

Spyware Doctor 3.2.1 Build. 359

<http://www.pctools.com/>



In-line patching tutorial

1. Target analysis

After install it use PEiD to know more about the target coding (if this one is packed or not):

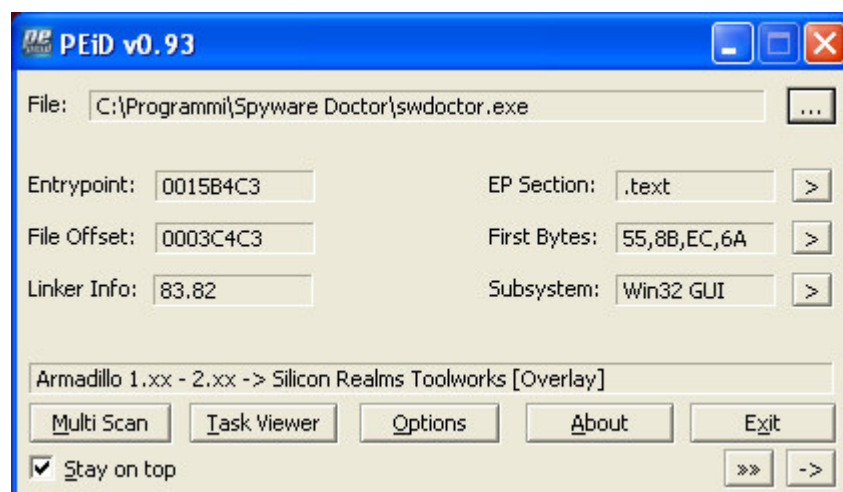


Fig. 1 PEiD target scanning



Fig. 2 RDG Packer Detector target analysis

2. In-line patching a brief introduction

This is the third tutorial about the Spyware Doctor 3.2.1 cracking saga, after the first one mainly related to cracking this target with loader generation we have write a second one related to manual unpacking (MUP) and full working patching, now is time to show how approach this target in another way called in-line patching, this is needed in order to understand all the possibility available in order to crack a packed executable target.

In-line patching is a type of patching mainly used in order to apply patching to a packed executable, when a target is packed (regard the type of packer) the executable code isn't directly accessible because this one isn't in a clear form due to the packer compression/cryptation. There are some step which have been take in before keep in running the packed code:

- start the packer stub at the EP (Entry Point)
- memory allocation for the packer/protector code
- decompression/decryption of the packer code
- execution of the packer code (CRC checking, licensing check and so on...)
- if all was right the packer can start the target code decompression/decryption and perform on the original code some anti-hacking measure (IAT scrambling, code splicing and other things focused to make hard the target unpacking/reversing)
- jumping to the OEP (Original Entry Point)
- execution of the original target

this is more generally, because everyone have play, e.g. with UPX, some stage isn't executed (like decompression and execution of the packer code or IAT destruction), by the way this list should be keep like as reference before attempt to study a new type of packer.

Well if target code isn't really be accessible by direct inspection we have to find another way to reach this one and then apply the patch (of course we can have also the need for skip some CRC check or other protection if this is present). A very effective way is based on the simple concept to follow the packer code execution until we're able to access at the real unpacked target code; in order to follow this flow we have to search the point into the packer code where execution start into the dynamically allocated memory area (by using **VirtualAlloc** API) and then jump into this area too, then search for other memory allocation and follow the flow until this process go to the end and the original code is fully decrypted. In order to take memory about every jump we have also to search into the executable for some free space where we can place our patching code, this free space will be call shortly "cave".

Then first we have to found where the packer code request to use some memory (by searching for a call to the **VirtualAlloc** API) then we have to store in a safe place the start of the memory allocated area (this area may be variable because depend how is organized the memory during the target execution, then may be different also into the same PC other then different PC), to do this first step, simply, after the **VirtualAlloc** call we have to redirect the execution to our cave and in this place apply some patch in order to modify the packer code (into the virtual allocated memory) to force it to point into our another cave and the repeat this steps until we're able to reach the target decrypted code. But also remember that in our cave we have also to patch some CRC check in order to skip the integrity control performed by the packer, then we need to do some consideration other than simple redirection to the next cave from the various dynamically allocated layer of code.

Finally this tut will be written in order to show a complete way for cracking then very deeping analysis is taken in order to show in a clever way how approach a packer and then how write a in-line code then sit down and take a little bit of free time before to start in reading this essay.

3. Packer analysis

We are ready to start, load the target in OllyDbg:

0055B4C3	55	PUSH EBP	
0055B4C4	8BEC	MOV EBP,ESP	
0055B4C6	6A FF	PUSH -1	
0055B4C8	68 205B5800	PUSH swdoctor.005B5820	
0055B4CD	68 00B25500	PUSH swdoctor.005B2500	SE handler installation
0055B4D2	64:A1 000000	MOV EAX,DWORD PTR FS:[0]	
0055B4D8	50	PUSH EAX	
0055B4D9	64:8925 0000	MOV DWORD PTR FS:[0],ESP	
0055B4E0	83EC 58	SUB ESP,58	
0055B4E3	53	PUSH EBX	
0055B4E4	56	PUSH ESI	
0055B4E5	57	PUSH EDI	
0055B4E6	8965 E8	MOV DWORD PTR SS:[EBP-18],ESP	
0055B4E9	FF15 88015800	CALL DWORD PTR DS:[<&KERNEL32.GetVersion	kernel32.GetVersion
0055B4EF	33D2	XOR EDX,EDX	
0055B4F1	8AD4	MOV DL,AH	

Fig. 3 Entry Point (EP)

Now we are into the packer entry point (EP), press Alt+O in order to open the debugging options window and sure about this settings:

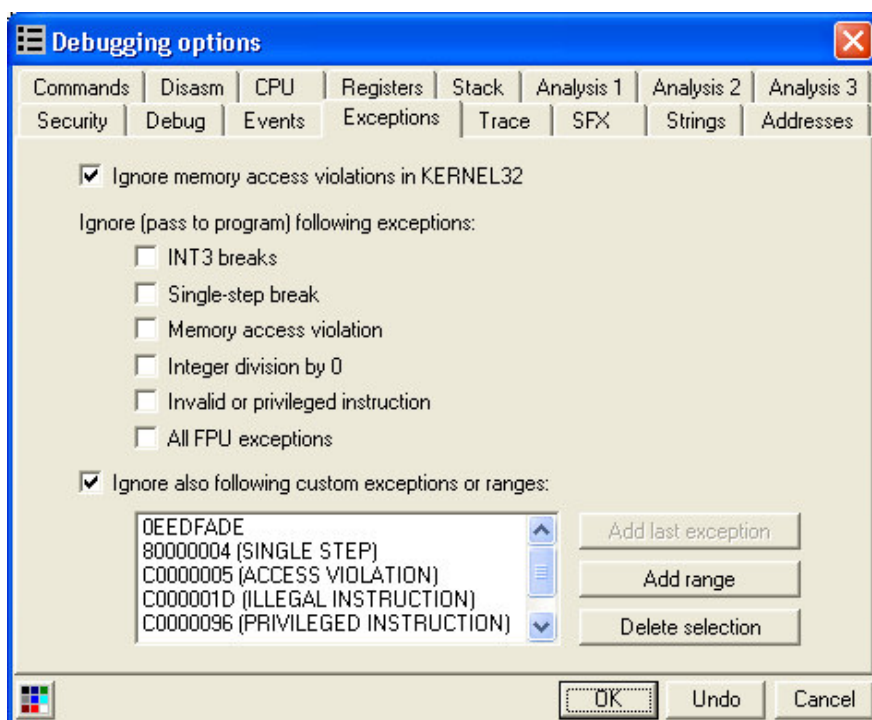


Fig. 4 Debugging options

now we have to search the point where we've to place the first redirection point to our patch cave, then place a memory breakpoint into the VirtualAlloc API function (press ALT+F1 and write BP VirtualAlloc then hit enter):

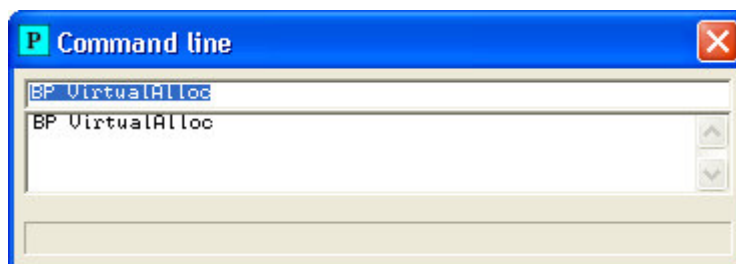


Fig. 5 VirtualAlloc memory breakpoint

then run the target by pressing SHIFT+F9.

When OllyDbg break remove the memory breakpoint and place a new memory breakpoint into the RETN 10 instruction then press F9:

77E5ABC5	55	PUSH EBP
77E5ABC6	8BEC	MOV EBP,ESP
77E5ABC8	FF75 14	PUSH DWORD PTR SS:[EBP+14]
77E5ABC9	FF75 10	PUSH DWORD PTR SS:[EBP+10]
77E5ABCA	FF75 0C	PUSH DWORD PTR SS:[EBP+C]
77E5ABD1	FF75 08	PUSH DWORD PTR SS:[EBP+8]
77E5ABD4	6A FF	PUSH -1
77E5ABD6	E8 9CFFFFFF	CALL kernel32.VirtualAllocEx
77E5ABD8	5D	POP EBP
77E5ABDC	C2 1000	RETN 10

Fig. 6

Registers (FPU)	
EAX	00000000
ECX	77E5ABC2 kernel32.77E5ABC2
EDX	00000000
EBX	7FFDF000
ESP	0012D6C4 ASCII "thT"
EBP	0012D74C
ESI	00000000
EDI	0012FF2C
EIP	77E5ABDC kernel32.77E5ABDC

Fig. 7

EAX keep the base address for the new allocated memory area, if you look into the registers window we have EAX=0 then this call isn't interesting for our purpose, press again SHIFT+F9 and again OllyDbg break in our breakpoint but now EAX is different than zero:

Registers (FPU)	
EAX	003A0000
ECX	77E5ABC2 kernel32.77E5ABC2
EDX	7FFE0304
EBX	7FFDF000
ESP	0012D6C4
EBP	0012D74C
ESI	00000000
EDI	0012FF2C
EIP	77E5ABDC kernel32.77E5ABDC

Fig. 8 New memory allocated area was from EAX=0x003A0000

We have now a valid address for some memory allocated area starting from the base address 0x003A0000 now press F7 once to return to the original code.

005468F6	> 61	POPAD	Protect = PAGE_READWRITE AllocationType = MEM_COMMIT Size Address = NULL VirtualAlloc
005468F7	. 9D	POPPD	
005468F8	. 0FC9	BSWAP EAX	
005468FA	. F7D1	NOT ECX	
005468FC	. 0FC8	BSWAP EAX	
005468FE	. F7D1	NOT ECX	
00546900	. 6A 04	PUSH 4	
00546902	. 68 00100000	PUSH 1000	
00546907	. 8B55 F0	MOV EDX,DWORD PTR SS:[EBP-10]	
0054690A	. 52	PUSH EDX	
0054690B	. 6A 00	PUSH 0	
0054690D	. FF15 6C005800	CALL DWORD PTR DS:[<&kernel32.VirtualAlloc>]	
00546913	. 8945 E0	MOV DWORD PTR SS:[EBP-20],EAX	
00546916	> 837D E0 00	CMP DWORD PTR SS:[EBP-20],0	
0054691A	. 75 11	JNZ SHORT swdoctor.0054692D	
0054691C	. C705 6C6C5800	MOV DWORD PTR DS:[586C6C1,2	
00546926	. 33C0	XOR EAX,EAX	
00546928	. E9 F2020000	JMP swdoctor.00546C1F	
0054692D	> 8B45 D8	MOV EAX,DWORD PTR SS:[EBP-28]	

Fig. 9

NOTE

We are talking about dynamically allocated memory area, then is important keep in mind that absolute address may be different in other OS and also into the same PC when you restart the target in OllyDbg, this is due to the memory state when the process perform a call to the **VirtualAlloc** API in order to request some memory reservation for own task.

In my system I've see this base address 0x003A0000 or 0x00F60000 then in the following exposition you can see different address, this is not really important rather you've to refer to relative address or offset from the base address.

The return address will be stored in [EBP-20] and a check is performed in order to sure about if the memory was really allocated or not (in this last case, as we've look, the returning value in EAX is set to 0).

Now the packer have request for some allocated memory area (which start from EAX=0x003A0000 / width = 0x4B000) then we can think about some decryption step act to write code into this area.

Restart the target with CTRL+F2 then goto into the memory dump area and press CTRL+G, write the OEP address 0x00503F68 and place a hardware breakpoint on write:

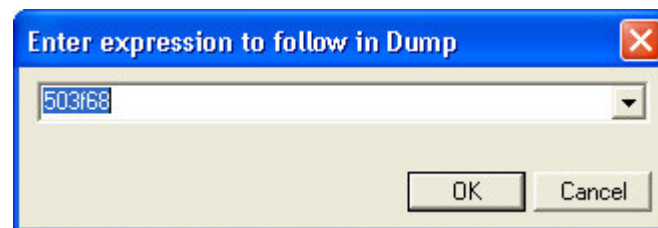


Fig. 10

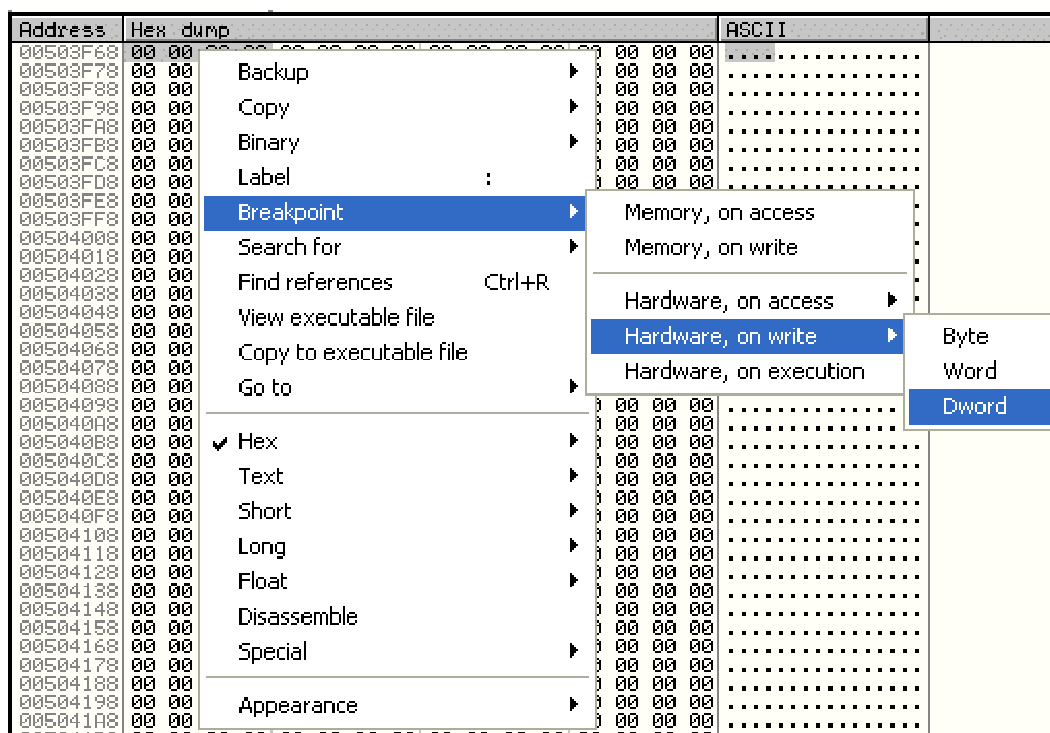


Fig. 11

Then press Shift+F9, OllyDbg will break in msvcrt:

77C12F43	F3:A5	REP MOVSD DWORD PTR ES:[EDI],DWORD PTR DS:[ESI]
77C12F45	FF2495 5830C17	JMP DWORD PTR DS:[EDX*4+77C13058]
77C12F4C	8BC7	MOV EAX,EDI
77C12F4E	BA 03000000	MOV EDX,3
77C12F53	83E9 04	SUB ECX,4
77C12F56	72 0C	JB SHORT msvcrt.77C12F64
77C12F58	83E0 03	AND EAX,3
77C12F5B	03C8	ADD ECX,EAX
77C12F5D	FF2485 702EC17	JMP DWORD PTR DS:[EAX*4+77C12F70]

Fig. 12

now press CTRL+F9 and then F7 once in order to return to the original code:

003C7D7D	6A 04	PUSH 4	
003C7D7F	FFB5 B4C4FFFF	PUSH DWORD PTR SS:[EBP-3B4C]	
003C7D85	8B85 F8C6FFFF	MOV EAX,DWORD PTR SS:[EBP-3908]	
003C7D8B	0385 B0C4FFFF	ADD EAX,DWORD PTR SS:[EBP-3B50]	
003C7D91	50	PUSH EAX	
003C7D92	FF15 4C213D00	CALL DWORD PTR DS:[3D214C]	kernel32.VirtualProtect
003C7D98	FFB5 B4C4FFFF	PUSH DWORD PTR SS:[EBP-3B4C]	
003C7D9E	FFB5 B8C4FFFF	PUSH DWORD PTR SS:[EBP-3B48]	
003C7DA4	8B85 F8C6FFFF	MOV EAX,DWORD PTR SS:[EBP-3908]	
003C7DA9	0385 B0C4FFFF	ADD EAX,DWORD PTR SS:[EBP-3B50]	
003C7DB0	50	PUSH EAX	
003C7DB1	E8 B4950000	CALL 003D136A	JMP to msvcrt.memcpy
003C7DB6	83C4 0C	ADD ESP,0C	
003C7DB9	8D85 ACC4FFFF	LEA EAX,DWORD PTR SS:[EBP-3B54]	
003C7DBF	50	PUSH EAX	
003C7DC0	FFB5 ACC4FFFF	PUSH DWORD PTR SS:[EBP-3B54]	
003C7DC6	FFB5 B4C4FFFF	PUSH DWORD PTR SS:[EBP-3B4C]	
003C7DCC	8B85 F8C6FFFF	MOV EAX,DWORD PTR SS:[EBP-3908]	
003C7DD2	0385 B0C4FFFF	ADD EAX,DWORD PTR SS:[EBP-3B50]	
003C7DD8	50	PUSH EAX	
003C7DD9	FF15 4C213D00	CALL DWORD PTR DS:[3D214C]	kernel32.VirtualProtect
003C7DDF	8B85 B8C4FFFF	MOV EAX,DWORD PTR SS:[EBP-3B48]	
003C7DE5	8985 B89EFFFF	MOV DWORD PTR SS:[EBP+FFFF9EB8],EAX	
003C7DEB	FFB5 B89EFFFF	PUSH DWORD PTR SS:[EBP+FFFF9EB8]	
003C7DF1	E8 6E950000	CALL 003D1364	JMP to msvcrt.??3@VAXPAX@Z
003C7DF6	59	POP ECX	
003C7DF7	^ E9 94FAFFFF	JMP 003C7890	
003C7DFC	8325 F41E3E00	AND DWORD PTR DS:[3E1EF4],0	
003C7E03	83BD 5CC8FFFF	CMP DWORD PTR SS:[EBP-37A4],0	
003C7E0A	74 33	JE SHORT 003C7E3F	
003C7E0C	8D85 84C4FFFF	LEA EAX,DWORD PTR SS:[EBP-3B7C]	
003C7E12	50	PUSH EAX	
003C7E13	6A 20	PUSH 20	
003C7E15	FFB5 5CC8FFFF	PUSH DWORD PTR SS:[EBP-37A4]	
003C7E1B	FF35 9C203E00	PUSH DWORD PTR DS:[3E209C]	
003C7E1D	FF15 4C213D00	CALL DWORD PTR DS:[3D214C]	kernel32.VirtualProtect
003C7E27	8B85 ECC6FFFF	MOV EAX,DWORD PTR SS:[EBP-3914]	
003C7E2D	8985 B49EFFFF	MOV DWORD PTR SS:[EBP+FFFF9EB4],EAX	
003C7E33	FFB5 B49EFFFF	PUSH DWORD PTR SS:[EBP+FFFF9EB4]	
003C7E39	E8 26950000	CALL 003D1364	JMP to msvcrt.??3@VAXPAX@Z
003C7E3E	59	POP ECX	

Fig. 13

Before executing the call in 0x003C7DB6 the target isn't unpacked this call start to unpack the target code and after some loop this end in 0x003C7DFC:

003C7DF6	59	POP ECX	
003C7DF7	^ E9 94FAFFFF	JMP 003C7890	
003C7DFC	8325 F41E3E00	AND DWORD PTR DS:[3E1EF4],0	
003C7E03	83BD 5CC8FFFF	CMP DWORD PTR SS:[EBP-37A4],0	
003C7E0A	74 33	JE SHORT 003C7E3F	
003C7E0C	8D85 84C4FFFF	LEA EAX,DWORD PTR SS:[EBP-3B7C]	
003C7E12	50	PUSH EAX	
DS:[003E1EF4]=1D6E6CEB			

Fig. 14

Remind also that now we are into the memory allocated area and we have also to find where this area will be written by the packer code, but this analysis is left at the end of this analysis just before to start with the inline patching section.

Now you've to trace a little using F8 until you reach 0x003C81B5 (offset 0x00281B5 from the base address):

003C81B3	EB CA	JMP SHORT 003C817F	
003C81B5	6A 00	PUSH 0	
003C81B7	FF35 087C3D00	PUSH DWORD PTR DS:[3D7C08]	
003C81BD	A1 EC1E3E00	MOV EAX,DWORD PTR DS:[3E1EEC]	
003C81C2	0305 087A3D00	ADD EAX,DWORD PTR DS:[3D7A08]	
003C81C8	50	PUSH EAX	
003C81C9	FFB5 1CC8FFFF	PUSH DWORD PTR SS:[EBP-37E4]	
003C81CF	E8 D892FDFF	CALL 003A14AC	
003C81D4	83C4 10	ADD ESP,10	
003C81D7	A1 840C3E00	MOV EAX,DWORD PTR DS:[3E0C84]	
003C81DC	8985 58C4FFFF	MOV DWORD PTR SS:[EBP-3BA8],EAX	
003C81E2	83BD 58C4FFFF	CMP DWORD PTR SS:[EBP-3BA8],0	
003C81E9	74 36	JE SHORT 003C8221	
003C81EB	8B85 58C4FFFF	MOV EAX,DWORD PTR SS:[EBP-3BA8]	
003C81F1	8338 00	CMP DWORD PTR DS:[EAX],0	
003C81F4	74 2B	JE SHORT 003C8221	
003C81F6	8B85 58C4FFFF	MOV EAX,DWORD PTR SS:[EBP-3BA8]	
003C81FC	8B00	MOV EAX,DWORD PTR DS:[EAX]	
003C81FE	8B00	MOV EAX,DWORD PTR DS:[EAX]	
003C8200	0305 C0233E00	ADD EAX,DWORD PTR DS:[3E23C0]	
003C8206	8B8D 58C4FFFF	MOV ECX,DWORD PTR SS:[EBP-3BA8]	
003C820C	8B09	MOV ECX,DWORD PTR DS:[ECX]	
003C820E	8901	MOV DWORD PTR DS:[ECX],EAX	
003C8210	8B85 58C4FFFF	MOV EAX,DWORD PTR SS:[EBP-3BA8]	
003C8216	83C0 04	ADD EAX,4	
003C8219	8985 58C4FFFF	MOV DWORD PTR SS:[EBP-3BA8],EAX	
003C821F	EB CA	JMP SHORT 003C81EB	
003C8221	93	XCHG EAX,EBX	
003C8222	67:E3 00	JCXZ SHORT 003C8225	
003C8225	93	XCHG EAX,EBX	
003C8226	C8 AAA23D	ENTER 0A2AA,3D	
003C822A	17	POP SS	Modification of segment register
003C822B	BE 7B8C250E	MOV ESI,0E258C7B	
003C8230	0C 50	OR AL,50	
003C8232	79 68	JNS SHORT 003C829C	
003C8234	6E	OUTS DX,BYTE PTR ES:[EDI]	I/O command
003C8235	0C A4	OR AL,0A4	
003C8237	E1 3F	LOOPDE SHORT 003C8278	
003C8239	B9 B186612F	MOV ECX,2F6186B1	
003C823E	A7	CMPS DWORD PTR DS:[ESI],DWORD PTR ES:[EDI]	
003C823F	0C A6	OR AL,0A6	
003C8241	DDFC	FSTSW ESP	Illegal use of register
003C8243	82A5 EBD3BB9C	AND BYTE PTR SS:[EBP+9CBBD3EB],FFFFFFF6	
003C8244	6F	OUTS DX,DWORD PTR ES:[EDI]	I/O command
003C824B	338A C0AE0413	XOR ECX,DWORD PTR DS:[EDX+1304AEC0]	
003C8251	A2 3975B03E	MOV BYTE PTR DS:[3EB07539],AL	
003C8256	8CCC	MOV SP,CS	

Fig. 15

Now look the code below the 0x003C8226 address, this look very strange the we can think that the packer have to take some decryption before execute it. To check this thing step through the code with F8 and when you reach and execute the call at 0x003C81CF the code over 0x003C8226 will be full decrypted.

003C81B3	EB CA	JMP SHORT 003C817F	
003C81B5	6A 00	PUSH 0	
003C81B7	FF35 087C3D00	PUSH DWORD PTR DS:[3D7C08]	
003C81B9	A1 EC1E3E00	MOV EAX,DWORD PTR DS:[3E1EEC]	
003C81C2	0305 087A3D00	ADD EAX,DWORD PTR DS:[3D7A08]	
003C81C8	58	PUSH EAX	
003C81C9	FFB5 1CC8FFFF	PUSH DWORD PTR SS:[EBP-37E4]	
003C81CF	E8 D892FDFF	CALL 003A14AC	Decryption call
003C81D4	83C4 10	ADD ESP,10	
003C81D7	A1 840C3E00	MOV EAX,DWORD PTR DS:[3E0C84]	
003C81DC	8985 58C4FFFF	MOV DWORD PTR SS:[EBP-3BA8],EAX	
003C81E2	83BD 58C4FFFF	CMP DWORD PTR SS:[EBP-3BA8],0	
003C81E9	74 36	JE SHORT 003C8221	
003C81EB	8B85 58C4FFFF	MOV EAX,DWORD PTR SS:[EBP-3BA8]	
003C81F1	8338 00	CMP DWORD PTR DS:[EAX],0	
003C81F4	74 2B	JE SHORT 003C8221	
003C81F6	8B85 58C4FFFF	MOV EAX,DWORD PTR SS:[EBP-3BA8]	
003C81FC	8B00	MOV EAX,DWORD PTR DS:[EAX]	
003C81FE	8B00	MOV EAX,DWORD PTR DS:[EAX]	
003C8200	0305 C0233E00	ADD EAX,DWORD PTR DS:[3E23C0]	
003C8206	8B8D 58C4FFFF	MOV ECX,DWORD PTR SS:[EBP-3BA8]	
003C820C	8B09	MOV ECX,DWORD PTR DS:[ECX]	
003C820E	8901	MOV DWORD PTR DS:[ECX],EAX	
003C8210	8B85 58C4FFFF	MOV EAX,DWORD PTR SS:[EBP-3BA8]	
003C8216	83C0 04	ADD EAX,4	
003C8219	8985 58C4FFFF	MOV DWORD PTR SS:[EBP-3BA8],EAX	
003C821F	EB CA	JMP SHORT 003C81EB	
003C8221	93	XCHG EAX,EBX	
003C8222	67:E3 00	JCXZ SHORT 003C8225	
003C8225	93	XCHG EAX,EBX	
003C8226	A1 F81E0410	MOV EAX,DWORD PTR DS:[10041EF8]	Decrypted code
003C822B	8B8D F81E0410	MOV ECX,DWORD PTR DS:[10041EF8]	
003C8231	8B40 5C	MOV EAX,DWORD PTR DS:[EAX+5C]	
003C8234	3341 64	XOR EAX,DWORD PTR DS:[ECX+64]	
003C8237	8B8D F81E0410	MOV ECX,DWORD PTR DS:[10041EF8]	
003C823D	3341 68	XOR EAX,DWORD PTR DS:[ECX+68]	
003C8240	25 80000000	AND EAX,80	
003C8245	85C0	TEST EAX,EAX	
003C8247	74 47	JE SHORT 003C8290	
003C8249	A1 F81E0410	MOV EAX,DWORD PTR DS:[10041EF8]	
003C824E	8B8D F81E0410	MOV ECX,DWORD PTR DS:[10041EF8]	
003C8254	8B40 5C	MOV EAX,DWORD PTR DS:[EAX+5C]	
003C8257	3341 64	XOR EAX,DWORD PTR DS:[ECX+64]	
003C825A	8B8D F81E0410	MOV ECX,DWORD PTR DS:[10041EF8]	
003C8260	3341 68	XOR EAX,DWORD PTR DS:[ECX+68]	
003C8263	83E0 10	AND EAX,10	
003C8266	85C0	TEST EAX,EAX	
003C8268	74 26	JE SHORT 003C8290	
003C826A	C785 C4C3FFFF	MOV DWORD PTR SS:[EBP-3C3C],94	

Fig. 16

But if you step again same code is rewritten and the cycle end into the 0x003C8221 address then we can place a memory breakpoint on it and simply press F9.

The decrypted code start from 0x003C8221 or in other word at one address which have a offset of **0x0028221** from the base address (absolute 0x00F88221 if the base address will be 0x00F60000).

003C81B3	EB CA	JMP SHORT 003C817F	
003C81B5	6A 00	PUSH 0	
003C81B7	FF35 007C3D00	PUSH DWORD PTR DS:[3D7C00]	
003C81B9	A1 EC1E3E00	MOV EAX,DWORD PTR DS:[3E1EEC]	
003C81C2	0305 007A3D00	ADD EAX,DWORD PTR DS:[3D7A00]	
003C81C8	5B	PUSH EAX	
003C81C9	FFB5 1CC8FFFF	PUSH DWORD PTR SS:[EBP-37E4]	
003C81CF	E8 D892FDFF	CALL 003A14AC	First decryption loop
003C81D4	83C4 10	ADD ESP,10	
003C81D7	A1 040C3E00	MOV EAX,DWORD PTR DS:[3E0C84]	
003C81DC	8985 58C4FFFF	MOV DWORD PTR SS:[EBP-3BA8],EAX	
003C81E2	838D 58C4FFFF	CMP DWORD PTR SS:[EBP-3BA8],0	
003C81E9	74 36	JE SHORT 003C8221	
003C81EB	8B85 58C4FFFF	MOV EAX,DWORD PTR SS:[EBP-3BA8]	
003C81F1	8338 00	CMP DWORD PTR DS:[EAX],0	
003C81F4	74 2B	JE SHORT 003C8221	
003C81F6	8B85 58C4FFFF	MOV EAX,DWORD PTR SS:[EBP-3BA8]	
003C81FC	8B00	MOV EAX,DWORD PTR DS:[EAX]	
003C81FE	8B00	MOV EAX,DWORD PTR DS:[EAX]	
003C8200	0305 C0233E00	ADD EAX,DWORD PTR DS:[3E23C0]	
003C8206	8B8D 58C4FFFF	MOV ECX,DWORD PTR SS:[EBP-3BA8]	
003C820C	8B09	MOV ECX,DWORD PTR DS:[ECX]	
003C820E	8901	MOV DWORD PTR DS:[ECX],EAX	
003C8210	8B85 58C4FFFF	MOV EAX,DWORD PTR SS:[EBP-3BA8]	
003C8216	83C0 04	ADD EAX,4	
003C8219	8985 58C4FFFF	MOV DWORD PTR SS:[EBP-3BA8],EAX	
003C821F	EB CA	JMP SHORT 003C81EB	
003C8221	93	XCHG EAX,EBX	
003C8222	67:E3 00	JCXZ SHORT 003C8225	
003C8225	93	XCHG EAX,EBX	
003C8226	A1 F81E3E00	MOV EAX,DWORD PTR DS:[3E1EF8]	
003C822B	8B0D F81E3E00	MOV ECX,DWORD PTR DS:[3E1EF8]	swdoctor.00580370
003C8231	8B40 5C	MOV EAX,DWORD PTR DS:[EAX+5C]	
003C8234	3341 64	XOR EAX,DWORD PTR DS:[ECX+64]	
003C8237	8B0D F81E3E00	MOV ECX,DWORD PTR DS:[3E1EF8]	swdoctor.00580370
003C823D	3341 68	XOR EAX,DWORD PTR DS:[ECX+68]	
003C8240	25 00000000	AND EAX,00	
003C8245	85C0	TEST EAX,EAX	
003C8247	74 47	JE SHORT 003C8290	
003C8249	A1 F81E3E00	MOV EAX,DWORD PTR DS:[3E1EF8]	
003C824E	8B0D F81E3E00	MOV ECX,DWORD PTR DS:[3E1EF8]	swdoctor.00580370
003C8254	8B40 5C	MOV EAX,DWORD PTR DS:[EAX+5C]	
003C8257	3341 64	XOR EAX,DWORD PTR DS:[ECX+64]	
003C825A	8B0D F81E3E00	MOV ECX,DWORD PTR DS:[3E1EF8]	swdoctor.00580370
003C8260	3341 68	XOR EAX,DWORD PTR DS:[ECX+68]	
003C8263	83E0 10	AND EAX,10	
003C8266	85C0	TEST EAX,EAX	
003C8268	74 26	JE SHORT 003C8290	
003C826A	C785 C4C3FFFF	MOV DWORD PTR SS:[EBP-3C3C],94	
003C8274	0D85 C4C3FFFF	LEA EAX,DWORD PTR SS:[EBP-3C3C]	
003C827A	5B	PUSH EAX	
003C827B	FF15 CC203D00	CALL DWORD PTR DS:[3D20CC]	kernel32.GetVersionExA
003C8281	83BD D4C3FFFF	CMP DWORD PTR SS:[EBP-3C2C],2	
003C8288	74 06	JE SHORT 003C8290	

