

# Inline patching Asprotect 2.x

**ThunderPwr (ARTeam)**

## 0. INTRODUCTION

This essay has been written to illustrate the steps that must be carry out to do the inline patching technique of programs compressed with ASProtect and also, from a more general point of view, in what way we have to approach packed target for inline patching.

The inline patching are not the simpler way that can be used in order to modify the behaviour of a program since require a good understanding of the packer protection mechanisms, but after this first step the way may be easily extendible to other different packer (e.g. Armadillo).

From the point of view of the efficiency it win regarding the unpacking approach because the size of the final patch will be more little than the decompressed executable (MUP approach), the only drawback is that this approach can be less algorithmic than the automatic unpacking.

In order to show the technique on a real example I've try the Chord Pickout target, release version 1.5. You can download the target to the [www.chordpickout.com](http://www.chordpickout.com) web site address.

**All the information contained in this tutorial does not have to be used in order to use in illegal way copyrighted and protected software. All the info in this tutorial must be used in order to only better know the protection scheme used from this particular protection. It is not encouraged therefore to the use of these information for illicit or various scopes different than the pure study.**

## 1. PACKER ANALYSIS (REVERSING STAGE)

By means of a file scanner we can check if the executable was packed or not (of course it is but this is to cover all the stuff in a more general way):



This is confirmed also by direct inspection of the Entry point structure which is the same for every ASProtected target:

00401000	68	01B04E00	PUSH Chordpic.004EB001
00401005	E8	01000000	CALL Chordpic.0040100B
0040100A	C3		RETN
0040100B	C3		RETN
0040100C	AA		DB AA
0040100D	1F		DB 1F
0040100E	5A		DB 5A
0040100F	D0		DB D0
00401010	D0		DB D0
00401011	D3		DB D3

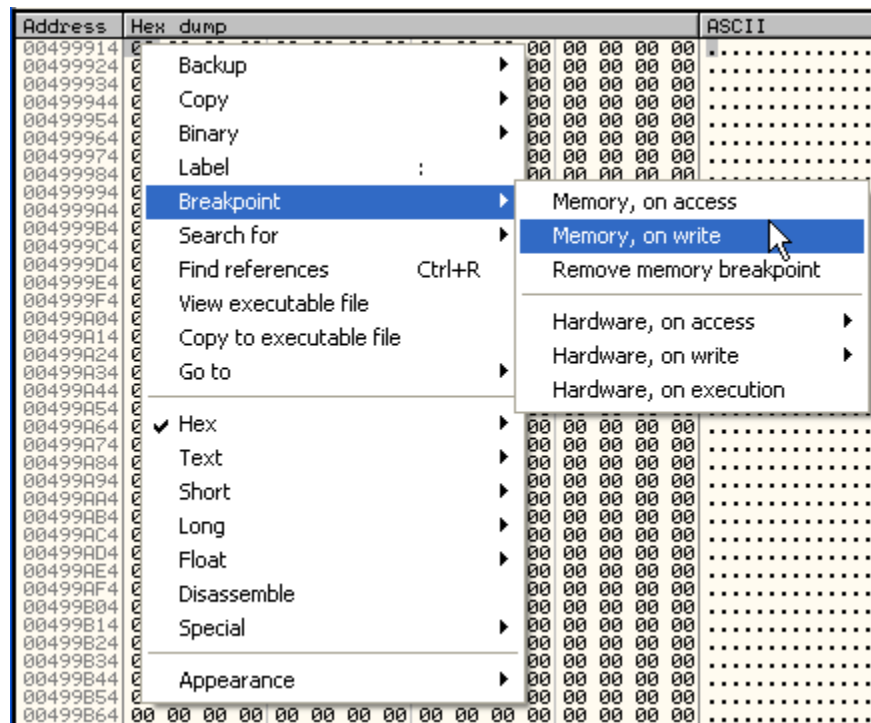
As general approach, in order to carry out inline patching, we've to found the OEP (or an area in proximity of same) of the application. This task depend from the packer used during the target protection, but for ASProtect we can use the exceptions methods and with it is easy found it to be in:

00499914	55	PUSH EBP	OEP
00499915	8BEC	MOV EBP,ESP	
00499917	83C4 EC	ADD ESP,-14	
0049991A	53	PUSH EBX	
0049991B	33C0	XOR EAX,EAX	
0049991D	8945 EC	MOV DWORD PTR SS:[EBP-14],EAX	
00499920	B8 24964900	MOV EAX,chordpic.00499624	
00499925	E8 86CDF6FF	CALL chordpic.004066B0	
0049992A	8B1D F0E04900	MOV EBX,DWORD PTR DS:[49E0F0]	chordpic.0049FBC0
00499930	33C0	XOR EAX,EAX	
00499932	55	PUSH EBP	
00499933	68 4E9A4900	PUSH chordpic.00499A4E	
00499938	64:FF30	PUSH DWORD PTR FS:[EAX]	

Now restart the target and look at the OllyDbg dump Window (the bottom area), press the CTRL+G keys and write the OEP address 0x00499914, in this way tries to characterize from which zone of memory of ASProtect it comes written the code that leaves from the OEP.



All the address at the OEP is a null byte since the code of the program must still be decompressed from the packer stub, then put a breakpoint on memory write into the byte located at the OEP:



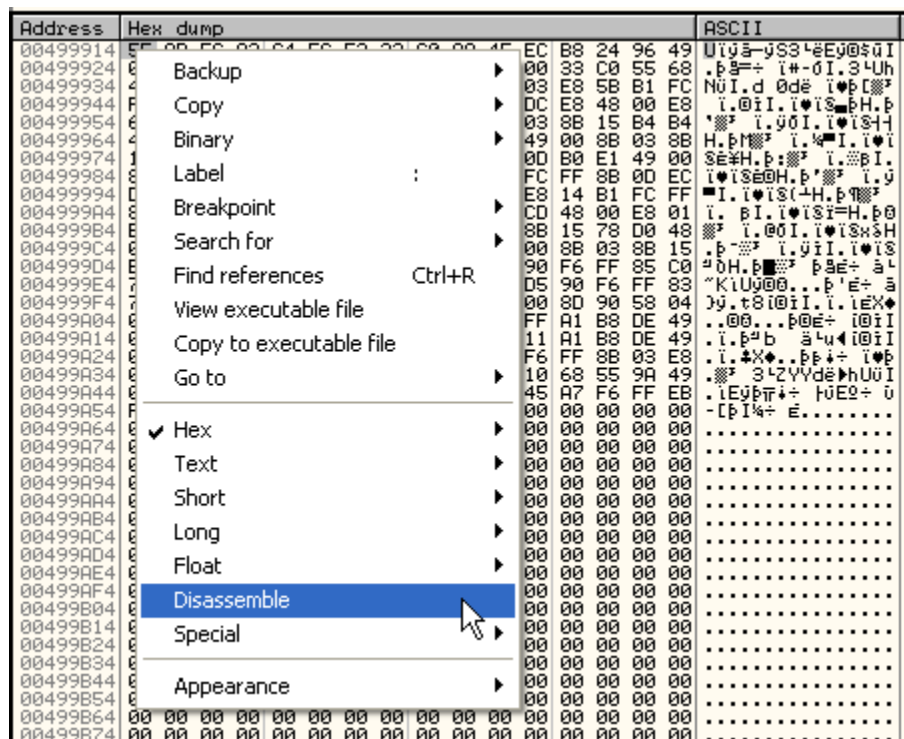
Now press ALT+O to show the OllyDbg options dialog and check all the exceptions then press Shift+F9 to run the target. After a little OllyDbg will be break in this part of code:

00B72663	F3:A5	REP MOVS DWORD PTR ES:[EDI],DWORD PTR DS:[ESI]
00B72665	89C1	MOV ECX,EAX
00B72667	83E1 03	AND ECX,3
00B7266A	F3:A4	REP MOVS BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]
00B7266C	5F	POP EDI
00B7266D	5E	POP ESI
00B7266E	C3	RETN
00B7266F	8D740E FC	LEA ESI,DWORD PTR DS:[ESI+ECX-4]

pressing F8 once the writing loop is executed, then look into the dump window:

Address	Hex dump	ASCII
00499914	55 8B EC 83 C4 EC 53 33 C0 89 45 EC B8 24 96 49	Ug3-gS3tEg0sUI
00499924	00 E8 86 CD F6 FF 8B 10 F0 E0 49 00 33 C0 55 68	.p3=i#-0I.3Uh
00499934	4E 9A 49 00 64 FF 30 64 89 20 8B 03 E8 5B B1 FC	NuId 0de i0p[
00499944	FF 8B 0D B8 DE 49 00 8B 03 8B 15 DC E8 48 00 E8	i.0iI.i0is.pH.p
00499954	60 B1 FC FF 8B 0D 98 E2 49 00 8B 03 8B 15 B4 B4	' i.00I.i0is-H
00499964	48 00 E8 40 B1 FC FF 8B 0D AC DF 49 00 8B 03 8B	H.p[ i.%I.i0i
00499974	15 D4 BE 48 00 E8 3A B1 FC FF 8B 0D B0 E1 49 00	SE#H.p: i.0BI.
00499984	8B 03 8B 15 D4 B8 48 00 E8 27 B1 FC FF 8B 0D EC	i0is0H.p' i.0
00499994	DF 49 00 8B 03 8B 15 28 C1 48 00 E8 14 B1 FC FF	I.i0is(-H.p[
004999A4	8B 0D 20 E1 49 00 8B 03 8B 15 D8 CD 48 00 E8 01	i. pI.i0is=H.p0
004999B4	B1 FC FF 8B 0D 40 E2 49 00 8B 03 8B 15 78 D0 48	' i.00I.i0isx3H
004999C4	00 E8 EE 80 FC FF 8B 0D 98 DE 49 00 8B 03 8B 15	.p- i.0iI.i0is
004999D4	BC E3 48 00 E8 D8 B0 FC FF E8 86 90 F6 FF 85 C0	0H.p[ p3e+ a
004999E4	7E 48 8D 55 EC B8 01 00 00 E8 D5 90 F6 FF 83	"KiUg00...p'e+ a
004999F4	7D EC 00 74 38 A1 B8 DE 49 00 8B 0D 8D 90 58 04	09.t8i0iI.i.iEX+
00499A04	00 00 B8 01 00 00 00 E8 B8 90 F6 FF A1 B8 DE 49	..00...p0e+ i0iI
00499A14	00 8B 00 E8 BC 62 FF FF C4 C0 75 11 A1 B8 DE 49	.i.p#b. a'u+ i0iI
00499A24	00 8B 00 55 58 04 00 00 E8 E7 AD F6 FF 8B 03 E8	.i.*X+.p3+ i0p
00499A34	00 B1 FC FF 33 C0 5A 59 59 64 89 10 68 55 9A 49	. i.3'ZYVdehUUI
00499A44	00 8D 45 EC E8 CB AD F6 FF C3 E9 45 A7 F6 FF EB	.iEy0p+ +0EQ+ 0
00499A54	F0 5B E8 49 AC F6 FF 90 00 00 00 00 00 00 00	-[pI%+ e.....
00499A64	00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....

To see the disassembled view:



Obtaining:

Address	Hex dump	Disassembly
00499914	55	DB 55
00499915	8B	DB 8B
00499916	EC	DB EC
00499917	83	DB 83
00499918	C4	DB C4
00499919	EC	DB EC
0049991A	53	DB 53
0049991B	33	DB 33
0049991C	C0	DB C0
0049991D	89	DB 89
0049991E	45	DB 45

The code obviously is not in a clear way since OllyDbg could not have analysed it from the action of the first target loading since the code has not been still decompressed from the packer stub. Therefore goes into the OllyDbg code window, press CTRL+G and writes the address from where we've to start the analysis (0x00499914) and presses Enter, then perform the code analysis by pressing CTRL+A keys:

00499914	55	PUSH EBP
00499915	8BEC	MOV EBP,ESP
00499917	83C4 EC	ADD ESP,-14
0049991A	53	PUSH EBX
0049991B	33C0	XOR EAX,EAX
0049991D	8945 EC	MOV DWORD PTR SS:[EBP-14],EAX
00499920	B8 24964900	MOV EAX,chordpic.00499624
00499925	E8 86CDF6FF	CALL chordpic.004066B0
0049992A	8B1D F0E04900	MOV EBX,DWORD PTR DS:[49E0F0]
00499930	33C0	XOR EAX,EAX
00499932	55	PUSH EBP
00499933	68 4E9A4900	PUSH chordpic.00499A4E

Bingo, the code has been decompressed and corresponds to what we have found after jumping the exceptions previously.

To this point we know that the writing of the code at OEP happens from one memory area that leaves from (ALT+M in order to visualize the map of memory of the process in execution):

00B40000	00001000				Priv	RWE	RWE	
00B50000	00001000				Priv	RWE	RWE	
00B60000	00001000				Priv	RWE	RWE	
00B70000	00033000				Priv	RWE	RWE	
00B80000	00004000				Priv	RW		
00B90000	0009C000				Priv	RW		
00CB0000	00050000				Map	R	R	
00D00000	00001000				Priv	RWE	RWE	

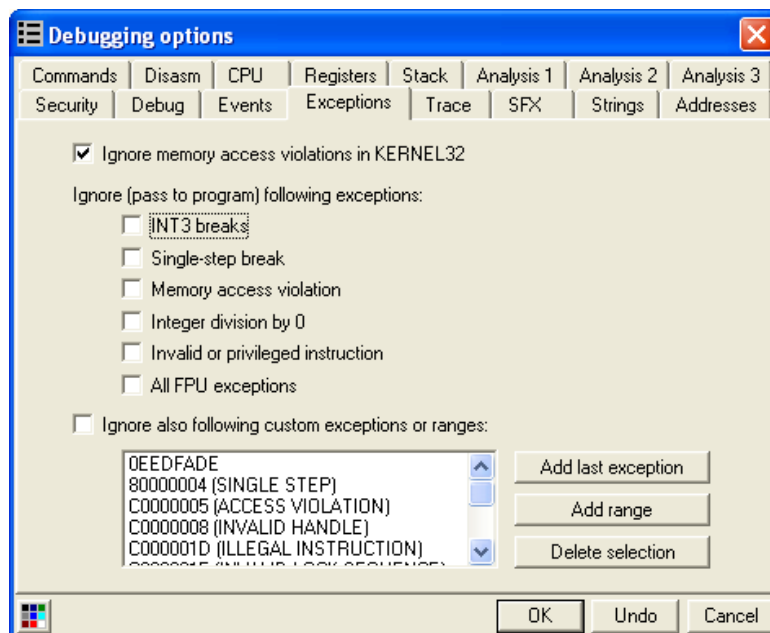
This area does not belong to the already existing sections on disc since it is not contemplated in the PE-Header and it is exactly an area allotted in dynamic way from the packer that before it demands memory to the system and later on fills up it with the code that then will go to execute and that in the event analysed it is taken care to decompress the code of the program.

Since this memory area will be set a runtime we can't find it when the target is load at the entry point, but we've to find when and in what point of the code this is allocated in order to trace it with the inline code, to do all the stuff we need to carry out a minimum reversing of the packer.

The first things then will be in finding some entry point from the stub of ASProtect that concurs to maintain memory to us of the first allocation of memory carried out from the packer.

Restart the target with CTRL+F2.

Uncheck all the exceptions (ALT+O) unless the first one:



Press CTRL+G and write **VirtualAlloc** that is the API used in order to demand memory to the system, this API is widely used from many packer and of course also ASProtect use it in order to allot the memory areas that they will come filled up with the code of the packer itself.

