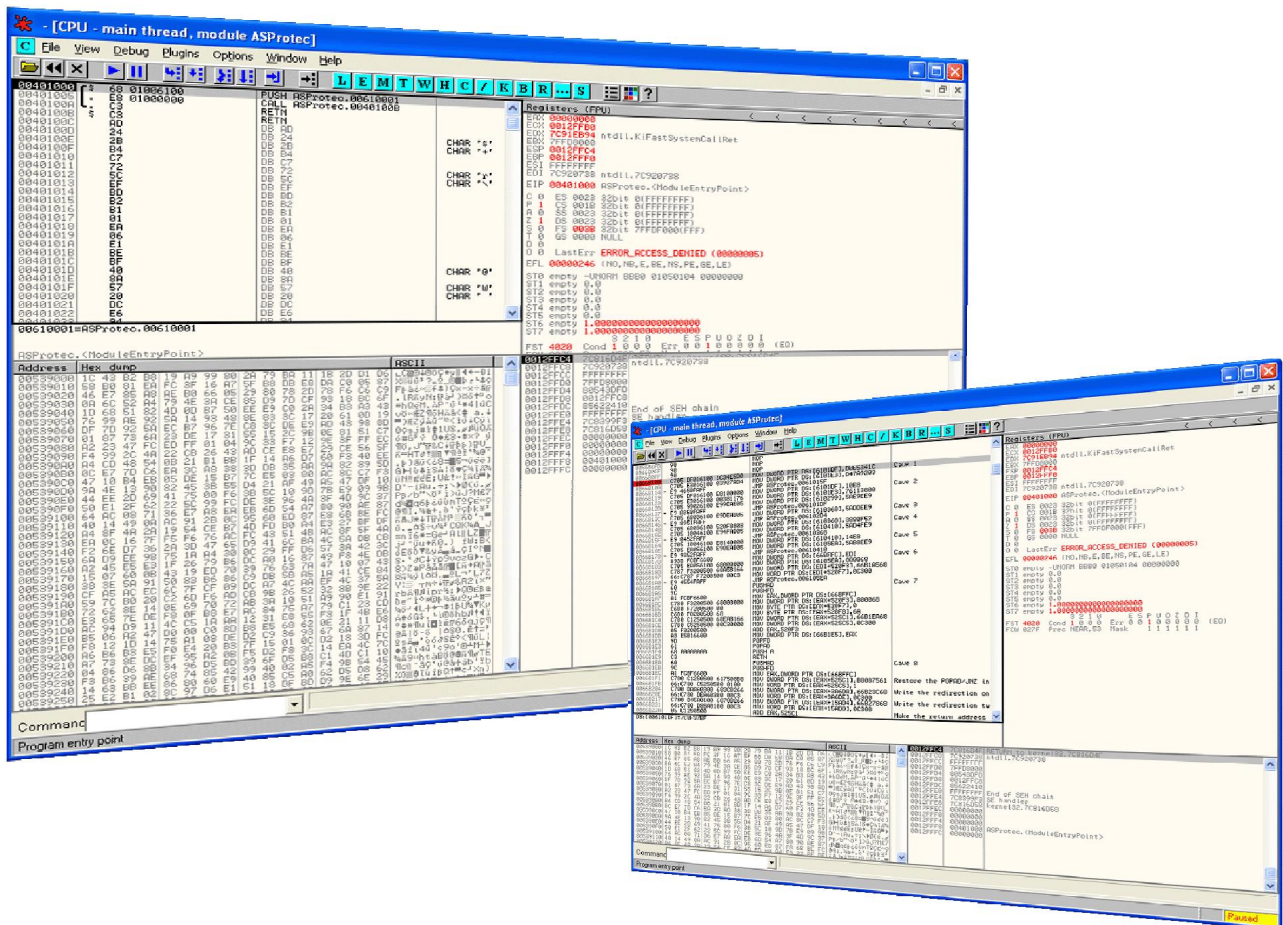




# Inline patching ASProtect 2.3 SKE

Build 04.26 and Build 06.19 tutorial

ThunderPwr

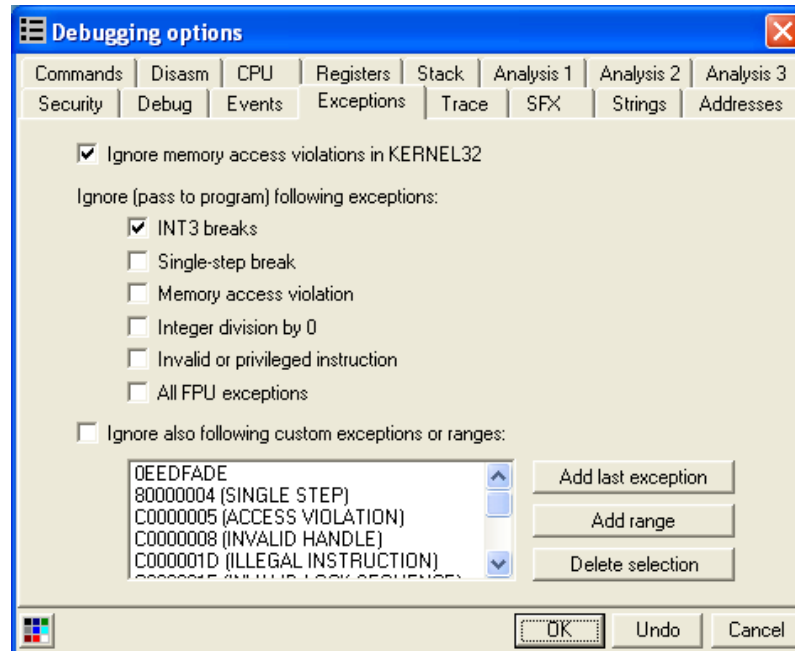


Ver.	Date	Description
1.0	10/08/2006	First private release.
	11/08/2006	First public release.

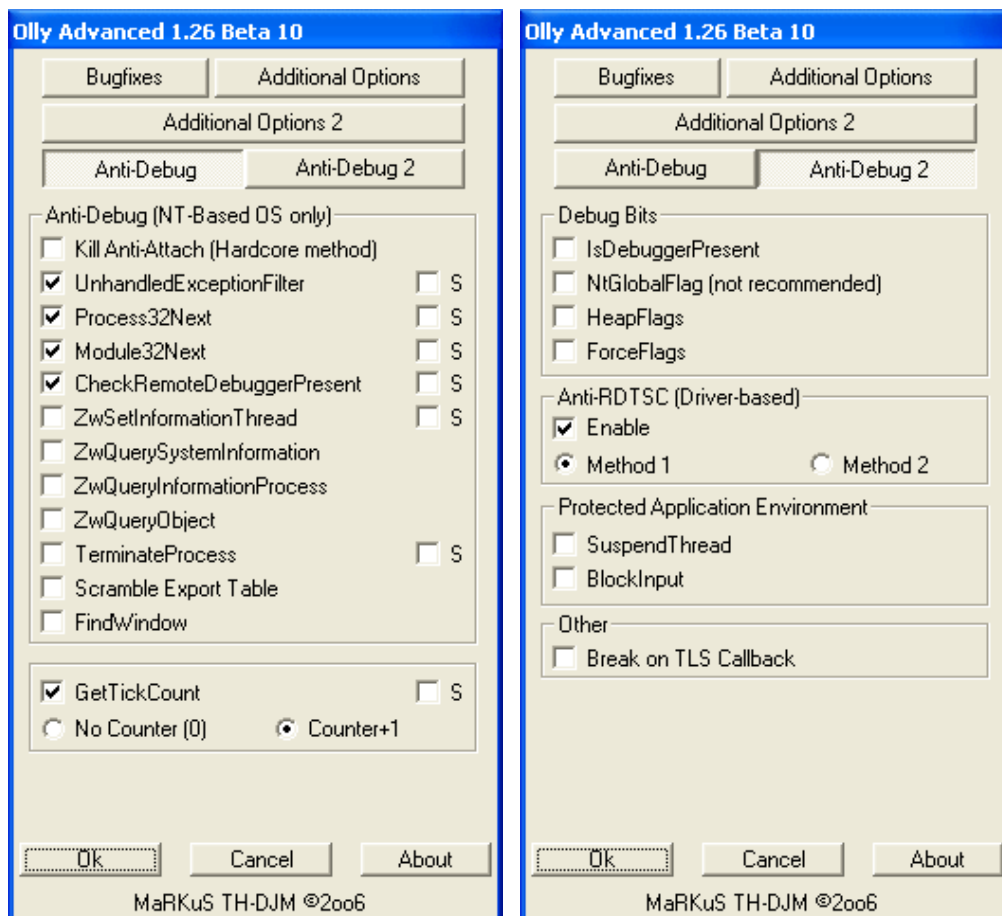
**ASProtect 2.3 SKE 04.26**

The target for this analysis can be downloaded from <http://www.tuts4you.com/> choose the **UnPackMe\_ASProtect.2.3.04.26.g.exe** file.