Fig. 17

Now step again a little bit until you reach the address 0x003C872D, another time the code after 0x003C879A looks strange then may be another decryption loop will be start:

003C872B	EB CA	JMP SHORT 003C86F7	
003C872D	6A 00	PUSH 0	
003C872F	FF35 107C3D00	PUSH DWORD PTR DS:[3D7C10]	
003C8735	A1 EC1E3E00	MOV EAX,DWORD PTR DS:[3E1EEC]	
003C873A	0305 107A3D00	ADD EAX,DWORD PTR DS:[3D7A10]	
003C8740	5B	PUSH EAX	
003C8741	FFB5 38C8FFFF	PUSH DWORD PTR SS:[EBP-37C8]	
003C8747	E8 608DFDFF	CALL 003A14AC	Decryption loop
003C874C	83C4 10	ADD ESP,10	
003C874F	A1 8C0C3E00	MOV EAX,DWORD PTR DS:[3E0C8C]	
003C8754	8985 94C3FFFF	MOV DWORD PTR SS:[EBP-3C6C],EAX	
003C875A	83BD 94C3FFFF	CMP DWORD PTR SS:[EBP-3C6C],0	
003C8761	74 36	JE SHORT 003C8799	
003C8763	8B85 94C3FFFF	MOV EAX,DWORD PTR SS:[EBP-3C6C]	
003C8769	8338 00	CMP DWORD PTR DS:[EAX],0	
003C876C	74 2B	JE SHORT 003C8799	
003C876E	8B85 94C3FFFF	MOV EAX,DWORD PTR SS:[EBP-3C6C]	
003C8774	8B00	MOV EAX,DWORD PTR DS:[EAX]	
003C8776	8B00	MOV EAX,DWORD PTR DS:[EAX]	
003C8778	0305 C0233E00	ADD EAX,DWORD PTR DS:[3E23C0]	
003C877E	8B8D 94C3FFFF	MOV ECX,DWORD PTR SS:[EBP-3C6C]	
003C8784	8B09	MOV ECX,DWORD PTR DS:[ECX]	
003C8786	8901	MOV DWORD PTR DS:[ECX],EAX	
003C8788	8B85 94C3FFFF	MOV EAX,DWORD PTR SS:[EBP-3C6C]	
003C878E	83C0 04	ADD EAX,4	
003C8791	8985 94C3FFFF	MOV DWORD PTR SS:[EBP-3C6C],EAX	
003C8797	EB CA	JMP SHORT 003C8763	
003C8799	93	XCHG EAX,EBX	
003C879A	56	PUSH ESI	
003C879B	90	NOP	
003C879C	5E	POP ESI	
003C879D	93	XCHG EAX,EBX	
003C879E	C599 1C4CA8A1	LDS EBX,FWORD PTR DS:[ECX+A1A84C]	Modification of segment register
003C87A4	C09E 255B8276	ROR BYTE PTR DS:[ESI+76825B25],0F7	Shift constant out of range 1..31
003C87AB	3E:8AD5	MOV DL,CH	Superfluous prefix
003C87AE	D4 65	AAA 65	
003C87B0	FD	STD	
003C87B1	55	PUSH EBP	
003C87B2	CC	INT3	
003C87B3	EB	RET	

Fig. 18

Step until you reach the 0x003C8747 address, in this place there is another decryption call:

003C872B	EB CA	JMP SHORT 003C86F7	
003C872D	6A 00	PUSH 0	
003C872F	FF35 107C3D00	PUSH DWORD PTR DS:[3D7C10]	
003C8735	A1 EC1E3E00	MOV EAX,DWORD PTR DS:[3E1EEC]	
003C873A	0305 107A3D00	ADD EAX,DWORD PTR DS:[3D7A10]	
003C8740	50	PUSH EAX	
003C8741	FFB5 38C8FFFF	PUSH DWORD PTR SS:[EBP-37C8]	
003C8747	E8 608DFDFF	CALL 003A14AC	Decryption loop
003C874C	83C4 10	ADD ESP,10	
003C874F	A1 8C0C3E00	MOV EAX,DWORD PTR DS:[3E0C8C]	
003C8754	8985 94C3FFFF	MOV DWORD PTR SS:[EBP-3C6C],EAX	
003C875A	83BD 94C3FFFF	CMP DWORD PTR SS:[EBP-3C6C],0	
003C8761	74 36	JE SHORT 003C8799	
003C8763	8B85 94C3FFFF	MOV EAX,DWORD PTR SS:[EBP-3C6C]	
003C8769	8338 00	CMP DWORD PTR DS:[EAX],0	
003C876C	74 2B	JE SHORT 003C8799	
003C876E	8B85 94C3FFFF	MOV EAX,DWORD PTR SS:[EBP-3C6C]	
003C8774	8B00	MOV EAX,DWORD PTR DS:[EAX]	
003C8776	8B00	MOV EAX,DWORD PTR DS:[EAX]	
003C8778	0305 C0233E00	ADD EAX,DWORD PTR DS:[3E23C0]	
003C877E	8B8D 94C3FFFF	MOV ECX,DWORD PTR SS:[EBP-3C6C]	
003C8784	8B09	MOV ECX,DWORD PTR DS:[ECX]	
003C8786	8901	MOV DWORD PTR DS:[ECX],EAX	
003C8788	8B85 94C3FFFF	MOV EAX,DWORD PTR SS:[EBP-3C6C]	
003C878E	83C0 04	ADD EAX,4	
003C8791	8985 94C3FFFF	MOV DWORD PTR SS:[EBP-3C6C],EAX	
003C8797	EB CA	JMP SHORT 003C8763	
003C8799	93	XCHG EAX,EBX	
003C879A	56	PUSH ESI	
003C879B	90	NOP	
003C879C	5E	POP ESI	
003C879D	93	XCHG EAX,EBX	
003C879E	C705 E0790310	MOV DWORD PTR DS:[100379E0],100385A4	
003C87A8	00A5 20C8FFFF	AND BYTE PTR SS:[EBP-37E0],0	
003C87AF	A1 F81E0410	MOV EAX,DWORD PTR DS:[10041EF8]	
003C87B4	8B40 40	MOV EAX,DWORD PTR DS:[EAX+40]	
003C87B7	8985 C89CFFFF	MOV DWORD PTR SS:[EBP+FFFF9CC8],EAX	
003C87BD	8B85 C89CFFFF	MOV EAX,DWORD PTR SS:[EBP+FFFF9CC8]	
003C87C3	8985 24C8FFFF	MOV DWORD PTR SS:[EBP-37DC],EAX	
003C87C9	A1 F81E0410	MOV EAX,DWORD PTR DS:[10041EF8]	
003C87CE	8B40 64	MOV EAX,DWORD PTR DS:[EAX+64]	
003C87D1	6BC0 0A	IMUL EAX,EAX,0A	

Fig. 19

And this one is completed when execution reach 0x003C8799 / 0x00F88799 (offset **0x0028799**):

003C8725	8985 98C3FFFF	MOV DWORD PTR SS:[EBP-3C60],EAX	
003C872B	EB CA	JMP SHORT 003C86F7	
003C872D	6A 00	PUSH 0	
003C872F	FF35 107C3D00	PUSH DWORD PTR DS:[3D7C10]	
003C8735	A1 EC1E3E00	MOV EAX,DWORD PTR DS:[3E1EEC]	
003C873A	0305 107A3D00	ADD EAX,DWORD PTR DS:[3D7A10]	
003C8740	50	PUSH EAX	
003C8741	FFB5 38C8FFFF	PUSH DWORD PTR SS:[EBP-37C8]	
003C8747	E8 608DFDFF	CALL 003A14AC	Decryption loop
003C874C	83C4 10	ADD ESP,10	
003C874F	A1 8C0C3E00	MOV EAX,DWORD PTR DS:[3E0C8C]	
003C8754	8985 94C3FFFF	MOV DWORD PTR SS:[EBP-3C6C],EAX	
003C875A	83BD 94C3FFFF	CMP DWORD PTR SS:[EBP-3C6C],0	
003C8761	74 36	JE SHORT 003C8799	
003C8763	8B85 94C3FFFF	MOV EAX,DWORD PTR SS:[EBP-3C6C]	
003C8769	8338 00	CMP DWORD PTR DS:[EAX],0	
003C876C	74 2B	JE SHORT 003C8799	
003C876E	8B85 94C3FFFF	MOV EAX,DWORD PTR SS:[EBP-3C6C]	
003C8774	8B00	MOV EAX,DWORD PTR DS:[EAX]	
003C8776	8B00	MOV EAX,DWORD PTR DS:[EAX]	
003C8778	0305 C0233E00	ADD EAX,DWORD PTR DS:[3E23C0]	
003C877E	8B8D 94C3FFFF	MOV ECX,DWORD PTR SS:[EBP-3C6C]	
003C8784	8B09	MOV ECX,DWORD PTR DS:[ECX]	
003C8786	8901	MOV DWORD PTR DS:[ECX],EAX	
003C8788	8B85 94C3FFFF	MOV EAX,DWORD PTR SS:[EBP-3C6C]	
003C878E	83C0 04	ADD EAX,4	
003C8791	8985 94C3FFFF	MOV DWORD PTR SS:[EBP-3C6C],EAX	
003C8797	EB CA	JMP SHORT 003C8763	
003C8799	93	XCHG EAX,EBX	
003C879A	56	PUSH ESI	
003C879B	90	NOP	
003C879C	5E	POP ESI	
003C879D	93	XCHG EAX,EBX	
003C879E	C705 E0793D00	MOV DWORD PTR DS:[3D79E0],3D85A4	
003C87A8	00A5 20C8FFFF	AND BYTE PTR SS:[EBP-37E0],0	
003C87AF	A1 F81E3E00	MOV EAX,DWORD PTR DS:[3E1EF8]	
003C87B4	8B40 40	MOV EAX,DWORD PTR DS:[EAX+40]	
003C87B7	8985 C89CFFFF	MOV DWORD PTR SS:[EBP+FFFF9CC8],EAX	
003C87BD	8B85 C89CFFFF	MOV EAX,DWORD PTR SS:[EBP+FFFF9CC8]	
003C87C3	8985 24C8FFFF	MOV DWORD PTR SS:[EBP-37DC],EAX	
003C87C9	A1 F81E3E00	MOV EAX,DWORD PTR DS:[3E1EF8]	
003C87CE	8B40 64	MOV EAX,DWORD PTR DS:[EAX+64]	
003C87D1	6BC0 0A	IMUL EAX,EAX,0A	ASCII "E2"

Fig. 20

Now step again a little bit using F8 until you reach the address 0x003C8C91, from this point we have the IAT destruction cycle from Armadillo, in this place you can place the magic jump

which is able to recover the major part of the IAT entry, used during the manual unpacking approach [1], now scroll down into the code and skip the antidebug control (GetThickCount API calling) by change the JBE instruction into a unconditional jump instruction (JMP) at address 0x003C9061.

Now you can place a memory breakpoint in 0x003C9125 / 0x00F89125 (offset **0x0029125**) and press F9 in order to skip all the IAT scrambling cycle.

Step again through the code by using F8 until you reach the address 0x003C957A, there is another decryption loop but Armadillo code is able to detect our JBE - JMP trick used to skip the GetThickCount API call and the subsequent decryption isn't correct anyway, to continue the code analysis simply place an hardware breakpoint in 0x003C957A and restart the target by using the CTRL+F2 keys. Run the target until OllyDbg break into our breakpoint:

003C9578	EB CA	JMP SHORT 003C9544	
003C957A	6A 00	PUSH 0	
003C957C	FF35 147C3D00	PUSH DWORD PTR DS:[3D7C14]	
003C9582	A1 EC1E3E00	MOV EAX,DWORD PTR DS:[3E1EEC]	
003C9587	0305 147A3D00	ADD EAX,DWORD PTR DS:[3D7A14]	
003C958D	50	PUSH EAX	
003C958E	FFB5 E0C6FFFF	PUSH DWORD PTR SS:[EBP-3920]	
003C9594	E8 137FFDFF	CALL 003A14AC	Decryption call
003C9599	83C4 10	ADD ESP,10	
003C959C	A1 900C3E00	MOV EAX,DWORD PTR DS:[3E0C90]	
003C95A1	8985 00B0FFFF	MOV DWORD PTR SS:[EBP+FFFFB000],EAX	
003C95A7	83BD 00B0FFFF	CMP DWORD PTR SS:[EBP+FFFFB000],0	
003C95AE	74 36	JE SHORT 003C95E6	
003C95B0	8B85 00B0FFFF	MOV EAX,DWORD PTR SS:[EBP+FFFFB000]	
003C95B6	8338 00	CMP DWORD PTR DS:[EAX],0	
003C95B9	74 2B	JE SHORT 003C95E6	
003C95BB	8B85 00B0FFFF	MOV EAX,DWORD PTR SS:[EBP+FFFFB000]	
003C95C1	8B00	MOV EAX,DWORD PTR DS:[EAX]	
003C95C3	8B00	MOV EAX,DWORD PTR DS:[EAX]	
003C95C5	0305 C0233E00	ADD EAX,DWORD PTR DS:[3E23C0]	
003C95CB	8B8D 00B0FFFF	MOV ECX,DWORD PTR SS:[EBP+FFFFB000]	
003C95D1	8B07	MOV ECX,DWORD PTR DS:[ECX]	
003C95D3	8901	MOV DWORD PTR DS:[ECX],EAX	
003C95D5	8B85 00B0FFFF	MOV EAX,DWORD PTR SS:[EBP+FFFFB000]	
003C95DB	83C0 04	ADD EAX,4	
003C95DE	8985 00B0FFFF	MOV DWORD PTR SS:[EBP+FFFFB000],EAX	
003C95E4	EB CA	JMP SHORT 003C95B0	
003C95E6	56	PUSH ESI	
003C95E7	67:E3 00	JCXZ SHORT 003C95EA	
003C95EA	5E	POP ESI	
003C95EB	15 50F5FD63	ADC EAX,63FDF550	
003C95F0	9C	PUSHFD	
003C95F1	EC	IN AL,DX	I/O command
003C95F2	C2 F571	RETN 71F5	
003C95F5	5F	POP EDI	
003C95F6	AA	STOS BYTE PTR ES:[EDI]	
003C95F7	FA	CLI	
...	

Fig. 21

Place a memory breakpoint on 0x003C95E6 and press F9 in order to run all the cycle:

003C9572	8985 04B0FFFF	MOV DWORD PTR SS:[EBP+FFFFB004],EAX	
003C9578	EB CA	JMP SHORT 003C9544	
003C957A	6A 00	PUSH 0	
003C957C	FF35 147C3D00	PUSH DWORD PTR DS:[3D7C14]	
003C9582	A1 EC1E3E00	MOV EAX,DWORD PTR DS:[3E1EEC]	
003C9587	0305 147A3D00	ADD EAX,DWORD PTR DS:[3D7A14]	
003C958D	50	PUSH EAX	
003C958E	FFB5 E0C6FFFF	PUSH DWORD PTR SS:[EBP-3920]	
003C9594	E8 137FFDFF	CALL 003A14AC	Decryption call
003C9599	83C4 10	ADD ESP,10	
003C959C	A1 900C3E00	MOV EAX,DWORD PTR DS:[3E0C90]	
003C95A1	8985 00B0FFFF	MOV DWORD PTR SS:[EBP+FFFFB000],EAX	
003C95A7	83BD 00B0FFFF	CMP DWORD PTR SS:[EBP+FFFFB000],0	
003C95AE	74 36	JE SHORT 003C95E6	
003C95B0	8B85 00B0FFFF	MOV EAX,DWORD PTR SS:[EBP+FFFFB000]	
003C95B6	8338 00	CMP DWORD PTR DS:[EAX],0	
003C95B9	74 2B	JE SHORT 003C95E6	
003C95BB	8B85 00B0FFFF	MOV EAX,DWORD PTR SS:[EBP+FFFFB000]	
003C95C1	8B00	MOV EAX,DWORD PTR DS:[EAX]	
003C95C3	8B00	MOV EAX,DWORD PTR DS:[EAX]	
003C95C5	0305 C0233E00	ADD EAX,DWORD PTR DS:[3E23C0]	
003C95CB	8B8D 00B0FFFF	MOV ECX,DWORD PTR SS:[EBP+FFFFB000]	
003C95D1	8B07	MOV ECX,DWORD PTR DS:[ECX]	
003C95D3	8901	MOV DWORD PTR DS:[ECX],EAX	
003C95D5	8B85 00B0FFFF	MOV EAX,DWORD PTR SS:[EBP+FFFFB000]	
003C95DB	83C0 04	ADD EAX,4	
003C95DE	8985 00B0FFFF	MOV DWORD PTR SS:[EBP+FFFFB000],EAX	
003C95E4	EB CA	JMP SHORT 003C95B0	
003C95E6	56	PUSH ESI	
003C95E7	67:E3 00	JCXZ SHORT 003C95EA	
003C95EA	5E	POP ESI	
003C95EB	C705 E0793D00	MOV DWORD PTR DS:[3D79E0],3D8544	
003C95F5	A1 D0233E00	MOV EAX,DWORD PTR DS:[3E23D0]	
003C95FA	8B80 F03D0000	MOV EAX,DWORD PTR DS:[EAX+3DF0]	
003C9600	8985 00C8FFFF	MOV DWORD PTR SS:[EBP-3800],EAX	

