchit] other addresses. It is reloaded program and we pass to address **00694583** and it is seen:



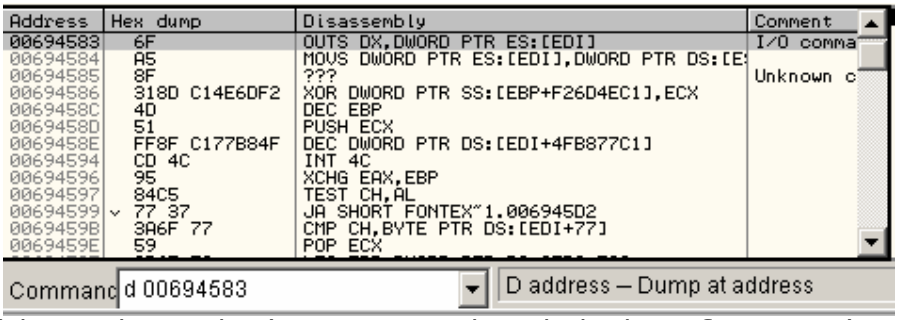
However, where be divided the instructions, which we did see here past time? But these are they and there is, only encoded are deciphered by gradually several decoders.
Now there are two versions:
To enter so as Alex. To write [dekriptor], which will decipher this code and if it deciphers correctly, then write [kriptor] and encode by them our code, and then replace with them original. For the application of this method it is necessary to find all decoders of this code, to clean of the rubbish and to try to write its decoder.

To find the code, which deciphers this place, to [perezagruzit] program. To pass to the address of decoder and if there rubbish, then find the decoder of this decoder, otherwise enter passage to our [patch]. And so until we find very first decoder.
I selected the second version. We place [brjak] on address **00694583** to the record and start program.

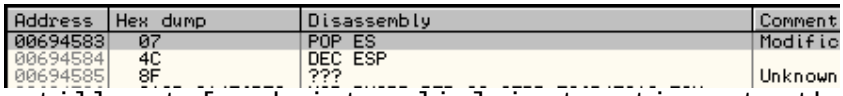


We are interrupted in this place:

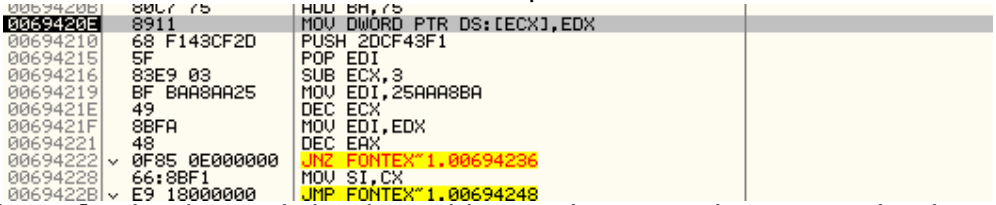
We harvest F8 and we look that with address **00694583**:



As is evident the code is not yet decoded therefore again we harvest F9. And again we are interrupted on the same address. Again we harvest F8 and we look that with address **00694583**:



Well, then still not [raskriptovalis] instructions to the end, again we harvest F9. We are interrupted here:



F8 and we look that with the address interesting. Not it is [raskriptovan]? Again F9, F8. And so until instructions on address

00694583 are restored. After the fulfillment of [instuktsii] of mov in this place:

```
00694368   81C3 19AB5007   ADD EBX,750AB19
0069436E   8918            MOV DWORD PTR DS:[EAX],EBX
00694370   BF 626C254B     MOV EDI,4B256C62
00694375   81E8 B07E6C03   SUB EAX,36C7EB0
0069437B   66:BA DCD5      MOV DX,0D5DC
0069437F   81C0 AC7E6C03   ADD EAX,36C7EAC
00694385 v E9 10000000     JMP FONTEX~1.0069439A
0069438A   C8 618647       ENTER 8661,47
0069438E ^ 74 9D           JE SHORT FONTEX~1.0069432D
00694390   12E3            ADC AH,BL
00694392 ^ E0 99           LOOPDNE SHORT FONTEX~1.0069432D
```

We see that th instructions on address **00694583** e interesting us are completely [raskriptovany]:

```
Address   Hex dump          Disassembly                          Comment
00694583   8985 C7010000    MOV DWORD PTR SS:[EBP+1C7],EAX
00694589   8B55 5B          MOV EDX,DWORD PTR SS:[EBP+5B]
0069458C   8B85 C7010000    MOV EAX,DWORD PTR SS:[EBP+1C7]
00694592   8942 0C          MOV DWORD PTR DS:[EDX+C],EAX
00694595   8D9D 0D040000    LEA EBX,DWORD PTR SS:[EBP+40D]
0069459B   53               PUSH EBX
0069459C   6A 00            PUSH 0
0069459E   6A 00            PUSH 0
006945A0   6A 01            PUSH 1
006945A2   57               PUSH EDI
006945A3   8B5E 08          MOV EBX,DWORD PTR DS:[ESI+8]
006945A6   03DF             ADD EBX,EDI
006945A8   53               PUSH EBX
Command  d 00694583                  D address — Dump at address
```

It leaves, that decoder with address **0069436E** latter and after it it is possible to [patchit] the code with address **00694583**. By [perezapustim] program let us pass to the address of the last decoder:

```
0069436E   32C5            XOR AL,CH
00694370 v 76 1C           JBE SHORT FONTEX~1.0069438E
00694372   D6              SALC
00694373   D6              SALC
00694374   C2 7B52          RETN 527B
00694377   AC              LODS BYTE PTR DS:[ESI]
```

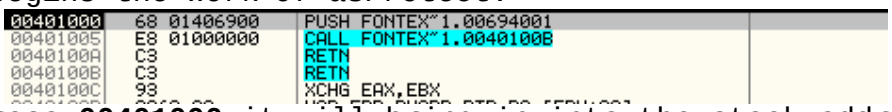
We see that the instructions of decoder are also [poshifrovany] and are deciphered by another decoder. On the whole we repeat all previous actions, until decoder deciphers. Then we pass to the address of the decoder, which deciphers this decoder even we look, that there. If bytes are there [poshifrovany], we again repeat all actions until we reach the decoder, which not is [pokriptovan] in the packed program. In me this chain of the decoders (in this sequence they they decipher each other) came out:
69415A - > 69420E - > 6942D 3-> 69436E - > 694583.

It is now necessary to study the first decoder and to find in it such place, with passage of which the following decoder will be already [raskriptovan]. This is how the first decoder appears:

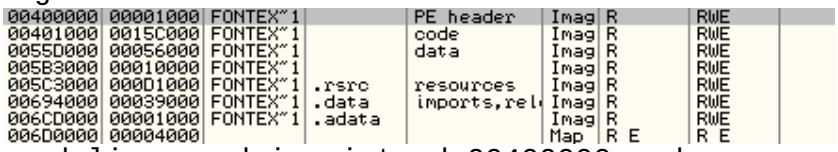
```
0069415A   8F0413          POP DWORD PTR DS:[EBX+EDX]
0069415D   66:B8 1B4B      MOV AX,4B1B
00694161   66:81C9 6414    OR CX,1464
00694166   81EA 829BAA1A   SUB EDX,1AAA9B82
0069416C   80C1 FC         ADD CL,0FC
0069416F   81C2 7E9BAA1A   ADD EDX,1AAA9B7E
00694175   81C8 A6E8620C   OR EAX,0C62E8A6
0069417B   81FA 9CF8FFFF   CMP EDX,-764
00694181 v 0F85 19000000   JNZ FONTEX~1.006941A0
00694187   81E9 00D00B2C   SUB ECX,2C0BD000
0069418D v E9 1F000000     JMP FONTEX~1.006941B1
00694192 ^ 7E DF           JLE SHORT FONTEX~1.00694173
00694194   2C F5           SUB AL,0F5
```

A little [potrassirovav] it it is possible to understand, that this is cycle, and that after working out it will pass to **00694187**. It means with this address necessary to place **jmp** on our [patch]. Since this **jmp** will rub over the command **of sub of ecx, 2c0BD000** i.e. will have to carry out in our [patche], then to [propatchit] the following decoder and to pass to address **0069418D** for continuing the normal operation of program. I decided not to [patchit] this code statically (i.e. immediately in the packed

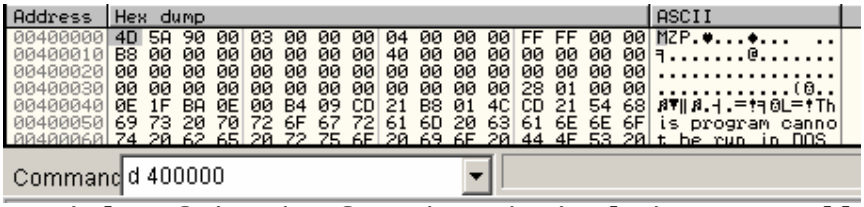
file), but to [propatchit] only with the starting of program. Will look as begins the work Of aSProtect:



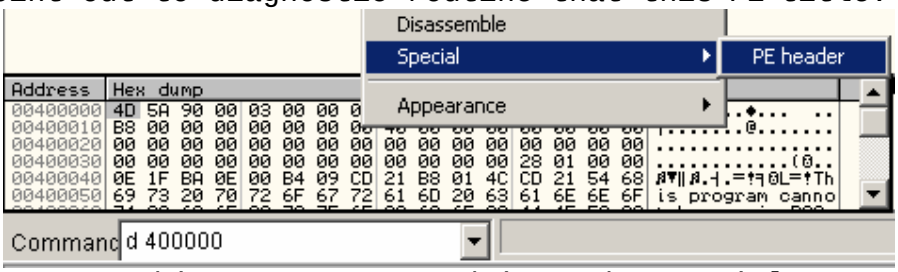
With address **00401000** it will bring in into the stack address **00694001**. Through this address is located the body of [raspakovshchika]. If we will replace this address by the address of our [patcha], then ASProtect with the starting immediately will pass to our [patch]! Now let us calculate the address of our [patcha]. As I already spoke write him will be between the end PE of title and beginning of the first section. Let us open the map of memory even we see that PE the title begins with **00400000**, and the beginning of the first section with **00401000**:



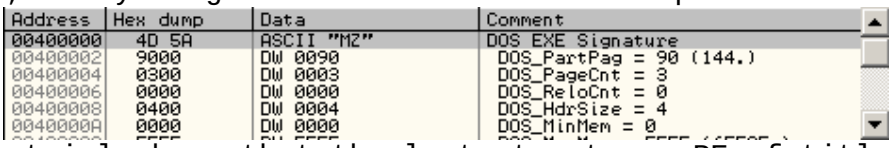
In the command line we drive into d 00400000 and we see:



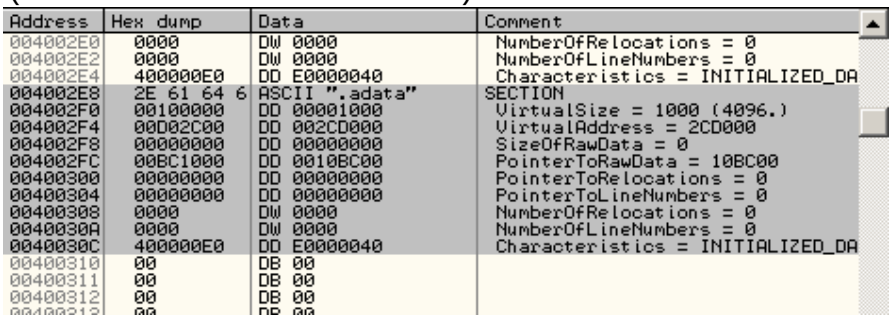
This PE the title of in the form hexadecimal dump. In OllyDbg it is possible to look PE title in the more convenient idea. For this let us point out to diagnostic routine that this PE title.



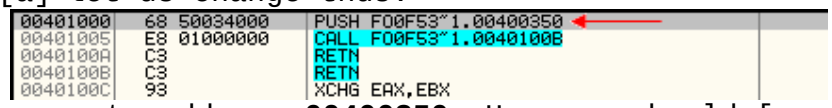
Well here, everything was converted into the special structures:



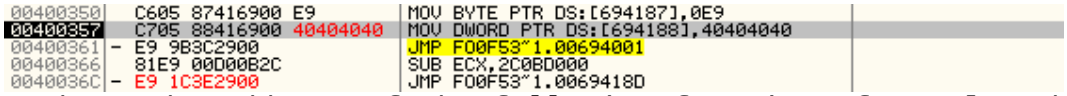
As you certainly know that the last structures PE of title are the descriptions of sections. We search for the descriptions of the sections (we twist window downward):



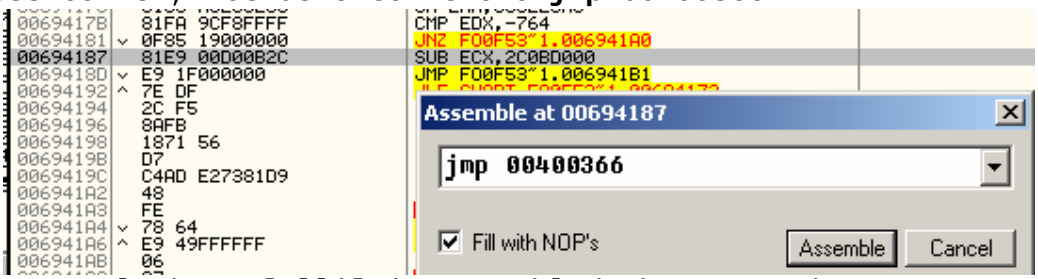
We see that after the address **of 0040030[S]** the description of sections conclude and begin zero. I decided to write [patch] from address **00400350**, suddenly he will be required to write any data (name of user for example ☺). Let us approach. The beginning Of aSProtect'[a] let us change thus:



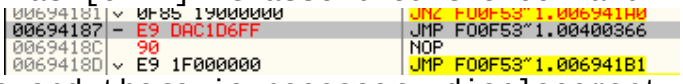
Now let us pass to address **00400350**. Here we should [propatchit] the first decoder. Let us write for this
MOV BYTE PTR DS:[694187], 0E9.
This command will prescribe with the address **00694187** [opkod] of the instruction **of jmp**. It is further necessary to enter after [dzhampa] the address, to which ASProtect will pass after the decoding of the second decoder. Thus far, that we do not have function, whose address must be entered; therefore let us enter for the moment any value **MOV DWORD PTR DS:[694188], 40404040** after this it is necessary to pass to that address, where ASProtect would pass, if we did not change its beginning. We recall what address it was after the command of push we at first and write **WITH JMP 00694001**. It is now necessary to write the function, to which will pass ASProtect after the decoding of the second decoder. You do remember that the command **of sub of ecx, 2c0BD000** will be rubbed over by [dzhampom]? Therefore let us carry out first it, and then let us return to the decoder. All this let us write after **JMP 00694001**:



Now we know the address of the following function of our [patcha] (**00400366**) and we can enter him instead of **40404040** with address **00400357** but not all so simply. It is there necessary to enter displacement relative to address **00694187**. You will look as it appears **jmp 0069418D** to **0040036[S]**:
E9 1C3E2900. **E9** – this [opkod] of [dzhampa], and **1C3E2900** (in the machine idea) = **of 00293E1C** (in the human) = **0069418D-0040036C-shch**. I.e. this nothing else but displacement. In order not to suffer and not to calculate these displacement let us pass to address **694187**, let us enter there **jmp 00400366**



and let us look as [Olli] it assembled the command:



Well it is here and there is necessary displacement. Let us enter it instead of **40404040** (to enter necessary in “human” idea! **FFD6C1DA** why thus? [Izuchi] the basics of assembler). Well here we already have a beginning of [patcha]:

```
00400350    C605 87416900 E9        MOV BYTE PTR DS:[694187],0E9
00400357    C705 88416900 DAC1D6FF  MOV DWORD PTR DS:[694188],FFD6C1DA
00400361  - E9 9B3C2900             JMP FO0F53~1.00694001
00400366    81E9 00D00B2C           SUB ECX,2C0BD000
0040036C  - E9 1C3E2900             JMP FO0F53~1.0069418D
```

After the second decoder will be completely decoded ASProtect it will fall into the function with address **00400366**, where the rubbed over command will be carried out and control will pass again to the decoder (well this only thus far we we will not find the place, where it is possible to [propatchit] the second decoder.). We place [brjak] on**0040036C**and start program. They interrupted, it means the second decoder it is deciphered. Let us look how it appears:

```
0069420E    8911                MOV DWORD PTR DS:[ECX],EDX
00694210    68 F143CF2D         PUSH 2DCF43F1
00694215    5F                  POP EDI
00694216    83E9 03             SUB ECX,3
00694219    BF BAA8AA25         MOV EDI,25AAA8BA
0069421E    49                  DEC ECX
0069421F    8BFA                MOV EDI,EDX
00694221    48                  DEC EAX
00694222  v 0F85 0E000000       JNZ FO0F53~1.00694236
00694228    66:8BF1             MOV SI,CX
0069422B  v E9 18000000         JMP FO0F53~1.00694248
00694230    36:37               AAA
```

[Potrassirovav], we understand, that this again cycle and after finalizing control will pass to address **00694228**, it means instead of [dzhampa] with address**0040036C**necessary to write commands, [propatchivajushchie] instructions on address **00694228**. But after them to carry out the instruction **of mov si, cx**, to [propatchit] the third decoder and to return the secondly. To more shortly repeat all that the fact that they made with the previous decoder. In me this code came out:

```
00400350    C605 87416900 E9        MOV BYTE PTR DS:[694187],0E9
00400357    C705 88416900 DAC1D6FF  MOV DWORD PTR DS:[694188],FFD6C1DA
00400361  - E9 9B3C2900             JMP FO0F53~1.00694001
00400366    81E9 00D00B2C           SUB ECX,2C0BD000
0040036C    C605 28426900 E9        MOV BYTE PTR DS:[694228],0E9
00400373    C705 29426900 55C1D6FF  MOV DWORD PTR DS:[694229],FFD6C155
0040037D  - E9 0B3E2900             JMP FO0F53~1.0069418D
00400382    66:8BF1                 MOV SI,CX
00400385  - E9 BE3E2900             JMP FO0F53~1.00694248
```