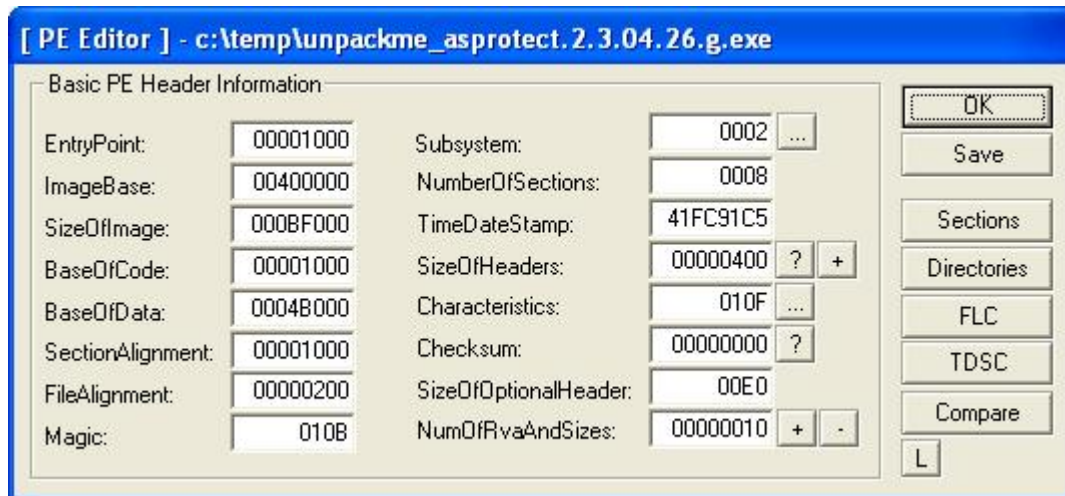
Start your OllyDbg, press ALT+O and sure about this settings:



also set this check into the Olly Advanced plugin.



Run LordPE and take also a look at the packed executable sections:



press the Sections button:

[ Section Table ]						
Name	VOffset	VSize	ROffset	RSize	Flags	
	00001000	0004A000	00000400	00023600	E0000040	
	0004B000	0000C000	00023A00	00002E00	E0000040	
	00057000	00009000	00026800	00001400	E0000040	
	00060000	00003000	00027C00	00002C00	E0000040	
.rsrc	00063000	00008000	0002A800	00000600	E0000040	
	0006B000	00001000	0002AE00	00000200	E0000040	
.data	0006C000	00052000	0002B000	00052000	E0000040	
.adata	000BE000	00001000	0007D000	00000000	E0000040	

as in previous ASProtect version the last section namely **.adata** keep a RAW-SIZE equal to 0.

Well this is enough to start our analysis then run OllyDbg and load in it the packed executable, we have the classical ASProtect entry point:

00401000	68 01C04600	PUSH UnPackMe.0046C001	
00401005	E8 01000000	CALL UnPackMe.0040100B	
0040100A	C3	RETN	
0040100B	C3	RETN	
0040100C	54	DB 54	CHAR 'T'
0040100D	04	DB 04	
0040100E	97	DB 97	

As usual we've to start with some code stepping by using F7, and after a little we should reach the first decryption loop.

This loop have some obfuscated code, but by using NOP instructions it will look good for us.

0046C11A	B9 E7010000	MOV ECX,1E7	
0046C11F	8B10	MOV EDX,DWORD PTR DS:[EAX]	LOOP#1 START
0046C121	81E6 8ABE8250	AND ESI,5082BE8A	
0046C127	81C2 2F0BC00F	ADD EDX,0FC00B2F	
0046C12D	68 7137C420	PUSH 20C43771	
0046C132	E8 09000000	CALL UnPackMe.0046C140	
0046C137	AD	LODS DWORD PTR DS:[ESI]	
0046C138	✓ E2 73	LOOPE SHORT UnPackMe.0046C1AD	
0046C13A	30A9 2ECF5C65	XOR BYTE PTR DS:[ECX+655CCF2E],CH	
0046C140	✓ E9 00000000	JMP UnPackMe.0046C152	
0046C145	✓ E1 06	LOOPE SHORT UnPackMe.0046C14D	
0046C147	C7	???	Unknown command
0046C148	F4	HLT	Privileged command
0046C149	1D 92636019	SBB EAX,19606392	
0046C14E	90	NOP	
0046C14F	90	NOP	
0046C150	90	NOP	
0046C151	90	NOP	
0046C152	5F	POP EDI	
0046C153	5E	POP ESI	
0046C154	81F2 3C45874D	XOR EDX,4D87453C	
0046C15A	0FBFFE	MOVSX EDI,SI	
0046C15D	81EA C5F21802	SUB EDX,218F2C5	
0046C163	BE 89EA457A	MOV ESI,7A45EA89	
0046C168	8910	MOV DWORD PTR DS:[EAX],EDX	
0046C16A	81E8 CBCA906B	SUB EAX,6B90CACB	
0046C170	✓ 0F8D 05000000	JGE UnPackMe.0046C17B	
0046C176	66:81F7 43C8	XOR DI,0C843	
0046C17B	81C0 C7CA906B	ADD EAX,6B90CAC7	
0046C181	E8 13000000	CALL UnPackMe.0046C199	
0046C186	9F	LAHF	
0046C187	EC	IN AL,DX	I/O command
0046C188	B5 4A	MOV CH,4A	
0046C18A	BB D8311697	MOV EBX,971631D8	
0046C18F	846D A2	TEST BYTE PTR SS:[EBP-5E],CH	
0046C192	33F0	XOR ESI,EAX	
0046C194	69EE 8F1C2581	IMUL EBP,ESI,81251C8F	
0046C19A	DF87 26DE0A5F	FILD WORD PTR DS:[EDI+5F0ADE26]	
0046C1A0	49	DEC ECX	
0046C1A1	✓ 0F85 1B000000	JNZ UnPackMe.0046C1C2	
0046C1A7	BE 7F5FDB22	MOV ESI,22DB5F7F	
0046C1AC	✓ E9 38000000	JMP UnPackMe.0046C1E9	LOOP#1 EXIT POINT
0046C1B1	95	XCHG EAX,EBP	
0046C1B2	AA	STOS BYTE PTR ES:[EDI]	
0046C1B3	9B	WAIT	
0046C1B4	3811	CMP BYTE PTR DS:[ECX],DL	
0046C1B6	✓ 76 77	JBE SHORT UnPackMe.0046C22F	
0046C1B8	E4 4D	IN AL,4D	I/O command
0046C1BA	0213	ADD DL,BYTE PTR DS:[EBX]	
0046C1BC	50	PUSH EAX	
0046C1BD	49	DEC ECX	
0046C1BE	4E	DEC ESI	
0046C1BF	6F	OUTS DX,DWORD PTR ES:[EDI]	I/O command
0046C1C0	✓ 7C 05	JL SHORT UnPackMe.0046C1C7	
0046C1C2	68 68F71E42	PUSH 421EF768	
0046C1C7	✓ E9 0C000000	JMP UnPackMe.0046C1D8	
0046C1CC	14 BD	ADC AL,0BD	
0046C1CE	B2 03	MOV DL,3	
0046C1D0	80B9 FE5FAC75	CMP BYTE PTR DS:[ECX+75AC5FFE],0A	
0046C1D7	90	NOP	
0046C1D8	5B	POP EBX	
0046C1D9	^ E9 41FFFFFF	JMP UnPackMe.0046C11F	Continue LOOP#1
0046C1DE	57	PUSH EDI	
0046C1DF	44	INC ESP	
0046C1E0	2D 62F3B029	SUB EAX,29B0F362	
0046C1E5	AE	SCAS BYTE PTR ES:[EDI]	
0046C1E6	4F	DEC EDI	
0046C1E7	DCE5	FSUBR ST(5),ST	
0046C1E9	✓ B9 015DA160	MOV ECX,60A15D01	
0046C1EE	E8 0B000000	CALL UnPackMe.0046C1FE	
0046C1F3	E7 94	OUT 94,EAX	I/O command
0046C1F5	3D 32830039	CMP EAX,39008332	
0046C1FA	^ 7E DF	JLE SHORT UnPackMe.0046C1DB	
0046C1FC	2C F5	SUB AL,0F5	