7C809A81	8BFF	MOV EDI,EDI	<b>API Entry Point</b>
7C809A83	55	PUSH EBP	
7C809A84	8BEC	MOV EBP,ESP	
7C809A86	FF75 14	PUSH DWORD PTR SS:[EBP+14]	
7C809A89	FF75 10	PUSH DWORD PTR SS:[EBP+10]	
7C809A8C	FF75 0C	PUSH DWORD PTR SS:[EBP+C]	
7C809A8F	FF75 08	PUSH DWORD PTR SS:[EBP+8]	
7C809A92	6A FF	PUSH -1	
7C809A94	E8 09000000	CALL kernel32.VirtualAllocEx	
7C809A99	5D	POP EBP	
7C809A9A	C2 1000	RETN 10	
7C809A9D	90	NOP	
7C809A9E	90	NOP	
7C809A9F	90	NOP	

Press Shift+F9 and OllyDbg will be break into the API, press ALT+F9 to go back into the caller code and take also a look of the address contained in EAX:

004EB4C1	43	INC EBX	
004EB4C2	BB 3219122A	MOV EBX,2A121932	
004EB4C7	00B2 5C332589	ADD BYTE PTR DS:[EDX+8925335C],DH	
004EB4CD	A5	MOVS DWORD PTR ES:[EDI],DWORD PTR DS:[ESI]	
004EB4CE	290400	SUB DWORD PTR DS:[EAX+EAX],EAX	
004EB4D1	006A 40	ADD BYTE PTR DS:[EDX+40],CH	
004EB4D4	68 00100000	PUSH 1000	
004EB4D9	FFB5 08040000	PUSH DWORD PTR SS:[EBP+408]	
004EB4DF	6A 00	PUSH 0	
004EB4E1	FF95 F0030000	CALL NEAR DWORD PTR SS:[EBP+3F0]	Richiama VirtualAlloc per la prima volta
004EB4E7	8985 CC010000	MOV DWORD PTR SS:[EBP+1CC],EAX	
004EB4ED	8B9D 00040000	MOV EBX,DWORD PTR SS:[EBP+400]	
004EB4F3	039D 00040000	ADD EBX,DWORD PTR SS:[EBP+400]	
004EB4F9	50	PUSH EAX	
004EB4FA	53	PUSH EBX	

#### NOTE

In my machine at the time of writing this tutorial EAX is equal to 0x00B30000, but take care about this value, generally speaking this one may be quite different in other machine since this is one is related to a run-time memory area and not a fixed absolute address.

The memory map of the of process under investigation (ALT+M) turns out following:

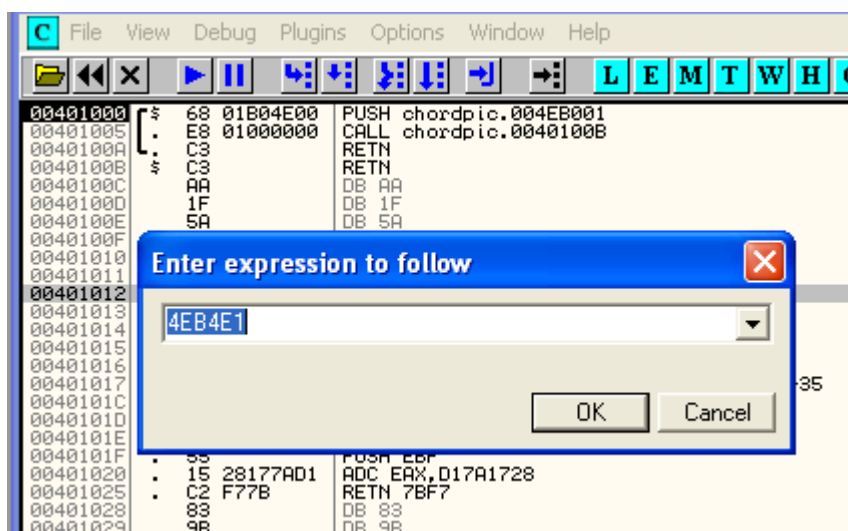
003F0000	00002000			Map	R	E	
00400000	00001000	chordpic	PE header	Imag	R	RWE	
00401000	00099000	chordpic	code	Imag	R	RWE	
0049A000	00005000	chordpic	data	Imag	R	RWE	
0049F000	00002000	chordpic		Imag	R	RWE	
004A1000	00003000	chordpic		Imag	R	RWE	
004A4000	00001000	chordpic		Imag	R	RWE	
004A5000	00001000	chordpic		Imag	R	RWE	
004A6000	00001000	chordpic		Imag	R	RWE	
004A7000	0000A000	chordpic		Imag	R	RWE	
004B1000	0003A000	chordpic	.rsrc	Imag	R	RWE	
004EB000	00022000	chordpic	.data	Imag	R	RWE	
0050D000	00001000	chordpic	.adata	Imag	R	RWE	
00510000	00103000			Map	R	R	
00620000	00001000			Priv	RW	RW	
00630000	00129000			Map	R	E	
00930000	00001000			Priv	RW	RW	
00940000	00004000			Priv	RW	RW	
00950000	00003000			Map	R	R	
00960000	00003000			Priv	RW	RW	
00970000	00002000			Map	R	R	
00980000	00001000			Priv	RW	RW	
00990000	00004000			Priv	RW	RW	
00B30000	00033000			Priv	RWE	RWE	
011A0000	00002000			Map	R	R	
5A000000	00001000	swpg	PE header	Imag	R	RWE	
5A001000	00012000	swpg	code	Imag	R	RWE	
5A013000	00001000	swpg	DATA	Imag	R	RWE	
5A014000	00001000	swpg	BSS	Imag	R	RWE	
5A015000	00001000	swpg	.idata	Imag	R	RWE	
5A016000	00001000	swpg	.reloc	Imag	R	RWE	
5A017000	00001000	swpg	.rsrc	Imag	R	RWE	
5F000000	00001000			Priv	RWE	RWE	

As we can note code relating to the OEP decompression isn't still present, this means that this area is build with a layer of code which is not yet decompressed then we've to find it.

Now we have to see if the caller code for the **VirtualAlloc** API is available when the target is at own entry-point or if we've to surf again through some other layer of the code before reach them.

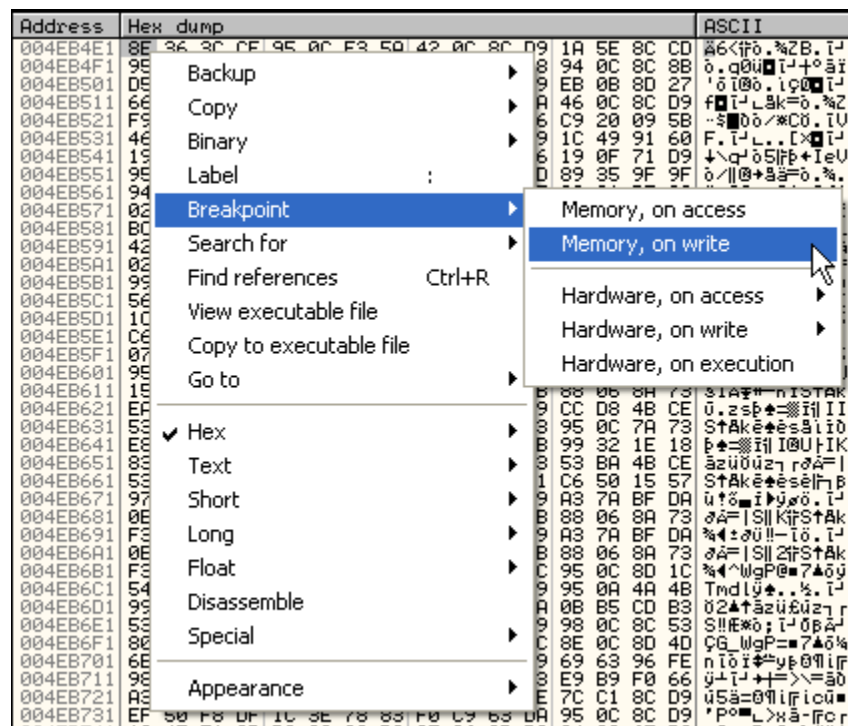


Press CTRL+F2 and go to see that what we find to the address 0x004EB4E1:



004EB4E1	8E36	MOV SEG?,WORD PTR DS:[ESI]	Richiama VirtualAlloc per la prima volta
004EB4E3	3C CE	CMP AL,0CE	
004EB4E5	95	XCHG EAX,EBP	
004EB4E6	0C F3	OR AL,0F3	
004EB4E8	5A	POP EDX	
004EB4E9	42	INC EDX	
004EB4EA	0C 8C	OR AL,8C	
004EB4EC	D91A	FSTP DWORD PTR DS:[EDX]	

This piece of code is quite different than the one that we've show above, this means that ASProtect executes some preliminary code decryptions before calling the **VirtualAlloc**, then we have to found where this code comes written. To do this search we have to restart OllyDbg and put a memory breakpoint in writing to the address 0x004EB4E1 to the aim to characterize the point in which the code it comes effectively written.



Press Shift+F9 and OllyDbg break in 0x004EB13F, the next call, if executed decrypt all the code.

004EB13F	8F0413	POP DWORD PTR DS:[EBX+EDX]	
004EB142	E8 0F000000	CALL chordpic.004EB156	Decrittazione di tutto il codice a 0x004EB4E1
004EB147	1BB8 91F6F764	SBB EDI, DWORD PTR DS:[EAX+64F7F691]	
004EB14D	CD 82	INT 82	
004EB14F	93	XCHG EAX, EBX	

Now restart OllyDbg with CTRL+F2 and from the entry point we go to see what we have on the address 0x004EB142, yup, now the code is exactly which would have to be, therefore we have found a good starting point to trace over the code.

Make some step by using F7, it easy to note that we're into a decompression loop, this look more clear after some little code manipulation by using some NOP instruction.



004EB0F6	81C3 74080000	ADD EBX,874	
004EB0FC	BF F8DF4C62	MOV EDI,624C0FF8	
004EB101	2BD2	SUB EDX,EDX	
004EB103	81EE C23C2260	SUB ESI,60223CC2	
004EB109	FF341A	PUSH DWORD PTR DS:[EDX+EBX]	LOOP START
004EB10C	59	POP ECX	
004EB10D	E8 0D000000	CALL chordpic.004EB11F	
004EB112	41	INC ECX	
004EB113	E6 27	OUT 27,AL	I/O command
004EB115	D4 7D	AAM 7D	
004EB117	72 C3	JB SHORT chordpic.004EB0DC	
004EB119	40	INC EAX	
004EB11A	79 BE	JNS SHORT chordpic.004EB0DA	
004EB11C	1F	POP DS	Modification of segment register
004EB11D	6C	INS BYTE PTR ES:[EDI],DX	I/O command
004EB11E	90	NOP	
004EB11F	66:8BF9	MOV DI,CX	
004EB122	5E	POP ESI	
004EB123	81E9 7C9A3773	SUB ECX,73379A7C	
004EB129	52	PUSH EDX	
004EB12A	8AE1	MOV AH,CL	
004EB12C	5E	POP ESI	
004EB12D	81E9 05BFA677	SUB ECX,77A6BF05	
004EB133	81E9 5A0B1143	SUB ECX,43110B5A	
004EB139	51	PUSH ECX	
004EB13A	BE FF2D774A	MOV ESI,4A772DFF	
004EB13F	8F0413	POP DWORD PTR DS:[EBX+EDX]	
004EB142	E8 0F000000	CALL chordpic.004EB156	
004EB147	1BB8 91F6F764	SBB EDI,DWORD PTR DS:[EAX+64F7F691]	
004EB14D	CD 82	INT 82	
004EB14F	93	XCHG EAX,EBX	
004EB150	D0C9	ROR CL,1	
004EB152	CE	INTO	
004EB153	EF	OUT DX,EAX	I/O command
004EB154	FC	CLD	
004EB155	90	NOP	
004EB156	66:BF A6C8	MOV DI,0C8A6	
004EB15A	5E	POP ESI	
004EB15B	66:BE DF97	MOV SI,97DF	
004EB15F	83EA 03	SUB EDX,3	
004EB162	66:BE 719D	MOV SI,9D71	
004EB166	4A	DEC EDX	
004EB167	66:81DE E2E2	SBB SI,0E2E2	
004EB16C	81FA 78F8FFFF	CMP EDX,-788	
004EB172	0F85 3C000000	JNZ chordpic.004EB1B4	Non eseguito alla fine del loop
004EB178	E8 13000000	CALL chordpic.004EB190	Primo salto per fine loop
004EB17D	CF	IRETD	
004EB17E	5C	POP ESP	
004EB17F	65:3AEB	CMP CH,BL	Superfluous prefix
004EB182	48	DEC EAX	
004EB183	E1 06	LOOPDE SHORT chordpic.004EB18B	
004EB185	C7	???	Unknown command
004EB186	F4	HLT	Privileged command
004EB187	1D 92636019	SBB EAX,19606392	
004EB18C	90	NOP	
004EB18D	90	NOP	
004EB18E	90	NOP	
004EB18F	90	NOP	
004EB190	E8 0D000000	CALL chordpic.004EB1A2	
004EB195	51	PUSH ECX	
004EB196	B6 B7	MOV DH,0B7	
004EB198	24 8D	AND AL,8D	
004EB19A	42	INC EDX	
004EB19B	53	PUSH EBX	
004EB19C	90	NOP	
004EB19D	898E AFBC4568	MOV DWORD PTR DS:[ESI+6845BCAF],ECX	
004EB1A3	A8 20	TEST AL,20	
004EB1A5	E1 35	LOOPDE SHORT chordpic.004EB1DC	
004EB1A7	5E	POP ESI	
004EB1A8	5E	POP ESI	
004EB1A9	5F	POP EDI	
004EB1AA	E9 1B000000	JMP chordpic.004EB1CA	USCITA DAL LOOP
004EB1AF	F9	STC	
004EB1B0	3E:9F	LAHF	Superfluous prefix
004EB1B2	EC	IN AL,DX	I/O command
004EB1B3	90	NOP	
004EB1B4	80D4 F0	ADC AH,0F0	
004EB1B7	E9 4DFFFFFF	JMP chordpic.004EB109	LOOP END
004EB1BC	EE	OUT DX,AL	I/O command
004EB1BD	8F	???	Unknown command
004EB1BE	1C 25	SBB AL,25	
004EB1C0	FA	CLI	
004EB1C1	AB	STOS DWORD PTR ES:[EDI]	

The loop exit point can be located at the address 0x004EB1AA, puts therefore a memory breakpoint at this address and press Shift+F9 to run all the loop, when OllyDbg stop look at the code after the JMP destination, this code should be changed due to the decompression loop.