Fig. 22

Remove the memory breakpoint and step again with F8 until you reach in 0x003C9B1E another decrypting loop:

003C9B1C	EB CA	JMP SHORT 003C9AE8	
003C9B1E	6A 00	PUSH 0	
003C9B20	FF35 187C3D00	PUSH DWORD PTR DS:[3D7C18]	
003C9B26	A1 EC1E3E00	MOV EAX,DWORD PTR DS:[3E1EEC]	
003C9B2B	0305 187A3D00	ADD EAX,DWORD PTR DS:[3D7A18]	
003C9B31	50	PUSH EAX	
003C9B32	FFB5 F0C6FFFF	PUSH DWORD PTR SS:[EBP-3910]	
003C9B38	E8 6F79FDFF	CALL 003A14AC	Decryption call
003C9B3D	83C4 10	ADD ESP,10	
003C9B40	A1 940C3E00	MOV EAX,DWORD PTR DS:[3E0C94]	
003C9B45	8985 B89FFFFF	MOV DWORD PTR SS:[EBP+FFFF9FB8],EAX	
003C9B4B	83BD B89FFFFF	CMP DWORD PTR SS:[EBP+FFFF9FB8],0	
003C9B52	74 36	JE SHORT 003C9B8A	
003C9B54	8B85 B89FFFFF	MOV EAX,DWORD PTR SS:[EBP+FFFF9FB8]	
003C9B5A	8338 00	CMP DWORD PTR DS:[EAX],0	
003C9B5D	74 2B	JE SHORT 003C9B8A	
003C9B5F	8B85 B89FFFFF	MOV EAX,DWORD PTR SS:[EBP+FFFF9FB8]	
003C9B65	8B00	MOV EAX,DWORD PTR DS:[EAX]	
003C9B67	8B00	MOV EAX,DWORD PTR DS:[EAX]	
003C9B69	0305 C0233E00	ADD EAX,DWORD PTR DS:[3E23C0]	
003C9B6F	8B8D B89FFFFF	MOV ECX,DWORD PTR SS:[EBP+FFFF9FB8]	
003C9B75	8B09	MOV ECX,DWORD PTR DS:[ECX]	
003C9B77	8901	MOV DWORD PTR DS:[ECX],EAX	
003C9B79	8B85 B89FFFFF	MOV EAX,DWORD PTR SS:[EBP+FFFF9FB8]	
003C9B7F	83C0 04	ADD EAX,4	
003C9B82	8985 B89FFFFF	MOV DWORD PTR SS:[EBP+FFFF9FB8],EAX	
003C9B88	EB CA	JMP SHORT 003C9B54	
003C9B8A	87F3	XCHG EBX,ESI	
003C9B8C	90	NOP	
003C9B8D	87F3	XCHG EBX,ESI	
003C9B8F	81CE B8E29862	OR ESI,6298E2B8	
003C9B95	7E C6	JLE SHORT 003C9B5D	
003C9B97	3B32	CMP ESI,DWORD PTR DS:[EDX]	

Fig. 23

to skip also this loop place a memory breakpoint in 0x003C9B8A and press F9:

003C9B16	8985 BC9FFFFF	MOV DWORD PTR SS:[EBP+FFFF9FBC],EAX	
003C9B1C	EB CA	JMP SHORT 003C9AE8	
003C9B1E	6A 00	PUSH 0	
003C9B20	FF35 187C3D00	PUSH DWORD PTR DS:[3D7C18]	
003C9B26	A1 EC1E3E00	MOV EAX,DWORD PTR DS:[3E1EEC]	
003C9B2B	0305 187A3D00	ADD EAX,DWORD PTR DS:[3D7A18]	
003C9B31	50	PUSH EAX	
003C9B32	FFB5 F0C6FFFF	PUSH DWORD PTR SS:[EBP-3910]	
003C9B38	E8 6F79FDFF	CALL 003A14AC	Decryption call
003C9B3D	83C4 10	ADD ESP,10	
003C9B40	A1 940C3E00	MOV EAX,DWORD PTR DS:[3E0C94]	
003C9B45	8985 B89FFFFF	MOV DWORD PTR SS:[EBP+FFFF9FB8],EAX	
003C9B4B	83BD B89FFFFF	CMP DWORD PTR SS:[EBP+FFFF9FB8],0	
003C9B52	74 36	JE SHORT 003C9B8A	
003C9B54	8B85 B89FFFFF	MOV EAX,DWORD PTR SS:[EBP+FFFF9FB8]	
003C9B5A	8338 00	CMP DWORD PTR DS:[EAX],0	
003C9B5D	74 2B	JE SHORT 003C9B8A	
003C9B5F	8B85 B89FFFFF	MOV EAX,DWORD PTR SS:[EBP+FFFF9FB8]	
003C9B65	8B00	MOV EAX,DWORD PTR DS:[EAX]	
003C9B67	8B00	MOV EAX,DWORD PTR DS:[EAX]	
003C9B69	0305 C0233E00	ADD EAX,DWORD PTR DS:[3E23C0]	
003C9B6F	8B8D B89FFFFF	MOV ECX,DWORD PTR SS:[EBP+FFFF9FB8]	
003C9B75	8B09	MOV ECX,DWORD PTR DS:[ECX]	
003C9B77	8901	MOV DWORD PTR DS:[ECX],EAX	
003C9B79	8B85 B89FFFFF	MOV EAX,DWORD PTR SS:[EBP+FFFF9FB8]	
003C9B7F	83C0 04	ADD EAX,4	
003C9B82	8985 B89FFFFF	MOV DWORD PTR SS:[EBP+FFFF9FB8],EAX	
003C9B88	EB CA	JMP SHORT 003C9B54	
003C9B8A	87F3	XCHG EBX,ESI	
003C9B8C	90	NOP	
003C9B8D	87F3	XCHG EBX,ESI	
003C9B8F	C705 E0793D00	MOV DWORD PTR DS:[3D79E0],3D8540	
003C9B95	A1 FC1E3E00	MOV EAX,DWORD PTR DS:[3E1EEC]	
003C9B9E	8B00	MOV EAX,DWORD PTR DS:[EAX]	
003C9BA0	8985 28C8FFFF	MOV DWORD PTR SS:[EBP-37D8],EAX	

Fig. 24

Remove the memory breakpoint and start again with F8 after some step we are into some scrambled code, step until you reach the 0x003CA0AB address, we have a WriteProcessMemory API call:

003CA0A0	61	POPAD	
003CA0A3	FF35 6C223D00	PUSH DWORD PTR DS:[3D226C]	kernel32.WriteProcessMemory
003CA0A8	E8 10240000	CALL 003CC4C6	
003CA0B6	59	POP ECX	
003CA0B7	8B85 48C8FFFF	MOV BYTE PTR SS:[EBP-37B8],AL	
003CA0BD	50	PUSH EAX	
003CA0BE	F7D0	NOT EAX	
003CA0C0	0FC8	BSWAP EAX	
003CA0C2	58	POP EAX	

Fig. 25

Skip another loop by placing a memory breakpoint in 0x003CA15C and press F9 the target will be start.

Now is time to found a good point where we can apply our patching, first thing may be after the IAT rebuilding, now the target is unpacked, last work is about possible CRC checking, this will be cover later.

Well, press CTRL+F2 to restart the target and place one hardware breakpoint in 0x003C9125 / 0x00F89125 (offset **0x0029125** from the base address), this place is just after the IAT writing. Run the target with SHIFT+F9 and when OllyDbg break press F8 once in order to skip the jump, this place may be good to take redirection to our final patching cave. When OllyDbg break we can take this offset (**0x0029284**) to apply your final patch (target patching):

```

00F89284 FFB5 34C8FFFF PUSH DWORD PTR SS:[EBP-37CC]
00F8928A E8 5580FDFF CALL 00F612E4
00F8928F 59 POP ECX
00F89290 EB 03 JMP SHORT 00F89295
00F89292 D6 SALC
00F89293 D6 SALC
00F89294 8BA1 8C0CFA00 MOV ESP,DWORD PTR DS:[ECX+FA0C8C]
00F8929A 8985 28B0FFFF MOV DWORD PTR SS:[EBP+FFFFB028],EAX
00F892A0 83BD 28B0FFFF CMP DWORD PTR SS:[EBP+FFFFB028],0
00F892A7 74 36 JF SHORT 00F8929F

```

Fig. 26

Address	Patching
50301F	MOV AL,1
4F2CA7	change to JMP
44C698	change to JMP

Well, first break must be in 0x00546913 to capture the base address for the allocated memory area but now, if you remember, we have also to find the point where the allocated memory area will be written, to do this, when you're in 0x00546913 press ALT+M and place a break on access into the new allocated area then press F9. OllyDbg will break in 0x0055A3A3:

```

0055A3A1 72 29 JB SHORT swdoctor.0055A3CC
0055A3A3 F3:A5 REP MOVSD WORD PTR ES:[EDI],DWORD PTR DS:[ESI]
0055A3A5 FF2495 B8A45 JMP DWORD PTR DS:[EDX*4+55A4B8]
0055A3AC 8BC7 MOV EAX,EDI
0055A3AE BA 03000000 MOV EDX,3
0055A3B3 83E9 04 SUB ECX,4
0055A3B6 72 0C JB SHORT swdoctor.0055A3C4

```

Fig. 27

Now look into the stack:

```

0012D6BC 00000000
0012D6C0 0012FF2C
0012D6C4 0012D74C
0012D6C8 00546941 RETURN to swdoctor.00546941 from swdoctor.0055A370
0012D6CC 00F60000
0012D6D0 00E60048
0012D6D4 00001000

```

Fig. 28

go to the address 0x00546941, place a memory breakpoint and press F9:

005468FE	. F7D1	NOT ECX	
00546900	. 6A 04	PUSH 4	
00546902	. 68 00100000	PUSH 1000	
00546907	. 8B55 F0	MOV EDX,DWORD PTR SS:[EBP-10]	
0054690A	. 52	PUSH EDX	
0054690E	. 6A 00	PUSH 0	
00546910	. FF15 6C005800	CALL DWORD PTR DS:[<%KERNEL32.VirtualAlloc>]	Protect = PAGE_READWRITE AllocationType = MEM_COMMIT
00546913	. 8945 E0	MOV DWORD PTR SS:[EBP-20],EAX	Size Address = NULL
00546916	. 837D E0 00	CMP DWORD PTR SS:[EBP-20],0	VirtualAlloc EAX = base address
0054691A	. 75 11	JNZ SHORT swdoctor.0054692D	
0054691C	. C705 6C6C5800	MOV DWORD PTR DS:[586C6C],2	
00546926	. 33C0	XOR EAX,EAX	
00546928	. E9 F2020000	JMP swdoctor.00546C1F	
0054692D	. 8B45 D8	MOV EAX,DWORD PTR SS:[EBP-28]	
00546930	. 8B48 3C	MOV ECX,DWORD PTR DS:[EAX+3C]	
00546933	. 51	PUSH ECX	
00546934	. 8B55 D0	MOV EDX,DWORD PTR SS:[EBP-30]	
00546937	. 52	PUSH EDX	
00546938	. 8B45 E0	MOV EAX,DWORD PTR SS:[EBP-20]	
0054693B	. 50	PUSH EAX	
0054693C	. E8 2F3A0100	CALL swdoctor.0055A370	Write into the allocated memory area
00546941	. 83C4 0C	ADD ESP,0C	
00546944	. 70 07	JO SHORT swdoctor.0054694D	
00546946	. 7C 03	JL SHORT swdoctor.0054694B	
00546948	. EB 05	JMP SHORT swdoctor.0054694F	
0054694A	. E8	DB E8	
0054694B	. 74 FB	JE SHORT swdoctor.00546948	
0054694D	. EB F9	JMP SHORT swdoctor.00546948	
0054694F	. 8B4D E8	MOV ECX,DWORD PTR SS:[EBP-18]	

Fig. 29

Now we can check if the code about the first layer is unpacked, press CTRL+G and write a address equal to the base_address+0x00281B5.

Code isn't unpacked then we've to step again with F8 until you're in 0x005469B2:

00546996	? 61	POPAD	
00546997	? 8B45 F4	MOV EAX,DWORD PTR SS:[EBP-C]	
0054699A	. 8B48 10	MOV ECX,DWORD PTR DS:[EAX+10]	
0054699D	. 51	PUSH ECX	
0054699E	. 8B55 F4	MOV EDX,DWORD PTR SS:[EBP-C]	
005469A1	. 8B45 D0	MOV EAX,DWORD PTR SS:[EBP-30]	
005469A4	. 0342 14	ADD EAX,DWORD PTR DS:[EDX+14]	
005469A7	. 50	PUSH EAX	
005469A8	. 8B4D F4	MOV ECX,DWORD PTR SS:[EBP-C]	
005469AB	. 8B55 E0	MOV EDX,DWORD PTR SS:[EBP-20]	
005469AE	. 0351 0C	ADD EDX,DWORD PTR DS:[ECX+C]	
005469B1	. 52	PUSH EDX	
005469B2	. E8 B9390100	CALL swdoctor.0055A370	
005469B7	. 83C4 0C	ADD ESP,0C	
005469BA	. 8B45 F4	MOV EAX,DWORD PTR SS:[EBP-C]	
005469BD	. 83C0 28	ADD EAX,28	
005469C0	. 8945 F4	MOV DWORD PTR SS:[EBP-C],EAX	
005469C3	. EB A4	JMP SHORT swdoctor.00546969	
005469C5	. 8B4D E0	MOV ECX,DWORD PTR SS:[EBP-20]	
005469C8	. 3B4D DC	CMP ECX,DWORD PTR SS:[EBP-24]	
005469CB	. 0F84 E9000000	JE swdoctor.00546ABA	
005469D1	. 50	PUSH EAX	

Fig. 30

after this call the code about the first layer will be fully decrypted.

Place a memory breakpoint (F2) at 0x005469C5 and press F9 we have some interesting thinks:

005469AE	. 0351 0C	ADD EDX,DWORD PTR DS:[ECX+C]	
005469B1	. 52	PUSH EDX	
005469B2	. E8 B9390100	CALL swdoctor.0055A370	
005469B7	. 83C4 0C	ADD ESP,0C	
005469BA	. 8B45 F4	MOV EAX,DWORD PTR SS:[EBP-C]	
005469BD	. 83C0 28	ADD EAX,28	
005469C0	. 8945 F4	MOV DWORD PTR SS:[EBP-C],EAX	
005469C3	. EB A4	JMP SHORT swdoctor.00546969	
005469C5	. 8B4D E0	MOV ECX,DWORD PTR SS:[EBP-20]	
005469C8	. 3B4D DC	CMP ECX,DWORD PTR SS:[EBP-24]	
005469CB	. 0F84 E9000000	JE swdoctor.00546ABA	
005469D1	. 50	PUSH EAX	
005469D2	. F7D0	NOT EAX	
005469D4	. 0FC8	BSWAP EAX	

Stack SS:[0012D72C]=00F60000
ECX=00000000
Jump from 0054696F

Fig. 31

In [EBP-20] we have the base range address (take care the address is 0x00F60000 and may be different from the previous one, this is due to some restart into the code during my first analysis).

Now we know all the layer which have to pass before reach the unpacked target code and also when the first layer will be fully decrypted then we are ready to start with in-line patching.

4. In-line patching (part I)

First of all we have to save the base address in a safe place, to do that we have to found a free space inside our target where to redirect the execution from the Armadillo area. Run LordPE and use the PE Editor feature with the **swdoctor.exe**, take a look at the sections:

Name	VOffset	VSize	ROffset	RSize	Flags
.reloc	00120000	0000FB28	00001000	00010000	50000040
.text	00130000	00040000	00011000	00031000	E0000020
.adata	00170000	00010000	00042000	0000D000	E0000020
.data	00180000	00010000	0004F000	0000A000	C0000040
.reloc1	00190000	00010000	00059000	00003000	42000040
.pdata	001A0000	00110000	0005C000	00106000	C0000040
.rsrc	002B0000	000C1000	00162000	0000C000	50000040

Fig. 32

Go to OllyDbg and into the dump window press CTRL+G and write 0x570000 (we check the .text section), scroll down into the code and as you can see from 0x00560AD0 to the end we have a lot of free space, the we can start our first patch cave from address 0x00560B00.

Restart the target by using CTRL+F2 and remove all breakpoint (memory and hardware), then simply put a memory breakpoint into the our first redirection point in 0x005469C5 then press SHIFT+F9 and wait when OllyDbg break.

Now we can place a jump to our first cave code located at 0x00560B00:

005469C0	. 8945 F4	MOV DWORD PTR SS:[EBP-C],EAX	
005469C3	. ^ EB A4	JMP SHORT swdoctor.00546969	
005469C5	. E9 36A10100	JMP swdoctor.00560B00	Jump to our first cave
005469CA	. 90	NOP	
005469CB	. v 0F84 E9000000	JE swdoctor.00546ABA	
005469D1	. 50	PUSH EAX	
005469D2	. F7D0	NOT EAX	

Fig. 33 First redirection.