**LOOP#1 EXIT POINT (Redirection #1 - hardcoded): 0x0046C1AC**  
**ORIGINAL CODE: JMP 46C19E (E9 38 00 00 00)**

The first decryption loop was gone then step again to reach the other one, also we've obfuscated code and some NOP may be help into our analysis:

0046C224	BF DE8AB152	MOV EDI,52B18ADE	
0046C229	FF32	PUSH DWORD PTR DS:[EDX]	LOOP#2 START
0046C22B	81C3 51132116	ADD EBX,16211351	
0046C231	5E	POP ESI	
0046C232	✓ E9 09000000	JMP UnPackMe.0046C240	
0046C237	8D42 53	LEA EAX,DWORD PTR DS:[EDX+53]	
0046C23A	90	NOP	
0046C23B	90	NOP	
0046C23C	90	NOP	
0046C23D	90	NOP	
0046C23E	90	NOP	
0046C23F	90	NOP	
0046C240	81EE 9375080F	SUB ESI,0F087593	
0046C246	68 A817E14E	PUSH 4EE117A8	
0046C24B	68 A7419146	PUSH 469141A7	
0046C250	56	PUSH ESI	
0046C251	5B	POP EBX	
0046C252	5F	POP EDI	
0046C253	59	POP ECX	
0046C254	81F6 D0091F3D	XOR ESI,3D1F09D0	
0046C25A	80C7 BB	ADD BH,0BB	
0046C25D	81C6 C9BD1462	ADD ESI,6214BDC9	
0046C263	56	PUSH ESI	
0046C264	✓ E9 07000000	JMP UnPackMe.0046C270	
0046C269	33F0	XOR ESI,EAX	
0046C26B	90	NOP	
0046C26C	90	NOP	
0046C26D	90	NOP	
0046C26E	90	NOP	
0046C26F	90	NOP	
0046C270	8F02	POP DWORD PTR DS:[EDX]	
0046C272	66:81E1 C683	AND CX,83C6	
0046C277	81EA B43EC971	SUB EDX,71C93EB4	
0046C27D	✓ 0F8D 03000000	JGE UnPackMe.0046C286	
0046C283	66:8BD9	MOV BX,CX	
0046C286	81C2 B03EC971	ADD EDX,71C93EB0	
0046C28C	68 AA1D4352	PUSH 52431DAA	
0046C291	66:B9 4D15	MOV CX,154D	
0046C295	5F	POP EDI	
0046C296	48	DEC EAX	
0046C297	✓ 0F85 24000000	JNZ UnPackMe.0046C2C1	
0046C29D	E8 11000000	CALL UnPackMe.0046C2B3	
0046C2A2	05 5A8B6881	ADD EAX,81688B5A	
0046C2A7	26:67:14 BD	ADC AL,0BD	Superfluons prefix
0046C2AB	B2 03	MOV DL,3	
0046C2AD	80B9 FEFAC75	CMP BYTE PTR DS:[ECX+75AC5FFE],0F	
0046C2B4	B7 CB	MOV BH,0CB	
0046C2B6	5F	POP EDI	
0046C2B7	✓ E9 1A000000	JMP UnPackMe.0046C2D6	LOOP#2 EXIT
0046C2BC	90	NOP	
0046C2BD	90	NOP	
0046C2BE	90	NOP	
0046C2BF	90	NOP	
0046C2C0	90	NOP	
0046C2C1	66:BF 8666	MOV DI,6686	
0046C2C5	^ E9 5FFFFFFF	JMP UnPackMe.0046C229	LOOP#2 END
0046C2CA	^ 74 9D	JE SHORT UnPackMe.0046C269	
0046C2CC	12E3	ADC AH,BL	
0046C2CE	^ E0 99	LOOPDNE SHORT UnPackMe.0046C269	
0046C2D0	5E	POP ESI	
0046C2D1	3F	AAA	
0046C2D2	0C 55	OR AL,55	
0046C2D4	6A 5B	PUSH 5B	
0046C2D6	68 167C0D7F	PUSH 7F0D7C16	
0046C2DB	68 6D0B6B79	PUSH 796B0B6D	
0046C2E0	0FB7DA	MOVZX EBX,DX	

**LOOP#2 EXIT POINT (Redirection #2):** 0x0046C2B7

**ORIGINAL CODE:** JMP 46C19E (E9 1A 00 00 00)

Same as above after some step we reach another loop, again we can use NOP to bring the code in a more readable form:

0046C327	4F	DEC EDI	
0046C328	DCE5	FSUBR ST(5),ST	
0046C32A	90	NOP	
0046C32B	90	NOP	
0046C32C	8B10	MOV EDX,DWORD PTR DS:[EAX]	LOOP#3 START
0046C32E	B7 0A	MOV BH,0A	
0046C330	81F2 F917A07E	XOR EDX,7EA017F9	
0046C336	66:BF 5EB3	MOV DI,0B35E	
0046C33A	81F2 3E430E75	XOR EDX,750E433E	
0046C340	81C2 9FB5836F	ADD EDX,6F83B59F	
0046C346	66:8BF1	MOV SI,CX	
0046C349	52	PUSH EDX	
0046C34A	66:81C6 0EB0	ADD SI,0B00E	
0046C34F	8F00	POP DWORD PTR DS:[EAX]	
0046C351	BB 1ABF7761	MOV EBX,6177BF1A	
0046C356	83E8 01	SUB EAX,1	
0046C359	48	DEC EAX	
0046C35A	48	DEC EAX	
0046C35B	48	DEC EAX	
0046C35C	^ E9 05000000	JMP UnPackMe.0046C366	
0046C361	^ 79 BE	JNS SHORT UnPackMe.0046C321	
0046C363	1F	POP DS	Modification of segment register
0046C364	6C	INS BYTE PTR ES:[EDI],DX	I/O command
0046C365	90	NOP	
0046C366	81E9 01000000	SUB ECX,1	
0046C36C	^ 0F85 13000000	JNZ UnPackMe.0046C385	
0046C372	81D6 2241E016	ADC ESI,16E04122	
0046C378	^ E9 16000000	JMP UnPackMe.0046C393	LOOP#3 EXIT
0046C37D	^ 70 E9	J0 SHORT UnPackMe.0046C368	
0046C37F	6E	OUTS DX,BYTE PTR ES:[EDI]	I/O command
0046C380	90	NOP	
0046C381	90	NOP	
0046C382	90	NOP	
0046C383	90	NOP	
0046C384	90	NOP	
0046C385	^ E9 A2FFFFFF	JMP UnPackMe.0046C32C	LOOP#3 END
0046C38A	5D	POP EBP	
0046C38B	D2A3 A0591EFF	SHL BYTE PTR DS:[EBX+FF1E59A0],CL	
0046C391	CC	INT3	
0046C392	15 8BC3E810	ADC EAX,10E8C38B	
0046C397	0000	ADD BYTE PTR DS:[EAX],AL	
0046C399	00C8	ADD AL,CL	
0046C39B	61	POPAD	
0046C39C	8647 74	XCHG BYTE PTR DS:[EDI+74],AL	