Further we repeat everything. We place [brjak] on **00400385** F9 they interrupted. The third decoder is deciphered. We look, that there:

```
006942D3    8F043B              POP DWORD PTR DS:[EBX+EDI]
006942D6    B5 21               MOV CH,21
006942D8    BA B2DA4D62         MOV EDX,624DDAB2
006942DD    83EF 01             SUB EDI,1
006942E0    81E2 755EF474       AND EDX,74F45E75
006942E6    4F                  DEC EDI
006942E7    4F                  DEC EDI
006942E8    4F                  DEC EDI
006942E9    BE F16C3952         MOV ESI,52396CF1
006942EE    81FF ECF9FFFF       CMP EDI,-614
006942F4  ^ 0F85 8AFFFFFF       JNZ FO0F53~1.00694284
006942FA    BA 62548E0A         MOV EDX,0A8E5462
006942FF    5E                  POP ESI
```

Entire also the very upon transfer to **006942FA** the fourth decoder will be completely decoded. I think and here you will manage themselves:

```
00400361  - E9 9B3C2900             JMP FO0F53~1.00694001
00400366    81E9 00D00B2C           SUB ECX,2C0BD000
0040036C    C605 28426900 E9        MOV BYTE PTR DS:[694228],0E9
00400373    C705 29426900 55C1D6FF  MOV DWORD PTR DS:[694229],FFD6C155
0040037D  - E9 0B3E2900             JMP FO0F53~1.0069418D
00400382    66:8BF1                 MOV SI,CX
00400385    C605 FA426900 E9        MOV BYTE PTR DS:[6942FA],0E9
0040038C    C705 FB426900 9CC0D6FF  MOV DWORD PTR DS:[6942FB],FFD6C09C
00400396  - E9 AD3E2900             JMP FO0F53~1.00694248
0040039B    BA 62548E0A             MOV EDX,0A8E5462
004003A0  - E9 5A3F2900             JMP FO0F53~1.006942FF
```

We repeat. We place [brjak] on **004003A0** and start program. The fourth decoder is deciphered. We look that there:

```
0069436E    8918            MOV DWORD PTR DS:[EAX],EBX
00694370    BF 626C254B     MOV EDI,4B256C62
00694375    81E8 B07E6C03   SUB EAX,36C7EB0
0069437B    66:BA DCD5      MOV DX,0D5DC
0069437F    81C0 AC7E6C03   ADD EAX,36C7EAC
00694385  v E9 10000000     JMP FO0F53~1.0069439A
0069438A    C8 618647       ENTER 8661,47
0069438E  ^ 74 9D           JE SHORT FO0F53~1.0069432D
00694390    12E3            ADC AH,BL
00694392  ^ E0 99           LOOPDNE SHORT FO0F53~1.0069432D
00694394    5E              POP ESI
00694395    3F              AAS
00694396    0C 55           OR AL,55
00694398    6A 5B           PUSH 5B
0069439A    83EE 01         SUB ESI,1
0069439D  v 0F85 35000000   JNZ FO0F53~1.006943D8
006943A3  v 0F82 1F000000   JB FO0F53~1.006943C8
006943A9    68 D3A51663     PUSH 6316A5D3
006943AE    E8 13000000     CALL FO0F53~1.006943C6
006943B3    2F              DAS
006943B4    3C C5           CMP AL,0C5
006943B6    1A4B 28         SBB CL,BYTE PTR DS:[EBX+28]
006943B9    41              INC ECX
006943BA    E6 27           OUT 27,AL
006943BC    D4 7D           AAM 7D
006943BE  ^ 72 C3           JB SHORT FO0F53~1.00694383
006943C0    40              INC EAX
006943C1  ^ 79 BE           JNS SHORT FO0F53~1.00694381
006943C3    1F              POP DS
006943C4    6C              INS BYTE PTR ES:[EDI],DX
006943C5    35 595AE91F     XOR EAX,1FE95A59
006943CA    0000            ADD BYTE PTR DS:[EAX],AL
006943CC    0017            ADD BYTE PTR DS:[EDI],DL
```

But here here increasingly more complex. Cycle after finalizing will pass to the address **of 006943A3** but as it is seen the there conditional jump, which it did not be desirable [ba] to rub over, since it it is sufficiently complicated to carry out in [patche] (in the plan of the size of [patcha]). Well it is good. Then it is possible to rub over the following command (although known will not be carried out the conditional jump or not). But here here that exactly and problem. The fact is that this last decoder will be decoded not to the end and deciphers not only the code, which to us is necessary, but also itself. All which is located below address **of 006943A3** encoded it will decipher only during the work of this decoder. Then we make thus. Let us replace this passage to our, and in the function of our [patcha], to which will pass ASProtect after the interpretation of th code e interesting us we let us restore this conditional [dzhamp] let us return to it.

```
0040037D - E9 0B3E2900          JMP FO0F53~1.0069418D
00400382   66:8BF1              MOV SI,CX
00400385   C605 FA426900 E9     MOV BYTE PTR DS:[6942FA],0E9
0040038C   C705 FB426900 9CC0D6FF MOV DWORD PTR DS:[6942FB],FFD6C09C
00400396 - E9 AD3E2900          JMP FO0F53~1.00694248
0040039B   BA 62548E0A          MOV EDX,0A8E5462
004003A0   C605 A3436900 E9     MOV BYTE PTR DS:[6943A3],0E9
004003A7   C705 A4436900 0EC0D6FF MOV DWORD PTR DS:[6943A4],FFD6C00E
004003B1 - E9 493F2900          JMP FO0F53~1.006942FF
004003B6   C605 A3436900 0F     MOV BYTE PTR DS:[6943A3],0F
004003BD   C705 A4436900 821F0000 MOV DWORD PTR DS:[6943A4],1F82
004003C7 - E9 D73F2900          JMP FO0F53~1.006943A3
```

We place [brjak] on **004003C7** and start program. Everything! The interesting us code with address **00694581** is completely [raskriptovan], the thanks to you industrious decoders, [propatchivat] it we will be thus:

To

```
00694581    03C7            ADD EAX,EDI
00694583    8985 C7010000   MOV DWORD PTR SS:[EBP+1C7],EAX
00694589    8B55 5B         MOV EDX,DWORD PTR SS:[EBP+5B]
0069458C    8B85 C7010000   MOV EAX,DWORD PTR SS:[EBP+1C7]
00694592    8942 0C         MOV DWORD PTR DS:[EDX+C],EAX
00694595    8D9D 0D040000   LEA EBX,DWORD PTR SS:[EBP+40D]
```

Afterward

```
00694581    BA E7034000     MOV EDX,FO0F53~1.004003E7
00694586    FFD2            CALL EDX
00694588    90              NOP
00694589    8B55 5B         MOV EDX,DWORD PTR SS:[EBP+5B]
0069458C    8B85 C7010000   MOV EAX,DWORD PTR SS:[EBP+1C7]
00694592    8942 0C         MOV DWORD PTR DS:[EDX+C],EAX
```

Why thus? First of all you will look to [opkod] of the command **mov of edx, 004003E7**.

You do see? BA of – of [opkod] **of mov of edx**, and further goes address without any displacement (in “machine” idea). Then we cause function with the address, which is located into edx. This method is good even and fact that with the fulfillment of the instruction **of call** in the stack there will be the address of the recovery, to which we can pass, after carrying out, the instruction **of ret**. Why then we earlier did not use this method? But you will look, how many bytes it occupies. Thus far we have each byte on the calculation. The register of edx I used because further it it is rerecorded By aSProtect'[om]. We finish writing [patch]:

```
00400385   C605 FA426900 E9        MOV BYTE PTR DS:[6942FA],0E9
0040038C   C705 FB426900 9CC0D6FF  MOV DWORD PTR DS:[6942FB],FFD6C09C
00400396 - E9 AD3E2900             JMP FO0F53~1.00694248
0040039B   BA 62548E0A             MOV EDX,0A8E5462
004003A0   C605 A3436900 E9        MOV BYTE PTR DS:[6943A3],0E9
004003A7   C705 A4436900 0EC0D6FF  MOV DWORD PTR DS:[6943A4],FFD6C00E
004003B1 - E9 493F2900             JMP FO0F53~1.006942FF
004003B6   C605 A3436900 0F        MOV BYTE PTR DS:[6943A3],0F
004003BD   C705 A4436900 821F0000  MOV DWORD PTR DS:[6943A4],1F82
004003C7   C605 81456900 BA        MOV BYTE PTR DS:[694581],0BA
004003CE   C705 82456900 E7034000  MOV DWORD PTR DS:[694582],FO0F53~1.004003E7
004003D8   C705 86456900 FFD2908B  MOV DWORD PTR DS:[694586],8B90D2FF
004003E2 - E9 BC3F2900             JMP FO0F53~1.006943A3
004003E7   03C7                    ADD EAX,EDI
004003E9   8985 C7010000          MOV DWORD PTR SS:[EBP+1C7],EAX
004003EF   C3                      RETN
```