And write the first patching code (redirection from Armadillo static code to our cave area), redirection is made also to skip the CRC check, this will be show later, now keep it just as a example for the first patch cave:

00560ACD	. 00000000	DD 00000000	Return address (from cave to Armadillo code) address
00560AD1	. 00000000	DD 00000000	Base address storing register
00560AD5	. > 66:C705 C569	MOV WORD PTR DS:[5469C5],4D8B	First cave, restore
00560ADE	. 66:C705 C769	MOV WORD PTR DS:[5469C7],3BE0	the original
00560AE7	. 66:C705 C969	MOV WORD PTR DS:[5469C9],0DC4D	code at the redirection
00560AF0	. 60	PUSHAD	Save the registers
00560AF1	. 9C	PUSHFD	Save the flags
00560AF2	. 8B45 E0	MOV EAX,DWORD PTR SS:[EBP-20]	Retrieve the base address
00560AF5	. A3 D10A5600	MOV DWORD PTR DS:[560AD1],EAX	and save it in 560AD1 for future reference
00560AFA	. C780 4E1C0200	MOV DWORD PTR DS:[EAX+21C4E1],560B1568	Write the first redirection -> PUSH 0x00560B15
00560B04	. C780 521C0200	MOV DWORD PTR DS:[EAX+21C521],9090C300	Write the first redirection -> RETN
00560B0E	. 9D	POPAD	Restore the registers
00560B0F	. 61	POPAD	Restore the registers
00560B10	. ^ E9 B05EFFFF	JMP swdoctor.005469C5	Return to the main (static) code

Fig. 34 First cave (first redirection)

If you run this code step by step using F8 first the original code at the redirection address will be restored and then the code at the end of the first decryption loop is changed to redirect the flow from Armadillo code to the second cave section which start from the address 00560B15. To check it step with F8 until you reach the address 0x00560B45 and then check the code at the end of the first decryption loop.

00560A05	>	66:C705 C569	MOV WORD PTR DS:[5469C5],408B	First cave, restore
00560A0E	.	66:C705 C769	MOV WORD PTR DS:[5469C7],3BE0	the original
00560AE7	.	66:C705 C969	MOV WORD PTR DS:[5469C9],0DC4D	code at the redirection
00560AF0	.	60	PUSHAD	Save the registers
00560AF1	.	9C	PUSHFD	Save the flags
00560AF2	.	8B45 E0	MOV EAX,DWORD PTR SS:[EBP-20]	Retrieve the base address
00560AF5	.	A3 010A5600	MOV DWORD PTR DS:[560AD1],EAX	and save it in 560AD1 for future reference
00560AFA	.	C780 4E1C020	MOV DWORD PTR DS:[EAX+21C4E],560B1568	Write the first redirection -> PUSH 0x00560B15
00560B04	.	C780 521C020	MOV DWORD PTR DS:[EAX+21C52],9090C300	Write the first redirection -> RETN
00560B0E	.	9D	POPAD	Restore the flags
00560B0F	.	61	POPAD	Restore the registers
00560B10	^	E9 B05EFEFF	JMP swdoctor.005469C5	Return to the main (static) code

DS:[003C1C4E]=FEB48589

Address	Hex dump	Disassembly	Comment
003C1C4E	8985 B4FEFFFF	MOV DWORD PTR SS:[EBP-14C],EAX	
003C1C54	3DC3	CMP EAX,EBX	
003C1C56	74 19	JE SHORT 003C1C71	
003C1C58	8B08	MOV ECX,DWORD PTR DS:[EAX]	

Fig. 33a Execution of the first redirection and patch code.

00560A05	>	66:C705 C569	MOV WORD PTR DS:[5469C5],408B	First cave, restore
00560A0E	.	66:C705 C769	MOV WORD PTR DS:[5469C7],3BE0	the original
00560AE7	.	66:C705 C969	MOV WORD PTR DS:[5469C9],0DC4D	code at the redirection
00560AF0	.	60	PUSHAD	Save the registers
00560AF1	.	9C	PUSHFD	Save the flags
00560AF2	.	8B45 E0	MOV EAX,DWORD PTR SS:[EBP-20]	Retrieve the base address
00560AF5	.	A3 010A5600	MOV DWORD PTR DS:[560AD1],EAX	and save it in 560AD1 for future reference
00560AFA	.	C780 4E1C020	MOV DWORD PTR DS:[EAX+21C4E],560B1568	Write the first redirection -> PUSH 0x00560B15
00560B04	.	C780 521C020	MOV DWORD PTR DS:[EAX+21C52],9090C300	Write the first redirection -> RETN
00560B0E	.	9D	POPAD	Restore the flags
00560B0F	.	61	POPAD	Restore the registers
00560B10	^	E9 B05EFEFF	JMP swdoctor.005469C5	Return to the main (static) code

Stack [0012D6B4]=00000246 (00 D0 T0 S0 Z1 A0 P1 C0)

Address	Hex dump	Disassembly	Comment
003C1C4E	68 150B5600	PUSH 560B15	
003C1C53	C3	RETN	
003C1C54	90	NOP	
003C1C55	90	NOP	
003C1C56	74 19	JE SHORT 003C1C71	
003C1C58	8B08	MOV ECX,DWORD PTR DS:[EAX]	
003C1C5A	3BCB	CMP ECX,EBX	

Fig. 33 b Execution of the patching instruction fro the first layer.

Now save the injected code by right click and Copy to executable in order to check the target behaviour with this first in-line code.

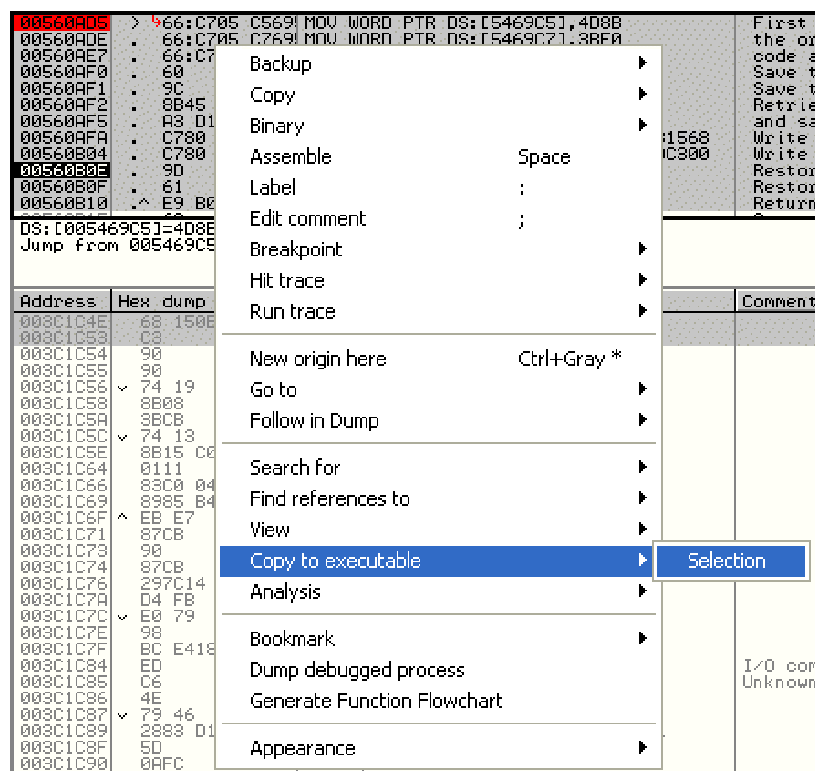


Fig. 34 Save the first patch.

After saving, restart the target with CTRL+F2, and run it with SHIFT+F9, we get an error message:

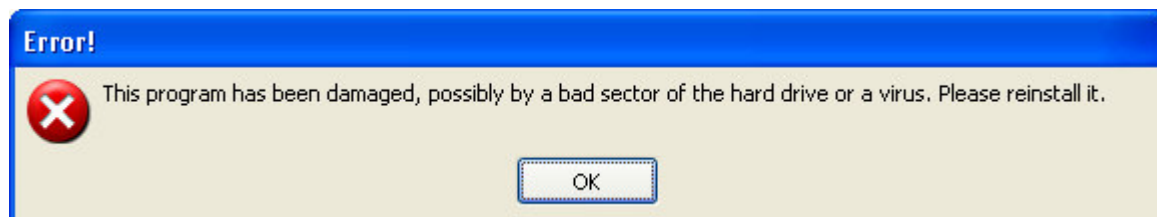


Fig. 35 Integrity check error message

Well this error mean some CRC check inside the code, no problem, press F12 and then with ALT+K take a look into the call stack:

0012CBDC	77F4B7F4	Includes 7FFE0304	ntdll.77F4B7F2	0012CC34
0012CBE0	77E5A2C0	ntdll.ZwDelayExecution	kernel32.77E5A2C7	0012CC34
0012CC38	77E41BF5	kernel32.SleepEx	kernel32.77E41BF0	0012CC34
0012CC3C	00000001	Timeout = 1. ms		
0012CC40	00000000	Alertable = FALSE		
0012CC44	00F8EE4E	? kernel32.Sleep	00F8EE48	
0012CC48	00000001	Timeout = 1. ms		
0012CCD8	00F827FA	? 00F8EBB0	00F827F5	
0012D750	00546202	Includes 00F827FA	swdoctor.005461FF	0012D74C
0012D7B8	0054774A	swdoctor.00545FAB	swdoctor.00547745	0012D7B4
0012DF18	00547818	swdoctor.00546FF7	swdoctor.00547813	0012DF14
0012DF20	00548043	swdoctor.00547810	swdoctor.0054803E	0012DF1C
0012FF38	0055B591	swdoctor.00547D10	swdoctor.<ModuleEntryPoint>	0012FF34
0012FF3C	00400000	Arg1 = 00400000 ASCII "MZP"		
0012FF40	00000000	Arg2 = 00000000		
0012FF44	00151F0A	Arg3 = 00151F0A		
0012FF48	0000000A	Arg4 = 0000000A		

Fig. 36 Call stack when the integrity check is triggered.

Double click into the highlighted row (first jump into the dynamically allocated memory aread for the packer) we have:

005461FF	. FF55 F0	CALL DWORD PTR SS:[EBP-10]
00546202	. 83C4 04	ADD ESP,4
00546205	. 8945 FC	MOV DWORD PTR SS:[EBP-4],EAX
00546208	. 837D EC FF	CMP DWORD PTR SS:[EBP-14],-1
0054620C	~ 74 0B	JE SHORT swdoctor.00546219
0054620E	. 8B55 EC	MOV EDX,DWORD PTR SS:[EBP-14]
00546211	. 8915 1C6F5800	MOV DWORD PTR DS:[586F1C],EDX
00546217	~ EB 10	JMP SHORT swdoctor.00546229
00546219	> 837D FC 01	CMP DWORD PTR SS:[EBP-4],1

Fig. 37

Place a memory breakpoint and restart the target, when OllyDbg break sure about the call address, for me I've 0x00F81BAA and I've same address for the original target then we have to step inside the call (this check is useful to find some redirection trick used by some protection, if some was wrong the indirect call can take different way from the right behaviour, and if you don't do this check all the work may be take very long time in debugging before discover it).

If you take a look at the call code there is another decrypting loop with end to 0x00F81C71 (offset **0x21C71**), to keep short for now follow the code in a similar way which we have do in our previous analysis then you should have:

00F81BA8	55	PUSH EBP	
00F81BA9	8BEC	MOV EBP,ESP	
00F81BAD	6A FF	PUSH -1	
00F81BAF	68 2020F900	PUSH 0F92020	
00F81BB4	68 E013F900	PUSH 0F913E0	
00F81BB9	64:A1 00000000	MOV EAX,DWORD PTR FS:[0]	JMP to MSUCRT._except_handler8
00F81BBF	50	PUSH EAX	
00F81BC0	64:8925 00000000	MOV DWORD PTR FS:[0],ESP	
00F81BC7	81EC 440A0000	SUB ESP,0A44	
00F81BCD	53	PUSH EBX	
00F81BCE	56	PUSH ESI	
00F81BCF	57	PUSH EDI	
00F81BD0	8965 E8	MOV DWORD PTR SS:[EBP-18],ESP	
00F81BD3	C745 E4 020000	MOV DWORD PTR SS:[EBP-1C],2	
00F81BDA	330B	XOR EBX,EBX	
00F81BDC	895D FC	MOV DWORD PTR SS:[EBP-4],EBX	
00F81BDF	E8 090E0000	CALL 00F829ED	
00F81BE4	68 680EFA00	PUSH 0FA0E68	
00F81BE9	FF15 7C20F900	CALL DWORD PTR DS:[F9207C]	ntdll.RtlEnterCriticalSection
00F81BEF	A1 E479F900	MOV EAX,DWORD PTR DS:[F979E4]	
00F81BF4	A3 C423FA00	MOV DWORD PTR DS:[FA23C4],EAX	
00F81BF9	C745 D4 FF20B7	MOV DWORD PTR SS:[EBP-2C],8CB720FF	
00F81C00	8B0D 600CFA00	MOV ECX,DWORD PTR DS:[FA0C60]	
00F81C06	898D B8FEFFFF	MOV DWORD PTR SS:[EBP-148],ECX	
00F81C0C	3BCB	CMP ECX,EBX	
00F81C0E	74 19	JE SHORT 00F81C29	
00F81C10	8B01	MOV EAX,DWORD PTR DS:[ECX]	
00F81C12	3BC3	CMP EAX,EBX	
00F81C14	74 13	JE SHORT 00F81C29	
00F81C16	8B15 C023FA00	MOV EDX,DWORD PTR DS:[FA23C0]	
00F81C1C	2910	SUB DWORD PTR DS:[EAX],EDX	
00F81C1E	83C1 04	ADD ECX,4	
00F81C21	898D B8FEFFFF	MOV DWORD PTR SS:[EBP-148],ECX	
00F81C27	EB E7	JMP SHORT 00F81C10	
00F81C29	53	PUSH EBX	
00F81C2A	FF35 E47BF900	PUSH DWORD PTR DS:[F97BE4]	
00F81C30	A1 EC1EFA00	MOV EAX,DWORD PTR DS:[FA1EEC]	
00F81C35	8B0D E479F900	MOV ECX,DWORD PTR DS:[F979E4]	
00F81C3B	03C1	ADD EAX,ECX	
00F81C3D	50	PUSH EAX	
00F81C3E	FF75 D4	PUSH DWORD PTR SS:[EBP-2C]	
00F81C41	E8 66F8FDFF	CALL 00F614AC	
00F81C46	83C4 10	ADD ESP,10	
00F81C49	A1 600CFA00	MOV EAX,DWORD PTR DS:[FA0C60]	
00F81C4E	8985 B4FEFFFF	MOV DWORD PTR SS:[EBP-14C],EAX	
00F81C54	3BC3	CMP EAX,EBX	
00F81C56	74 19	JE SHORT 00F81C71	
00F81C58	8B08	MOV ECX,DWORD PTR DS:[EAX]	
00F81C5A	3BCB	CMP ECX,EBX	
00F81C5C	74 13	JE SHORT 00F81C71	
00F81C5E	8B15 C023FA00	MOV EDX,DWORD PTR DS:[FA23C0]	
00F81C64	0111	ADD DWORD PTR DS:[ECX],EDX	
00F81C66	83C0 04	ADD EAX,4	
00F81C69	8985 B4FEFFFF	MOV DWORD PTR SS:[EBP-14C],EAX	
00F81C6F	EB E7	JMP SHORT 00F81C58	
00F81C71	87CB	XCHG EBX,ECX	End of the decryption loop
00F81C73	90	NOP	
00F81C74	87CB	XCHG EBX,ECX	
00F81C76	8B7D 08	MOV EDI,DWORD PTR SS:[EBP+8]	
00F81C79	8B47 10	MOV EAX,DWORD PTR DS:[EDI+10]	
00F81C7C	A3 8C1AFA00	MOV DWORD PTR DS:[FA1A8C],EAX	
00F81C81	8B77 20	MOV ESI,DWORD PTR DS:[EDI+20]	
00F81C84	8935 BC23FA00	MOV DWORD PTR DS:[FA23BC],ESI	
00F81C8A	8B47 08	MOV EAX,DWORD PTR DS:[EDI+8]	
00F81C8D	A3 F81EFA00	MOV DWORD PTR DS:[FA1EF8],EAX	
00F81C92	8B48 68	MOV ECX,DWORD PTR DS:[EAX+68]	
00F81C95	3348 64	XOR ECX,DWORD PTR DS:[EAX+64]	
00F81C98	3348 5C	XOR ECX,DWORD PTR DS:[EAX+5C]	
00F81C9B	F6C1 80	TEST CL,80	
00F81C9E	74 04	JE SHORT 00F81CA4	
00F81CA0	6A 01	PUSH 1	

Fig. 38 Code for the original swdoctor.exe target.