004EB0F6	81C3 74080000	ADD EBX,874	
004EB0FC	BF F8DF4C62	MOV EDI,624C0FF8	
004EB101	2B02	SUB EDX,EDX	
004EB103	81EE C23C2260	SUB ESI,60223CC2	
004EB109	FF341A	PUSH DWORD PTR DS:[EDX+EBX]	LOOP START
004EB10C	59	POP ECX	
004EB10D	E8 0D000000	CALL chordpic.004EB11F	
004EB112	41	INC ECX	
004EB113	E6 27	OUT 27,AL	I/O command
004EB115	04 7D	AAM 7D	
004EB117	72 C3	JB SHORT chordpic.004EB0DC	
004EB119	40	INC EAX	
004EB11A	79 BE	JNS SHORT chordpic.004EB0DA	
004EB11C	1F	POP DS	Modification of segment register
004EB11D	6C	INS BYTE PTR ES:[EDI],DX	I/O command
004EB11E	90	NOP	
004EB11F	66:8BF9	MOV DI,CX	
004EB122	5E	POP ESI	
004EB123	81E9 7C9A3773	SUB ECX,73379A7C	
004EB129	52	PUSH EDX	
004EB12A	8AE1	MOV AH,CL	
004EB12C	5E	POP ESI	
004EB12D	81E9 05BFA677	SUB ECX,77A6BF05	
004EB133	81E9 5A0B1143	SUB ECX,43110B5A	
004EB139	51	PUSH ECX	
004EB13A	BE FF2D774A	MOV ESI,4A772DFF	
004EB13F	8F0413	POP DWORD PTR DS:[EBX+EDX]	
004EB142	E8 0F000000	CALL chordpic.004EB156	
004EB147	1BB8 91F6F764	SBB EDI,DWORD PTR DS:[EAX+64F7F691]	
004EB14D	CD 82	INT 82	
004EB14F	93	XCHG EAX,EBX	
004EB150	D0C9	ROR CL,1	
004EB152	CE	INTO	
004EB153	EF	OUT DX,EAX	I/O command
004EB154	FC	CLD	
004EB155	90	NOP	
004EB156	66:BF A6C8	MOV DI,0C8A6	
004EB15A	5E	POP ESI	
004EB15B	66:BE DF97	MOV SI,97DF	
004EB15F	83EA 03	SUB EDX,3	
004EB162	66:BE 719D	MOV SI,9D71	
004EB166	4A	DEC EDX	
004EB167	66:81DE E2E2	SBB SI,0E2E2	
004EB16C	81FA 78F8FFFF	CMP EDX,-788	
004EB172	0F85 3C000000	JNZ chordpic.004EB1B4	Non eseguito alla fine del loop
004EB178	E8 13000000	CALL chordpic.004EB190	Primo salto per fine loop
004EB17D	CF	IRETD	
004EB17E	5C	POP ESP	
004EB17F	65:3AEB	CMP CH,BL	Superfluous prefix
004EB182	48	DEC EAX	
004EB183	E1 06	LOOPDE SHORT chordpic.004EB18B	
004EB185	C7	???	Unknown command
004EB186	F4	HLT	Privileged command
004EB187	1D 92636019	SBB EAX,19606392	
004EB18C	90	NOP	
004EB18D	90	NOP	
004EB18E	90	NOP	
004EB18F	90	NOP	
004EB190	E8 0D000000	CALL chordpic.004EB1A2	
004EB195	51	PUSH ECX	
004EB196	B6 B7	MOV DH,0B7	
004EB198	24 8D	AND AL,8D	
004EB19A	42	INC EDX	
004EB19B	53	PUSH EBX	
004EB19C	90	NOP	
004EB19D	898E AFBC4568	MOV DWORD PTR DS:[ESI+6845BCAF],ECX	
004EB1A3	A8 20	TEST AL,20	
004EB1A5	E1 35	LOOPDE SHORT chordpic.004EB1DC	
004EB1A7	5E	POP ESI	
004EB1A8	5E	POP ESI	
004EB1A9	5F	POP EDI	
004EB1AA	E9 1B000000	JMP chordpic.004EB1CA	USCITA DAL LOOP
004EB1AF	F9	STC	
004EB1B0	3E:9F	LAHF	Superfluous prefix
004EB1B2	EC	IN AL,DX	I/O command
004EB1B3	90	NOP	
004EB1B4	80D4 F0	ADC AH,0F0	
004EB1B7	E9 4DFFFFFF	JMP chordpic.004EB109	LOOP END
004EB1BC	EE	OUT DX,AL	I/O command
004EB1BD	8F	???	Unknown command
004EB1BE	1C 25	SBB AL,25	
004EB1C0	FA	CLI	
004EB1C1	AB	STOS DWORD PTR ES:[EDI]	
004EB1C2	08A1 C687B4DD	OR BYTE PTR DS:[ECX+DDB487C6],AH	
004EB1C8	52	PUSH EDX	
004EB1C9	2366 81	AND ESP,DWORD PTR DS:[ESI-7F]	
004EB1CC	DFC5	FFREEP ST(5)	
004EB1CE	9C	PUSHFD	
004EB1CF	E8 0F000000	CALL chordpic.004EB1E3	

Now we've to restart again OllyDbg with CTRL+F2 and go straight to see the code to the address 0x004EB1AA:

004EB1AA	E9 1B000000	JMP chordpic.004EB1CA	USCITA DAL LOOP
004EB1AF	F9	STC	
004EB1B0	3E:9F	LAHF	Superfluous prefix
004EB1B2	EC	IN AL,DX	I/O command
004EB1B3	B5 80	MOV CH,80	
004EB1B5	D4 F0	AAM 0F0	
004EB1B7	E9 40FFFFFF	JMP chordpic.004EB109	LOOP END
004EB1BC	EE	OUT DX,AL	I/O command
004EB1BD	8F	???	Unknown command
004EB1BE	1C 25	SBB AL,25	
004EB1C0	FA	CLI	
004EB1C1	AB	STOS DWORD PTR ES:[EDI]	
004EB1C2	08A1 C687B4DD	OR BYTE PTR DS:[ECX+DD8487C6],AH	
004EB1C8	52	PUSH EDX	
004EB1C9	2341 E6	AND EAX,DWORD PTR DS:[ECX-1A]	
004EB1CC	CE	INT0	
004EB1CD	F3:	PREFIX REP:	Superfluous prefix
004EB1CE	77 40	JA SHORT chordpic.004EB21D	
004EB1D0	FF2D DB643A56	JMP FAR FWORD PTR DS:[563A64DB]	Far jump
004EB1D6	1C 4B	SBB AL,4B	
004EB1D8	17	POP SS	Modification of segment register
004EB1D9	0258 D7	ADD BL,BYTE PTR DS:[EAX-29]	
004EB1DC	B2 6E	MOV DL,6E	
004EB1DE	54	PUSH ESP	
004EB1DF	230F	AND ECX,DWORD PTR DS:[EDI]	
004EB1E1	9A 10B6D73A DB	CALL FAR 64DB:3AD7B610	Far call
004EB1E8	EF	OUT DX,EAX	I/O command
004EB1E9	31C8	XOR EAX,ECX	

Yup we have the code in a clear way, this means that this jump was untouched from the target Entry Point and carried out to the end of the first decryption loop, this is a very good stuff for us because this is our first redirection point to our patching cave area, then take a note for this address.

#### **END OF THE LOOP #1 (redirection #1): 0x004EB1AA** **ORIGINAL INSTRUCTION: JMP 0x004EB1CA**

After the loop execution we have also to check if the code to the address of the first **VirtualAlloc** API call has been effectively written.

Then simply press CTRL+G and write 0x004EB4E1, the code for our misfortune still has not been decrypted, then we've to step through the code by using F7 and after some instructions we newly meet a decryption loop with some scrambled code.

004EB1B7	^ E9 4DFFFFFF	JMP chordpic.004EB109	LOOP END
004EB1BC	EE	OUT DX,AL	I/O command
004EB1BD	8F	???	Unknown command
004EB1BE	1C 25	SBB AL,25	
004EB1C0	FA	CLI	
004EB1C1	AB	STOS DWORD PTR ES:[EDI]	
004EB1C2	08A1 C687B4D0	OR BYTE PTR DS:[ECX+DDB487C6],AH	
004EB1C8	52	PUSH EDX	
004EB1C9	90	NOP	
004EB1CA	66:81DF C59C	SBB DI,9CC5	
004EB1CF	E8 0F000000	CALL chordpic.004EB1E3	
004EB1D4	4B	DEC EBX	
004EB1D5	2841 E6	SUB BYTE PTR DS:[ECX-1A],AL	
004EB1D8	27	DAA	
004EB1D9	D4 7D	ARM 7D	
004EB1DB	72 C3	JB SHORT chordpic.004EB1A0	
004EB1DD	40	INC EAX	
004EB1DE	^ 79 BE	JNS SHORT chordpic.004EB19E	
004EB1E0	1F	POP DS	Modification of segment register
004EB1E1	6C	INS BYTE PTR ES:[EDI],DX	I/O command
004EB1E2	90	NOP	
004EB1E3	51	PUSH ECX	
004EB1E4	E8 0C000000	CALL chordpic.004EB1F5	
004EB1E9	04 ED	ADD AL,0ED	
004EB1EB	22B3 70E96E0F	AND DH,BYTE PTR DS:[EBX+F6EE970]	
004EB1F1	9C	PUSHFD	
004EB1F2	A5	MOVS DWORD PTR ES:[EDI],DWORD PTR	
004EB1F3	7A 2B	JPE SHORT chordpic.004EB220	
004EB1F5	66:8BDE	MOV BX,SI	
004EB1F8	58	POP EAX	
004EB1F9	5B	POP EBX	
004EB1FA	5E	POP ESI	
004EB1FB	B0 7E	MOV AL,7E	
004EB1FD	81C6 7C070000	ADD ESI,77C	
004EB203	80F7 93	XOR BH,93	
004EB206	68 B1010000	PUSH 1B1	
004EB20B	81E3 8516025A	AND EBX,5A021685	
004EB211	59	POP ECX	
004EB212	8ADD	MOV BL,CH	
004EB214	FF36	PUSH DWORD PTR DS:[ESI]	
004EB216	B0 5A	MOV AL,5A	
004EB218	5A	POP EDX	
004EB219	81C2 5B6F3451	ADD EDX,51346F5B	
004EB21F	^ 0F83 05000000	JNB chordpic.004EB22A	
004EB225	53	PUSH EBX	
004EB226	80C4 30	ADD AH,30	
004EB229	5B	POP EBX	
004EB22A	81F2 F853A348	XOR EDX,48A353F8	
004EB230	81C2 D180F01A	ADD EDX,1AF080D1	
004EB236	E8 0D000000	CALL chordpic.004EB248	
004EB23B	^ E1 06	LOOPDE SHORT chordpic.004EB243	
004EB23D	C7	???	Unknown command
004EB23E	F4	HLT	Privileged command
004EB23F	1D 92636019	SBB EAX,19606392	
004EB244	90	NOP	
004EB245	90	NOP	
004EB246	90	NOP	
004EB247	90	NOP	
004EB248	8AD8	MOV BL,AL	
004EB24A	5B	POP EBX	
004EB24B	8916	MOV DWORD PTR DS:[ESI],EDX	
004EB24D	83EE 03	SUB ESI,3	
004EB250	BF 661F5A5E	MOV EDI,5E5A1F66	
004EB255	4E	DEC ESI	
004EB256	66:81CB 43C2	OR BX,0C243	
004EB258	49	DEC ECX	
004EB25C	^ 0F85 19000000	JNZ chordpic.004EB27B	
004EB262	B8 B5D4A229	MOV EAX,29A2D4B5	
004EB267	^ E9 24000000	JMP chordpic.004EB290	USCITA DAL LOOP#2
004EB26C	BB D8311697	MOV EBX,971631D8	
004EB271	846D A2	TEST BYTE PTR SS:[EBP-5E],CH	
004EB274	33F0	XOR ESI,EAX	
004EB276	90	NOP	
004EB277	90	NOP	
004EB278	90	NOP	
004EB279	90	NOP	
004EB27A	90	NOP	
004EB27B	^ 0F87 02000000	JA chordpic.004EB283	
004EB281	8BC2	MOV EAX,EDX	
004EB283	^ E9 8CFFFFFF	JMP chordpic.004EB214	
004EB288	20D9	AND CL,BL	
004EB28A	9E	SAHF	
004EB28B	^ 7F 4C	JG SHORT chordpic.004EB2D9	
004EB28D	95	XCHG EAX,EBP	
004EB28E	AA	STOS BYTE PTR ES:[EDI]	
004EB28F	9B	WAIT	
004EB290	BA E5429C47	MOV EDX,479C42E5	
004EB295	C585 5C7CBD1A	LDS EAX,FWORD PTR SS:[EBP+1ABD7C5C]	Modification of segment register

At the address 0x004EB267 we've the loop exit point, therefore we can simply put a software breakpoint to this address (F2) and press Shift+F9 in order to execute all the loop in a fast way.

004EB23B	✓ E1 06	LOOPDE SHORT chordpic.004EB243	
004EB23D	C7	???	Unknown command
004EB23E	F4	HLT	Privileged command
004EB23F	1D 92636019	SBB EAX,19606392	
004EB244	90	NOP	
004EB245	90	NOP	
004EB246	90	NOP	
004EB247	90	NOP	
004EB248	8AD8	MOV BL,AL	
004EB24A	5B	POP EBX	
004EB24B	8916	MOV DWORD PTR DS:[ESI],EDX	
004EB24D	83EE 03	SUB ESI,3	
004EB250	BF 661F5A5E	MOV EDI,5E5A1F66	
004EB255	4E	DEC ESI	
004EB256	66:81CB 43C2	OR BX,0C243	
004EB25B	49	DEC ECX	
004EB25C	✓ 0F85 19000000	JNZ chordpic.004EB27B	
004EB262	B8 B5D4A229	MOV EAX,29A2D4B5	
004EB267	✓ E9 24000000	JMP chordpic.004EB290	USCITA DAL LOOP#2
004EB26C	BB 08311697	MOV EBX,97163108	
004EB271	846D A2	TEST BYTE PTR SS:[EBP-5E1],CH	
004EB274	33F0	XOR ESI,EAX	
004EB276	90	NOP	
004EB277	90	NOP	
004EB278	90	NOP	
004EB279	90	NOP	
004EB27A	90	NOP	
004EB27B	✓ 0F87 02000000	JA chordpic.004EB283	
004EB281	8BC2	MOV EAX,EDX	
004EB283	^ E9 8CFFFFFF	JMP chordpic.004EB214	
004EB288	20D9	AND CL,BL	
004EB28A	9E	SAHF	
004EB28B	✓ 7F 4C	JG SHORT chordpic.004EB2D9	
004EB28D	95	XCHG EAX,EBP	
004EB28E	AA	STOS BYTE PTR ES:[EDI]	
004EB28F	9B	WAIT	
004EB290	BE 87C4C02B	MOV ESI,2BC0C487	
004EB295	E8 09000000	CALL chordpic.004EB2A3	
004EB296	0000 0000	RET 00000000	

As usual take a note of the loop exit address, this is the second redirection point to the patch cave.

**END OF THE LOOP #2 (redirection #2): 0x004EB267**  
**INSTRUCTION ORIGINATES THEM: JMP 0x004EB290**

Also as we've do previously, take a look if the caller code for the **VirtualAlloc** API has been written or not, but the searched code isn't still present then we've to step again a little.