You will memorize, that to before return to the body Of aSProtect'[a] in the register of eax is located the address of that isolated to [oboasti] memory, where will pass ASProtect and this key place, after which ASProtect will work only in the chosen memory. You will memorize this number! [Potrassiruem], until we see here this code.

```
00C610F3   68 00800000     PUSH 8000
00C610F8   6A 00           PUSH 0
00C610FA   50              PUSH EAX
00C610FB   FF95 7D294400   CALL DWORD PTR SS:[EBP+44297D]   VirtualFree
00C61101   8D0E            LEA ECX,DWORD PTR DS:[ESI]
00C61103   8551 2C         TEST DWORD PTR DS:[ECX+2C],EDX
00C61106   44              INC ESP
00C61107   07              POP ES                           Modification c
00C61108   50              PUSH EAX
00C61109   C3              RETN
```

We see that after the call Of virtualFree Of aSPr again somewhere it passes and (here [khitrjuga]!) instructions after the address **of 00C610FB** are encoded and are deciphered not long before the fulfillment. It means necessarily to cause our [patch] in such place, where these instructions are already deciphered. To me they were pleased to instruction, that they are carried out before the instruction **of call**.

But as to us them to [propatchit]? Indeed this region to memory is allotted dynamically. You do remember, I did request to memorize value from the register of eax? With the aid of this address we will be able to [propatchit] the interesting us instructions. Let us calculate the displacement of data of instructions relative to address from eax. **00C610F of 3-00C61000=F3**. Let us write our [patch].

```
004003E2 - E9 BC3F2900             JMP FO0F53~1.006943A3
004003E7   03C7                    ADD EAX,EDI
004003E9   8985 C7010000          MOV DWORD PTR SS:[EBP+1C7],EAX
004003EF   83C2 22                 ADD EDX,22
004003F2   C680 F3000000          MOV BYTE PTR DS:[EAX+F3],0B9
004003F9   8990 F4000000          MOV DWORD PTR DS:[EAX+F4],EDX
004003FF   66:C780 F8000 MOV WORD PTR DS:[EAX+F8],0D1FF
00400408   C3                      RETN
00400409   5B                      POP EBX
0040040A   68 00800000             PUSH 8000
0040040F   6A 00                   PUSH 0
00400411   FFE3                    JMP EBX
```

We place [brjak] on **00400411** and start program. We are interrupted and look at the deciphered instructions.

```
00C610ED   8B85 75294400   MOV EAX,DWORD PTR SS:[EBP+442975]
00C610F3   B9 09044000     MOV ECX,400409
00C610F8   FFD1            CALL ECX
00C610FA   50              PUSH EAX
00C610FB   FF95 7D294400   CALL DWORD PTR SS:[EBP+44297D]
00C61101   8D85 512C4400   LEA EAX,DWORD PTR SS:[EBP+442C51]
00C61107   50              PUSH EAX
00C61108   C3              RETN
00C61109   0000            ADD BYTE PTR DS:[EAX],AL
00C6110B   0000            ADD BYTE PTR DS:[EAX],AL
00C6110D   0000            ADD BYTE PTR DS:[EAX],AL
00C6110F   40              INC EAX
```

Passage is accomplished to the address, which is located into eax, and here into eax this value falls from **ebp+442C51**. I can certainly not rights, but I am not confident, that the number **of 442C51** is constant, and is not generated each time (or each) anew. Therefore in order not to risk, let us replace the instruction **of retn** by the passage into our [patch], but in [patche] let us restore everything in the place (since we will rub over instructions after the instruction **of retn**) and let us carry out the instruction of retn in its [patche]. We finish writing [patch].

```
004003E9   8985 C7010000   MOV DWORD PTR SS:[EBP+1C7],EAX
004003EF   83C2 22         ADD EDX,22
004003F2   C680 F3000000   MOV BYTE PTR DS:[EAX+F3],0B9
004003F9   8990 F4000000   MOV DWORD PTR DS:[EAX+F4],EDX
004003FF   66:C780 F80000  MOV WORD PTR DS:[EAX+F8],0D1FF
00400408   C3              RETN
00400409   83C1 1A         ADD ECX,1A
0040040C   C643 07 BB      MOV BYTE PTR DS:[EBX+7],0BB
00400410   894B 08         MOV DWORD PTR DS:[EBX+8],ECX
00400413   66:C743 0C FFD  MOV WORD PTR DS:[EBX+C],0D3FF
00400419   5B              POP EBX
0040041A   68 00800000     PUSH 8000
0040041F   6A 00           PUSH 0
00400421   FFE3            JMP EBX
00400423   58              POP EAX
00400424   C740 F9 C30000  MOV DWORD PTR DS:[EAX-7],0C3
0040042B   66:C740 FE 000  MOV WORD PTR DS:[EAX-2],0
00400431   C3              RETN
```

We place [brjak] on **00400431** and will neglect program. They interrupted, we harvest F8 and fall into the program.

```
00C6130B   0000            ADD BYTE PTR DS:[EAX],AL
00C6130D   8B9D 552A4400   MOV EBX,DWORD PTR SS:[EBP+442A55]
00C61313   0BDB            OR EBX,EBX
00C61315   74 0A           JE SHORT 00C61321
00C61317   8B03            MOV EAX,DWORD PTR DS:[EBX]
```

Now we twist window downward in search of the instruction **of retn** and at the sufficiently large removal we see:

```
00C6159D   83C6 14         ADD ESI,14
00C615A0   8B95 D8304400   MOV EDX,DWORD PTR SS:[EBP+4430D8]
00C615A6   E9 EBFEFFFF     JMP 00C61496
00C615AB   8B85 652A4400   MOV EAX,DWORD PTR SS:[EBP+442A65]
00C615B1   50              PUSH EAX
00C615B2   0385 D8304400   ADD EAX,DWORD PTR SS:[EBP+4430D8]
00C615B8   5B              POP EBX
00C615B9   0BDB            OR EBX,EBX
00C615BB   8985 112F4400   MOV DWORD PTR SS:[EBP+442F11],EAX
00C615C1   61              POPAD
00C615C2   75 08           JNZ SHORT 00C615CC
00C615C4   B8 01000000     MOV EAX,1
00C615C9   C2 0C00         RETN 0C
00C615CC   68 00000000     PUSH 0
00C615D1   C3              RETN
00C615D2   8B85 DC304400   MOV EAX,DWORD PTR SS:[EBP+4430DC]
```

, in addition I am not confident, that this code will be always located on one and the same displacement relative to that address, on which we left [patcha]. Therefore I decided to search for this section on the signature and to place passage to [patch]. And, attention! Beginning from this address necessary to restore all rubbed over by us commands, since beginning from this address (can and earlier, we indeed already was restored the code) ASProtect begins to observe its integrity and will not make it possible so simple to rummage in its code. We finish writing [patch].

```
0040041A    68 00800000     PUSH 8000
0040041F    6A 00           PUSH 0
00400421    FFE3            JMP EBX
00400423    58              POP EAX
00400424    C740 F9 C30000  MOV DWORD PTR DS:[EAX-7],0C3
0040042B    66:C740 FE 000  MOV WORD PTR DS:[EAX-2],0
00400431    60              PUSHAD
00400432    8B4424 20       MOV EAX,DWORD PTR SS:[ESP+20]
00400436    40              INC EAX
00400437    8B08            MOV ECX,DWORD PTR DS:[EAX]
00400439    81F9 7508B801   CMP ECX,1B80875
0040043F ^  75 F5           JNZ SHORT F00F53~1.00400436
00400441    83C3 2F         ADD EBX,2F
00400444    C600 B8         MOV BYTE PTR DS:[EAX],0B8
00400447    8958 01         MOV DWORD PTR DS:[EAX+1],EBX
0040044A    66:C740 05 FFD  MOV WORD PTR DS:[EAX+5],0D0FF
00400450    61              POPAD
00400451    C3              RETN
00400452    58              POP EAX
00400453    83E8 07         SUB EAX,7
00400456    C700 7508B801   MOV DWORD PTR DS:[EAX],1B80875
0040045C    C740 04 000000  MOV DWORD PTR DS:[EAX+4],C2000000
00400463    50              PUSH EAX
00400464    C3              RETN
```