Step with F8 until you see this code:

00F81F41	83C4 0C	ADD ESP,0C	
00F81F44	50	PUSH EAX	
00F81F45	FF15 6822F900	CALL DWORD PTR DS:[F92268]	kernel32.OutputDebugStringA
00F81F4B	A1 F81EFA00	MOV EAX,DWORD PTR DS:[FA1EF8]	
00F81F50	8B48 68	MOV ECX,DWORD PTR DS:[EAX+68]	
00F81F53	3348 64	XOR ECX,DWORD PTR DS:[EAX+64]	
00F81F56	3348 5C	XOR ECX,DWORD PTR DS:[EAX+5C]	
00F81F59	F7C1 00003000	TEST ECX,300000	
00F81F5F	74 05	JE SHORT 00F81F66	
00F81F61	E8 10D90000	CALL 00F8F876	
00F81F66	8B45 08	MOV EAX,DWORD PTR SS:[EBP+8]	
00F81F69	8B40 0C	MOV EAX,DWORD PTR DS:[EAX+C]	
00F81F6C	8945 CC	MOV DWORD PTR SS:[EBP-34],EAX	
00F81F6F	8045 CC	LEA EAX,DWORD PTR SS:[EBP-34]	
00F81F72	50	PUSH EAX	
00F81F73	E8 27920000	CALL 00F8B19F	

Fig. 39

Take care about the OutputDebugStringA, look into the registers window EAX is a pointer to a very long string, this is a countermeasure able to defeat our OllyDbg if this isn't patched (use the Repair tools).

Step again a little with F8 until you reach another **OutputDebugStringA** API call:

00F824DA	C705 E079F900	MOV DWORD PTR DS:[F979E0],0F984AC	ASCII "CC5"
00F824E4	FFB5 7CFCFFFF	PUSH DWORD PTR SS:[EBP-384]	
00F824EA	FF15 A820F900	CALL DWORD PTR DS:[F920A8]	kernel32.CloseHandle
00F824F0	68 00010000	PUSH 100	
00F824F5	8085 C0FEFFFF	LEA EAX,DWORD PTR SS:[EBP-140]	
00F824FB	50	PUSH EAX	
00F824FC	FF35 0C7FF900	PUSH DWORD PTR DS:[F97F0C]	
00F82502	E8 AC6AFEFF	CALL 00F68FB3	
00F82507	83C4 0C	ADD ESP,0C	
00F8250A	50	PUSH EAX	
00F8250B	FF15 6822F900	CALL DWORD PTR DS:[F92268]	kernel32.OutputDebugStringA
00F82511	C705 E079F900	MOV DWORD PTR DS:[F979E0],0F984A8	ASCII "CC6"
00F8251B	8B0D F81EFA00	MOV ECX,DWORD PTR DS:[FA1EF8]	swdoctor.00580370
00F82521	8B41 6C	MOV EAX,DWORD PTR DS:[ECX+6C]	
00F82524	3341 68	XOR EAX,DWORD PTR DS:[ECX+68]	
00F82527	3341 34	XOR EAX,DWORD PTR DS:[ECX+34]	
00F8252A	3345 DC	XOR EAX,DWORD PTR SS:[EBP-24]	
00F8252D	8985 D4F6FFFF	MOV DWORD PTR SS:[EBP-92C],EAX	EAX=1AE8752F
00F82533	8945 C4	MOV DWORD PTR SS:[EBP-3C],EAX	
00F82536	A1 C01CFA00	MOV EAX,DWORD PTR DS:[FA1CC0]	
00F8253B	8BF8	MOV EDI,EAX	
00F8253D	897D C0	MOV DWORD PTR SS:[EBP-40],EDI	
00F82540	C1EE 02	SHR ESI,2	
00F82543	8D34B0	LEA ESI,DWORD PTR DS:[EAX+ESI*4]	
00F82546	89B5 BCFEFFFF	MOV DWORD PTR SS:[EBP-144],ESI	
00F8254C	3BFE	CMP EDI,ESI	
00F8254E	73 12	JNB SHORT 00F82562	
00F82550	8D4D C4	LEA ECX,DWORD PTR SS:[EBP-3C]	
00F82553	E8 A8EAFDFF	CALL 00F61000	
00F82558	3107	XOR DWORD PTR DS:[EDI],EAX	
00F8255A	83C7 04	ADD EDI,4	
00F8255D	897D C0	MOV DWORD PTR SS:[EBP-40],EDI	
00F82560	EB EA	JMP SHORT 00F8254C	
00F82562	C705 E079F900	MOV DWORD PTR DS:[F979E0],0F984A4	ASCII "LP4"

Fig. 40 Code for the patched executable (first cave).

The jump in 0x00F8254E isn't taken.

Now perform the same analysis on the original executable:

00F824DA	C705 E079F900 AC84F900	MOV DWORD PTR DS:[F979E0],0F984AC	ASCII "CC5"
00F824E4	FFB5 7CFCFFFF	PUSH DWORD PTR SS:[EBP-384]	
00F824EA	FF15 A820F900	CALL DWORD PTR DS:[F920A8]	kernel32.CloseHandle
00F824F0	68 00010000	PUSH 100	
00F824F5	8085 C0FEFFFF	LEA EAX,DWORD PTR SS:[EBP-140]	
00F824FB	50	PUSH EAX	
00F824FC	FF35 0C7FF900	PUSH DWORD PTR DS:[F97F0C]	
00F82502	E8 AC6AFEFF	CALL 00F68FB3	
00F82507	83C4 0C	ADD ESP,0C	
00F8250A	50	PUSH EAX	
00F8250B	FF15 6822F900	CALL DWORD PTR DS:[F92268]	kernel32.OutputDebugStringA
00F82511	C705 E079F900 A884F900	MOV DWORD PTR DS:[F979E0],0F984A8	ASCII "CC6"
00F8251B	8B0D F81EFA00	MOV ECX,DWORD PTR DS:[FA1EF8]	swdoctor.00580370
00F82521	8B41 6C	MOV EAX,DWORD PTR DS:[ECX+6C]	DS:[005803DC]=B3BC88D9
00F82524	3341 68	XOR EAX,DWORD PTR DS:[ECX+68]	DS:[005803D8]=18E8D04C
00F82527	3341 34	XOR EAX,DWORD PTR DS:[ECX+34]	DS:[005803A4]=AED2E732
00F8252A	3345 DC	XOR EAX,DWORD PTR SS:[EBP-24]	Stack SS:[0012D728]=AD3243A9
00F8252D	8985 D4F6FFFF	MOV DWORD PTR SS:[EBP-92C],EAX	EAX=A8B4FF0E
00F82533	8945 C4	MOV DWORD PTR SS:[EBP-3C],EAX	
00F82536	A1 C01CFA00	MOV EAX,DWORD PTR DS:[FA1CC0]	
00F8253B	8BF8	MOV EDI,EAX	
00F8253D	897D C0	MOV DWORD PTR SS:[EBP-40],EDI	
00F82540	C1EE 02	SHR ESI,2	
00F82543	8D34B0	LEA ESI,DWORD PTR DS:[EAX+ESI*4]	
00F82546	89B5 BCFEFFFF	MOV DWORD PTR SS:[EBP-144],ESI	
00F8254C	3BFE	CMP EDI,ESI	
00F8254E	73 12	JNB SHORT 00F82562	
00F82550	8D4D C4	LEA ECX,DWORD PTR SS:[EBP-3C]	
00F82553	E8 A8EAFDFF	CALL 00F61000	
00F82558	3107	XOR DWORD PTR DS:[EDI],EAX	
00F8255A	83C7 04	ADD EDI,4	
00F8255D	897D C0	MOV DWORD PTR SS:[EBP-40],EDI	
00F82560	EB EA	JMP SHORT 00F8254C	
00F82562	C705 E079F900 A884F900	MOV DWORD PTR DS:[F979E0],0F984A4	ASCII "LP4"

Fig. 43 Original executable

We can see about different value into the EAX register, may be this one is related to the CRC value (I know this is our CRC but I'll show how find that with a complete analysis, now just keep in mind the offset related to the address 0x00F8252D which is equal to **0x002252D**).

Step again:

00F8277E	53	PUSH EBX	
00F8277F	68 5080F900	PUSH 0F98050	ASCII "InvalidKey"
00F82784	E8 0054FEFF	CALL 00F67B89	
00F82789	68 103E0000	PUSH 3E10	
00F8278E	E8 D0EB0000	CALL 00F91370	JMP to MSVCRT.??2@YAPAXI0Z
00F82793	83C4 14	ADD ESP,14	
00F82796	8985 E4F6FFFF	MOV DWORD PTR SS:[EBP-91C],EAX	
00F8279C	3BC3	CMP EAX,EBX	
00F8279E	74 0F	JE SHORT 00F827AF	
00F827A0	FF75 C8	PUSH DWORD PTR SS:[EBP-38]	
00F827A3	FF75 CC	PUSH DWORD PTR SS:[EBP-34]	
00F827A6	8BC8	MOV ECX,EAX	
00F827A8	E8 E6040000	CALL 00F82C93	
00F827AD	EB 02	JMP SHORT 00F827B1	
00F827AF	33C0	XOR EAX,EAX	
00F827B1	8BF0	MOV ESI,EAX	
00F827B3	8975 D0	MOV DWORD PTR SS:[EBP-30],ESI	
00F827B6	E8 0B7FFFFF	CALL 00F7A6C6	
00F827BB	83F8 17	CMP EAX,17	EAX=17 (in-lined)
00F827BE	75 48	JNZ SHORT 00F82808	Jump is NOT taken
00F827C0	68 10000100	PUSH 10010	UNICODE "NC=C:\IAR\EW22\78000\inc\"
00F827C5	53	PUSH EBX	
00F827C6	53	PUSH EBX	
00F827C7	53	PUSH EBX	
00F827C8	53	PUSH EBX	

Fig. 44 In-lined code, jump is NOT taken

After some step you keep again the error message, now perform the same check with the original executable:

00F8277E	53	PUSH EBX	
00F8277F	68 5080F900	PUSH 0F98050	ASCII "InvalidKey"
00F82784	E8 0054FEFF	CALL 00F67B89	
00F82789	68 103E0000	PUSH 3E10	
00F8278E	E8 D0EB0000	CALL 00F91370	JMP to MSVCRT.??2@YAPAXI0Z
00F82793	83C4 14	ADD ESP,14	
00F82796	8985 E4F6FFFF	MOV DWORD PTR SS:[EBP-91C],EAX	
00F8279C	3BC3	CMP EAX,EBX	
00F8279E	74 0F	JE SHORT 00F827AF	
00F827A0	FF75 C8	PUSH DWORD PTR SS:[EBP-38]	
00F827A3	FF75 CC	PUSH DWORD PTR SS:[EBP-34]	
00F827A6	8BC8	MOV ECX,EAX	
00F827A8	E8 E6040000	CALL 00F82C93	
00F827AD	EB 02	JMP SHORT 00F827B1	
00F827AF	33C0	XOR EAX,EAX	
00F827B1	8BF0	MOV ESI,EAX	
00F827B3	8975 D0	MOV DWORD PTR SS:[EBP-30],ESI	
00F827B6	E8 0B7FFFFF	CALL 00F7A6C6	
00F827BB	83F8 17	CMP EAX,17	EAX=0
00F827BE	75 48	JNZ SHORT 00F82808	Jump is taken
00F827C0	68 10000100	PUSH 10010	UNICODE "NC=C:\IAR\EW22\78000\inc\"
00F827C5	53	PUSH EBX	
00F827C6	53	PUSH EBX	
00F827C7	53	PUSH EBX	
00F827C8	53	PUSH EBX	

Fig. 45 Jump is taken into the original executable code.

Yup we've found one point related to the CRC checking, in order to skip the check we can try with a simple patching and change the conditional jump JNZ with a fixed jump and see if this one is able to work, make this change and press F9:

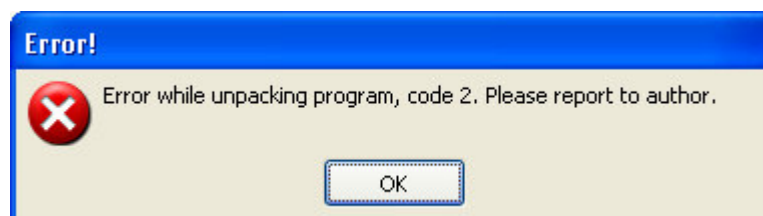


Fig. 46

Another error message arise (this is due to some error into the unpacking stage, every time some code is unpacked, now we have a bad CRC then the code isn't decrypted and the execution make one exception which show this message)

Simple patching isn't really effective then we have to backtrace a little in order to find where the check or some calculation was performed.

Because the jump is triggered by the CMP EAX,17 we have to step into the CALL 0x00F7A6C6 in order to show if there is the check, place one hardware breakpoint into the 0x00F827B6 and restart the target, when you reach this breakpoint press F7 once to see the call code.

00F7A6C6	A1 B823FA00	MOV EAX,DWORD PTR DS:[FA23B8]	DS:[00FA23B8]=00000017
00F7A6CB	C3	RETN	

Fig. 47

Yup, more simple but useful, restart the target again and place a memory breakpoint on byte write to the address 0x00FA23B8:

Address	Hex dump	ASCII
00FA23B8	17 00 00 00 00 00 40 00 00 00 F6 F0 76 1C 02 00@..
00FA23C8	00	
00FA23D8	00 Backup	
00FA23E8	02	
00FA23F8	00 Copy	
00FA2408	00	
00FA2418	00 Binary	
00FA2428	00	
00FA2438	02 Breakpoint	
00FA2448	00	
00FA2458	00 Search for	
00FA2468	00	
00FA2478	00 Go to	
00FA2488	00	
00FA2498	00 Hex	
00FA24A8	00	
00FA24B8	00	
00FA24C8	00	
00FA24D8	00	
00FA24E8	00	
00FA24F8	00	
00FA2508	00	
00FA2518	00	
00FA2528	2F Disassemble	
00FA2538	06	
00FA2548	32 Special	
00FA2558	0E	
00FA2568	00	
00FA2578	00	
00FA2588	00	

Fig. 48

now we are looking for the place where this byte will be written, OllyDbg break on this code:

00F7A6BC	8B4424 04	MOV EAX,DWORD PTR SS:[ESP+4]
00F7A6C0	A3 B823FA00	MOV DWORD PTR DS:[FA23B8],EAX
00F7A6C5	C3	RETN

Fig. 49

We have a first break but this place was write by 0 then not interesting, press again F9 there is another break and this time the address was written by 0x17, then press F7 once in order to return into the main code and we can see a nice code:

00F7B127	A3 FC1EFA00	MOV DWORD PTR DS:[FA1EFC],EAX
00F7B12C	8B10	MOV EDX,DWORD PTR DS:[EAX]
00F7B12E	8D48 04	LEA ECX,DWORD PTR DS:[EAX+4]
00F7B131	A1 F81EFA00	MOV EAX,DWORD PTR DS:[FA1EF8]
00F7B136	57	PUSH EDI
00F7B137	890D FC1EFA00	MOV DWORD PTR DS:[FA1EFC],ECX
00F7B13D	8B78 7C	MOV EDI,DWORD PTR DS:[EAX+7C]
00F7B140	3378 40	XOR EDI,DWORD PTR DS:[EAX+40]
00F7B143	3378 34	XOR EDI,DWORD PTR DS:[EAX+34]
00F7B146	3B07	CMPL EDX,EDI
00F7B148	74 0F	JE SHORT 00F7B159
00F7B14A	6A 17	PUSH 17
00F7B14C	E8 6BF5FFFF	CALL 00F7A6BC
00F7B151	59	POP ECX
00F7B152	32C0	XOR AL,AL
00F7B154	E9 5C020000	JMP 00F7B3B5
00F7B159	8B19	MOV EBX,DWORD PTR DS:[ECX]
00F7B15B	BF FFFF0000	MOV EDI,0FFFF
00F7B160	83C1 04	ADD ECX,4
00F7B163	3BDF	CMPL EBX,EDI
00F7B165	890D FC1EFA00	MOV DWORD PTR DS:[FA1EFC],ECX

Fig. 50

We've another conditional jump, then remove the hardware breakpoint on write and place another hardware breakpoint in execution at 0x00F7B127 and restart again the target.

00F7B127	A3 FC1EFA00	MOV DWORD PTR DS:[FA1EFC],EAX	EAX=005C9DD6 (swdoctor.005C9DD6)
00F7B12C	8B10	MOV EDX,DWORD PTR DS:[EAX]	DS:[005C9DD6]=62813606
00F7B12E	8D48 04	LEA ECX,DWORD PTR DS:[EAX+4]	Address=005C9DDA
00F7B131	A1 F81EFA00	MOV EAX,DWORD PTR DS:[FA1EF8]	DS:[00FA1EF8]=00580370
00F7B136	57	PUSH EDI	EDI=1
00F7B137	890D FC1EFA00	MOV DWORD PTR DS:[FA1EFC],ECX	DS:[005803EC]=28C843FD
00F7B13D	8B78 7C	MOV EDI,DWORD PTR DS:[EAX+7C]	DS:[005803B0]=D2B4F10B
00F7B140	3378 40	XOR EDI,DWORD PTR DS:[EAX+40]	DS:[005803A4]=AED2E732
00F7B143	3378 34	XOR EDI,DWORD PTR DS:[EAX+34]	EDX=62813606 / EDI = 54AE55C4
00F7B146	3B07	CMP EDX,EDI	
00F7B148	74 0F	JE SHORT 00F7B159	Jump is NOT taken
00F7B14A	6A 17	PUSH 17	
00F7B14C	E8 6BF5FFFF	CALL 00F7A6BC	
00F7B151	59	POP ECX	
00F7B152	32C0	XOR AL,AL	
00F7B154	E9 5C020000	JMP 00F7B3B5	
00F7B159	8B19	MOV EBX,DWORD PTR DS:[ECX]	
00F7B15B	BF FFFF0000	MOV EDI,0FFFF	

Fig. 51 Flow into the in-lined executable.