We've another decompression loop:

004EB267	✓ E9 24000000	JMP chordpic.004EB290	USCITA DAL LOOP#2
004EB26C	BB D8311697	MOV EBX,97163108	
004EB271	846D A2	TEST BYTE PTR SS:[EBP-5E],CH	
004EB274	33F0	XOR ESI,EAX	
004EB276	69EE 8F1C250F	IMUL EBP,ESI,0F251C8F	
004EB27C	8702	XCHG DWORD PTR DS:[EDX],EAX	
004EB27E	0000	ADD BYTE PTR DS:[EAX],AL	
004EB280	008B C2E98CFF	ADD BYTE PTR DS:[EBX+FF8CE9C2],CL	
004EB286	FFFF	???	Unknown command
004EB288	20D9	AND CL,BL	
004EB28A	9E	SAHF	
004EB28B	✓ 7F 4C	JG SHORT chordpic.004EB2D9	
004EB28D	95	XCHG EAX,EBP	
004EB28E	AA	STOS BYTE PTR ES:[EDI]	
004EB28F	9B	WAIT	
004EB290	BE 87C4C02B	MOV ESI,2BC0C487	
004EB295	E8 09000000	CALL chordpic.004EB2A3	
004EB29A	DD52 23	FST QWORD PTR DS:[EDX+23]	
004EB29D	20D9	AND CL,BL	
004EB29F	9E	SAHF	
004EB2A0	✓ 7F 4C	JG SHORT chordpic.004EB2EE	
004EB2A2	95	XCHG EAX,EBP	
004EB2A3	81E2 77DDFD0A	AND EDX,0AFDD077	
004EB2A9	5B	POP EBX	
004EB2AA	68 0221407E	PUSH 7E402102	
004EB2AF	✓ E9 0E000000	JMP chordpic.004EB2C2	
004EB2B4	4E	DEC ESI	
004EB2B5	6F	OUTS DX,DWORD PTR ES:[EDI]	I/O command
004EB2B6	✓ 7C 05	JL SHORT chordpic.004EB2BD	
004EB2B8	5A	POP EDX	
004EB2B9	8B68 81	MOV EBP,DWORD PTR DS:[EAX-7F]	
004EB2BC	26:67:14 BD	ADC AL,0BD	Superfluous prefix
004EB2C0	B2 03	MOV DL,3	
004EB2C2	5B	POP EAX	
004EB2C3	81C3 B5060000	ADD EBX,6B5	
004EB2C9	2BFF	SUB EDI,EDI	
004EB2CB	B8 F170BA61	MOV EAX,61BA70F1	
004EB2D0	8B0C3B	MOV ECX,DWORD PTR DS:[EBX+EDI]	INIZIO DEL LOOP#3
004EB2D3	66:81DE AE88	SBB SI,88AE	
004EB2D8	81F1 F09DD73D	XOR ECX,3DD79DF0	
004EB2DE	81DA 6B18CC6F	SBB EDX,6FCC186B	
004EB2E4	81C1 698F8D1E	ADD ECX,1E8D8F69	
004EB2EA	81E8 128EBB70	SUB EAX,70BB8E12	
004EB2F0	81E9 EE3D2E38	SUB ECX,382E3DEE	
004EB2F6	81F6 0C6A4C13	XOR ESI,134C6A0C	
004EB2FC	51	PUSH ECX	
004EB2FD	0FB7F3	MOVZX ESI,BX	
004EB300	8F043B	POP DWORD PTR DS:[EBX+EDI]	
004EB303	66:81E6 1051	AND SI,5110	
004EB308	66:81F2 4B28	XOR DX,284B	
004EB30D	81EF 41BB3515	SUB EDI,1535BB41	
004EB313	0FB7D2	MOVZX EDX,DX	
004EB316	81C7 3DBB3515	ADD EDI,1535BB3D	
004EB31C	0FB7F3	MOVZX ESI,BX	
004EB31F	81FF 20FAFFFF	CMP EDI,-5E0	
004EB325	✓ 0F85 19000000	JNZ chordpic.004EB344	
004EB32B	8BC6	MOV EAX,ESI	
004EB32D	✓ E9 41000000	JMP chordpic.004EB373	USCITA DAL LOOP#3
004EB332	22B3 70E96E0F	AND DH,BYTE PTR DS:[EBX+F6EE970]	
004EB338	9C	PUSHFD	
004EB339	A5	MOVS DWORD PTR ES:[EDI],DWORD PTR	
004EB33A	7A 2B	JPE SHORT chordpic.004EB367	
004EB33C	8821	MOV BYTE PTR DS:[ECX],AH	
004EB33E	46	INC ESI	
004EB33F	07	POP ES	
004EB340	34 5D	XOR AL,5D	Modification of segment register
004EB342	90	NOP	
004EB343	90	NOP	
004EB344	✓ 0F84 14000000	JE chordpic.004EB35E	
004EB34A	53	PUSH EBX	
004EB34B	✓ E9 0D000000	JMP chordpic.004EB35D	
004EB350	91	XCHG EAX,ECX	
004EB351	F6F7	DIV BH	
004EB353	64:CD 82	INT 82	Superfluous prefix
004EB356	93	XCHG EAX,EBX	
004EB357	D0C9	ROR CL,1	
004EB359	CE	INTO	
004EB35A	EF	OUT DX,EAX	I/O command
004EB35B	FC	CLD	
004EB35C	90	NOP	
004EB35D	58	POP EAX	chordpic.004EB94F
004EB35E	✓ E9 6DFFFFFF	JMP chordpic.004EB2D0	
004EB363	E8 01A6E794	CALL 95365969	
004EB368	3D 32830039	CMP EAX,39008332	
004EB36D	✓ 7E DF	JLE SHORT chordpic.004EB34E	
004EB36F	2C F5	SUB AL,0F5	
004EB371	8AFB	MOV BH,BL	
004EB373	9E	SAHF	
004EB374	25 77247596	AND EAX,96752477	
004EB379	A4	MOVS BYTE PTR ES:[EDI],BYTE PTR DS	

as usual skip the loop by using a memory breakpoint on the exit address for the loop and press Shift+F9 to reach this point.

004EB30D	81EF 41B83515	SUB EDI,1535BB41	
004EB313	0FB7D2	MOVZX EDX,DX	
004EB316	81C7 3DBB3515	ADD EDI,1535BB3D	
004EB31C	0FB7F3	MOVZX ESI,BX	
004EB31F	81FF 20FAFFFF	CMP EDI,-5E0	
004EB325	0F85 19000000	JNZ chordpic.004EB344	
004EB32B	8BC6	MOV EAX,ESI	
004EB32D	E9 41000000	JMP chordpic.004EB373	USCITA DAL LOOP#3
004EB332	22B3 70E96E0F	AND DH,BYTE PTR DS:[EBX+F6EE970]	
004EB338	9C	PUSHFD	
004EB339	A5	MOVS DWORD PTR ES:[EDI],DWORD PTR	
004EB33A	7A 2B	JPE SHORT chordpic.004EB367	
004EB33C	8821	MOV BYTE PTR DS:[ECX],AH	
004EB33E	46	INC ESI	
004EB33F	07	POP ES	Modification of segment register
004EB340	34 5D	XOR AL,5D	
004EB342	90	NOP	
004EB343	90	NOP	
004EB344	0F84 14000000	JE chordpic.004EB35E	
004EB34A	53	PUSH EBX	
004EB34B	E9 00000000	JMP chordpic.004EB35D	
004EB350	91	XCHG EAX,ECX	
004EB351	F6F7	DIV BH	
004EB353	64:CD 82	INT 82	Superfluous prefix
004EB356	93	XCHG EAX,EBX	
004EB357	D0C9	ROR CL,1	
004EB359	CE	INT0	
004EB35A	EF	OUT DX,EAX	I/O command
004EB35B	FC	CLD	
004EB35C	90	NOP	
004EB35D	58	POP EAX	
004EB35E	E9 6DFFFFFF	JMP chordpic.004EB2D0	
004EB363	E8 01A6E794	CALL 95365969	
004EB368	3D 32830039	CMP EAX,39008332	
004EB36D	7E DF	JLE SHORT chordpic.004EB34E	
004EB36F	2C F5	SUB AL,0F5	
004EB371	8AFB	MOV BH,BL	
004EB373	E9 09000000	JMP chordpic.004EB381	
004EB378	5D	POP EBP	
004EB379	D2A3 A0591EFF	SHL BYTE PTR DS:[EBX+FF1E59A0],CL	
004EB37F	CC	INT3	
004EB380	15 E80F0000	ADC EAX,0FE8	

Take a note for this exit address, this is the redirection #3.

**END OF THE LOOP #3 (redirection #3): 0x004EB32D**  
**ORIGINAL INSTRUCTION: JMP 0x004EB373**

The code on the address 0x004EB4E1 isn't present then we've to trace again.



We have another decompression loop.

004EB320	E9 41000000	JMP chordpic.004EB373	USCITA DAL LOOP#3
004EB332	22B3 70E96E0F	AND DH,BYTE PTR DS:[EBX+F6EE970]	
004EB338	9C	PUSHFD	
004EB339	A5	MOVS DWORD PTR ES:[EDI],DWORD PTR	
004EB33A	7A 2B	JPE SHORT chordpic.004EB367	
004EB33C	8821	MOV BYTE PTR DS:[ECX],AH	
004EB33E	46	INC ESI	
004EB33F	07	POP ES	Modification of segment register
004EB340	34 5D	XOR AL,5D	
004EB342	D2A3 0F841400	SHL BYTE PTR DS:[EBX+14840F],CL	
004EB348	0000	ADD BYTE PTR DS:[EAX],AL	
004EB34A	53	PUSH EBX	
004EB34B	E9 0D000000	JMP chordpic.004EB35D	
004EB350	91	XCHG EAX,ECX	
004EB351	F6F7	DIU BH	
004EB353	64:CD 82	INT 82	Superfluous prefix
004EB356	93	XCHG EAX,EBX	
004EB357	D0C9	ROR CL,1	
004EB359	CE	INT0	
004EB35A	EF	OUT DX,EAX	I/O command
004EB35B	FC	CLD	
004EB35C	8558 E9	TEST DWORD PTR DS:[EAX-17],EBX	
004EB35F	6D	INS DWORD PTR ES:[EDI],DX	I/O command
004EB360	FFFF	???	Unknown command
004EB362	FFE8	JMP FAR EAX	Illegal use of register
004EB364	01A6 E7943D32	ADD DWORD PTR DS:[ESI+323D94E7],ESI	
004EB36A	8300 39	ADD DWORD PTR DS:[EAX],39	
004EB36D	7E DF	JLE SHORT chordpic.004EB34E	
004EB36F	2C F5	SUB AL,0F5	
004EB371	8AFB	MOV BH,BL	
004EB373	E9 09000000	JMP chordpic.004EB381	
004EB378	5D	POP EBP	
004EB379	D2A3 A0591EFF	SHL BYTE PTR DS:[EBX+FF1E59A0],CL	
004EB37F	CC	INT3	
004EB380	90	NOP	
004EB381	E8 0F000000	CALL chordpic.004EB395	
004EB386	1BB8 91F6F764	SBB EDI,DWORD PTR DS:[EAX+64F7F691]	
004EB38C	CD 82	INT 82	
004EB38E	93	XCHG EAX,EBX	
004EB38F	D0C9	ROR CL,1	
004EB391	CE	INT0	
004EB392	EF	OUT DX,EAX	I/O command
004EB393	FC	CLD	
004EB394	90	NOP	
004EB395	0F8F 03000000	JG chordpic.004EB39E	
004EB398	0FBFF8	MOVSX EDI,AX	
004EB39E	59	POP ECX	
004EB39F	66:BF 7E9A	MOV DI,9A7E	
004EB3A3	81C1 C9050000	ADD ECX,5C9	
004EB3A9	0F8A 02000000	JPE chordpic.004EB3B1	
004EB3AF	8AF1	MOV DH,CL	
004EB3B1	68 4A010000	PUSH 14A	
004EB3B6	58	POP EAX	
004EB3B7	BE 657B2843	MOV ESI,43287B65	
004EB3BC	8B19	MOV EBX,DWORD PTR DS:[ECX]	INIZIO DEL LOOP#4
004EB3BE	8BD7	MOV EDX,EDI	
004EB3C0	81F3 0F27B513	XOR EBX,13B5270F	
004EB3C6	66:BA 6062	MOV DX,6260	
004EB3CA	81F3 9C63A21A	XOR EBX,1AA2639C	
004EB3D0	66:8BFA	MOV DI,DX	
004EB3D3	81EB A5212678	SUB EBX,782621A5	
004EB3D9	66:81DE B65C	SBB SI,5CB6	
004EB3DE	53	PUSH EBX	
004EB3DF	66:81F7 8934	XOR DI,3489	
004EB3E4	8F01	POP DWORD PTR DS:[ECX]	
004EB3E6	81EE 9AA34F6F	SUB ESI,6F4FA39A	
004EB3EC	81E9 A8D4641F	SUB ECX,1F64D4A8	
004EB3F2	BF 5493861D	MOV EDI,1D869354	
004EB3F7	81C1 A4D4641F	ADD ECX,1F64D4A4	
004EB3FD	68 43026048	PUSH 4B600243	
004EB402	81C7 BB764D47	ADD EDI,474D76BB	
004EB408	5E	POP ESI	
004EB409	48	DEC EAX	
004EB40A	0F85 ACFFFFFF	JNZ chordpic.004EB3BC	
004EB410	E8 12000000	CALL chordpic.004EB427	USCITA DAL LOOP#4
004EB415	A2 33F069EE	MOV BYTE PTR DS:[EE69F033],AL	
004EB41A	8F	???	Unknown command
004EB41B	1C 25	SBB AL,25	
004EB41D	FA	CLI	
004EB41E	AB	STOS DWORD PTR ES:[EDI]	
004EB41F	08A1 C687B4D0	OR BYTE PTR DS:[ECX+D0B487C6],AH	
004EB425	52	PUSH EDX	
004EB426	2366 8B	AND ESP,DWORD PTR DS:[ESI-75]	
004EB429	F1	INT1	
004EB42A	5E	POP ESI	
004EB42B	E8 00000000	CALL chordpic.004EB430	
004EB430	5D	POP EBP	
004EB431	5B	POP EBX	
004EB432	895D 5B	MOV DWORD PTR SS:[EBP+5B],EBX	
004EB435	5B	POP EBX	

A software memory breakpoint (F2) is put to the end of the loop and it is executed with Shift+F9, this point give us the redirection #4.

**END OF THE LOOP #4 (redirection #4): 0x004EB410**  
**ORIGINAL INSTRUCTION: 0x004EB427**

Finally our code was written:

004EB4C1	43	INC EBX	
004EB4C2	BB 3219122A	MOV EBX,2A121932	
004EB4C7	00B2 5C332589	ADD BYTE PTR DS:[EDX+8925335C],0H	
004EB4CD	A5	MOVS DWORD PTR ES:[EDI],DWORD PTR	
004EB4CE	290400	SUB DWORD PTR DS:[EAX+EAX],EAX	
004EB4D1	006A 40	ADD BYTE PTR DS:[EDX+40],CH	
004EB4D4	68 00100000	PUSH 1000	
004EB4D9	FFB5 00040000	PUSH DWORD PTR SS:[EBP+400]	
004EB4DF	6A 00	PUSH 0	
004EB4E1	FF95 F0030000	CALL NEAR DWORD PTR SS:[EBP+3F0]	Richiama VirtualAlloc per la prima volta
004EB4E7	8985 CC010000	MOV DWORD PTR SS:[EBP+1CC],EAX	
004EB4ED	8B9D 00040000	MOV EBX,DWORD PTR SS:[EBP+400]	
004EB4F3	039D 00040000	ADD EBX,DWORD PTR SS:[EBP+400]	
004EB4F9	50	PUSH EAX	
004EB4FA	53	PUSH EBX	
004EB4FB	E8 04010000	CALL chordpic.004EB604	
004EB500	6A 40	PUSH 40	

Well, at this point we have do this step:

1. We are to the packer entry point.
2. The decryption loop #1 is executed, the exit point is in 0x004EB1AA
3. The decryption loop #2 is executed, the exit point is in 0x004EB267
4. The decryption loop #3 is executed, the exit point is in 0x004EB32D
5. The decryption loop #4 is executed, the exit point is in 0x004EB410
6. We reach the **VirtualAlloc** calling address in 0x004EB4E1

The breakpoint list (activate each of this one in sequence beginning from the Entry Point and to happened execution of every loop):

Address	Module	Active	Disassembly	Comment
004EB1AA	chordpic	Always	JMP chordpic.004EB1CA	USCITA DAL LOOP#1
004EB267	chordpic	Always	JMP chordpic.004EB290	USCITA DAL LOOP#2
004EB32D	chordpic	Always	JMP chordpic.004EB373	USCITA DAL LOOP#3
004EB410	chordpic	Always	CALL chordpic.004EB427	USCITA DAL LOOP#4

After other tracing we reach the instructions that perform the **VirtualAlloc** calling.