Well here, almost everything. Program is practically completely unpacked. BUT! If now it are neglected, then it will fall down with the cry about the fact that the virus is discovered. Now for us one must find testing the integrity of file and somehow mix ASProtect'[u] us to reveal. I decided to separately not [zamorachivatsja] and made just as Alex. We press Ctrl+G, we write **MapViewOfFile**, harvest Enter and we on this function. To place [brjak] on it is impossible in what place, since ASProtect completely [dizassembliruet] entire function and it searches for interceptions. It is possible to place [brjak] on the memory with this address, but, in addition this [brjak] will frequently operate because of constant checkings. But iron [brjaki] Of aSProtect in me constantly were discarded.

```
7C80B78D    8BFF            MOV EDI,EDI
7C80B78F    55              PUSH EBP
7C80B790    8BEC            MOV EBP,ESP
7C80B792    6A 00           PUSH 0
7C80B794    FF75 18         PUSH DWORD PTR SS:[EBP+18]
7C80B797    FF75 14         PUSH DWORD PTR SS:[EBP+14]
7C80B79A    FF75 10         PUSH DWORD PTR SS:[EBP+10]
7C80B79D    FF75 0C         PUSH DWORD PTR SS:[EBP+C]
7C80B7A0    FF75 08         PUSH DWORD PTR SS:[EBP+8]
7C80B7A3    E8 76FFFFFF     CALL kernel32.MapViewOfFileEx
7C80B7A8    5D              POP EBP
7C80B7A9    C2 1400         RETN 14
```

But it is possible to enter differently. We see that the function **Of mapViewOfFile** is altogether only adapter to the function **Of mapViewOfFileEx**. Here on it let us place [brjak]. We harvest F9. We are interrupted and look into the window of stack.

```
0012FDEC  7C80B7A8  CALL to MapViewOfFileEx from kerne
0012FDF0  0000007C  hMapObject = 0000007C (window)
0012FDF4  00000004  AccessMode = FILE_MAP_READ
0012FDF8  00000000  OffsetHigh = 0
0012FDFC  00000000  OffsetLow = 0
0012FE00  00000000  MapSize = 0
0012FE04  00000000  BaseAddr = NULL
0012FE08  0012FE3C
0012FE0C  00C4867A  RETURN to 00C4867A
```

We see that the address of recovery exactly falls into the body Of aSProtect'[a]. it is passed to it.

```
00C48670    A1 E497C500     MOV EAX,DWORD PTR DS:[C597E4]
00C48675    8B40 08         MOV EAX,DWORD PTR DS:[EAX+8]
00C48678    FFD0            CALL EAX
00C4867A    8BD8            MOV EBX,EAX
00C4867C    50              PUSH EAX
00C4867D    E8 4A010000     CALL 00C487CC
00C48682    56              PUSH ESI
00C48683    C1CE 99         ROR ESI,99
00C48686    BE 2AFD4700     MOV ESI,47FD2A
00C4868B    8B7424 10       MOV ESI,DWORD PTR SS:[ESP+10]
```

We see that into the register of ebx sends the indicator to the file. Theory is such, that it is necessary to emulate the function **Of mapViewOfFile**. To place interception on the command of mov of ebx, eax then to cause the function **Of virtualAlloc** for the isolation of memory under the file, then to copy the file, which loaded into the memory Of aSProtect (it is it's a pity, that it

only for reading) into the isolated by us memory and to there already lead file to that state, in which it was to [propatchivanija], and then restore the rubbed over instruction of mov of ebx, eax and that the fact that they rubbed over after it, then to substitute the address our of the original file to the address, cleaned. The place of the call **Of mapViewOfFile** we will also search for on the signature. We finish writing [patch].

```
0040044A   66:C740 05 FFD(  MOV WORD PTR DS:[EAX+5],0D0FF
00400450   61               POPAD
00400451   C3               RETN
00400452   58               POP EAX
00400453   83E8 07          SUB EAX,7
00400456   C700 7508B801    MOV DWORD PTR DS:[EAX],1B80875
0040045C   C740 04 000000(  MOV DWORD PTR DS:[EAX+4],C2000000
00400463   50               PUSH EAX
00400464   60               PUSHAD
00400465   9C               PUSHFD
00400466   83C0 0B          ADD EAX,0B
00400469   8B00             MOV EAX,DWORD PTR DS:[EAX]
0040046B   48               DEC EAX
0040046C   8B08             MOV ECX,DWORD PTR DS:[EAX]
0040046E   81F9 FFD08BD8    CMP ECX,D88BD0FF
00400474 ^ 75 F5            JNZ SHORT FO45F3~1.0040046B
00400476   83C0 02          ADD EAX,2
00400479   C600 BB          MOV BYTE PTR DS:[EAX],0BB
0040047C   C740 01 8D0440(  MOV DWORD PTR DS:[EAX+1],FO45F3~1.0040048D
00400483   C740 05 FFD390(  MOV DWORD PTR DS:[EAX+5],5690D3FF
0040048A   9D               POPFD
0040048B   61               POPAD
0040048C   C3               RETN
```

Now after ASProtect will cause the function **Of mapViewOfFile** control immediately it will fall on address 0040048D, where we should cause the function **Of vitualAlloc**. But here indeed the misfortune Of aSProtect [bolshe] does not include this function in its IAT, which fills charger Windows. Now it obtains it dynamically. However, and that to us to now return back and to search for where ASProtect does cause this function and to memorize somewhere this address? But where we can it write down? I think more simply to itself to add this [fuktsiju] in IAT Of aSProtect'a. And let its Windows gives to us. Let us look with the aid of PE Of tools, where the directory of import begins.

Export Directory	00000000	00000000	...	H
Import Directory	00294A50	00000398	...	H

In principle it is possible to add function, also, with the aid of **PE Of tools** on the automaton. But me does not please itself as it this makes; therefore it is better by knobs. Let us calculate delta the displacement (read [manualy] on the import). For the beginning let us determine in what section it is located the directory of import.

Sections Editor

Sections Informations [HEX]

Name	Virtual Size	Virtual Offset	Raw Size	Raw Offset	Characteristics
	0015C000	00001000	0008B400	00001000	E0000040
	00056000	0015D000	00017200	0008C400	E0000040
	00010000	001B3000	00001E00	000A3600	E0000040
.rsrc	000D1000	001C3000	0002D800	000A5400	E0000040
.data	00039000	00294000	00039000	000D2C00	E0000040
.adata	00001000	002CD000	00000000	0010BC00	E0000040

Close

We see that this .data, because VA of the directory of import is located exactly after VA of this section. It means Delta=00294000-000D **of 2c00= of 1C1400**. Well here now we can calculate RVA of the directory of import. **RVA= of 00294A 50-1C1400=D3650**. Let us open

file into Hex editor and will pass to obtained RVA. We see massif IID.

```
000D3650  00 00 00 00  00 00 00 00  00 00 00 00  10 4A 29 00
000D3660  FC 49 29 00  00 00 00 00  00 00 00 00  00 00 00 00
000D3670  B8 4B 29 00  74 4C 29 00  00 00 00 00  00 00 00 00
000D3680  00 00 00 00  C4 4B 29 00  7C 4C 29 00  00 00 00 00
```

Let us look the name of first imported DLL. **[Adres]=00294A of 10-1C1400=D3610**. We look, what do we have with this address.

```
000D3610  6B 65 72 6E  65 6C 33 32  2E 64 6C 6C  00 00 00 47  kernel32.dll...G
000D3620  65 74 50 72  6F 63 41 64  64 72 65 73  73 00 00 00  etProcAddress...
000D3630  47 65 74 4D  6F 64 75 6C  65 48 61 6E  64 6C 65 41  GetModuleHandleA
```

It is excellent! That which is necessary. Now let us look, where is located the massif of indicators to the names of functions. **[Adres]=002949FC-1C1400=D35FC**.

```
000D35F0  00 00 00 00  00 00 00 00  00 00 00 00  1D 4A 29 00
000D3600  2E 4A 29 00  41 4A 29 00  00 00 00 00  00 00 00 00
```

As we see they are imported only three functions. As you know this massif it must conclude **with 00000000**, and, it concludes **with 00000000 00000000**. As it is successful! Instead of next-to-last 00000000 we can enter address for the line **“Of virtualAlloc”** and charger it automatically changes it to the real address of function. But there are no lines **“Of virtualAlloc”** in the file! But we will enter our its scarcely higher [patcha]. Let us make this in the same Of hex editor.

```
0000340  56 69 72 74  75 61 6C 41  6C 6C 6F 63  00 00 00 00  VirtualAlloc....
0000350  C6 05 87 41  69 00 E9 C7  05 88 41 69  00 DA C1 D6  Ж.‡Ai.йЗ.€Ai.Ъ
```

We enter instead of next-to-last **00000000 - > 0000033E (00000340-2)**

```
000D35F0  00 00 00 00  00 00 00 00  00 00 00 00  1D 4A 29 00
000D3600  2E 4A 29 00  41 4A 29 00  3E 03 00 00  00 00 00 00
```