Now take the same check into the original target:

00F7B127	A3 FC1EFA00	MOV DWORD PTR DS:[FA1EFC],EAX	EAX=005C9DD6 (swdoctor.005C9DD6)
00F7B12C	8B10	MOV EDX,DWORD PTR DS:[EAX]	DS:[005C9DD6]=54AE55C4
00F7B12E	8D48 04	LEA ECX,DWORD PTR DS:[EAX+4]	Address=005C9DDA
00F7B131	A1 F81EFA00	MOV EAX,DWORD PTR DS:[FA1EF8]	DS:[00FA1EF8]=00580370
00F7B136	57	PUSH EDI	EDI = 1
00F7B137	890D FC1EFA00	MOV DWORD PTR DS:[FA1EFC],ECX	ECX=005C9DDA (swdoctor.005C9DDA)
00F7B13D	8B78 7C	MOV EDI,DWORD PTR DS:[EAX+7C]	DS:[005803EC]=28C843FD
00F7B140	3378 40	XOR EDI,DWORD PTR DS:[EAX+40]	DS:[005803B0]=D2B4F10B
00F7B143	3378 34	XOR EDI,DWORD PTR DS:[EAX+34]	DS:[005803A4]=AED2E732
00F7B146	3B07	CMP EDX,EDI	EDX = EDI = 54AE55C4
00F7B148	74 0F	JE SHORT 00F7B159	Jump is taken
00F7B14A	6A 17	PUSH 17	
00F7B14C	E8 6BF5FFFF	CALL 00F7A6BC	
00F7B151	59	POP ECX	
00F7B152	32C0	XOR AL,AL	
00F7B154	E9 5C020000	JMP 00F7B3B5	
00F7B159	8B19	MOV EBX,DWORD PTR DS:[ECX]	
00F7B15B	BF FFFF0000	MOV EDI,0FFFF	

Fig. 52 Flow into the original executable.

Well this is another conditional check about the CRC and EDX keep the calculated CRC, again if we try to fool the conditional JE with a JMP an error message is shown:



Fig. 53

Then we have to search where the wrong CRC will be calculated and change it with the right one (original target), to perform this search simply restart the target and place a memory breakpoint on access at 0x005C9DD6, this register keep a value which is related to the calculated CRC.

OllyDbg break in this interesting place:

00F824D5	^ E9 B5FEFFFF	JMP 00F8238F	
00F824DA	C705 E079F900	MOV DWORD PTR DS:[F979E0],0F984AC	ASCII "CC5"
00F824E4	FFB5 7CFCFFFF	PUSH DWORD PTR SS:[EBP-384]	
00F824EA	FF15 A820F900	CALL DWORD PTR DS:[F920A8]	kernel32.CloseHandle
00F824F0	68 00010000	PUSH 100	
00F824F5	8D85 C0FEFFFF	LEA EAX,DWORD PTR SS:[EBP-140]	
00F824FB	50	PUSH EAX	
00F824FC	FF35 0C7FF900	PUSH DWORD PTR DS:[F97F0C]	
00F82502	E8 AC6AFEFF	CALL 00F68FB3	
00F82507	83C4 0C	ADD ESP,0C	
00F8250A	50	PUSH EAX	
00F8250B	FF15 6822F900	CALL DWORD PTR DS:[F92268]	kernel32.OutputDebugStringA
00F82511	C705 E079F900	MOV DWORD PTR DS:[F979E0],0F984A8	ASCII "CC6"
00F8251B	8B0D F81EFA00	MOV ECX,DWORD PTR DS:[FA1EF8]	swdoctor.00580370
00F82521	8B41 6C	MOV EAX,DWORD PTR DS:[ECX+6C]	
00F82524	3341 68	XOR EAX,DWORD PTR DS:[ECX+68]	
00F82527	3341 34	XOR EAX,DWORD PTR DS:[ECX+34]	
00F8252A	3345 DC	XOR EAX,DWORD PTR SS:[EBP-24]	
00F8252D	98B5 D4F6FFFF	MOV DWORD PTR SS:[EBP-92C],EAX	
00F82533	8945 C4	MOV DWORD PTR SS:[EBP-3C],EAX	
00F82536	A1 C01CFA00	MOV EAX,DWORD PTR DS:[FA1CC0]	
00F8253B	8BF8	MOV EDI,EAX	
00F8253D	897D C0	MOV DWORD PTR SS:[EBP-40],EDI	
00F82540	C1EE 02	SHR ESI,2	
00F82543	8D34B0	LEA ESI,DWORD PTR DS:[EAX+ESI*4]	
00F82546	89B5 BCFEFFFF	MOV DWORD PTR SS:[EBP-144],ESI	
00F8254C	3BFE	CMP EDI,ESI	
00F8254E	^ 73 12	JNB SHORT 00F82562	
00F82550	8D4D C4	LEA ECX,DWORD PTR SS:[EBP-3C]	
00F82553	E8 A8EAFDFF	CALL 00F61000	
00F82558	3107	XOR DWORD PTR DS:[EDI],EAX	EAX=4514797E / [5C9DD6]=27954F78
00F8255A	83C7 04	ADD EDI,4	[005C9DD6]=62813606
00F8255D	897D C0	MOV DWORD PTR SS:[EBP-40],EDI	
00F82560	^ EB EA	JMP SHORT 00F8254C	
00F82562	C705 E079F900	MOV DWORD PTR DS:[F979E0],0F984A4	ASCII "LP4"
00F8256C	A0 A8A2F900	MOV AL,BYTE PTR DS:[F9A2A8]	

Fig. 54 Fig. 54 In-lined executable.

And registers window look as below:

Registers (FPU)	
EAX	4514797E
ECX	00002710
EDX	00001920
EBX	00000000
ESP	0012CCEC
EBP	0012D74C
ESI	005CC5D6 swdoctor.005CC5D6
EDI	005C9DD6 swdoctor.005C9DD6
EIP	00F82558

Fig. 55 In-lined executable.

00F824D5	^ E9 B5FEFFFF	JMP 00F8238F	
00F824DA	C705 E079F900	MOV DWORD PTR DS:[F979E0],0F984AC	ASCII "CC5"
00F824E4	FFB5 7CFCFFFF	PUSH DWORD PTR SS:[EBP-384]	
00F824EA	FF15 A820F900	CALL DWORD PTR DS:[F920A8]	kernel32.CloseHandle
00F824F0	68 00010000	PUSH 100	
00F824F5	8D85 C0FEFFFF	LEA EAX,DWORD PTR SS:[EBP-140]	
00F824FB	50	PUSH EAX	
00F824FC	FF35 0C7FF900	PUSH DWORD PTR DS:[F97F0C]	
00F82502	E8 AC6AFEFF	CALL 00F68FB3	
00F82507	83C4 0C	ADD ESP,0C	
00F8250A	50	PUSH EAX	
00F8250B	FF15 6822F900	CALL DWORD PTR DS:[F92268]	kernel32.OutputDebugStringA
00F82511	C705 E079F900	MOV DWORD PTR DS:[F979E0],0F984A8	ASCII "CC6"
00F8251B	8B0D F81EFA00	MOV ECX,DWORD PTR DS:[FA1EF8]	swdoctor.00580370
00F82521	8B41 6C	MOV EAX,DWORD PTR DS:[ECX+6C]	
00F82524	3341 68	XOR EAX,DWORD PTR DS:[ECX+68]	
00F82527	3341 34	XOR EAX,DWORD PTR DS:[ECX+34]	
00F8252A	3345 DC	XOR EAX,DWORD PTR SS:[EBP-24]	
00F8252D	98B5 D4F6FFFF	MOV DWORD PTR SS:[EBP-92C],EAX	
00F82533	8945 C4	MOV DWORD PTR SS:[EBP-3C],EAX	
00F82536	A1 C01CFA00	MOV EAX,DWORD PTR DS:[FA1CC0]	
00F8253B	8BF8	MOV EDI,EAX	
00F8253D	897D C0	MOV DWORD PTR SS:[EBP-40],EDI	
00F82540	C1EE 02	SHR ESI,2	
00F82543	8D34B0	LEA ESI,DWORD PTR DS:[EAX+ESI*4]	
00F82546	89B5 BCFEFFFF	MOV DWORD PTR SS:[EBP-144],ESI	
00F8254C	3BFE	CMP EDI,ESI	
00F8254E	^ 73 12	JNB SHORT 00F82562	
00F82550	8D4D C4	LEA ECX,DWORD PTR SS:[EBP-3C]	
00F82553	E8 A8EAFDFF	CALL 00F61000	
00F82558	3107	XOR DWORD PTR DS:[EDI],EAX	EAX=733B1ABC / [005C9DD6]=27954F78
00F8255A	83C7 04	ADD EDI,4	[EDI]=54AE55C4
00F8255D	897D C0	MOV DWORD PTR SS:[EBP-40],EDI	
00F82560	^ EB EA	JMP SHORT 00F8254C	

Fig. 56 Original code

EAX looks different between the original and in-lined executable, then we have to step into the previous call at 0x00F82553, in this one we have that the EAX value, stored in [EBP-3C] at the address 0x00F82533, will be read at 0x00F61048 then from this point some calculation was performed, by the way because the final result will be related to this starting value we can think that we have to force in EAX, at 0x00F8252D (offset **0x002252D**), the value which we have into the original executable, if this think will be right we can skip all the CRC checking.

To sure simply restart the target and when you're in 0x00F8252D change the EAX value to **0x00A8B4FF0E**, with this change the in-lined target is able to run like a piece of cake then have succeed into the CRC defeating task.

00F824B8	8985 ECF6FFFF	MOV DWORD PTR SS:[EBP-914],EAX	
00F824C1	50	PUSH EAX	
00F824C2	E8 9DEE0000	CALL 00F91364	JMP to MSVCRT.??3@YAXPAX0Z
00F824C7	59	POP ECX	
00F824C8	FF85 64F8FFFF	INC DWORD PTR SS:[EBP-79C]	
00F824CE	8385 90FCFFFF	ADD DWORD PTR SS:[EBP-370],20	
00F824D5	E9 B5FEFFFF	JMP 00F8238F	
00F824DA	C705 E079F900	MOV DWORD PTR DS:[F979E0],0F984AC	ASCII "CC5"
00F824E4	FFB5 7CFCFFFF	PUSH DWORD PTR SS:[EBP-384]	
00F824EA	FF15 A820F900	CALL DWORD PTR DS:[F920A8]	kernel32.CloseHandle
00F824F0	68 00010000	PUSH 100	
00F824F5	8D85 C0FEFFFF	LEA EAX,DWORD PTR SS:[EBP-140]	
00F824FB	50	PUSH EAX	
00F824FC	FF35 0C7FF900	PUSH DWORD PTR DS:[F97F0C]	
00F82502	E8 AC6AFEFF	CALL 00F68FB3	
00F82507	83C4 0C	ADD ESP,0C	
00F8250A	50	PUSH EAX	
00F8250B	FF15 6822F900	CALL DWORD PTR DS:[F92268]	kernel32.OutputDebugStringA
00F82511	C705 E079F900	MOV DWORD PTR DS:[F979E0],0F984A8	ASCII "CC6"
00F8251B	8B0D F81EFA00	MOV ECX,DWORD PTR DS:[FA1EF8]	swdoctor.00580370
00F82521	8B41 6C	MOV EAX,DWORD PTR DS:[ECX+6C]	
00F82524	3341 68	XOR EAX,DWORD PTR DS:[ECX+68]	
00F82527	3341 34	XOR EAX,DWORD PTR DS:[ECX+34]	
00F8252A	3345 DC	XOR EAX,DWORD PTR SS:[EBP-24]	
00F8252D	8985 04F6FFFF	MOV DWORD PTR SS:[EBP-92C],EAX	Force EAX=0x00A8B4FF0E to fool the CRC checking
00F82533	8945 C4	MOV DWORD PTR SS:[EBP-3C],EAX	
00F82536	A1 C01CFA00	MOV EAX,DWORD PTR DS:[FA1CC0]	
00F8253B	8BF8	MOV EDI,EAX	
00F8253D	897D C0	MOV DWORD PTR SS:[EBP-40],EDI	

Fig. 57 Change EAX in order to skip the CRC check.

Now we have to do a think, before patch the first decrypting loop and then reach the 0x00F88221 address (offset 0x0028221) we have also to defeat the CRC check, but this one is wrapped into another decrypting loop hence we have to move our first redirection from this point, patch the CRC check and then going into the next decryption layer.

5. In-line patching (part II)

From our first analysis we have found a CRC check which we have to skip before make patching, then now we have to write the code in our first cave in order to meet this request then we have also to perform a first redirection from offset 0x0021C71 before patch the CRC check. To do this restart the target and place a memory breakpoint in 0x005469C5 then press SHIFT+F9 and when OllyDbg break write the first redirection code:

005469C5	> E9 36A10100	JMP swdoctor.00560B00	Jump to our first cave
005469CA	90	NOP	
005469CB	0F84 E9000000	JE swdoctor.005469EH	
005469D1	50	PUSH EAX	

Fig. 58 First redirection (from main code to cave 1)

We have for the first cave this code:

Cave 1

00560AD5	>	66:C705 C5695400 8B4D	MOV WORD PTR DS:[5469C5],4D8B	First cave, restore
00560ADE	.	66:C705 C7695400 E03B	MOV WORD PTR DS:[5469C7],3BE0	the original
00560AE7	.	66:C705 C9695400 4DDC	MOV WORD PTR DS:[5469C9],0DC4D	code at the redirection
00560AF0	.	60	PUSHAD	Save the registers
00560AF1	.	9C	PUSHFD	Save the flags
00560AF2	.	8B45 E0	MOV EAX,DWORD PTR SS:[EBP-20]	Retrieve the base address
00560AF5	.	A3 D10A5600	MOV DWORD PTR DS:[560AD1],EAX	and save it in 560AD1 for future reference
00560AFA	.	C780 4E1C0200 68150B56	MOV DWORD PTR DS:[EAX+21C4E],560B1568	Write the first redirection -> PUSH 0x00560B15
00560B04	.	C780 521C0200 00C39090	MOV DWORD PTR DS:[EAX+21C52],9090C300	Write the first redirection -> RETN
00560B0E	.	9D	POPPD	Restore the flags
00560B0F	.	61	POPAD	Restore the registers
00560B10	^	E9 B05EFEFF	JMP swdoctor.005469C5	Return to the main (static) code
00560B15	.	60	PUSHAD	Second cave
00560B16	.	9C	PUSHFD	

Fig. 59 OllyDbg screenshot (cave 1)

00560AD5	>	\66:C705 C5695400 8B4D	MOV WORD PTR DS:[5469C5],4D8B	; First cave
00560ADE	.	66:C705 C7695400 E03B	MOV WORD PTR DS:[5469C7],3BE0	
00560AE7	.	66:C705 C9695400 4DDC	MOV WORD PTR DS:[5469C9],0DC4D	
00560AF0	.	60	PUSHAD	
00560AF1	.	9C	PUSHFD	
00560AF2	.	8B45 E0	MOV EAX,DWORD PTR SS:[EBP-20]	
00560AF5	.	A3 D10A5600	MOV DWORD PTR DS:[560AD1],EAX	
00560AFA	.	C780 4E1C0200 68150B56	MOV DWORD PTR DS:[EAX+21C4E],560B1568	; PUSH 0x00560B15
00560B04	.	C780 521C0200 00C39090	MOV DWORD PTR DS:[EAX+21C52],9090C300	; RETN
00560B0E	.	9D	POPPD	
00560B0F	.	61	POPAD	
00560B10	^	E9 B05EFEFF	JMP swdoctor.005469C5	; return to main code
00560B15	.	60	PUSHAD	; Second cave
00560B16	.	9C	PUSHFD	

Now remove all the breakpoint (memory and hardware) and put only a memory breakpoint into the begin address of the second cave (0x00560B0E) we reach this point from the first decrypting loop at offset 0x0021C4E. Now we have the CRC code fully decrypted and in this new cave we have to redirect from offset 0x0021C4E and keep EAX to be equal to the real CRC value which we have see equal to 0x00A8B4FF0E.