004EB4D9	FFB5 08040000	PUSH DWORD PTR SS:[EBP+408]	
004EB4DF	6A 00	PUSH 0	
004EB4E1	FF95 F0030000	CALL NEAR DWORD PTR SS:[EBP+3F0]	
004EB4E7	8985 CC010000	MOV DWORD PTR SS:[EBP+1CC],EAX	Richiama VirtualAlloc per la prima volta
004EB4ED	8B9D 00040000	MOV EBX,DWORD PTR SS:[EBP+400]	
004EB4F3	039D 00040000	ADD EBX,DWORD PTR SS:[EBP+400]	
004EB4F9	50	PUSH EAX	
004EB4FA	53	PUSH EBX	
004EB4FB	E8 04010000	CALL chordpic.004EB604	
004EB500	6A 40	PUSH 40	
004EB502	68 00100000	PUSH 1000	
004EB507	FFB5 08040000	PUSH DWORD PTR SS:[EBP+408]	
004EB50D	6A 00	PUSH 0	
004EB50F	FF95 F0030000	CALL NEAR DWORD PTR SS:[EBP+3F0]	Richiama Virtual Alloc per la seconda volta
004EB515	8985 31040000	MOV DWORD PTR SS:[EBP+431],EAX	EAX = indirizzo sezione a run-time che scrive OEP
004EB51B	8985 D0010000	MOV DWORD PTR SS:[EBP+1D0],EAX	
004EB521	64:67:A1 0000	MOV EAX,DWORD PTR FS:[0]	
004EB526	8985 2D040000	MOV DWORD PTR SS:[EBP+42D],EAX	
004EB52C	8B85 5B	MOV EDI,DWORD PTR SS:[EBP+5B]	
004EB52F	8B85 D0010000	MOV EAX,DWORD PTR SS:[EBP+1D0]	
004EB535	8902	MOV DWORD PTR DS:[EDX],EAX	
004EB537	8B85 08040000	MOV EAX,DWORD PTR SS:[EBP+408]	
004EB53D	8942 04	MOV DWORD PTR DS:[EDX+4],EAX	
004EB540	8D85 9F030000	LEA EAX,DWORD PTR SS:[EBP+39F]	
004EB546	8B40 55	MOV EAX,DWORD PTR DS:[EAX+55]	
004EB549	8942 08	MOV DWORD PTR DS:[EDX+8],EAX	
004EB54C	8B85 EC030000	MOV EAX,DWORD PTR SS:[EBP+3EC]	
004EB552	8942 10	MOV DWORD PTR DS:[EDX+10],EAX	
004EB555	8B85 E8030000	MOV EAX,DWORD PTR SS:[EBP+3E8]	
004EB55B	8942 14	MOV DWORD PTR DS:[EDX+14],EAX	
004EB55E	8B95 CC010000	MOV EDI,DWORD PTR SS:[EBP+1CC]	
004EB564	BB F8010000	MOV EBX,1F8	
004EB569	8B7C1A 0C	MOV EDI,DWORD PTR DS:[EDX+EBX+C]	
004EB56D	0BFF	OR EDI,EDI	
004EB56F	74 1E	JE SHORT chordpic.004EB58F	
004EB571	8B4C1A 10	MOV ECX,DWORD PTR DS:[EDX+EBX+10]	
004EB575	0BC9	OR ECX,ECX	
004EB577	74 11	JE SHORT chordpic.004EB58A	
004EB579	03BD D0010000	ADD EDI,DWORD PTR SS:[EBP+1D0]	
004EB57F	8B741A 14	MOV ESI,DWORD PTR DS:[EDX+EBX+14]	
004EB583	03F2	ADD ESI,EDX	
004EB585	C1F9 02	SAR ECX,2	
004EB588	F3:A5	REP MOVS DWORD PTR ES:[EDI],DWORD PTR DS:[ESI]	
004EB58A	83C3 28	ADD EBX,28	
004EB58D	EB DA	JMP SHORT chordpic.004EB569	
004EB58F	8B85 CC010000	MOV EAX,DWORD PTR SS:[EBP+1CC]	Fine loop scrittura area a run-time
004EB595	50	PUSH EAX	
004EB596	8B95 D0010000	MOV EDI,DWORD PTR SS:[EBP+1D0]	
004EB59C	52	PUSH EDI	
004EB59D	8B18	MOV EBX,DWORD PTR DS:[EAX]	
004EB59F	83DA	ADD EBX,EDX	
004EB5A1	8B85 E4030000	MOV EAX,DWORD PTR SS:[EBP+3E4]	
004EB5A7	8903	MOV DWORD PTR DS:[EBX],EAX	
004EB5A9	8B85 E8030000	MOV EAX,DWORD PTR SS:[EBP+3E8]	
004EB5AF	8943 04	MOV DWORD PTR DS:[EBX+4],EAX	
004EB5B2	8B85 EC030000	MOV EAX,DWORD PTR SS:[EBP+3EC]	
004EB5B8	8943 08	MOV DWORD PTR DS:[EBX+8],EAX	
004EB5BB	5F	POP EDI	
004EB5BC	5E	POP ESI	
004EB5BD	8B46 04	MOV EAX,DWORD PTR DS:[ESI+4]	
004EB5C0	03C7	ADD EAX,EDI	
004EB5C2	8985 C7010000	MOV DWORD PTR SS:[EBP+1C7],EAX	
004EB5C8	8B85 5B	MOV EDI,DWORD PTR SS:[EBP+5B]	
004EB5CB	8B85 C7010000	MOV EAX,DWORD PTR SS:[EBP+1C7]	
004EB5D1	8942 0C	MOV DWORD PTR DS:[EDX+C],EAX	
004EB5D4	8D9D 00040000	LEA EBX,DWORD PTR SS:[EBP+40D]	
004EB5DA	53	PUSH EBX	
004EB5DB	6A 00	PUSH 0	
004EB5DD	6A 00	PUSH 0	
004EB5DF	6A 01	PUSH 1	
004EB5E1	57	PUSH EDI	
004EB5E2	8B8E 08	MOV EBX,DWORD PTR DS:[ESI+8]	
004EB5E5	03DF	ADD EBX,EDI	
004EB5E7	53	PUSH EBX	
004EB5E8	68 00800000	PUSH 8000	
004EB5ED	6A 00	PUSH 0	
004EB5EF	56	PUSH ESI	
004EB5F0	FF95 F4030000	CALL NEAR DWORD PTR SS:[EBP+3F4]	
004EB5F6	68 00100A00	PUSH 0BA1000	
004EB5F8	C3	RETN	Salta nell'area allocata a run-time
004EB5FC	0000	ADD BYTE PTR DS:[EAX],AL	
004EB5FE	B3 00	MOV BL,0	
004EB600	0000	ADD BYTE PTR DS:[EAX],AL	

The push 0BA1000 instruction followed from the RETN perform a jumps in to the new allocated area. But is also interesting take a look at the PUSH 8000 instruction because it is possible to recover in EDI the base address of the new fresh allocated memory area.

We can perform another redirection to our patching code by using the byte of the PUSH 8000 instruction.

**ABSOLUTE ADDRESS (redirection #5):** 0x004EB5E8

**ORIGINAL INSTRUCTION:** PUSH 8000 - > 0x68 0x00 0x80 0x00 0x00

The execution jumps in the region allotted from ASProtect to the offset one 0x0031000.

From this point all the future redirection deal about offset rather absolute address because all the code still in a run-time memory area.

Now we find ourselves in the memory area that will execute the decompression of the code to the OEP.

00BA1000	90	NOP	Entry point
00BA1001	60	PUSHAD	
00BA1002	E8 40060000	CALL 00BA1647	
00BA1007	EB 44	JMP SHORT 00BA104D	
00BA1009	0000	ADD BYTE PTR DS:[EAX],AL	
00BA100B	0000	ADD BYTE PTR DS:[EAX],AL	
00BA100D	0000	ADD BYTE PTR DS:[EAX],AL	
00BA100F	0000	ADD BYTE PTR DS:[EAX],AL	
00BA1011	87DB	XCHG EBX,EBX	
00BA1013	90	NOP	
00BA1014	0000	ADD BYTE PTR DS:[EAX],AL	
00BA1016	0000	ADD BYTE PTR DS:[EAX],AL	
00BA1018	0000	ADD BYTE PTR DS:[EAX],AL	
00BA101A	0000	ADD BYTE PTR DS:[EAX],AL	
00BA101C	0000	ADD BYTE PTR DS:[EAX],AL	
00BA101E	0000	ADD BYTE PTR DS:[EAX],AL	
00BA1020	0000	ADD BYTE PTR DS:[EAX],AL	
00BA1022	0000	ADD BYTE PTR DS:[EAX],AL	
00BA1024	0000	ADD BYTE PTR DS:[EAX],AL	
00BA1026	0000	ADD BYTE PTR DS:[EAX],AL	
00BA1028	0000	ADD BYTE PTR DS:[EAX],AL	
00BA102A	0000	ADD BYTE PTR DS:[EAX],AL	
00BA102C	0000	ADD BYTE PTR DS:[EAX],AL	
00BA102E	1003	ADC BYTE PTR DS:[EBX],AL	
00BA1030	0000	ADD BYTE PTR DS:[EAX],AL	
00BA1032	0000	ADD BYTE PTR DS:[EAX],AL	
00BA1034	0000	ADD BYTE PTR DS:[EAX],AL	
00BA1036	0000	ADD BYTE PTR DS:[EAX],AL	
00BA1038	0000	ADD BYTE PTR DS:[EAX],AL	
00BA103A	0000	ADD BYTE PTR DS:[EAX],AL	
00BA103C	0000	ADD BYTE PTR DS:[EAX],AL	
00BA103E	0000	ADD BYTE PTR DS:[EAX],AL	
00BA1040	0000	ADD BYTE PTR DS:[EAX],AL	
00BA1042	0000	ADD BYTE PTR DS:[EAX],AL	
00BA1044	0000	ADD BYTE PTR DS:[EAX],AL	
00BA1046	0000	ADD BYTE PTR DS:[EAX],AL	
00BA1048	0000	ADD BYTE PTR DS:[EAX],AL	
00BA104A	0000	ADD BYTE PTR DS:[EAX],AL	
00BA104C	90	NOP	
00BA104D	BB 44294400	MOV EBX,442944	
00BA1052	03D0	ADD EBX,EBP	
00BA1054	2B9D 71294400	SUB EBX,DWORD PTR SS:[EBP+442971]	
00BA105A	83BD D8304400	CMF DWORD PTR SS:[EBP+4430D8],0	
00BA1061	899D 2F2E4400	MOV DWORD PTR SS:[EBP+442E2F],EBX	
00BA1067	0F85 3E050000	JNZ 00BA15AB	
00BA106D	8D85 E0304400	LEA EAX,DWORD PTR SS:[EBP+4430E0]	

If we go to see the code to the offset 0x002663 from the base address, we notice that the code is quite different from that we've see at the beginning of our analysis, this because we've to step through some other decompression layer before reach the code able to unpack all the target code.

00B72663	01D9	ADD ECX,EBX	
00B72665	3D B4944C36	CMF EAX,364C94B4	
00B7266A	48	DEC EAX	
00B7266B	98	CWDE	
00B7266C	2C 44	SUB AL,44	
00B7266E	CB	RETF	Far return
00B7266F	E8 3403230A	CALL 0ADA29A8	
00B72674	C545 A3	LDS EAX,FWORD PTR SS:[EBP-5D]	Modification of segment register
00B72677	44	INC ESP	
00B72678	74 24	JE SHORT 00B7269E	
00B7267A	188D 533CB246	SBB BYTE PTR SS:[EBP+46B23C53],CL	
00B72680	06	PUSH ES	

We resume the analysis from the address 0x00BA1007, we've another call to the **VirtualAlloc** API on 0x00BA10C4, then we've a writing loop which decrypt some other code.

00BA10D6	50	PUSH EAX	
00BA10D7	53	PUSH EBX	
00BA10D8	E8 74050000	CALL 00BA1651	
00BA10DD	8BC8	MOV ECX,EAX	
00BA10DF	8DBD 452A4400	LEA EDI,DWORD PTR SS:[EBP+442A45]	
00BA10E5	8B85 75294400	MOV ESI,DWORD PTR SS:[EBP+442975]	
00BA10EB	F3:A4	REP MOVSB BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]	Decritta il codice che segue da 0x00BA1101
00BA10ED	8B85 75294400	MOV EAX,DWORD PTR SS:[EBP+442975]	
00BA10F3	68 00800000	PUSH 8000	
00BA10F8	6A 00	PUSH 0	
00BA10FA	50	PUSH EAX	
00BA10FB	FF95 7D294400	CALL NEAR DWORD PTR SS:[EBP+44297D]	
00BA1101	8D85 512C4400	LEA EAX,DWORD PTR SS:[EBP+442C51]	
00BA1107	50	PUSH EAX	0x00BA130D -> offset 0x008130D
00BA1108	C3	RETN	
00BA1109	0000	ADD BYTE PTR DS:[EAX],AL	
00BA110B	0000	ADD BYTE PTR DS:[EAX],AL	

this finally keep us to a new jump to the offset 0x003130D.

Take a note of the point where we can put our next redirection:

**OFFSET (redirection #6):** 0x00310F3

**ORIGINAL INSTRUCTION:** PUSH 8000 - > 0x68 0x00 0x80 0x00 0x00

Take also a look if the code at the address 0x00B72663 into the OllyDbg dump window comes decrypted, for now the code isn't present then we've to step a little.

We take the RETN instruction, step with F8, we reach the offset 0x0031343 related another call to the **VirtualAlloc** API.