All, now we know that we can cause **VirtualAlloc** to **D3608+1C1400+00400000=694A08** at any time ☺. let us write our [patch]. Let us preserve registers and let us isolate memory with size our file:

```
0040048D   60              PUSHAD
0040048E   BB 00C01000     MOV EBX,10C000
00400493   6A 04           PUSH 4
00400495   68 00100000     PUSH 1000
0040049A   53              PUSH EBX
0040049B   6A 00           PUSH 0
0040049D   B8 084A6900     MOV EAX,<&kernel32.VirtualAlloc>
004004A2   FF10            CALL DWORD PTR DS:[EAX]
```

Let us copy into the chosen memory the file, loaded By aSProtect'[om] and will substitute indicator to our:

```
004004A4   33D2            XOR EDX,EDX
004004A6   8B7424 1C       MOV ESI,DWORD PTR SS:[ESP+1C]
004004AA   8BF8            MOV EDI,EAX
004004AC   8BCB            MOV ECX,EBX
004004AE   F3:A4           REP MOVS BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]
004004B0   894424 1C       MOV DWORD PTR SS:[ESP+1C],EAX
```

Let us clean [patcha] from PE title:

```
004004B4   B9 10030000     MOV ECX,310
004004B9   83C1 04         ADD ECX,4
004004BC   C70408 00000000 MOV DWORD PTR DS:[EAX+ECX],0
004004C3   81F9 8A050000   CMP ECX,58A
004004C9 ^ 76 EE           JBE SHORT FO0F53~1.004004B9
```

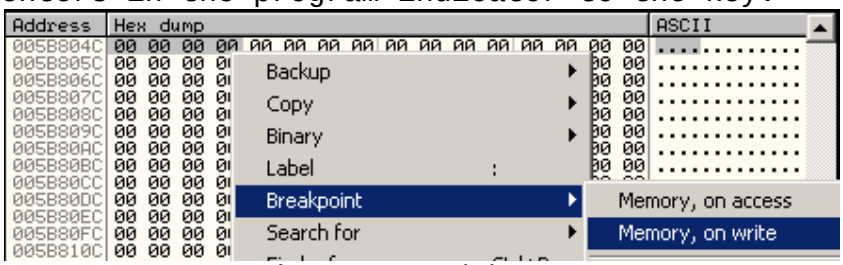
Let us restore the table of import and the beginning Of aSProtect'[a]:

```
004004CB   B9 08360D00     MOV ECX,0D3608
004004D0   C70408 0000000( MOV DWORD PTR DS:[EAX+ECX],0
004004D7   B9 01100000     MOV ECX,1001
004004DC   C70408 0140690( MOV DWORD PTR DS:[EAX+ECX],FO0F53~1.00694001
004004E3   61              POPAD
```

Well let us restore the intercepted instructions and let us return to the body Of aSProtect'[a].

```
004004E4   5E              POP ESI
004004E5   83EE 07         SUB ESI,7
004004E8   C706 8BD850E8   MOV DWORD PTR DS:[ESI],E850D88B
004004EE   C746 04 4A0100( MOV DWORD PTR DS:[ESI+4],14A
004004F5   FFE6            JMP ESI
```

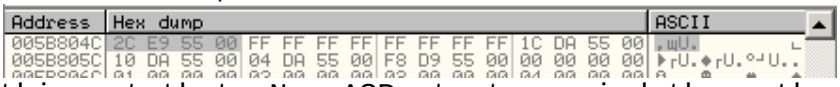
Now, if we neglect program, then it will be neglected, and it will completely normally work. Now let us find the place, where ASProtect enters in the program indicator to the key.



We start program, we are interrupted here.

```
00C32663   F3:A5           REP MOVS DWORD PTR ES:[EDI],DWORD PTR DS:[ESI]
00C32665   89C1            MOV ECX,EAX
00C32667   83E1 03         AND ECX,3
```

We harvest F8 and look, that we have to **005B804C**.



As we see this not that. Now ASProtect carried there the address, which was to the packing and where the further it it changes. Therefore again we harvest F9 and are interrupted here.

```
0044CD75   8B4424 04       MOV EAX,DWORD PTR SS:[ESP+4]
0044CD79   A3 4C805B00     MOV DWORD PTR DS:[5B804C],EAX
0044CD7E   C2 0400         RETN 4
```

But here this is already very interesting! Here ASProtect writes value from eax concretely with th address e interesting us. We harvest two times F8 and look, where they left:

```
00C4CABA   8B47 04         MOV EAX,DWORD PTR DS:[EDI+4]
00C4CABD   FFD0            CALL EAX
00C4CABF   A1 E097C500     MOV EAX,DWORD PTR DS:[C597E0]
00C4CAC4   8B40 48         MOV EAX,DWORD PTR DS:[EAX+48]
```

I decided to intercept the function **of call of eax** and to write down with th address e interesting us indicator to th line e necessary to us, and after this to return control to the code, after **call of eax**. We will also search for this place on the signature. As I not tried, but so also I could not compose the unique signature of the long of 8 bytes. But that that it came out it is encountered in the file of 2 times and to us is necessary it is 2nd 4. let us write [patch].

```
004004E3   61              POPAD
004004E4   5E              POP ESI
004004E5   83EE 07         SUB ESI,7
004004E8   C706 8BD850E8   MOV DWORD PTR DS:[ESI],E850D88B
004004EE   C746 04 4A0100( MOV DWORD PTR DS:[ESI+4],14A
004004F5   60              PUSHAD
004004F6   33FF            XOR EDI,EDI
004004F8   46              INC ESI
004004F9   8B06            MOV EAX,DWORD PTR DS:[ESI]
004004FB   3D 4704FFD0     CMP EAX,D0FF0447
00400500 ^ 75 F6           JNZ SHORT FO0F53~1.004004F8
00400502   47              INC EDI
00400503   83FF 01         CMP EDI,1
00400506 ^ 74 F0           JE SHORT FO0F53~1.004004F8
00400508   83EE 02         SUB ESI,2
0040050B   C606 B8         MOV BYTE PTR DS:[ESI],0B8
0040050E   81C3 93000000   ADD EBX,93
00400514   895E 01         MOV DWORD PTR DS:[ESI+1],EBX
00400517   66:C746 05 FFD( MOV WORD PTR DS:[ESI+5],0D0FF
0040051D   61              POPAD
0040051E   FFE6            JMP ESI
```

Well, then we intercepted the function Of aSProtect'[a], which corresponds for the registration and now we must it emulate. For this let us enter to **005B804C** indicator to the key. But indeed we do not have key! But we and it will enter scarcely higher our [patcha] (above the line “Of virtualAlloc”). We will use Hex by editor.

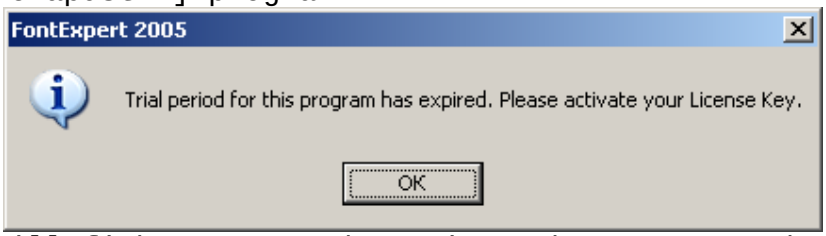
```
320  53 70 65 63  69 61 6C 20  62 75 69 6C  64 20 66 72  Special build fr
330  6F 6D 20 50  45 5F 4B 69  6C 6C 00 00  00 00 00 00  om PE_Kill.....
340  56 69 72 74  75 61 6C 41  6C 6C 6F 63  00 00 00 00  VirtualAlloc....
```

And let us write [patch].

```
0040051E  FFE6            JMP ESI
00400520  60              PUSHAD
00400521  B9 4C805B00     MOV ECX,F00F53~1.005B804C
00400526  C701 20034000   MOV DWORD PTR DS:[ECX],F00F53~1.00400320  ASCII "Special build from PE_Kill"
0040052C  61              POPAD
0040052D  58              POP EAX
0040052E  48              DEC EAX
0040052F  C740 FA 508B47( MOV DWORD PTR DS:[EAX-6],4478B50
00400536  66:C740 FE FFD( MOV WORD PTR DS:[EAX-2],0D0FF
0040053C  C600 A1         MOV BYTE PTR DS:[EAX],0A1
0040053F  FFE0            JMP EAX
```

Now, if we neglect program, then let us see, that NAG disappeared and program was neglected as [zareganaja]. Let us open window “about the program…” :

As we see – all good. But now let us try to transfer time forward and by [perezapustim] program.