As we with the first cave we have first to restore the original code and then write the patch for the redirection to the third cave:

Cave 2

00560B15	.	60	PUSHAD	Second cave, save the registers
00560B16	.	9C	PUSHFD	Save the flags
00560B17	.	A1 D10A5600	MOV EAX,DWORD PTR DS:[560AD1]	Keep back the base address
00560B1C	.	C780 4E1C0200 8985B4FE	MOV DWORD PTR DS:[EAX+21C4E],FEB48589	and restore the
00560B26	.	C780 521C0200 FFFF3BC3	MOV DWORD PTR DS:[EAX+21C52],C33BFFFF	original code on the redirection address
00560B30	.	C780 2D250200 68580B56	MOV DWORD PTR DS:[EAX+2252D],560B5868	Write the redirection -> PUSH 0x00560B58
00560B3A	.	66:C780 31250200 00C3	MOV WORD PTR DS:[EAX+22531],0C300	Write the redirection -> RETN
00560B43	.	8D98 461C0200	LEA EBX,DWORD PTR DS:[EAX+21C46]	EBX = return address
00560B49	.	891D CD0A5600	MOV DWORD PTR DS:[560ACD],EBX	and store it into the return register
00560B4F	.	9D	POPPD	Restore the flags
00560B50	.	61	POPAD	Restore the registers
00560B51	.	FF35 CD0A5600	PUSH DWORD PTR DS:[560ACD]	Push the return address
00560B57	.	C3	RETN	Use RETN like a JMP to the Armadillo code
00560B58	.	60	PUSHAD	Third cave
00560B59	.	9C	PUSHFD	

Fig. 60 OllyDbg screenshot (cave 2)

00560B15	.	60	PUSHAD	; Second cave
00560B16	.	9C	PUSHFD	
00560B17	.	A1 D10A5600	MOV EAX,DWORD PTR DS:[560AD1]	
00560B1C	.	C780 4E1C0200 8985B4FE	MOV DWORD PTR DS:[EAX+21C4E],FEB48589	; Restore code
00560B26	.	C780 521C0200 FFFF3BC3	MOV DWORD PTR DS:[EAX+21C52],C33BFFFF	
00560B30	.	C780 2D250200 68580B56	MOV DWORD PTR DS:[EAX+2252D],560B5868	; PUSH 0x00560B58
00560B3A	.	66:C780 31250200 00C3	MOV WORD PTR DS:[EAX+22531],0C300	; RETN
00560B43	.	8D98 461C0200	LEA EBX,DWORD PTR DS:[EAX+21C46]	
00560B49	.	891D CD0A5600	MOV DWORD PTR DS:[560ACD],EBX	
00560B4F	.	9D	POPPD	
00560B50	.	61	POPAD	
00560B51	.	FF35 CD0A5600	PUSH DWORD PTR DS:[560ACD]	
00560B57	.	C3	RETN	
00560B58	.	60	PUSHAD	; Third cave
00560B59	.	9C	PUSHFD	

in order to go back into the original code we keep the returning address in [560AF8] and then push it into the stack after context restoring (POPFD / POPAD instructions) then by using the RETN instruction we're able to perform the the jump.

Save our second cave patch, remove all the breakpoint and restart the target by using CTRL+F2, then when you're into the EP (Entry Point) place one memory breakpoint at the next entry cave at the address 0x00560B58 and press F9.

Now OllyDbg break into the entry of the third cave, as usual we have to save the context (registers and flags) and restore the original code at the redirection point then patch the CRC and perform the next redirection into the offset 0x0023198, finally return to the Armadillo code.

Cave 3

00560B58	. 60	PUSHAD	Third cave
00560B59	. 9C	PUSHFD	
00560B5A	. A1 D10A5600	MOV EAX,DWORD PTR DS:[560AD1]	
00560B5F	. C780 2D250200 8985D4F6	MOV DWORD PTR DS:[EAX+2252D],F6D48589	
00560B69	. 66:C780 31250200 FFFF	MOV WORD PTR DS:[EAX+22531],0FFFF	
00560B72	. C780 98310200 689F0B56	MOV DWORD PTR DS:[EAX+23198],560B9F68	Write the redirection -> PUSH 0x00560B9F
00560B7C	. 66:C780 9C310200 00C3	MOV WORD PTR DS:[EAX+2319C],0C300	Write the redirection -> RETN
00560B85	. 8D98 2D250200	LEA EBX,DWORD PTR DS:[EAX+2252D]	
00560B8B	. 891D CD0A5600	MOV DWORD PTR DS:[560ACD],EBX	
00560B91	. 9D	POPFD	
00560B92	. 61	POPAD	
00560B93	. FF35 CD0A5600	PUSH DWORD PTR DS:[560ACD]	
00560B99	. B8 0EFFB4A8	MOV EAX,A8B4FF0E	Force the right CRC value
00560B9E	. C3	RETN	
00560BA0	. 60	PUSHAD	Fourth cave

Fig. 61 OllyDbg screenshot (cave 3)

00560B58	. 60	PUSHAD	; Third cave
00560B59	. 9C	PUSHFD	
00560B5A	. A1 D10A5600	MOV EAX,DWORD PTR DS:[560AD1]	
00560B5F	. C780 2D250200 8985D4F6	MOV DWORD PTR DS:[EAX+2252D],F6D48589	
00560B69	. 66:C780 31250200 FFFF	MOV WORD PTR DS:[EAX+22531],0FFFF	
00560B72	. C780 98310200 689F0B56	MOV DWORD PTR DS:[EAX+23198],560B9F68	; PUSH 0x00560B9F
00560B7C	. 66:C780 9C310200 00C3	MOV WORD PTR DS:[EAX+2319C],0C300	; RETN
00560B85	. 8D98 2D250200	LEA EBX,DWORD PTR DS:[EAX+2252D]	
00560B8B	. 891D CD0A5600	MOV DWORD PTR DS:[560ACD],EBX	
00560B91	. 9D	POPFD	
00560B92	. 61	POPAD	
00560B93	. FF35 CD0A5600	PUSH DWORD PTR DS:[560ACD]	
00560B99	. B8 0EFFB4A8	MOV EAX,A8B4FF0E	; Force the right CRC value
00560B9E	. C3	RETN	
00560B9F	. 60	PUSHAD	; Fourth cave
00560BA0	. 9C	PUSHFD	

Now restart the target, clear all the breakpoint and put one memory breakpoint into the four cave entry at 0x00560B9F, now we have to patch the redirection from the next decryption loop which is before the IAT scrambling at offset 0x28754 then we have:

Cave 4

00560B9F	: 60	PUSHAD	Fourth cave
00560BA0	: 9C	PUSHFD	
00560BA1	: A1 D10A5600	MOV EAX,DWORD PTR DS:[560AD1]	
00560BA6	: C780 98310200 898574FB	MOV DWORD PTR DS:[EAX+23198],FB748589	
00560BB0	: C780 9C310200 FFFF83BD	MOV DWORD PTR DS:[EAX+2319C],BD83FFFF	
00560BBA	: C780 54870200 68E20B56	MOV DWORD PTR DS:[EAX+28754],560BE268	Write the redirection -> PUSH 0x00560BE2
00560BC4	: 66:C780 58870200 00C3	MOV WORD PTR DS:[EAX+28758],0C300	Write the redirection -> RETN
00560BCD	: 8D98 98310200	LEA EBX,DWORD PTR DS:[EAX+23198]	
00560BD3	: 891D CD0A5600	MOV DWORD PTR DS:[560ACD],EBX	
00560BD9	: 9D	POPF	
00560BDA	: 61	POPAD	
00560BDB	: FF35 CD0A5600	PUSH DWORD PTR DS:[560ACD]	
00560BE1	: C3	RETN	
00560BE2	: 60	PUSHAD	Fifth cave

Fig. 62 OllyDbg screenshot (cave 4)

00560B9F	. 60	PUSHAD	; Fourth cave
00560BA0	. 9C	PUSHFD	
00560BA1	. A1 D10A5600	MOV EAX,DWORD PTR DS:[560AD1]	
00560BA6	. C780 98310200 898574FB	MOV DWORD PTR DS:[EAX+23198],FB748589	
00560BB0	. C780 9C310200 FFFF83BD	MOV DWORD PTR DS:[EAX+2319C],BD83FFFF	
00560BBA	. C780 54870200 68E20B56	MOV DWORD PTR DS:[EAX+28754],560BE268	; PUSH 0x00560BE2
00560BC4	. 66:C780 58870200 00C3	MOV WORD PTR DS:[EAX+28758],0C300	; RETN
00560BCD	. 8D98 98310200	LEA EBX,DWORD PTR DS:[EAX+23198]	
00560BD3	. 891D CD0A5600	MOV DWORD PTR DS:[560ACD],EBX	
00560BD9	. 9D	POPF	
00560BDA	. 61	POPAD	
00560BDB	. FF35 CD0A5600	PUSH DWORD PTR DS:[560ACD]	
00560BE1	. C3	RETN	
00560BE2	. 60	PUSHAD	; Fifth cave
00560BE3	. 9C	PUSHFD	

as usual save this patch and restart the target, again place one memory breakpoint into the start of the next cave on 0x00560BE2.

Cave 5

00560BE2	: 60	PUSHAD	Fifth cave
00560BE3	: 9C	PUSHFD	
00560BE4	: A1 D10A5600	MOV EAX,DWORD PTR DS:[560AD1]	
00560BE9	: C780 54870200 898594C3	MOV DWORD PTR DS:[EAX+28754],C3948589	
00560BF3	: C780 58870200 FFFF83BD	MOV DWORD PTR DS:[EAX+28758],BD83FFFF	
00560BFD	: C780 84920200 68250C56	MOV DWORD PTR DS:[EAX+29284],560C2568	Write the redirection -> PUSH 0x00560C25
00560C07	: 66:C780 88920200 00C3	MOV WORD PTR DS:[EAX+29288],0C300	Write the redirection -> RETN
00560C10	: 8D98 54870200	LEA EBX,DWORD PTR DS:[EAX+28754]	
00560C16	: 891D CD0A5600	MOV DWORD PTR DS:[560ACD],EBX	
00560C1C	: 9D	POPF	
00560C1D	: 61	POPAD	
00560C1E	: FF35 CD0A5600	PUSH DWORD PTR DS:[560ACD]	
00560C24	: C3	RETN	
00560C25	: 60	PUSHAD	Final cave

Fig. 63 OllyDbg screenshot (cave 5)

00560BE2	. 60	PUSHAD	; Fifth cave
00560BE3	. 9C	PUSHFD	
00560BE4	. A1 D10A5600	MOV EAX,DWORD PTR DS:[560AD1]	
00560BE9	. C780 54870200 898594C3	MOV DWORD PTR DS:[EAX+28754],C3948589	
00560BF3	. C780 58870200 FFFF83BD	MOV DWORD PTR DS:[EAX+28758],BD83FFFF	
00560BFD	. C780 84920200 68250C56	MOV DWORD PTR DS:[EAX+29284],560C2568	; PUSH 0x00560C25
00560C07	. 66:C780 88920200 00C3	MOV WORD PTR DS:[EAX+29288],0C300	; RETN
00560C10	. 8D98 54870200	LEA EBX,DWORD PTR DS:[EAX+28754]	
00560C16	. 891D CD0A5600	MOV DWORD PTR DS:[560ACD],EBX	
00560C1C	. 9D	POPF	
00560C1D	. 61	POPAD	
00560C1E	. FF35 CD0A5600	PUSH DWORD PTR DS:[560ACD]	
00560C24	. C3	RETN	
00560C25	. 60	PUSHAD	; Final cave
00560C26	. 9C	PUSHFD	

as usual save this patch and restart the target, place one memory breakpoint into the start of the next cave on 0x00560C25 this is our last cave and from this place we can apply the final target patching to keep it registered.

Cave 6

00560C25	60	PUSHAD	Final cave
00560C26	9C	PUSHFD	
00560C27	A1 D10A5600	MOV EAX,DWORD PTR DS:[560AD1]	
00560C2C	C780 84920200 FFB534C8	MOV DWORD PTR DS:[EAX+29284],C834B5FF	
00560C36	66:C780 88920200 FFFF	MOV WORD PTR DS:[EAX+29288],0FFFF	
00560C3F	66:C705 1F305000 B001	MOV WORD PTR DS:[50301F],1B0	Target patching
00560C48	C605 A72C4F00 EB	MOV BYTE PTR DS:[4F2CA7],0EB	Target patching
00560C4F	C605 98C64400 EB	MOV BYTE PTR DS:[44C698],0EB	Target patching
00560C56	8D98 84920200	LEA EBX,DWORD PTR DS:[EAX+29284]	
00560C5C	891D CD0A5600	MOV DWORD PTR DS:[560ACD],EBX	
00560C62	9D	POPPD	
00560C63	61	POPAD	
00560C64	FF35 CD0A5600	PUSH DWORD PTR DS:[560ACD]	
00560C6A	C3	RETN	
00560C6B	00	DB 00	

Fig. 64 OllyDbg screenshot (cave 6)

00560C25	. 60	PUSHAD	; Final cave
00560C26	. 9C	PUSHFD	
00560C27	. A1 D10A5600	MOV EAX,DWORD PTR DS:[560AD1]	
00560C2C	. C780 84920200 FFB534C8	MOV DWORD PTR DS:[EAX+29284],C834B5FF	
00560C36	. 66:C780 88920200 FFFF	MOV WORD PTR DS:[EAX+29288],0FFFF	
00560C3F	. 66:C705 1F305000 B001	MOV WORD PTR DS:[50301F],1B0	; Target patching
00560C48	. C605 A72C4F00 EB	MOV BYTE PTR DS:[4F2CA7],0EB	; Target patching
00560C4F	. C605 98C64400 EB	MOV BYTE PTR DS:[44C698],0EB	; Target patching
00560C56	. 8D98 84920200	LEA EBX,DWORD PTR DS:[EAX+29284]	
00560C5C	. 891D CD0A5600	MOV DWORD PTR DS:[560ACD],EBX	
00560C62	. 9D	POPPD	
00560C63	. 61	POPAD	
00560C64	. FF35 CD0A5600	PUSH DWORD PTR DS:[560ACD]	
00560C6A	. C3	RETN	

save our last cave and when you've do it close OllyDbg. If you run the target one access memory on write violation error pop-up, this is because into the last cave we've try to write into the .code section which is not Writable, to avoid this problem simply change the characteristics number for this section.

Open the Peditor tools and load the in-lined target then press the **section** button:

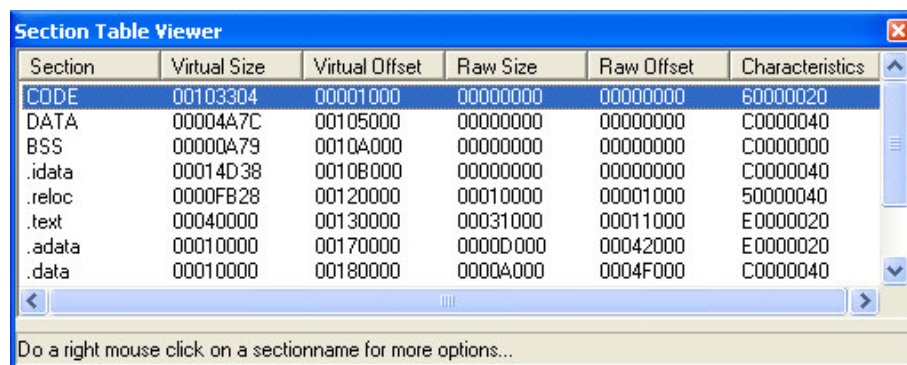


Fig. 65 Peditor: Section Table Viewer

Right click into the highlighted row:

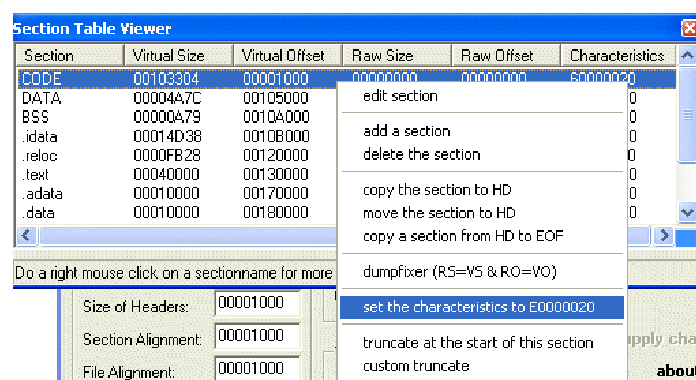


Fig. 66 Set the section characteristics to E0000020.

Close the PEditor and run the target, it works and is also still registered!

Works done!

6. Conclusion

I hope that this tutorial is able to show in a clear and simple way the in-line patching process, this one is more general and can be used also for other packer like ASProtect, about this approach we can point some advantage:

- MUP (manual unpacking) of the target isn't needed
- patch distribution is keep small in size

this method have also a drawback and isn't really effective when the packer work with many encrypted layer because before reach the real target code we have to step through all the layer and this may be a very time expensive work, may be a better way code a suitable loader or approach with the manual unpacking as last option.

Remember also the main things:

If you like this software you have to BUY IT or simply uninstall it after expiration.
This tutorial are used for educational purposes only.

7. Greetings

Thanks to the whole ARTeam:

**[Nilrem] [JDog45] [Shub - Nigurrath] [MaDMan_H3rCuL3s] [Ferrari]
[Kruger] [Teerayoot] [R@dier] [ThunderPwr] [Eggi] [EJ12N]
[Gabri3l][Stickman 373] [Bone Enterprise] [JohnWho] [Condzero]**

Thanks to all the people who take time to write tutorials.
Thanks to all the people who continue to develop better tools.
Thanks to Exetools, Woodmann, SND, TSRH, MP2K and all the others for being a great place of learning.
Thanks also to The Codebreakers Journal, and the Anticrack forum.

If you have any suggestions, comments or corrections fell free to contact me in ARTeam forum.

Byez

ThunderPwr