00BA130B	0000	ADD BYTE PTR DS:[EAX],AL	
00BA130D	8B9D 552A4400	MOV EBX,DWORD PTR SS:[EBP+442A55]	
00BA1313	000B	OR EBX,EBX	
00BA1315	74 0A	JE SHORT 00BA1321	
00BA1317	8B03	MOV EAX,DWORD PTR DS:[EBX]	
00BA1319	8785 592A4400	XCHG DWORD PTR SS:[EBP+442A59],EAX	
00BA131F	8903	MOV DWORD PTR DS:[EBX],EAX	
00BA1321	80B5 712A4400	LEA ESI,DWORD PTR SS:[EBP+442A71]	
00BA1327	833E 00	CMP DWORD PTR DS:[ESI],0	
00BA132A	0F84 D3000000	JE 00BA1403	
00BA1330	80B5 712A4400	LEA ESI,DWORD PTR SS:[EBP+442A71]	
00BA1336	8B46 04	MOV EAX,DWORD PTR DS:[ESI+4]	
00BA1339	6A 04	PUSH 4	
00BA133B	68 00100000	PUSH 1000	
00BA1340	50	PUSH EAX	
00BA1341	6A 00	PUSH 0	
00BA1343	FF95 79294400	CALL NEAR DWORD PTR SS:[EBP+442979]	call to VirtualAlloc
00BA1349	8985 75294400	MOV DWORD PTR SS:[EBP+442975],EAX	
00BA134F	56	PUSH ESI	
00BA1350	8B1E	MOV EBX,DWORD PTR DS:[ESI]	
00BA1352	039D D8304400	ADD EDI,DWORD PTR SS:[EBP+4430D8]	
00BA1358	50	PUSH EAX	
00BA1359	53	PUSH EBX	
00BA135A	E3 F2020000	CALL 00BA1651	
00BA135F	80BD 70294400	CMP BYTE PTR SS:[EBP+442970],0	
00BA1366	75 4C	JNZ SHORT 00BA13B4	
00BA1368	FE85 70294400	INC BYTE PTR SS:[EBP+442970]	
00BA136E	8B3E	MOV EDI,DWORD PTR DS:[ESI]	
00BA1370	03BD D8304400	ADD EDI,DWORD PTR SS:[EBP+4430D8]	
00BA1376	FF37	PUSH DWORD PTR DS:[EDI]	
00BA1378	C607 C3	MOV BYTE PTR DS:[EDI],0C3	
00BA137B	FFD7	CALL NEAR EDI	
00BA137D	8F07	POP DWORD PTR DS:[EDI]	
00BA137F	50	PUSH EAX	
00BA1380	51	PUSH ECX	
00BA1381	56	PUSH ESI	
00BA1382	53	PUSH EBX	
00BA1383	8BC8	MOV ECX,EAX	
00BA1385	83E9 06	SUB ECX,6	
00BA1388	8BB5 75294400	MOV ESI,DWORD PTR SS:[EBP+442975]	
00BA138E	330B	XOR EBX,EBX	
00BA1390	0BC9	OR ECX,ECX	
00BA1392	74 1C	JE SHORT 00BA13B0	
00BA1394	78 1A	JS SHORT 00BA13B0	
00BA1396	AC	LODS BYTE PTR DS:[ESI]	
00BA1397	3C E8	CMP AL,0E8	
00BA1399	74 08	JE SHORT 00BA13A3	
00BA139B	3C E9	CMP AL,0E9	
00BA139D	74 04	JE SHORT 00BA13A3	
00BA139F	43	INC EBX	
00BA13A0	49	DEC ECX	
00BA13A1	EB ED	JMP SHORT 00BA1390	
00BA13A3	291E	SUB DWORD PTR DS:[ESI],EBX	
00BA13A5	83C3 05	ADD EBX,5	
00BA13A8	83C6 04	ADD ESI,4	
00BA13AB	83E9 05	SUB ECX,5	
00BA13AE	EB E0	JMP SHORT 00BA1390	
00BA13B0	5B	POP EBX	
00BA13B1	5E	POP ESI	
00BA13B2	59	POP ECX	
00BA13B3	58	POP EAX	
00BA13B4	8BC8	MOV ECX,EAX	
00BA13B6	8B3E	MOV EDI,DWORD PTR DS:[ESI]	
00BA13B8	03BD D8304400	ADD EDI,DWORD PTR SS:[EBP+4430D8]	
00BA13BE	8BB5 75294400	MOV ESI,DWORD PTR SS:[EBP+442975]	
00BA13C4	C1F9 02	SAR ECX,2	
00BA13C7	F3:A5	REP MOVS DWORD PTR ES:[EDI],DWORD PTR DS:[ESI]	Scrittura codice
00BA13C9	8BC8	MOV ECX,EAX	
00BA13CB	83E1 03	AND ECX,3	
00BA13CE	F3:A4	REP MOVS BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]	
00BA13D0	5E	POP ESI	
00BA13D1	8B85 75294400	MOV EAX,DWORD PTR SS:[EBP+442975]	
00BA13D7	68 00800000	PUSH 8000	
00BA13DC	6A 00	PUSH 0	
00BA13DE	50	PUSH EAX	
00BA13DF	FF95 70294400	CALL NEAR DWORD PTR SS:[EBP+44297D]	
00BA13E5	83C6 08	ADD ESI,8	
00BA13E8	833E 00	CMP DWORD PTR DS:[ESI],0	
00BA13EB	0F85 45FFFFFF	JNZ 00BA1336	
00BA13F1	8B9D 552A4400	MOV EBX,DWORD PTR SS:[EBP+442A55]	
00BA13F7	000B	OR EBX,EBX	
00BA13F9	74 08	JE SHORT 00BA1403	
00BA13FB	8B03	MOV EAX,DWORD PTR DS:[EBX]	
00BA13FD	8785 592A4400	XCHG DWORD PTR SS:[EBP+442A59],EAX	
00BA1403	8B95 D8304400	MOV EDX,DWORD PTR SS:[EBP+4430D8]	
00BA1409	8B85 512A4400	MOV EAX,DWORD PTR SS:[EBP+442A51]	
00BA140F	2BD0	SUB EDX,EAX	
00BA1411	74 75	JE SHORT 00BA1488	

Step straight with F8 until the offset 0x00313C7, here we've the decryption loop for the code that will write, once executed, the unpacked code of the target.

Again take a note for the offset of the PUSH 8000 instruction, this is the our next redirection point.

**OFFSET (redirection #7): 0x00313D7**

**INSTRUCTION ORIGINATES THEM:** PUSH 8000 - > 0x68 0x00 0x80 0x00 0x00

When you press F8 the code will be written:

Address	Hex dump	Disassembly	Co
00B72663	F3:A5	REP MOVSD DWORD PTR ES:[EDI],DWORD PTR DS:[ESI]	
00B72665	89C1	MOV ECX,EAX	
00B72667	83E1 03	AND ECX,3	
00B7266A	F3:A4	REP MOVSB BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]	
00B7266C	5F	POP EDI	
00B7266D	5E	POP ESI	
00B7266E	C3	RETN	
00B7266F	8D740E FC	LEA ESI,DWORD PTR DS:[ESI+ECX-4]	

Step a little, we reach a block of instruction that may be rather interesting since it seems has to do with the IAT, we've two nested loop.

00BA1494	03F2	ADD ESI,EDX	
00BA1496	8B46 0C	MOV EAX,DWORD PTR DS:[ESI+C]	inizio loop esterno
00BA1499	85C0	TEST EAX,EAX	
00BA149B	0F84 0A010000	JE 00BA15AB	
00BA14A1	03C2	ADD EAX,EDX	
00BA14A3	8BD8	MOV EBX,EAX	
00BA14A5	50	PUSH EAX	
00BA14A6	FF95 EC314400	CALL NEAR DWORD PTR SS:[EBP+4431EC]	Nome API call to kernel32.GetModuleHandleA
00BA14AC	85C0	TEST EAX,EAX	
00BA14AE	75 07	JNZ SHORT 00BA14B7	
00BA14B0	53	PUSH EBX	
00BA14B1	FF95 F0314400	CALL NEAR DWORD PTR SS:[EBP+4431F0]	
00BA14B7	8985 4D294400	MOV DWORD PTR SS:[EBP+44294D],EAX	
00BA14B0	C785 51294400	MOV DWORD PTR SS:[EBP+442951],0	
00BA14C7	8B95 D8304400	MOV EDX,DWORD PTR SS:[EBP+4430D8]	inizio loop interno
00BA14CD	8B06	MOV EAX,DWORD PTR DS:[ESI]	
00BA14CF	85C0	TEST EAX,EAX	
00BA14D1	75 03	JNZ SHORT 00BA14D6	
00BA14D3	8B46 10	MOV EAX,DWORD PTR DS:[ESI+10]	
00BA14D6	03C2	ADD EAX,EDX	
00BA14D8	0385 51294400	ADD EAX,DWORD PTR SS:[EBP+442951]	
00BA14DE	8B18	MOV EBX,DWORD PTR DS:[EAX]	
00BA14E0	8B7E 10	MOV EDI,DWORD PTR DS:[ESI+10]	
00BA14E3	03FA	ADD EDI,EDX	
00BA14E5	038D 51294400	ADD EDI,DWORD PTR SS:[EBP+442951]	
00BA14E8	85C0	TEST EBX,EBX	
00BA14ED	0F84 A2000000	JE 00BA1595	
00BA14F3	F7C3 00000000	TEST EBX,80000000	
00BA14F9	75 04	JNZ SHORT 00BA14FF	
00BA14FB	03DA	ADD EBX,EDX	
00BA14FD	43	INC EBX	
00BA14FE	43	INC EBX	
00BA14FF	53	PUSH EBX	
00BA1500	31E3 FFFFFFFF	AND EBX,7FFFFFFF	
00BA1506	53	PUSH EBX	
00BA1507	FFB5 4D294400	PUSH DWORD PTR SS:[EBP+44294D]	nome API di cui si cerca l'indirizzo
00BA150D	FF95 E8314400	CALL NEAR DWORD PTR SS:[EBP+4431E8]	call to kernel32.GetProcAddress
00BA1513	85C0	TEST EAX,EAX	
00BA1515	5B	POP EBX	
00BA1516	75 6F	JNZ SHORT 00BA1587	
00BA1518	F7C3 00000000	TEST EBX,80000000	
00BA151E	75 19	JNZ SHORT 00BA1539	
00BA1520	53	PUSH EDI	
00BA1521	8B46 0C	MOV EAX,DWORD PTR DS:[ESI+C]	
00BA1524	0385 D8304400	ADD EAX,DWORD PTR SS:[EBP+4430D8]	
00BA152A	50	PUSH EAX	
00BA152B	53	PUSH EBX	
00BA152C	8D85 53314400	LEA EAX,DWORD PTR SS:[EBP+443153]	
00BA1532	50	PUSH EAX	
00BA1533	57	PUSH EDI	
00BA1534	E9 99000000	JMP 00BA15D2	
00BA1537	81E3 FFFFFFFF	AND EAX,7FFFFFFF	
00BA153F	8B85 DC304400	MOV EAX,DWORD PTR SS:[EBP+4430DC]	
00BA1545	3985 4D294400	CMP DWORD PTR SS:[EBP+44294D],EAX	
00BA154B	75 24	JNZ SHORT 00BA1571	
00BA154D	57	PUSH EDI	
00BA154E	8BD3	MOV EDX,EBX	
00BA1550	4A	DEC EDX	
00BA1551	C1E2 02	SHL EDX,2	
00BA1554	8B9D 4D294400	MOV EBX,DWORD PTR SS:[EBP+44294D]	
00BA155A	8B7B 3C	MOV EDI,DWORD PTR DS:[EBX+3C]	
00BA155D	8B7C3B 78	MOV EDI,DWORD PTR DS:[EBX+EDI+78]	
00BA1561	035C3B 1C	ADD EBX,DWORD PTR DS:[EBX+EDI+1C]	
00BA1565	8B0413	MOV EAX,DWORD PTR DS:[EBX+EDX]	
00BA1568	0385 4D294400	ADD EAX,DWORD PTR SS:[EBP+44294D]	
00BA156E	5F	POP EDI	
00BA156F	EB 16	JMP SHORT 00BA1587	
00BA1571	57	PUSH EDI	
00BA1572	8B46 0C	MOV EAX,DWORD PTR DS:[ESI+C]	
00BA1575	0385 D8304400	ADD EAX,DWORD PTR SS:[EBP+4430D8]	
00BA157B	50	PUSH EAX	
00BA157C	53	PUSH EBX	
00BA157D	8D85 A4314400	LEA EAX,DWORD PTR SS:[EBP+4431A4]	
00BA1583	50	PUSH EAX	
00BA1584	57	PUSH EDI	
00BA1585	EB 4B	JMP SHORT 00BA15D2	
00BA1587	8987	MOV DWORD PTR DS:[EDI],EAX	
00BA1589	8385 51294400	ADD DWORD PTR SS:[EBP+442951],4	Scrivo l'indirizzo risolto dell'API Si sposta nella prossima entry da scrivere continua loop interno
00BA1590	E9 32FFFFFF	JMP 00BA14C7	
00BA1595	8906	MOV DWORD PTR DS:[ESI],EAX	
00BA1597	8946 0C	MOV DWORD PTR DS:[ESI+C],EAX	
00BA159A	8946 10	MOV DWORD PTR DS:[ESI+10],EAX	
00BA159D	83C6 14	ADD ESI,14	
00BA15A0	8B95 D8304400	MOV EDX,DWORD PTR SS:[EBP+4430D8]	
00BA15A6	E9 EBFFFFFF	JMP 00BA1496	
00BA15A8	0385 652A4400	ADD EAX,DWORD PTR SS:[EBP+442A65]	continua loop esterno
00BA15B1	50	PUSH EAX	
00BA15B2	0385 D8304400	ADD EAX,DWORD PTR SS:[EBP+4430D8]	

the decoded API will be written from the offset 0x002C104, the loop exit point is 0x00315AB.



00B9C0F8	00000000	
00B9C0FC	00000000	
00B9C100	00000000	
00B9C104	7C809737	kernel32.GetCurrentThreadId
00B9C108	7C92188A	ntdll.RtlDeleteCriticalSection
00B9C10C	7C9110ED	ntdll.RtlLeaveCriticalSection
00B9C110	7C911005	ntdll.RtlEnterCriticalSection
00B9C114	7C809FA1	kernel32.InitializeCriticalSection
00B9C118	7C809B14	kernel32.VirtualFree
00B9C11C	7C809A81	kernel32.VirtualAlloc
00B9C120	7C80995D	kernel32.LocalFree
00B9C124	7C8099B0	kernel32.LocalAlloc
00B9C128	7C80B859	kernel32.VirtualQuery
00B9C12C	7C80A0C7	kernel32.WideCharToMultiByte
00B9C130	7C809CAD	kernel32.MultiByteToWideChar
00B9C134	7C80C6E0	kernel32.lstrlenA
00B9C138	7C810311	kernel32.lstrcpyA
00B9C13C	7C80C729	kernel32.lstrcpyA
00B9C140	7C801D4F	kernel32.LoadLibraryExA
00B9C144	7C80A405	kernel32.GetThreadLocale
00B9C148	7C801EEE	kernel32.GetStartupInfoA

Some other step keep us to the classic sequence of instructions that is found in ASPACK:

00BA1583	50	PUSH EAX	
00BA1584	57	PUSH EDI	
00BA1585	✓ EB 4B	JMP SHORT 00BA15D2	
00BA1587	8907	MOV DWORD PTR DS:[EDI],EAX	Scrive l'indirizzo risolto dell'API
00BA1589	8385 51294400	ADD DWORD PTR SS:[EBP+442951],4	Si sposta nella prossima entry da scrivere
00BA1590	^ E9 32FFFFFF	JMP 00BA14C7	continua loop interno
00BA1595	8906	MOV DWORD PTR DS:[ESI],EAX	
00BA1597	8946 0C	MOV DWORD PTR DS:[ESI+C],EAX	
00BA159A	8946 10	MOV DWORD PTR DS:[ESI+10],EAX	
00BA159D	83C6 14	ADD ESI,14	
00BA15A0	8B95 D8304400	MOV EDX,DWORD PTR SS:[EBP+4430D8]	
00BA15A6	^ E9 EBFEFFFF	JMP 00BA1496	continua loop esterno
00BA15AB	8B85 652A4400	MOV EAX,DWORD PTR SS:[EBP+442A65]	
00BA15B1	50	PUSH EAX	
00BA15B2	0385 D8304400	ADD EAX,DWORD PTR SS:[EBP+4430D8]	
00BA15B8	5B	POP EBX	
00BA15B9	0BDB	OR EBX,EBX	
00BA15BB	8985 112F4400	MOV DWORD PTR SS:[EBP+442F11],EAX	Scrive l'indirizzo a cui saltare
00BA15C1	61	POPAD	
00BA15C2	✓ 75 08	JNZ SHORT 00BA15CC	
00BA15C4	B8 01000000	MOV EAX,1	
00BA15C9	C2 0C00	RETN 0C	
00BA15CC	68 00000000	PUSH 0	
00BA15D1	C3	RETN	
00BA15D2	8B85 DC304400	MOV EAX,DWORD PTR SS:[EBP+4430DC]	

executing the MOV instruction set the address where we've to jump after the RETN execution.

00BA1587	8907	MOV DWORD PTR DS:[EDI],EAX	Scrive l'indirizzo risolto dell'API
00BA1589	8385 51294400	ADD DWORD PTR SS:[EBP+442951],4	Si sposta nella prossima entry da scrivere
00BA1590	^ E9 32FFFFFF	JMP 00BA14C7	continua loop interno
00BA1595	8906	MOV DWORD PTR DS:[ESI],EAX	
00BA1597	8946 0C	MOV DWORD PTR DS:[ESI+C],EAX	
00BA159A	8946 10	MOV DWORD PTR DS:[ESI+10],EAX	
00BA159D	83C6 14	ADD ESI,14	
00BA15A0	8B95 D8304400	MOV EDX,DWORD PTR SS:[EBP+4430D8]	
00BA15A6	^ E9 EBFEFFFF	JMP 00BA1496	continua loop esterno
00BA15AB	8B85 652A4400	MOV EAX,DWORD PTR SS:[EBP+442A65]	
00BA15B1	50	PUSH EAX	
00BA15B2	0385 D8304400	ADD EAX,DWORD PTR SS:[EBP+4430D8]	
00BA15B8	5B	POP EBX	
00BA15B9	0BDB	OR EBX,EBX	
00BA15BB	8985 112F4400	MOV DWORD PTR SS:[EBP+442F11],EAX	Scrive l'indirizzo a cui saltare
00BA15C1	61	POPAD	
00BA15C2	✓ 75 08	JNZ SHORT 00BA15CC	
00BA15C4	B8 01000000	MOV EAX,1	
00BA15C9	C2 0C00	RETN 0C	
00BA15CC	68 DC89B900	PUSH 0B89B900	
00BA15D1	C3	RETN	
00BA15D2	8B85 DC304400	MOV EAX,DWORD PTR SS:[EBP+4430DC]	

Our next redirection will be place at the offset 0x00315C1:

**OFFSET (redirection #8):** 0x00315C1

**ORIGINAL INSTRUCTION:** POPAD/JNZ



00B989DC	55	PUSH EBP	chordpic.004EB430
00B989DD	8BEC	MOV EBP,ESP	
00B989DF	83C4 B4	ADD ESP,-4C	
00B989E2	B8 D487B900	MOV EAX,0B987D4	
00B989E7	E8 50CCFDFF	CALL 00B7563C	
00B989EC	E8 DBAFDFFF	CALL 00B734CC	
00B989F1	8D40 00	LEA EAX,DWORD PTR DS:[EAX]	
00B989F4	0000	ADD BYTE PTR DS:[EAX],AL	
00B989F6	0000	ADD BYTE PTR DS:[EAX],AL	
00B989F8	0000	ADD BYTE PTR DS:[EAX],AL	
00B989FA	0000	ADD BYTE PTR DS:[EAX],AL	

At this point may be useful for further analysis to put a hardware breakpoint on execution at this new entry (offset 0x00289DC), this will save more time when we've to restart the target.