[Mda]… We will fight. Let us determine, where occurs checking to the time expiration of [triala].
Let us place [brjak] on **MessageBoxA**. But! To place is necessary also as to **MapViewOfFile**. For the beginning let us place [brjak] on**0040053F**in our [patche]. Let us neglect program. They interrupted. We harvest Ctrl+G, we write **MessageBoxA**, harvest Enter. Now we in this function.

```
77D7050B   8BFF            MOV EDI,EDI
77D7050D   55              PUSH EBP
77D7050E   8BEC            MOV EBP,ESP
77D70510   833D 1C04D977 ( CMP DWORD PTR DS:[77D9041C],0
77D70517   74 24           JE SHORT USER32.77D7053D
77D70519   64:A1 18000000 MOV EAX,DWORD PTR FS:[18]
77D7051F   6A 00           PUSH 0
77D70521   FF70 24         PUSH DWORD PTR DS:[EAX+24]
77D70524   68 F40AD977     PUSH USER32.77D90AF4
77D70529   FF15 1812D377   CALL DWORD PTR DS:[<&KERNEL32.Interlocke
77D7052F   85C0            TEST EAX,EAX
77D70531   75 0A           JNZ SHORT USER32.77D7053D
77D70533   C705 F00AD977 ( MOV DWORD PTR DS:[77D90AF0],1
77D7053D   6A 00           PUSH 0
77D7053F   FF75 14         PUSH DWORD PTR SS:[EBP+14]
77D70542   FF75 10         PUSH DWORD PTR SS:[EBP+10]
77D70545   FF75 0C         PUSH DWORD PTR SS:[EBP+C]
77D70548   FF75 08         PUSH DWORD PTR SS:[EBP+8]
77D7054B   E8 2D000000     CALL USER32.MessageBoxExA
77D70550   5D              POP EBP
77D70551   C2 1000          RETN 10
```

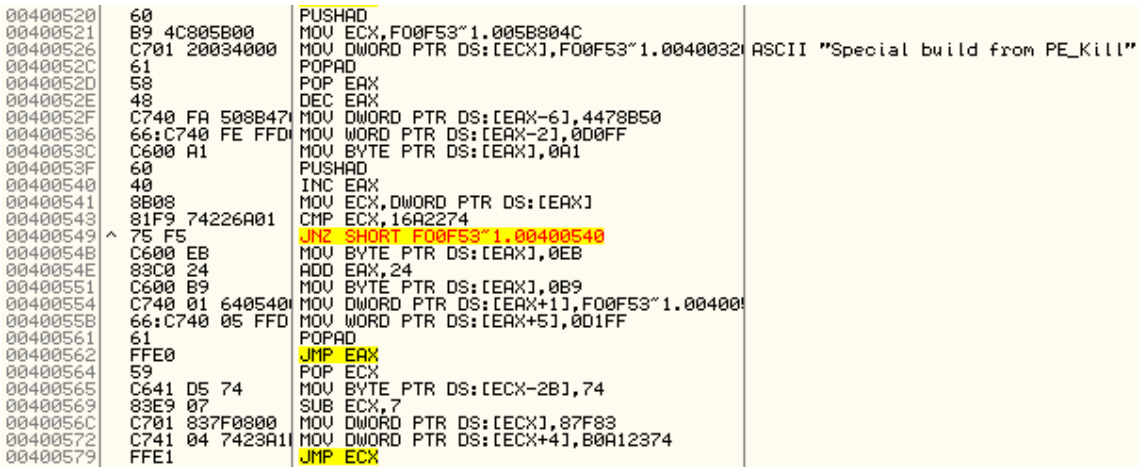
As we see this function it is also adapter to the function **Of messageBoxExA**. Let us pass to this function and let us place [brjak] on it. But now let us use [obaldennuju] [fichu] Of ollyDbg of – reverse laying out. For this let us first transfer diagnostic routine into the regime of the laying out of the code by the pressure of the keys For ctrl+F11. We await, until program interrupts on our [brjake]. It interrupted. Now we begin to harvest key “-” minus on the digital keyboard. In this case Olly begins to produce the reverse laying out of the code, in this case showing the state of registers and stack at that moment, when this instruction was carried out. We leave into this place.

```
0044CCB2   56              PUSH ESI
0044CCB3   57              PUSH EDI
0044CCB4   BE 80C55600     MOV ESI,FO0F53~1.0056C580          ASCII "FontExpert 2005"
0044CCB9   8D7D F0         LEA EDI,DWORD PTR SS:[EBP-10]
0044CCBC   A5              MOVS DWORD PTR ES:[EDI],DWORD PTR DS:[ESI]
0044CCBD   A5              MOVS DWORD PTR ES:[EDI],DWORD PTR DS:[ESI]
0044CCBE   8B1D C4D75500   MOV EBX,DWORD PTR DS:[55D7C4]       USER32.MessageBoxA
0044CCC4   A5              MOVS DWORD PTR ES:[EDI],DWORD PTR DS:[ESI]
0044CCC5   6A 13           PUSH 13
0044CCC7   A5              MOVS DWORD PTR ES:[EDI],DWORD PTR DS:[ESI]
0044CCC8   59              POP ECX
0044CCC9   BE 30C55600     MOV ESI,FO0F53~1.0056C530          ASCII "Trial period for this p
0044CCCE   8DBD 7CFFFFFF   LEA EDI,DWORD PTR SS:[EBP-84]
0044CCD4   F3:A5           REP MOVS DWORD PTR ES:[EDI],DWORD PTR DS:[ESI
0044CCD6   6A 40           PUSH 40
0044CCD8   8D45 F0         LEA EAX,DWORD PTR SS:[EBP-10]
0044CCDB   50              PUSH EAX
0044CCDC   8D85 7CFFFFFF   LEA EAX,DWORD PTR SS:[EBP-84]
0044CCE2   50              PUSH EAX
0044CCE3   6A 00           PUSH 0
0044CCE5   A4              MOVS BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]
0044CCE6   FFD3            CALL EBX                           USER32.MessageBoxA
0044CCE8   6A 40           PUSH 40
0044CCEA   59              POP ECX
0044CCEB   BE 28C45600     MOV ESI,FO0F53~1.0056C428          ASCII "012345678901234567890123
0044CCF0   8DBD 78FEFFFF   LEA EDI,DWORD PTR SS:[EBP-188]
0044CCF6   F3:A5           REP MOVS DWORD PTR ES:[EDI],DWORD PTR DS:[ESI
```

As we see precisely hence it was caused **MessageBox**. But there are no conditional jumps; therefore let us continue reverse laying out, until we leave here here.

```
00C4CE26   FFD0            CALL EAX
00C4CE28   837F 38 00      CMP DWORD PTR DS:[EDI+38],0
00C4CE2C   74 22           JE SHORT 00C4CE50
00C4CE2E   6A 01           PUSH 1
00C4CE30   68 2E57C300     PUSH 0C3572E
00C4CE35   832C24 02       SUB DWORD PTR SS:[ESP],2
00C4CE39   FF25 A8B4C500   JMP DWORD PTR DS:[C5B4A8]
00C4CE3F   EB 0F           JMP SHORT 00C4CE50
00C4CE41   837F 28 00      CMP DWORD PTR DS:[EDI+28],0
```

If passage to **00C4CE 2c** was carried out, then program would jump over through the leap **of 00C4CE39**, which leads for the communication about the end of [triala]. Will search for this place as you already they surmised on the signature. We finish writing [patch].



Now, if we neglect program, then it will be neglected, and it will consider itself [zareganoj], but if you visit into **Tools->Detect Of font Of problems->Installed Of fonts** and will press in the appeared window **OK**, then program will be thrown out into the system error. To catch the reason for error is simple. It is necessary to place [brjak] on reading to **005B804C**, then to open **Tools->Detect Of font Of problems->Installed Of fonts** to harvest OK and we will fall on checking of [zareganosti]. After which if we not [zaregany] window is shut normally, but if yes – that here here I did not greatly understand that he occurs. The whether this what first [izvratnaja] checking on [zareganost], then whether still that, but this something causes terrible Gluck and [proga] falls. These possible places are [poshifrovany] and are deciphered only with the introduction of key, can everything much simpler. On the whole it is necessary to read a little about the limitations, what me it does not absolutely want. If to you it is interesting to completely force open this program, then I think problems in you it will not arise, since last [propatchenaja] command was already in unpacked [proge] and now you can [patchit] any bytes of program. I round on this. Successes to you in all your undertakings! **PE_Kill**.