Now is time to deal about CRC checking (integrity check for the hard-disk code). Then press CTRL+B and write this binary string to search:

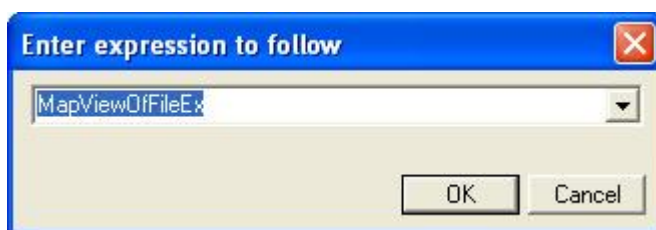
We press OK, the specific sequence comes beginning at offset 0x0018662:

00B88650	8B40 18	MOV EAX,DWORD PTR DS:[EAX+18]
00B88653	FFD0	CALL NEAR EAX
00B88655	833D 14B4B900	CMP DWORD PTR DS:[B9B414],0
00B8865C	0F84 66040000	JE 00B88AC8
00B88662	6A 00	PUSH 0
00B88664	6A 00	PUSH 0
00B88666	6A 00	PUSH 0
00B88668	6A 04	PUSH 4
00B8866A	A1 14B4B900	MOV EAX,DWORD PTR DS:[B9B414]
00B8866F	50	PUSH EAX
00B88670	A1 E497B900	MOV EAX,DWORD PTR DS:[B997E4]
00B88675	8B40 08	MOV EAX,DWORD PTR DS:[EAX+8]
00B88678	FFD0	CALL NEAR EAX
00B8867A	8BD8	MOV EBX,EAX
00B8867C	50	PUSH EAX
00B8867D	E8 4A010000	CALL 00B887CC

this code is relative to the file mapping through the **MapViewOfFileEx** which is called by the CALL EAX instruction through some emulated code (in my machine at the moment in which I write EAX pointed to the address 0x00A20000).

00A20000	8BFF	MOV EDI,EDI
00A20002	55	PUSH EBP
00A20003	8BEC	MOV EBP,ESP
00A20005	6A 00	PUSH 0
00A20007	FF75 18	PUSH DWORD PTR SS:[EBP+18]
00A20009	FF75 14	PUSH DWORD PTR SS:[EBP+14]
00A2000D	FF75 10	PUSH DWORD PTR SS:[EBP+10]
00A20010	FF75 0C	PUSH DWORD PTR SS:[EBP+0C]
00A20013	FF75 08	PUSH DWORD PTR SS:[EBP+08]
00A20016	68 A8B7807C	PUSH 7C80B7A8
00A20018	68 1EB7807C	PUSH kernel32.MapViewOfFileEx
00A20020	C3	RETN
00A20021	0000	ADD BYTE PTR DS:[EAX],AL
00A20023	0000	ADD BYTE PTR DS:[EAX],AL
00A20025	0000	ADD BYTE PTR DS:[EAX],AL

We put therefore the breakpoint in the entry point of the **MapViewOfFileEx**.



7C80B71E	8BFF	MOV EDI,EDI
7C80B720	55	PUSH EBP
7C80B721	8BEC	MOV EBP,ESP
7C80B723	51	PUSH ECX
7C80B724	51	PUSH ECX
7C80B725	8B45 14	MOV EAX,DWORD PTR SS:[EBP+14]
7C80B728	8945 F8	MOV DWORD PTR SS:[EBP-8],EAX
7C80B72B	8B45 10	MOV EAX,DWORD PTR SS:[EBP+10]
7C80B72E	8945 FC	MOV DWORD PTR SS:[EBP-4],EAX
7C80B731	8B45 18	MOV EAX,DWORD PTR SS:[EBP+18]
7C80B734	8945 10	MOV DWORD PTR SS:[EBP+10],EAX
7C80B737	8B45 1C	MOV EAX,DWORD PTR SS:[EBP+1C]
7C80B73A	8945 14	MOV DWORD PTR SS:[EBP+14],EAX
7C80B73D	8B45 0C	MOV EAX,DWORD PTR SS:[EBP+C]
7C80B740	83F8 01	CMP EAX,1
7C80B743	0F84 8F150300	JE kernel32.7C83CCD8
7C80B746	8B45 14	MOV EAX,DWORD PTR SS:[EBP+14]

We keep some break, and therefore we've to repeat the execution until we reach the code showed above (use combination ALT+M break on access in the run-time area then ALT+F9 and Shift+F9), after some step we're able to found the key point after that we've the file image mapped in memory, from this you can use the trick of the binary search to find the key point in a more fast way.

ASProtect, by using the **MapViewOfFile** API, has created an image of file which is in our hard-disk, this check is to discover some code alteration which have been written in hardcoded way, this check is also only for the original file because the code at the OEP isn't still written.

**NOTE for *MapViewOfFile***

From the MSDN it is had:

**MapViewOfFile**

The **MapViewOfFile** function maps a view of a file mapping into the address space of a calling process.

To specify a suggested base address for the view, use the [MapViewOfFileEx](#) function. However, this practice is not recommended.

```
LPVOID MapViewOfFile(
    HANDLE hFileMappingObject,
    DWORD dwDesiredAccess,
    DWORD dwFileOffsetHigh,
    DWORD dwFileOffsetLow,
    SIZE_T dwNumberOfBytesToMap
);
```

PUSH 4 sets up the flag for the modalities of access to the area that contains the file image (mapping) to value FILE\_MAP\_READ, we must have the way to modify the byte inside of the mapped image in order to fake ASProtect and show that the file was not modified. In order to do this we must change the attribute of access to FILE\_MAP\_COPY that correspond to the value 0x01, therefore we will have to replace the original PUSH 4 with a new PUSH 1 and later restore the original code to avoid other detection from ASProtect.

FILE\_MAP\_COPY = 0x01

FILE\_MAP\_WRITE = 0x02

FILE\_MAP\_READ = 0x04

Value	Meaning
FILE_MAP_WRITE	Read/write access. The mapping object must be created with PAGE_READWRITE protection. A read/write view of the file is mapped.
FILE_MAP_READ	Read-only access. The mapping object must be created with PAGE_READWRITE or PAGE_READONLY protection. A read-only view of the file is mapped.
FILE_MAP_COPY	Copy-on-write access. The mapping object must be created with PAGE_WRITECOPY protection.  The system commits physical storage from the paging file at the time that <b>MapViewOfFile</b> is called. The actual physical storage is not used until a thread in the process writes to an address in the view. At that time, the system copies the original page to a new page that is backed by the paging file, maps the page into the process address space, and changes the page protection to PAGE_READWRITE. The threads in the process can access only the local copy of the data, not the original data. If the page is ever trimmed from the working set of the process, it can be written to the paging file storage that is committed when <b>MapViewOfFile</b> is called.  This process only allocates physical memory when a virtual address is actually written to. Changes are never written back to the original file, and are freed when the thread in your process unmaps the view.  Paging file space for the entire view is committed when copy-on-write access is specified, because the thread in the process can write to every single page. Therefore, enough physical storage space must be obtained at the time <b>MapViewOfFile</b> is called.
FILE_MAP_EXECUTE	Execute access. Code can be run from the mapped memory. The mapping object must be created with PAGE_EXECUTE_READWRITE or PAGE_EXECUTE_READ access.  <b>Windows Server 2003 and Windows XP:</b> This feature is not available until Windows XP SP2 and Windows Server 2003 SP1.

The PUSH 4 offset is equal to **0x0018669**, and also we've to perform another redirection just after the **MapViewOfFile** API execution, this must be made at the offset **0x001867A**.

**OFFSET (redirection #9):** 0x001867A

**ORIGINAL INSTRUCTION:** MOV EBX, EAX

Into the redirection code cave we will use first the value which is in EAX since the content of this registry supplies the base address from where it begins the file mapping in memory.

After some tracing we reach the verification 61:

00B88EE9	83C5 18	ADD EBP,18	
00B88EEC	6A 04	PUSH 4	
00B88EEF	8D4D E4	LEA ECX,DWORD PTR SS:[EBP-1C]	
00B88EF1	8B15 20B4B900	MOV EDX,DWORD PTR DS:[B9B420]	
00B88EF7	A1 1CB4B900	MOV EAX,DWORD PTR DS:[B9B41C]	
00B88EFC	E8 3389FFFF	CALL 00B81834	
00B88F01	EB 0A	JMP SHORT 00B88F0D	
00B88F03	68 7892B800	PUSH 0B89278	
00B88F08	E8 0FC1FFFF	CALL 00B8501C	ASCII "61J"
00B88F0D	8BC6	MOV EAX,ESI	
00B88F0F	E8 0C9CFE9F	CALL 00B72B20	
00B88F14	52	PUSH EDX	
00B88F15	E8 2B000000	CALL 00B88F45	
00B88F1A	57	PUSH EDI	

we can step after this check and after some other tracing finally we keep the cycle that is taken care to decompress the code of the program.

00B890C6	C600 CF	MOV BYTE PTR DS:[EAX],0CF	
00B890C9	EB 71	JMP SHORT 00B8913C	
00B890CB	E8 7494FEFF	CALL 00B72544	Inizio loop
00B890D0	8BF0	MOV ESI,EAX	
00B890D2	8B03	MOV EAX,DWORD PTR DS:[EBX]	
00B890D4	0345 E8	ADD EAX,DWORD PTR SS:[EBP-18]	
00B890D7	8945 FC	MOV DWORD PTR SS:[EBP-4],EAX	
00B890DA	8B4B 04	MOV ECX,DWORD PTR DS:[EBX+4]	
00B890DD	8BD6	MOV EDX,ESI	
00B890DF	8B45 FC	MOV EAX,DWORD PTR SS:[EBP-4]	
00B890E2	E8 4DEFFFFF	CALL 00B88034	
00B890E7	8B7B 04	MOV EDI,DWORD PTR DS:[EBX+4]	
00B890EA	8A45 FC	MOV AL,BYTE PTR SS:[EBP-4]	
00B890ED	8B15 5497B900	MOV EDX,DWORD PTR DS:[B99754]	
00B890F3	8802	MOV BYTE PTR DS:[EDX],AL	
00B890F5	8BC7	MOV EAX,EDI	
00B890F7	8B15 6C97B900	MOV EDX,DWORD PTR DS:[B9976C]	
00B890FD	8802	MOV BYTE PTR DS:[EDX],AL	
00B890FF	007D F7 00	CMP BYTE PTR SS:[EBP-9],0	
00B89103	75 1E	JNZ SHORT 00B89123	
00B89105	C645 F7 01	MOV BYTE PTR SS:[EBP-9],1	
00B89109	56	PUSH ESI	
00B8910A	8B75 FC	MOV ESI,DWORD PTR SS:[EBP-4]	
00B8910D	83C6 14	ADD ESI,14	
00B89110	FF36	PUSH DWORD PTR DS:[ESI]	
00B89112	C606 C3	MOV BYTE PTR DS:[ESI],0C3	
00B89115	FFD6	CALL NEAR ESI	
00B89117	8F06	POP DWORD PTR DS:[ESI]	
00B89119	5E	POP ESI	
00B8911A	8BD7	MOV EDX,EDI	
00B8911C	8BC6	MOV EAX,ESI	
00B8911E	E8 75F0FFFF	CALL 00B88198	
00B89123	8BCF	MOV ECX,EDI	
00B89125	8BD6	MOV EDX,ESI	
00B89127	8B45 FC	MOV EAX,DWORD PTR SS:[EBP-4]	
00B8912A	E8 8DC7FEFF	CALL 00B758BC	Decomprime il codice
00B8912F	8B53 04	MOV EDX,DWORD PTR DS:[EBX+4]	
00B89132	8BC6	MOV EAX,ESI	
00B89134	E8 2394FEFF	CALL 00B7255C	
00B89139	83C3 0C	ADD EBX,0C	
00B8913C	8B43 04	MOV EAX,DWORD PTR DS:[EBX+4]	
00B8913F	85C0	TEST EAX,EAX	
00B89141	77 88	JL SHORT 00B890CB	Continua il loop
00B89143	837D F0 00	CMP DWORD PTR SS:[EBP-10],0	Fine loop
00B89147	74 08	JE SHORT 00B89151	
00B89149	8B45 F0	MOV EAX,DWORD PTR SS:[EBP-10]	
00B8914C	8B55 EC	MOV EDX,DWORD PTR SS:[EBP-14]	
00B8914F	8910	MOV DWORD PTR DS:[EAX],EDX	
00B89151	B8 03000000	MOV EAX,3	

Address	Hex dump	Disassembly	Comment
00499914	55	DB 55	CHAR 'U'
00499915	8B	DB 8B	
00499916	EC	DB EC	
00499917	83	DB 83	
00499918	C4	DB C4	
00499919	EC	DB EC	
0049991A	53	DB 53	CHAR 'S'
0049991B	33	DB 33	CHAR '3'
0049991C	C0	DB C0	
0049991D	89	DB 89	
0049991E	45	DB 45	CHAR 'E'
0049991F	EC	DB EC	

in order to see the code return to the OllyDbg code window, press CTRL+G and write the OEP address, then press CTRL+A to force the code analysis.

00499914	55	PUSH EBP	
00499915	8BEC	MOV EBP,ESP	
00499917	83C4 EC	ADD ESP,-14	
0049991A	53	PUSH EBX	
0049991B	33C0	XOR EAX,EAX	
0049991D	8945 EC	MOV DWORD PTR SS:[EBP-14],EAX	
00499920	B8 24964900	MOV EAX,chordpic.00499624	
00499925	E8 86CDF6FF	CALL chordpic.004066B0	
0049992A	8B1D F0E04900	MOV EBX,DWORD PTR DS:[49E0F0]	chordpic.0049FBC0
00499930	33C0	XOR EAX,EAX	
00499932	55	PUSH EBP	
00499933	68 4E9A4900	PUSH chordpic.00499A4E	

Good, start again with tracing, skip the exception and you can reach a loop able to read the decompressed code in order to detect some memory patching.