Complete [iskhodnik] of [patcha] (to **400350** it goes text, but [dizasm] accepted it for the code):

```
/* 400320 */PUSH EBX
/* 400321 */JO SHORT FO0F53~1.00400388
/* 400323 */ARPL WORD PTR DS:[ECX+61], BP
/* 400326 */INS BYTE PTR ES:[EDI], DX
/* 400327 */AND BYTE PTR DS:[EDX+75], AH
/* 40032A */IMUL EBP, DWORD PTR SS:[ESP+20], 6D6F7266
/* 400332 */AND BYTE PTR DS:[EAX+45], DL
/* 400335 */POP EDI
/* 400336 */DEC EBX
/* 400337 */IMUL EBP, DWORD PTR SS:[ESP+EBP * 2], 0
/* 40033F */ADD BYTE PTR DS:[ESI+69], DL
/* 400342 */JB SHORT FO0F53~1.004003B8
/* 400344 */JNZ SHORT FO0F53~1.004003A7
/* 400346 */INS BYTE PTR ES:[EDI], DX
/* 400347 */INC ECX
```

```
/* 400348 */INS BYTE PTR ES:[EDI], DX
/* 400349 */INS BYTE PTR ES:[EDI], DX
/* 40034A */OUTS DX, DWORD PTR ES:[EDI]
/* 40034B */ARPL WORD PTR DS:[EAX], AX
/* 40034D */ADD BYTE PTR DS:[EAX], AL
/* 40034F */ADD DH, AL
/* 400351 */ADD EAX, FO0F53~1.00694187
/* 400356 */JMP of 41C80922
/* 40035B */IMUL EAX, DWORD PTR DS:[EAX], FFD6C1 DA
/* 400361 */JMP FO0F53~1.00694001
/* 400366 */SUB ECX, 2c0BD000
/* 40036C*/OF MOV BYTE PTR DS:[694228], 0E9
/* 400373 */MOV DWORD PTR DS:[694229], FFD6C155
/* 40037D */JMP FO0F53~1.0069418D
/* 400382 */MOV SI, CX
/* 400385 */MOV BYTE PTR DS:[6942FA], 0E9
/* 40038C*/OF MOV DWORD PTR DS:[6942FB], FFD6C09C
/* 400396 */JMP FO0F53~1.00694248
/* 40039B */MOV EDX, 0A8E5462
/* 4003A0 */MOV BYTE PTR DS:[6943A3], 0E9
/* 4003A7 */MOV DWORD PTR DS:[6943A4], FFD6C00E
/* 4003B1 */JMP FO0F53~1.006942FF
/* 4003B6 */MOV BYTE PTR DS:[6943A3], 0F
/* 4003BD */MOV DWORD PTR DS:[6943A4], 1F82
/* 4003C7 */MOV BYTE PTR DS:[694581], 0BA
/* 4003CE */MOV DWORD PTR DS:[694582], FO0F53~1.004003E7
/* 4003D8 */MOV DWORD PTR DS:[694586], 8B90D2FF
/* 4003E2 */JMP FO0F53~1.006943A3
/* 4003E7 */ADD EAX, EDI
/* 4003E9 */MOV DWORD PTR SS:[EBP+1C7], EAX
/* 4003EF */ADD EDX, 22
/* 4003F2 */MOV BYTE PTR DS:[EAX+F3], 0B9
/* 4003F9 */MOV DWORD PTR DS:[EAX+F4], EDX
/* 4003FF */MOV WORD PTR DS:[EAX+F8], 0D1FF
/* 400408 */RETN
/* 400409 */ADD ECX, 1a
/* 40040C*/OF MOV BYTE PTR DS:[EBX+7], 0BB
/* 400410 */MOV DWORD PTR DS:[EBX+8], ECX
/* 400413 */MOV WORD PTR DS:[EBX+C], 0D3FF
/* 400419 */POP EBX
/* 40041A */PUSH 8000
/* 40041F */PUSH 0
/* 400421 */JMP EBX
/* 400423 */POP EAX
/* 400424 */MOV DWORD PTR DS:[EAX- "], 0C3
/* 40042B */MOV WORD PTR DS:[EAX-2], 0
/* 400431 */PUSHAD
/* 400432 */MOV EAX, DWORD PTR SS:[ESP+20]
/* 400436 */INC EAX
/* 400437 */MOV ECX, DWORD PTR DS:[EAX]
/* 400439 */CMP ECX, 1b80875
/* 40043F */JNZ SHORT FO0F53~1.00400436
/* 400441 */ADD EBX, 2F
```

```
/* 400444 */MOV BYTE PTR DS:[EAX], 0B8
/* 400447 */MOV DWORD PTR DS:[EAX+1], EBX
/* 40044A */MOV WORD PTR DS:[EAX+5], 0D0FF
/* 400450 */POPAD
/* 400451 */RETN
/* 400452 */POP EAX
/* 400453 */SUB EAX, "
/* 400456 */MOV DWORD PTR DS:[EAX], 1b80875
/* 40045C*/OF MOV DWORD PTR DS:[EAX+4], C2000000
/* 400463 */PUSH EAX
/* 400464 */PUSHAD
/* 400465 */PUSHFD
/* 400466 */ADD EAX, 0B
/* 400469 */MOV EAX, DWORD PTR DS:[EAX]
/* 40046B */DEC EAX
/* 40046C*/OF MOV ECX, OF DWORD PTR DS:[EAX]
/* 40046E */CMP ECX, D88BD0FF
/* 400474 */JNZ SHORT FO0F53~1.0040046B
/* 400476 */ADD EAX, 2
/* 400479 */MOV BYTE PTR DS:[EAX], 0BB
/* 40047C*/OF MOV DWORD PTR DS:[EAX+1], FO0F53~1.0040048D
/* 400483 */MOV DWORD PTR DS:[EAX+5], 5690D3FF
/* 40048A */POPFD
/* 40048B */POPAD
/* 40048C*/OF RETN
/* 40048D */PUSHAD
/* 40048E */MOV EBX, 10C000
/* 400493 */PUSH 4
/* 400495 */PUSH 1000
/* 40049A */PUSH EBX
/* 40049B */PUSH 0
/* 40049D */MOV EAX, <&kernel32.VirtualAlloc>
/* 4004A2 */CALL DWORD PTR DS:[EAX]
/* 4004A4 */XOR EDX, EDX
/* 4004A6 */MOV ESI, DWORD PTR SS:[ESP+1C]
/* 4004AA */MOV EDI, EAX
/* 4004AC */MOV ECX, EBX
/* 4004AE */REP MOVS BYTE PTR ES:[EDI], BYTE PTR DS:[ESI]
/* 4004B0 */MOV DWORD PTR SS:[ESP+1C], EAX
/* 4004B4 */MOV ECX, 310
/* 4004B9 */ADD ECX, 4
/* 4004BC */MOV DWORD PTR DS:[EAX+ECX], 0
/* 4004C3 */CMP ECX, 58A
/* 4004C9 */JBE SHORT FO0F53~1.004004B9
/* 4004CB */MOV ECX, 0D3608
/* 4004D0 */MOV DWORD PTR DS:[EAX+ECX], 0
/* 4004D7 */MOV ECX, 1001
/* 4004DC */MOV DWORD PTR DS:[EAX+ECX], FO0F53~1.00694001
/* 4004E3 */POPAD
/* 4004E4 */POP ESI
/* 4004E5 */SUB ESI, "
/* 4004E8 */MOV DWORD PTR DS:[ESI], E850D88B
/* 4004EE */MOV DWORD PTR DS:[ESI+4], 14A
```

```
/* 4004F5 */PUSHAD
/* 4004F6 */XOR EDI, EDI
/* 4004F8 */INC ESI
/* 4004F9 */MOV EAX, DWORD PTR DS:[ESI]
/* 4004FB */CMP EAX, D0FF0447
/* 400500 */JNZ SHORT FO0F53~1.004004F8
/* 400502 */INC EDI
/* 400503 */CMP EDI, 1
/* 400506 */JE SHORT FO0F53~1.004004F8
/* 400508 */SUB ESI, 2
/* 40050B */MOV BYTE PTR DS:[ESI], 0B8
/* 40050E */ADD EBX, 93
/* 400514 */MOV DWORD PTR DS:[ESI+1], EBX
/* 400517 */MOV WORD PTR DS:[ESI+5], 0D0FF
/* 40051D */POPAD
/* 40051E */JMP ESI
/* 400520 */PUSHAD
/* 400521 */MOV ECX, FO0F53~1.005B804C
/* 400526 */MOV DWORD PTR DS:[ECX], FO0F53~1.00400320
/* 40052C*/OF POPAD
/* 40052D */POP EAX
/* 40052E */DEC EAX
/* 40052F */MOV DWORD PTR DS:[EAX- '], 4478B50
/* 400536 */MOV WORD PTR DS:[EAX-2], 0D0FF
/* 40053C*/OF MOV BYTE PTR DS:[EAX], 0A1
/* 40053F */PUSHAD
/* 400540 */INC EAX
/* 400541 */MOV ECX, DWORD PTR DS:[EAX]
/* 400543 */CMP ECX, 16A2274
/* 400549 */JNZ SHORT FO0F53~1.00400540
/* 40054B */MOV BYTE PTR DS:[EAX], 0EB
/* 40054E */ADD EAX, 24
/* 400551 */MOV BYTE PTR DS:[EAX], 0B9
/* 400554 */MOV DWORD PTR DS:[EAX+1], FO0F53~1.00400564
/* 40055B */MOV WORD PTR DS:[EAX+5], 0D1FF
/* 400561 */POPAD
/* 400562 */JMP EAX
/* 400564 */POP ECX
/* 400565 */MOV BYTE PTR DS:[ECX-2b], 74
/* 400569 */SUB ECX, "
/* 40056C*/OF MOV DWORD PTR DS:[ECX], 87F83
/* 400572 */MOV DWORD PTR DS:[ECX+4], B0A12374
/* 400579 */JMP ECX
```