00B7CED7	83FB 40	CMP EBX,40	
00B7CEDA	7C 14	JL SHORT 00B7CEF0	
00B7CEDC	8BD7	MOV EDI,EDI	
00B7CEDE	8BC6	MOV EAX,ESI	
00B7CEE0	8B08	MOV ECX,DWORD PTR DS:[EAX]	
00B7CEE2	FF51 14	CALL NEAR DWORD PTR DS:[ECX+14]	
00B7CEE5	83C7 40	ADD EDI,40	
00B7CEE8	83EB 40	SUB EBX,40	
00B7CEEB	83FB 40	CMP EBX,40	
00B7CEEE	7D EC	JGE SHORT 00B7CEDC	
00B7CEF0	8B0424	MOV EAX,DWORD PTR SS:[ESP]	Loop lettura codice per verifica CRC chordpic.<ModuleEntryPoint>
00B7CEF3	03C5	ADD EAX,EBP	
00B7CEF5	8D56 08	LEA EDX,DWORD PTR DS:[ESI+8]	
00B7CEF8	8BCB	MOV ECX,EBX	
00B7CEFA	E8 5157FFFF	CALL 00B72650	
00B7CEFF	5A	POP EDX	
00B7CF00	5D	POP EBP	
00B7CF01	5F	POP EDI	
00B7CF02	5E	POP ESI	
00B7CF03	5B	POP EBX	
00B7CF04	C3	RETN	
00B7CF05	8D40 00	LEA EAX,DWORD PTR DS:[EAX]	

after some step we reach another good point, this is the check point 45. This check perform a memory check for the decompressed code, if you patch the target before this check one error arise.

00B8A32F	8B00	MOV EAX,DWORD PTR DS:[EAX]	
00B8A331	03C7	ADD EAX,EDI	
00B8A333	8B5424 0C	MOV EDX,DWORD PTR SS:[ESP+C]	
00B8A337	E8 D873FFFF	CALL 00B81714	
00B8A33C	8BE8	MOV EBP,EAX	EAX = 5A935349
00B8A33E	892D 3CB4B900	MOV DWORD PTR DS:[B9B43C],EBP	
00B8A344	3B6C24 10	CMP EBP,DWORD PTR SS:[ESP+10]	[ESP+10]=5A935349 (CRC)
00B8A348	74 0C	JE SHORT 00B8A356	
00B8A34A	68 DCA3B800	PUSH 00B8A3DC	ASCII "45J"
00B8A34F	E8 C8ACFFFF	CALL 00B8501C	
00B8A354	EB 12	JMP SHORT 00B8A368	
00B8A356	8B4424 0C	MOV EAX,DWORD PTR SS:[ESP+C]	
00B8A35A	A3 38B4B900	MOV DWORD PTR DS:[B9B438],EAX	
00B8A35F	8B4424 08	MOV EAX,DWORD PTR SS:[ESP+8]	
00B8A363	A3 34B4B900	MOV DWORD PTR DS:[B9B434],EAX	

The right CRC is keep into the stack at the address [ESP+10] and in this target is equal to 5A935349.

0012FF50	00000000	
0012FF54	00400000	ASCII "M2P"
0012FF58	00001000	
0012FF5C	00098C00	
0012FF60	5A935349	CRC
0012FF64	0012FF98	
0012FF68	00B70000	
0012FF6C	00B30000	

Now we can apply our patches to the address 0x0048CB79 just after the check, the program is able to run fine and the patches is able to work too.

0048CB67	B0 B0	MOV AL,0B0	
0048CB69	CD 45	INT 45	
0048CB6B	E8 C0F9FFFF	CALL chordpic.0048C530	
0048CB70	84C0	TEST AL,AL	
0048CB72	90	NOP	PATCH: NOP
0048CB73	90	NOP	
0048CB74	A1 ECDD4900	MOV EAX,DWORD PTR DS:[49DDEC]	
0048CB79	C600 00	MOV BYTE PTR DS:[EAX],0	PATCH: FORCE 0
0048CB7C	A1 ECDD4900	MOV EAX,DWORD PTR DS:[49DDEC]	
0048CB81	8038 00	CMP BYTE PTR DS:[EAX],0	
0048CB84	0F84 F2000000	JE chordpic.0048CC7C	LOOK AT THIS :-)
0048CB8A	33DB	XOR EBX,EBX	
0048CB8C	8D55 F8	LEA EDX,DWORD PTR SS:[EBP-8]	
0048CB8F	A1 84E14900	MOV EAX,DWORD PTR DS:[49E184]	

After this we can apply our patching after the check, then we can write our last redirection point at the offset **0x001A356**.

**OFFSET (redirection #10):** 0x001A356

**ORIGINAL INSTRUCTION:** MOV EAX, DWORD[ESP+C]

## 2. INLINE PATCHING

Now is time to write the code for the inline patching and we must also find a suitable area where to write all the patching code.

The last **.adata** section may be a good choice, this area have a RAW-SIZE set to 0 then we've to enlarge this to meet our need. Run LordPE and load in the packed executable, change the RAW-SIZE of the **.adata** section from 0 to 0x1000 (it must be big enough for all the patches) and save the changes.

Now run Hex-Workshop and load the packed executable, scroll down to the end of the file and insert a 0x1000 byte of value 0x90 then save all the change.

Now if you try to run the executable one error arise:



this is due to the integrity check of the file image through the **MapViewOfFile**.

Another important aspect is given from the fact that ASProtect, once executed the **MapViewOfFile**, fill with zeroes part of the **.adata** area, this means that we have to check what is the area used from ASProtect and redirect just after the last byte used from ASProtect itself

The code already previously written will be erased, but this isn't really important because this code has work previously and now isn't useful.

Making this analysis the first free area it turns out from the address 0x0050DCD6, this will be the address to which redirect after the **MapViewOfFile** execution.

We can write the following code of patching:

```
004EB1AA JMP chordpic.0050D100; Redirection to cave 1 from hardcoded jump

0050D100 MOV DWORD PTR DS:[4EB267], 21EA3E9; Cave 1
0050D10A JMP chordpic.004EB1CA

0050D10F MOV DWORD PTR DS:[4EB32D], 21DECE9; Cave 2
0050D119 JMP chordpic.004EB290

0050D11E MOV DWORD PTR DS:[4EB410], 21D18E9; Cave 3
0050D128 JMP chordpic.004EB373

0050D12D MOV DWORD PTR DS:[4EB410], 12E8 ; Cave 4 (restoration of the original code of the call)
0050D137 MOV DWORD PTR DS:[4EB5E8], 21B59E9; Redirection to the cave 5
0050D141 JMP chordpic.004EB410
```

```

0050D146 MOV DWORD PTR DS:[4EB5E8], 800068; Cave 5 (restoration the code of PUSH 8000)
0050D150 MOV DWORD PTR DS:[50DFFC], EDI; Storage for the base address (at the end of the area)
0050D156 MOV DWORD PTR DS:[EDI+310F3], 50D1E68; Redirection to the cave 6
0050D160 MOV WORD PTR DS:[EDI+310F7], 0C300
0050D169 JMP chordpic.004EB5E8

0050D16E PUSHAD ; Cave 6 (save the target context)
0050D16F PUSHFD
0050D170 MOV EAX, DWORD PTR DS:[50DFFC] ; Load the base address
0050D175 MOV DWORD PTR DS:[EAX+310F3], 800068; Restoration PUSH 8000 (offset 310F3)
0050D17F MOV BYTE PTR DS:[EAX+310F7], 0
0050D186 MOV BYTE PTR DS:[EAX+310F8], 6A
0050D18D MOV DWORD PTR DS:[EAX+313D7], 50D1B268; It goes to next PUSH 8000 (offset 313D7) for cave 7
0050D197 MOV WORD PTR DS:[EAX+313DB], 0C300
0050D1A0 ADD EAX, 310F3 ; Calculate the return address
0050D1A5 MOV DWORD PTR DS:[50D1AD], EAX
0050D1AA POPFD
0050D1AB POPAD
0050D1AC PUSH 0
0050D1B1 RETN

0050D1B2 PUSHAD ; Cave 7
0050D1B3 PUSHFD
0050D1B4 MOV EAX, DWORD PTR DS:[50DFFC]
0050D1B9 MOV DWORD PTR DS:[EAX+313D7], 800068 ; Restoration PUSH 8000 to the offset one 313D7
0050D1C3 MOV BYTE PTR DS:[EAX+313DB], 0
0050D1CA MOV BYTE PTR DS:[EAX+313DC], 6A
0050D1D1 MOV DWORD PTR DS:[EAX+315C1], 50D1F668; Cave 8 goes to the POPAD (offset 315C1) for cave 8
0050D1DB MOV WORD PTR DS:[EAX+315C5], 0C300
0050D1E4 ADD EAX, 313D7 ; Calculate the return address
0050D1E9 MOV DWORD PTR DS:[50D1F1], EAX
0050D1EE POPFD
0050D1EF POPAD
0050D1F0 PUSH 0
0050D1F5 RETN

0050D1F6 PUSHAD ; Cave 8
0050D1F7 PUSHFD
0050D1F8 MOV EAX, DWORD PTR DS:[50DFFC]
0050D1FD MOV DWORD PTR DS:[EAX+315C1], B8087561; Restoration POPAD/JNZ to the offset one 315C1
0050D207 MOV WORD PTR DS:[EAX+315C5], 1
0050D210 MOV BYTE PTR DS:[EAX+18669], 1 ; Patch the PUSH 4 to PUSH 1 (offset 18669)
0050D217 MOV DWORD PTR DS:[EAX+1867A], 50DCD668; It goes to MOV EBX, EAX for cave 9
0050D221 MOV WORD PTR DS:[EAX+1867E], 0C300
0050D22A ADD EAX, 315C1 ; Calculates the return address
0050D22F MOV DWORD PTR DS:[50D237], EAX
0050D234 POPFD
0050D235 POPAD
0050D236 PUSH 0BE13D7
0050D23B RETN

```

Now we have executed also the **MapViewOfFile**, the first area within the **.adata** section has been erased from ASProtect, then the cave 9 redirection will have to be made jumping the address **0x0050DCD6**.

When we're into the cave 9, since in EAX we've the base address of the file mapping image we've to restore into the image the RAW SIZE for the **.adata** section and restore the code of the first hardcoded jump which is to the address 0x004EB267.

The offset for the first JMP redirection into the mapping file image is easy to find, into the OllyDbg dump-window press CTRL+G and write the address which is in EAX (in my case 0x00D70000) then press OK:





Address	Hex dump	ASCII
00DF0000	4D 5A 50 00 02 00 00 00 04 00 0F 00 FF FF 00 00	MZP.0...0.0. ..
00DF0010	B8 00 00 00 00 00 00 00 40 00 1A 00 00 00 00	0.....0.+.....
00DF0020	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....0.....
00DF0030	00 00 00 00 00 00 00 00 00 00 00 00 01 00 00	.....0.....
00DF0040	BA 10 00 0E 1F B4 09 CD 21 B8 01 4C CD 21 90 90	.0?+.=?00L=?éé
00DF0050	54 68 69 73 20 70 72 6F 67 72 61 6D 20 6D 75 73	This program must
00DF0060	74 20 62 65 20 72 75 6E 20 75 6E 64 65 72 20 57	be run under W
00DF0070	69 6E 33 32 00 0A 24 37 00 00 00 00 00 00 00	in32..\$7.....
00DF0080	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00DF0090	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....

then press Ctrl+B and write the pattern that we have to search to looking for the JMP offset (remember also to check the **Entire block**):

JMP 0050D100 -> E9 51 1F 02 00

press OK:

Address	Hex dump	ASCII
00E537AA	E9 51 1F 02 00 F9 3E 9F EC B5 80 04 F0 E9 4D FF	úQv0.->f94CÉ-0M
00E537BA	FF FF EE 8F 1C 25 FA AB 08 A1 C6 87 B4 0D 52 23	-AL%:0i0iR#
00E537CA	41 E6 CE F3 77 4D FF 2D 0B 64 3A 56 1C 4B 17 02	Avi%wM-0d:ULK0
00E537DA	58 07 B2 6E 54 23 0F 9A 10 B6 D7 3A 0B 64 EF 31	Xi0nT#*00i:0d'1
00E537EA	C8 87 A2 9E C4 D3 FE C9 80 DF 1A 94 66 43 48 89	E00X-E=IF0000fChé

In order to see the code right click -> Disassemble:

Address	Hex dump	Disassembly	Comment
00E537AA	E9 511F0200	JMP 00E75700	
00E537AF	F9	STC	
00E537B0	3E:9F	LAHF	
00E537B2	EC	IN AL,DX	Superfluous prefix
00E537B3	B5 80	MOV CH,80	I/O command
00E537B5	04 F0	RAM 0F0	
00E537B7	E9 40FFFFFF	JMP 00E53709	
00E537BC	EE	OUT DX,AL	I/O command
00E537BD	8F	???	Unknown command
00E537BE	1C 25	SBB AL,25	
00E537C0	FA	CLI	

well done, this is the code that we're searching for.

Address	Hex dump	Disassembly	Comment
004EB1AA	E9 511F0200	JMP chordpic.0050D100	LOOP#1
004EB1AF	F9	STC	
004EB1B0	3E:9F	LAHF	
004EB1B2	EC	IN AL,DX	Superfluous prefix
004EB1B3	B5 80	MOV CH,80	I/O command
004EB1B5	04 F0	RAM 0F0	
004EB1B7	E9 40FFFFFF	JMP chordpic.004EB109	
004EB1BC	EE	OUT DX,AL	I/O command
004EB1BD	8F	???	Unknown command
004EB1BE	1C 25	SBB AL,25	
004EB1C0	FA	CLI	

The code to modify in order to restore the first jump in the file image is found therefore to the offset **0x00637AA**.

Now we can write the first code for the cave 9.

```
0050DCD6 MOV BYTE PTR DS:[EAX+399], 0; Cave 9 (restores size of raw given)
0050DCDD MOV DWORD PTR DS:[EAX+637AA], 1BE9; It restores first jump
```

Now we've restored the file mapped image in memory, remains to put the next redirection just after the memory checking.

Below the full code for cave 9:

```
0050DCD6 MOV BYTE PTR DS:[EAX+399], 0; Cave 9 (restores size of raw given)
0050DCDD MOV DWORD PTR DS:[EAX+637AA], 1BE9; It restores first jump
0050DCE7 PUSHAD
0050DCE8 PUSHFD
0050DCE9 MOV EAX, DWORD PTR DS:[50DFFC]; It loads the base address
0050DCEE MOV BYTE PTR DS:[EAX+18669], 4; PUSH 1 - > PUSH 4
0050DCF5 MOV DWORD PTR DS:[EAX+1867A], E850D88B; It restores MOV EBX, EAX
0050DCFF MOV WORD PTR DS:[EAX+1867E], 14A
0050DD08 MOV DWORD PTR DS:[EAX+1A356], 50DD2D68; Redirezione to cave 10
0050DD12 MOV WORD PTR DS:[EAX+1A35A], 0C300
0050DD1B ADD EAX, 1867A; it calculates the return address
0050DD20 MOV DWORD PTR DS:[50DD28], EAX
0050DD25 POPFD
0050DD26 POPAD
0050DD27 PUSH 0
0050DD2C RETN
```

From the previous analysis we know that we have to skip the check 45 before apply our patches then we can write our last cave code.

```
0050DD2D PUSHAD; Cave 10
0050DD2E PUSHFD
0050DD2F MOV EAX, DWORD PTR DS:[50DFFC]
0050DD34 MOV DWORD PTR DS:[EAX+1A356], 0C24448B
0050DD3E MOV WORD PTR DS:[EAX+1A35A], 38A3
0050DD47 MOV WORD PTR DS:[48CB72], 9090; Patch 1
0050DD50 MOV BYTE PTR DS:[48CB7B], 0; Patch 2
0050DD57 ADD EAX, 1A356
0050DD5C MOV DWORD PTR DS:[50DD64], EAX
0050DD61 POPFD
0050DD62 POPAD
0050DD63 PUSH 0
0050DD68 RETN
```

That's all.



## ThunderPwr of ARTeam

**Thanks to all ARTeam and special thanks goes to Madman H3rCu13S and John Who for the tutorial on ASProtect inline technique.**  
**Also great thanks to Ricardo Narvaja and all Cracks Latinos group.**  
**Thanks to you that have read all the tutorial.**