**LOOP#3 EXIT POINT (Redirection #3):** 0x0046C378

**ORIGINAL CODE:** JMP 46C393 (E9 16 00 00 00)



Now we've another loop, code isn't obfuscated:

0046C3C5	72 C3	JB SHORT UnPackMe.0046C38A	
0046C3C7	8B0C3A	MOV ECX,DWORD PTR DS:[EDX+EDI]	LOOP#4 START
0046C3CA	E8 14000000	CALL UnPackMe.0046C3E3	
0046C3CF	6C	INS BYTE PTR ES:[EDI],0X	I/O command
0046C3D0	35 CA3B58B1	XOR EAX,B1583BCA	
0046C3D5	96	XCHG EAX,ESI	
0046C3D6	17	POP SS	Modification of segment register
0046C3D7	04 ED	ADD AL,0ED	
0046C3D9	22B3 70E96E0F	AND DH,BYTE PTR DS:[EBX+F6EE970]	
0046C3DF	9C	PUSHFD	
0046C3E0	A5	MOVS DWORD PTR ES:[EDI],DWORD PTR DS:[E	
0046C3E1	7A 2B	JPE SHORT UnPackMe.0046C40E	
0046C3E3	0FB7DB	MOVZX EBX,BX	
0046C3E6	58	POP EAX	
0046C3E7	81E9 44C81B4E	SUB ECX,4E1BC844	
0046C3ED	8BDA	MOV EBX,EDX	
0046C3EF	81E9 2D2D7161	SUB ECX,61712D2D	
0046C3F5	68 B8A99761	PUSH 6197A9B8	
0046C3FA	81C3 935BC25D	ADD EBX,5DC25B93	
0046C400	58	POP EAX	
0046C401	81C1 6222C256	ADD ECX,56C22262	
0046C407	E8 14000000	CALL UnPackMe.0046C420	
0046C40C	FC	CLD	
0046C40D	85DA	TEST EDX,EBX	
0046C40F	0BE8	OR EBP,EAX	
0046C411	01A6 E7943D32	ADD DWORD PTR DS:[ESI+323D94E7],ESP	
0046C417	8300 39	ADD DWORD PTR DS:[EAX],39	
0046C41A	7E DF	JLE SHORT UnPackMe.0046C3FB	
0046C41C	2C F5	SUB AL,0F5	
0046C41E	8AFB	MOV BH,BL	
0046C420	5B	POP EBX	
0046C421	890C3A	MOV DWORD PTR DS:[EDX+EDI],ECX	
0046C424	68 731FAD17	PUSH 17AD1F73	
0046C429	80F0 3A	XOR AL,3A	
0046C42C	58	POP EAX	
0046C42D	57	PUSH EDI	
0046C42E	58	POP EAX	
0046C42F	83EF 01	SUB EDI,1	
0046C432	BB 78CE697F	MOV EBX,7F69CE78	
0046C437	4F	DEC EDI	
0046C438	4F	DEC EDI	
0046C439	4F	DEC EDI	
0046C43A	E9 0C000000	JMP UnPackMe.0046C44B	
0046C43F	24 8D	AND AL,8D	
0046C441	42	INC EDX	
0046C442	53	PUSH EBX	
0046C443	90	NOP	
0046C444	898E AFBC459A	MOV DWORD PTR DS:[ESI+9A45BCAF],ECX	
0046C44A	CB	RET	Far return
0046C44B	81FF D4FAFFFF	CMP EDI,-52C	
0046C451	0F85 70FFFFFF	JNZ UnPackMe.0046C3C7	LOOP#4 END
0046C457	8BC7	MOV EAX,EDI	
0046C459	E8 00000000	CALL UnPackMe.0046C45E	
0046C45E	5D	POP EBP	
0046C45F	5B	POP EBX	

it write some byte at address 0x0046C979 but don't care of it and go on:

Address	Hex dump	Disassembly
0046C979	AE	SCAS BYTE PTR ES:[EDI]
0046C97A	63E5	ARPL BP,SP
0046C97C	2B00 5E696A7E	SUB ECX,DWORD PTR DS:[7E6A695E]
0046C982	297D 15	SUB DWORD PTR SS:[EBP+15],EDI
0046C985	0C 31	OR AL,31
0046C987	0000	ADD BYTE PTR DS:[EAX],AL
0046C989	0000	ADD BYTE PTR DS:[EAX],AL
0046C98B	0000	ADD BYTE PTR DS:[EAX],AL

**LOOP#4 EXIT POINT (Redirection #4): 0x0046C457**

**ORIGINAL CODE:** MOV EAX,EDI

Step straight after some code we should reach the two classical **VirtualAlloc** API call:

0046C502	68 00100000	PUSH 1000	
0046C507	FFB5 00040000	PUSH DWORD PTR SS:[EBP+400]	
0046C50D	6A 00	PUSH 0	
0046C50F	FF95 F0030000	CALL NEAR DWORD PTR SS:[EBP+3F0]	VirtualAlloc #1
0046C515	8985 CC010000	MOV DWORD PTR SS:[EBP+1CC],EAX	9C0000
0046C518	8B9D 00040000	MOV EBX,DWORD PTR SS:[EBP+400]	
0046C521	039D 00040000	ADD EBX,DWORD PTR SS:[EBP+400]	
0046C527	50	PUSH EAX	
0046C529	53	PUSH EBX	
0046C52B	E8 04010000	CALL UnPackMe.0046C632	
0046C52E	6A 40	PUSH 40	
0046C536	68 00100000	PUSH 1000	
0046C538	FFB5 00040000	PUSH DWORD PTR SS:[EBP+400]	
0046C53B	6A 00	PUSH 0	
0046C53D	FF95 F0030000	CALL NEAR DWORD PTR SS:[EBP+3F0]	VirtualAlloc #2
0046C543	8985 31040000	MOV DWORD PTR SS:[EBP+431],EAX	A10000
0046C549	8985 D0010000	MOV DWORD PTR SS:[EBP+1D0],EAX	
0046C54F	64 67 A1 0000	MOV EAX,DWORD PTR FS:[0]	
0046C554	8985 2D040000	MOV DWORD PTR SS:[EBP+42D],EAX	
0046C55A	8B55 5B	MOV EDI,DWORD PTR SS:[EBP+5B]	
0046C55D	8B55 D0010000	MOV EAX,DWORD PTR SS:[EBP+1D0]	
0046C563	8902	MOV DWORD PTR DS:[EDI],EAX	
0046C565	8B55 08040000	MOV EAX,DWORD PTR SS:[EBP+408]	
0046C568	8942 04	MOV DWORD PTR DS:[EDI+4],EAX	
0046C56E	8D85 9F030000	LEA EAX,DWORD PTR SS:[EBP+39F]	
0046C574	8B40 55	MOV EAX,DWORD PTR DS:[EAX+55]	
0046C577	8942 08	MOV DWORD PTR DS:[EDI+8],EAX	
0046C57A	8B85 EC030000	MOV EAX,DWORD PTR SS:[EBP+3EC]	
0046C580	8942 10	MOV DWORD PTR DS:[EDI+10],EAX	
0046C583	8B85 E8030000	MOV EAX,DWORD PTR SS:[EBP+3E8]	
0046C589	8942 14	MOV DWORD PTR DS:[EDI+14],EAX	
0046C58C	8B95 CC010000	MOV EDI,DWORD PTR SS:[EBP+1CC]	
0046C592	BB F8010000	MOV EBX,1F8	
0046C597	8B7C1A 0C	MOV EDI,DWORD PTR DS:[EDI+EBX+C]	
0046C59B	0BFF	OR EDI,EDI	
0046C59D	74 1E	JE SHORT UnPackMe.0046C5BD	
0046C59F	8B4C1A 10	MOV ECX,DWORD PTR DS:[EDI+EBX+10]	
0046C5A3	0BC9	OR ECX,ECX	
0046C5A5	74 11	JE SHORT UnPackMe.0046C5B8	
0046C5A7	03BD D0010000	ADD EDI,DWORD PTR SS:[EBP+1D0]	
0046C5AD	8B741A 14	MOV ESI,DWORD PTR DS:[EDI+EBX+14]	
0046C5B1	03F2	ADD ESI,EDX	
0046C5B3	C1F9 02	SAR ECX,2	
0046C5B6	F3 A5	REP MOVS DWORD PTR ES:[EDI],DWORD PTR DS:[ESI]	Fill the last allocated area
0046C5B8	83C3 28	ADD EBX,28	
0046C5BB	EB DA	JMP SHORT UnPackMe.0046C597	
0046C5BD	8B85 CC010000	MOV EAX,DWORD PTR SS:[EBP+1CC]	
0046C5C3	50	PUSH EAX	
0046C5C4	8B95 D0010000	MOV EDX,DWORD PTR SS:[EBP+1D0]	
0046C5CA	52	PUSH EDX	
0046C5CB	8B18	MOV EBX,DWORD PTR DS:[EAX]	
0046C5CD	03DA	ADD EBX,EDX	
0046C5CF	8B85 E4030000	MOV EAX,DWORD PTR SS:[EBP+3E4]	
0046C5D5	8903	MOV DWORD PTR DS:[EBX],EAX	
0046C5D7	8B85 E8030000	MOV EAX,DWORD PTR SS:[EBP+3E8]	
0046C5DD	8943 04	MOV DWORD PTR DS:[EBX+4],EAX	
0046C5E0	8B85 EC030000	MOV EAX,DWORD PTR SS:[EBP+3EC]	
0046C5E6	8943 08	MOV DWORD PTR DS:[EBX+8],EAX	
0046C5E9	5F	POP EDI	
0046C5EA	5E	POP ESI	
0046C5EB	8B46 04	MOV EAX,DWORD PTR DS:[ESI+4]	
0046C5EE	03C7	ADD EAX,EDI	
0046C5F0	8985 C7010000	MOV DWORD PTR SS:[EBP+1C7],EAX	Set the push address
0046C5F6	8B55 5B	MOV EDI,DWORD PTR SS:[EBP+5B]	
0046C5F9	8B85 C7010000	MOV EAX,DWORD PTR SS:[EBP+1C7]	
0046C5FF	8942 0C	MOV DWORD PTR DS:[EDI+C],EAX	
0046C602	8D9D 00040000	LEA EBX,DWORD PTR SS:[EBP+40D]	
0046C608	53	PUSH EBX	
0046C609	6A 00	PUSH 0	
0046C60B	6A 00	PUSH 0	
0046C60D	6A 01	PUSH 1	
0046C60F	57	PUSH EDI	
0046C610	8B5E 08	MOV EBX,DWORD PTR DS:[ESI+8]	
0046C613	03DF	ADD EBX,EDI	
0046C615	53	PUSH EBX	
0046C616	68 00000000	PUSH 0000	Place for redirection#5 (keep the EDI address)
0046C61B	6A 00	PUSH 0	
0046C61D	56	PUSH ESI	
0046C61E	FF95 F4030000	CALL NEAR DWORD PTR SS:[EBP+3F4]	
0046C624	68 00E0A500	PUSH 0A5E000	
0046C629	C3	RETN	
0046C62A	0000	ADD BYTE PTR DS:[EAX],AL	First jump in run-time area

**ADDRESS FOR REDIRECTION #5:** 0x0046C616

**ORIGINAL CODE:** PUSH 8000 (68 00 80 00 00)

#### Note for redirection #5

Into the redirection we've also to save the base address of the newly allocated memory area, this value was stored into the EDI register.



Reach the RETN instruction and jump into the newly allocated area (the red coloured NOP instruction again is only to leave the obfuscated code):

00A5E000	90	NOP
00A5E001	60	PUSHAD
00A5E002	E8 40060000	CALL 00A5E647
00A5E007	EB 44	JMP SHORT 00A5E04D
00A5E009	0000	ADD BYTE PTR DS:[EAX],AL
00A5E00B	0000	ADD BYTE PTR DS:[EAX],AL
00A5E00D	0000	ADD BYTE PTR DS:[EAX],AL
00A5E00F	0000	ADD BYTE PTR DS:[EAX],AL
00A5E011	87DB	XCHG EBX,EBX
00A5E013	90	NOP
00A5E014	0000	ADD BYTE PTR DS:[EAX],AL
00A5E016	0000	ADD BYTE PTR DS:[EAX],AL
00A5E018	0000	ADD BYTE PTR DS:[EAX],AL
00A5E01A	0000	ADD BYTE PTR DS:[EAX],AL
00A5E01C	0000	ADD BYTE PTR DS:[EAX],AL
00A5E01E	0000	ADD BYTE PTR DS:[EAX],AL
00A5E020	0000	ADD BYTE PTR DS:[EAX],AL
00A5E022	0000	ADD BYTE PTR DS:[EAX],AL
00A5E024	0000	ADD BYTE PTR DS:[EAX],AL
00A5E026	0000	ADD BYTE PTR DS:[EAX],AL
00A5E028	0000	ADD BYTE PTR DS:[EAX],AL
00A5E02A	0000	ADD BYTE PTR DS:[EAX],AL
00A5E02C	0000	ADD BYTE PTR DS:[EAX],AL
00A5E02E	E0 04	LOOPNE SHORT 00A5E034
00A5E030	0000	ADD BYTE PTR DS:[EAX],AL
00A5E032	0000	ADD BYTE PTR DS:[EAX],AL
00A5E034	0000	ADD BYTE PTR DS:[EAX],AL
00A5E036	0000	ADD BYTE PTR DS:[EAX],AL
00A5E038	0000	ADD BYTE PTR DS:[EAX],AL
00A5E03A	0000	ADD BYTE PTR DS:[EAX],AL
00A5E03C	0000	ADD BYTE PTR DS:[EAX],AL
00A5E03E	0000	ADD BYTE PTR DS:[EAX],AL
00A5E040	0000	ADD BYTE PTR DS:[EAX],AL
00A5E042	0000	ADD BYTE PTR DS:[EAX],AL
00A5E044	0000	ADD BYTE PTR DS:[EAX],AL
00A5E046	0000	ADD BYTE PTR DS:[EAX],AL
00A5E048	0000	ADD BYTE PTR DS:[EAX],AL
00A5E04A	0000	ADD BYTE PTR DS:[EAX],AL
00A5E04C	90	NOP
00A5E04D	BB 44294400	MOV EBX,442944
00A5E052	03DD	ADD EBX,EBP
00A5E054	2B9D 71294400	SUB EBX,DWORD PTR SS:[EBP+442971]
00A5E05A	83BD 08304400	CMP DWORD PTR SS:[EBP+4430D8],0
00A5E061	899D 2F2E4400	MOV DWORD PTR SS:[EBP+442E2F],EBX
00A5E067	0F85 3E050000	JNZ 00A5E5AB
00A5E06D	8D85 E0304400	LEA EAX,DWORD PTR SS:[EBP+4430E0]
00A5E073	50	PUSH EAX
00A5E074	FF95 EC314400	CALL NEAR DWORD PTR SS:[EBP+4431EC]

After some step we've another **VirtualAlloc** API call followed by a writing loop, time for us to set a new redirection point:

00A5E067	0F85 3E050000	JNZ 00A5E5AB	
00A5E06D	8D85 E0304400	LEA EAX,DWORD PTR SS:[EBP+4430E0]	
00A5E073	50	PUSH EAX	
00A5E074	FF95 EC314400	CALL NEAR DWORD PTR SS:[EBP+4431EC]	GetModuleHandleA
00A5E07A	8985 DC304400	MOV DWORD PTR SS:[EBP+4430DC],EAX	
00A5E080	8BF8	MOV EDI,EAX	
00A5E082	8D9D ED304400	LEA EBX,DWORD PTR SS:[EBP+4430ED]	
00A5E088	53	PUSH EBX	
00A5E089	50	PUSH EAX	
00A5E08A	FF95 E8314400	CALL NEAR DWORD PTR SS:[EBP+4431E8]	GetProcAddress
00A5E090	8985 79294400	MOV DWORD PTR SS:[EBP+442979],EAX	
00A5E096	8D9D FA304400	LEA EBX,DWORD PTR SS:[EBP+4430FA]	
00A5E09C	53	PUSH EBX	
00A5E09D	57	PUSH EDI	
00A5E09E	FF95 E8314400	CALL NEAR DWORD PTR SS:[EBP+4431E8]	GetProcAddress
00A5E0A4	8985 7D294400	MOV DWORD PTR SS:[EBP+44297D],EAX	
00A5E0AA	8B85 2F2E4400	MOV EAX,DWORD PTR SS:[EBP+442E2F]	
00A5E0B6	8985 D8304400	MOV DWORD PTR SS:[EBP+4430D8],EAX	
00A5E0B6	6A 04	PUSH 4	
00A5E0B8	68 00100000	PUSH 1000	
00A5E0BD	68 46050000	PUSH 546	
00A5E0C2	6A 00	PUSH 0	
00A5E0C4	FF95 79294400	CALL NEAR DWORD PTR SS:[EBP+442979]	VirtualAlloc
00A5E0CA	8985 75294400	MOV DWORD PTR SS:[EBP+442975],EAX	EAX=009C0000
00A5E0D0	8D9D 452A4400	LEA EBX,DWORD PTR SS:[EBP+442A45]	
00A5E0D6	50	PUSH EAX	
00A5E0D7	53	PUSH EBX	
00A5E0D8	E8 74050000	CALL 00A5E651	
00A5E0DD	8BC8	MOV ECX,EAX	
00A5E0DF	8D8D 452A4400	LEA EDI,DWORD PTR SS:[EBP+442A45]	
00A5E0E5	8BB5 75294400	MOV ESI,DWORD PTR SS:[EBP+442975]	
00A5E0EB	F3:A4	REP MOVSB BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]	Writing loop
00A5E0ED	8B85 75294400	MOV EAX,DWORD PTR SS:[EBP+442975]	
00A5E0F3	68 00800000	PUSH 8000	Place for redirection #6
00A5E0F8	6A 00	PUSH 0	
00A5E0FA	50	PUSH EAX	
00A5E0FB	FF95 7D294400	CALL NEAR DWORD PTR SS:[EBP+44297D]	
00A5E101	8D85 512C4400	LEA EAX,DWORD PTR SS:[EBP+442C51]	
00A5E107	50	PUSH EAX	
00A5E108	C3	RETN	
00A5E109	0000	ADD BYTE PTR DS:[EAX],AL	
00A5E10B	0000	ADD BYTE PTR DS:[EAX],AL	
00A5E10D	0000	ADD BYTE PTR DS:[EAX],AL	
00A5E10F	40	INC EAX	

**OFFSET FOR REDIRECTION #5:** 0x004E0F3  
**ORIGINAL CODE:** PUSH 8000 (68 00 80 00 00)

Reach the RETN instruction at offset 0x004E108 (from the base address of the second VirtualAlloc API call) and go into the next code layer:

00A5E300	8B9D 552A4400	MOV EBX,DWORD PTR SS:[EBP+442A55]
00A5E313	0BD8	OR EBX,EBX
00A5E315	74 0A	JE SHORT 00A5E321
00A5E317	8B03	MOV EAX,DWORD PTR DS:[EBX]
00A5E319	8785 592A4400	XCHG DWORD PTR SS:[EBP+442A59],EAX
00A5E31F	8903	MOV DWORD PTR DS:[EBX],EAX
00A5E321	8DB5 712A4400	LEA ESI,DWORD PTR SS:[EBP+442A71]
00A5E327	833E 00	CMP DWORD PTR DS:[ESI],0
00A5E32A	0F84 D3000000	JE 00A5E403
00A5E330	8DB5 712A4400	LEA ESI,DWORD PTR SS:[EBP+442A71]

Place one hardware breakpoint on offset 0x004E30D to keep fast when you've to restart.

Now start to step again, you see another call to the **VirtualAlloc** API at offset 0x004E343, step again and you're able to reach the classical ASPACK sequence:

00A5E5B8	5B	POP EBX
00A5E5B9	0BD8	OR EBX,EBX
00A5E5BB	8985 112F4400	MOV DWORD PTR SS:[EBP+442F11],EAX
00A5E5C1	61	POPAD
00A5E5C2	75 08	JNZ SHORT 00A5E5CC
00A5E5C4	B8 01000000	MOV EAX,1
00A5E5C9	C2 0C00	RETN 0C
00A5E5CC	68 00000000	PUSH 0
00A5E5D1	C3	RETN
00A5E5D2	8B85 DC304400	MOV EAX,DWORD PTR SS:[EBP+4430DC]

press F8 one time:

00A5E5B8	5B	POP EBX
00A5E5B9	0BD8	OR EBX,EBX
00A5E5BB	8985 112F4400	MOV DWORD PTR SS:[EBP+442F11],EAX
00A5E5C1	61	POPAD
00A5E5C2	75 08	JNZ SHORT 00A5E5CC
00A5E5C4	B8 01000000	MOV EAX,1
00A5E5C9	C2 0C00	RETN 0C
00A5E5CC	68 00E5A400	PUSH 0A4E5B0
00A5E5D1	C3	RETN
00A5E5D2	8B85 DC304400	MOV EAX,DWORD PTR SS:[EBP+4430DC]
00A5E5D8	8D8D 15314400	LEA ECX,DWORD PTR SS:[EBP+443115]

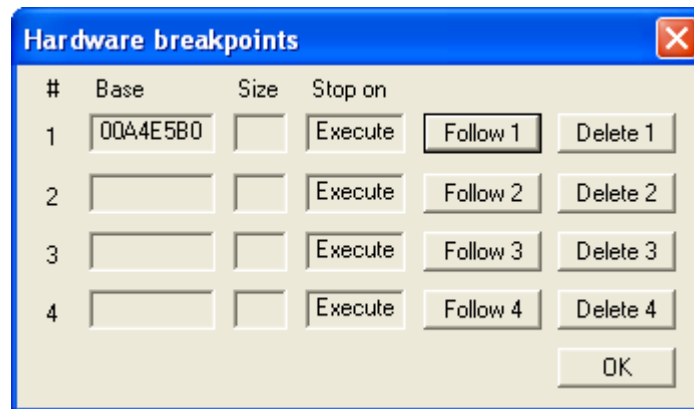
the MOV instruction write the new jump address, time for us to write down the address for the next redirection:

**OFFSET FOR REDIRECTION #6:** 0x004E5C1  
**ORIGINAL CODE:** POPAD / JNZ instructions

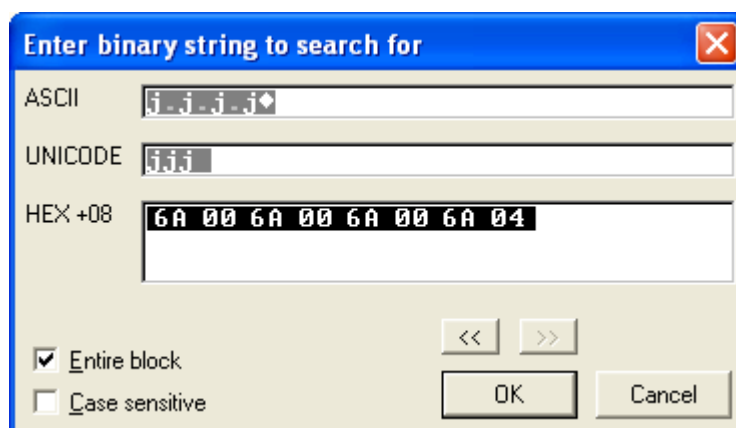
Reach the RETN instruction and we land into a new area:

00A4E5B0	55	PUSH EBP
00A4E5B1	8BEC	MOV EBP,ESP
00A4E5B3	83C4 B4	ADD ESP,-4C
00A4E5B6	B8 B8E2A400	MOV EAX,0A4E2B8
00A4E5BB	E8 C877FCFF	CALL 00A15D88
00A4E5C0	E8 FB4FFCFF	CALL 00A135C0
00A4E5C5	8D40 00	LEA EAX,DWORD PTR DS:[EAX]
00A4E5C8	0000	ADD BYTE PTR DS:[EAX],AL
00A4E5CA	0000	ADD BYTE PTR DS:[EAX],AL
00A4E5CC	0000	ADD BYTE PTR DS:[EAX],AL

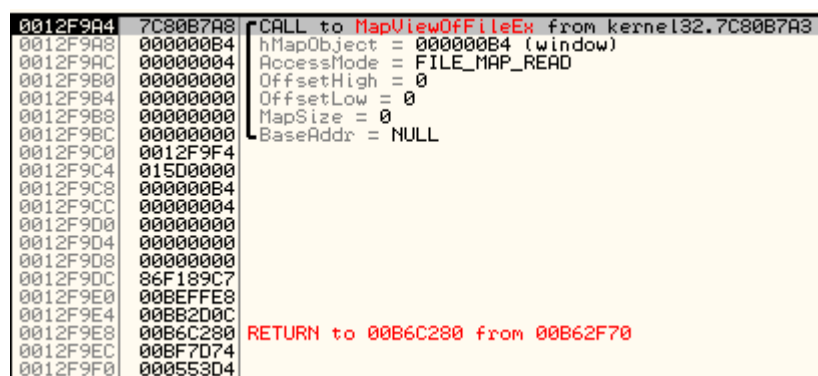
Place one hardware breakpoint on this offset 0x003E5B0 for future reference.



Now is time to search the keypoint related the **MapViewOfFile** API call (if this one exist) in order to deal about the CRC check (this check is for the file on the disk):



but we found anything then may be in this newer version this test was skipped or simply the call was totally obfuscated, well place a memory breakpoint (F2) into the **MapViewOfFileEx** entry point and press Shift+F9, you've to press Shift+F9 some time until you're able to see into the stack window some address referred our run-time area:



Press ALT+F9 to return into the caller code:

5B18B99C	33FF	XOR EDI,EDI	
5B18B99E	397E 08	CMP DWORD PTR DS:[ESI+8],EDI	
5B18B9A1	0F85 196C0100	JNZ 5B1A25C0	
5B18B9A7	8B5D 08	MOV EBX,DWORD PTR SS:[EBP+8]	
5B18B9AA	57	PUSH EDI	
5B18B9AB	57	PUSH EDI	
5B18B9AC	57	PUSH EDI	
5B18B9AD	FF75 0C	PUSH DWORD PTR SS:[EBP+C]	
5B18B9B0	53	PUSH EBX	
5B18B9B1	FF15 D410185B	CALL NEAR DWORD PTR DS:[5B1810D4]	kernel32.MapViewOfFile
5B18B9B7	3BC7	CMP EAX,EDI	
5B18B9B9	8946 08	MOV DWORD PTR DS:[ESI+8],EAX	
5B18B9BC	0F84 086C0100	JE 5B1A25CA	
5B18B9C2	57	PUSH EDI	
5B18B9C3	8BCE	MOV ECX,ESI	
5B18B9C5	895E 0C	MOV DWORD PTR DS:[ESI+C],EBX	
5B18B9C8	E8 18000000	CALL 5B18B9E5	

press again Shift+F9:

0012F9A4	7C80B7A3	CALL to MapViewOfFileEx from kernel32.7C80B7A3
0012F9A8	00000080	hMapObject = 00000080 (window)
0012F9AC	00000004	AccessMode = FILE_MAP_READ
0012F9B0	00000000	OffsetHigh = 0
0012F9B4	00000000	OffsetLow = 0
0012F9B8	00000000	MapSize = 0
0012F9BC	00000000	BaseAddr = NULL
0012F9C0	0012F9F4	

and ALT+F9, now we land into this piece of code:

014E0000	68 00004E01	PUSH 14E0000
014E0005	68 07C3A400	PUSH 0A4C307
014E000A	68 747DAD00	PUSH 0AD07D74
014E000F	E8 5C2F56FF	CALL 00A42F70
014E0014	0000	ADD BYTE PTR DS:[EAX],AL
014E0016	0000	ADD BYTE PTR DS:[EAX],AL
014E0018	0000	ADD BYTE PTR DS:[EAX],AL
014E001A	0000	ADD BYTE PTR DS:[EAX],AL

by using the single step (F7) reach the selected CALL and enter in.

00A42F70	60	PUSHAD	
00A42F71	89E0	MOV EAX,ESP	
00A42F73	9C	PUSHFD	
00A42F74	5A	POP EDX	
00A42F75	55	PUSH EBP	
00A42F76	89E5	MOV EBP,ESP	
00A42F78	83C5 24	ADD EBP,24	
00A42F7B	31C9	XOR ECX,ECX	
00A42F7D	64:8B09	MOV ECX,DWORD PTR FS:[ECX]	
00A42F80	81EC B80B0000	SUB ESP,0BB8	
00A42F86	FF75 08	PUSH DWORD PTR SS:[EBP+8]	
00A42F89	FF75 0C	PUSH DWORD PTR SS:[EBP+C]	
00A42F8C	52	PUSH EDX	
00A42F8D	51	PUSH ECX	
00A42F8E	50	PUSH EAX	
00A42F8F	FF75 04	PUSH DWORD PTR SS:[EBP+4]	
00A42F92	E8 F5FEFFFF	CALL 00A42E8C	<<--
00A42F97	81C4 DC0B0000	ADD ESP,0BDC	
00A42F9D	C2 0C00	RETN 0C	
00A42FA0	C3	RETN	

Now reach the call 00A22E8C just before the RETN 0C instruction and step inside this call:

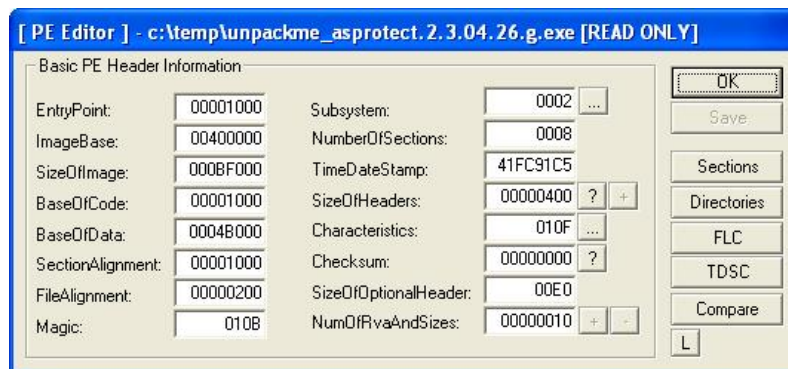
00A42E8C	55	PUSH EBP	
00A42E8D	8BEC	MOV EBP,ESP	
00A42E8E	81C4 40FFFFFF	ADD ESP,-0C0	
00A42E8F	53	PUSH EBX	
00A42E90	56	PUSH ESI	
00A42E91	57	PUSH EDI	
00A42E92	8B5D 0C	MOV EBX,DWORD PTR SS:[EBP+C]	
00A42E93	837D 18 00	CMPL DWORD PTR SS:[EBP+18],0	
00A42E94	74 08	JE SHORT 00A42EA9	
00A42EA1	8D45 18	LEA EAX,DWORD PTR SS:[EBP+18]	
00A42EA4	E8 9F6CFFFF	CALL 00A39B48	
00A42EA9	33C9	XOR ECX,ECX	
00A42EAB	55	PUSH EBP	
00A42EAC	68 272FA400	PUSH 0A42F27	
00A42EAD	64:FF31	PUSH DWORD PTR FS:[ECX]	
00A42EAE	64:8921	MOV DWORD PTR FS:[ECX],ESP	
00A42EAF	EB 01	JMP SHORT 00A42EBA	
00A42EB0	9A 8D8543FF FF	CALL FAR FFFF:FF43858D	Far call
00A42EB1	BA 8D000000	MOV EDX,0BD	
00A42EB2	E8 4232F0FF	CALL 00A1610C	
00A42EB3	8B45 14	MOV EAX,DWORD PTR SS:[EBP+14]	
00A42EB4	8945 8B	MOV DWORD PTR SS:[EBP-75],EAX	
00A42EB5	8B45 10	MOV EAX,DWORD PTR SS:[EBP+10]	
00A42EB6	8945 E0	MOV DWORD PTR SS:[EBP-20],EAX	
00A42EB7	8B45 1C	MOV EAX,DWORD PTR SS:[EBP+1C]	
00A42EB8	8945 87	MOV DWORD PTR SS:[EBP-79],EAX	
00A42EB9	EB 01	JMP SHORT 00A42EDF	
00A42EBA	E9 33D28D85	JMP 86320116	
00A42EBB	63FF	ARPL DI,DI	
00A42EBC	FFFF	???	Unknown command
00A42EBD	8D4B 20	LEA ECX,DWORD PTR DS:[EBX+20]	
00A42EBE	8BF2	MOV ESI,EDX	
00A42EBF	C1E6 02	SHL ESI,2	
00A42EC0	2BCB	SUB ECX,ESI	
00A42EC1	83E9 04	SUB ECX,4	
00A42EC2	8B09	MOV ECX,DWORD PTR DS:[ECX]	
00A42EC3	8908	MOV DWORD PTR DS:[EAX],ECX	
00A42EC4	42	INC EDX	
00A42EC5	83C0 04	ADD EAX,4	
00A42EC6	83FA 08	CMPL EDX,8	
00A42EC7	75 E6	JNZ SHORT 00A42EE7	
00A42EC8	8385 73FFFFFF	ADD DWORD PTR SS:[EBP-8D],10	
00A42EC9	EB 01	JMP SHORT 00A42F0B	
00A42ECA	9A 8D9543FF FF	CALL FAR FFFF:FF43958D	Far call
00A42ECB	8B45 08	MOV EAX,DWORD PTR SS:[EBP+8]	
00A42ECC	E8 F7FEFFFF	CALL 00A42E10	
00A42ECD	84C0	TEST AL,AL	
00A42ECE	75 EE	JNZ SHORT 00A42F0B	
00A42ECF	33C0	XOR EAX,EAX	
00A42ED0	5A	POP EDX	
00A42ED1	59	POP ECX	
00A42ED2	59	POP ECX	
00A42ED3	64:8910	MOV DWORD PTR FS:[EAX],EDX	
00A42ED4	EB 21	JMP SHORT 00A42F48	
00A42ED5	E9 3CFFFCFF	JMP 00A12E68	
00A42ED6	6A FF	PUSH -1	
00A42ED7	E8 852DFEFF	CALL 00A25CB8	
00A42ED8	8B55 08	MOV EDX,DWORD PTR SS:[EBP+8]	
00A42ED9	0142 50	ADD DWORD PTR DS:[EDX+50],EAX	
00A42EDA	68 682FA400	PUSH 0A42F68	ASCII "196J"
00A42EDB	E8 0D28FEFF	CALL 00A25750	
00A42EDC	E8 7C02F0FF	CALL 00A131C4	
00A42EDD	8D95 43FFFFFF	LEA EDX,DWORD PTR SS:[EBP-8D]	
00A42EDE	8B45 08	MOV EAX,DWORD PTR SS:[EBP+8]	
00A42EDF	E8 8AFDFFFF	CALL 00A42CE0	<<--
00A42EE0	5F	POP EDI	
00A42EE1	5E	POP ESI	
00A42EE2	5B	POP EBX	
00A42EE3	8BE5	MOV ESP,EBP	
00A42EE4	5D	POP EBP	
00A42EE5	C2 1800	RETN 18	
00A42EE6	00FF	ADD BH,BH	

step into the call which is after the string "196" and enter it.

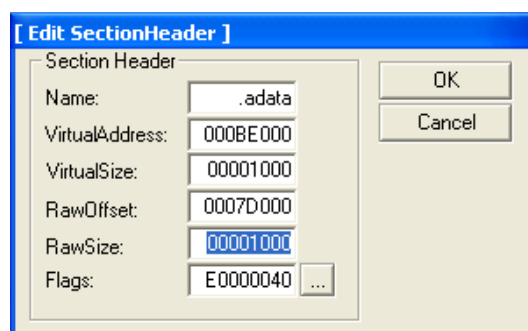
00A42CE0	55	PUSH EBP
00A42CE1	8BEC	MOV EBP,ESP
00A42CE3	83C4 F4	ADD ESP,-0C
00A42CE6	53	PUSH EBX
00A42CE7	8B42 30	MOV EAX,DWORD PTR DS:[EDX+30]
00A42CEA	83E8 20	SUB EAX,20
00A42CED	83E8 04	SUB EAX,4
00A42CF0	8945 F4	MOV DWORD PTR SS:[EBP-C],EAX
00A42CF3	33C0	XOR EAX,EAX
00A42CF5	8B4D F4	MOV ECX,DWORD PTR SS:[EBP-C]
00A42CF8	83C1 20	ADD ECX,20
00A42CFB	8BD8	MOV EBX,EAX
00A42CFD	C1E3 02	SHL EBX,2
00A42D00	2BCB	SUB ECX,EBX
00A42D02	83E9 04	SUB ECX,4
00A42D05	8B5C82 20	MOV EBX,DWORD PTR DS:[EDX+EAX*4+20]
00A42D09	8919	MOV DWORD PTR DS:[ECX],EBX
00A42D0B	40	INC EAX
00A42D0C	83F8 08	CMP EAX,8
00A42D0F	75 E4	JNZ SHORT 00A42CF5
00A42D11	8B45 F4	MOV EAX,DWORD PTR SS:[EBP-C]
00A42D14	83C0 20	ADD EAX,20
00A42D17	8B4A 44	MOV ECX,DWORD PTR DS:[EDX+44]
00A42D1A	8908	MOV DWORD PTR DS:[EAX],ECX
00A42D1C	8B42 48	MOV EAX,DWORD PTR DS:[EDX+48]
00A42D1F	8945 F8	MOV DWORD PTR SS:[EBP-8],EAX
00A42D22	8B82 9D000000	MOV EAX,DWORD PTR DS:[EDX+9D]
00A42D28	8945 FC	MOV DWORD PTR SS:[EBP-4],EAX
00A42D2B	31C0	XOR EAX,EAX
00A42D2D	8B55 FC	MOV EDX,DWORD PTR SS:[EBP-4]
00A42D30	64:8910	MOV DWORD PTR FS:[EAX],EDX
00A42D33	FF75 F8	PUSH DWORD PTR SS:[EBP-8]
00A42D36	9D	POPAD
00A42D37	8B65 F4	MOV ESP,DWORD PTR SS:[EBP-C]
00A42D3A	61	POPAD
00A42D3B	C3	RETN
00A42D3C	5B	POP EBX
00A42D3D	8BE5	MOV ESP,EBP
00A42D3F	5D	POP EBP
00A42D40	C3	RETN

we got the keypoint, this code will be executed more time after the **MapViewOfFileEx** breakpoint, and after some execution when the POPAD instruction is executed, ASProtect catch the integrity error.

Well let me explain a little more now; in order to see the integrity error we've to enlarge the last section (this is useful later when we've to write our patching code), then run LordPE and change the RAW-SIZE of the last section **.adata** from 0 to 0x1000.



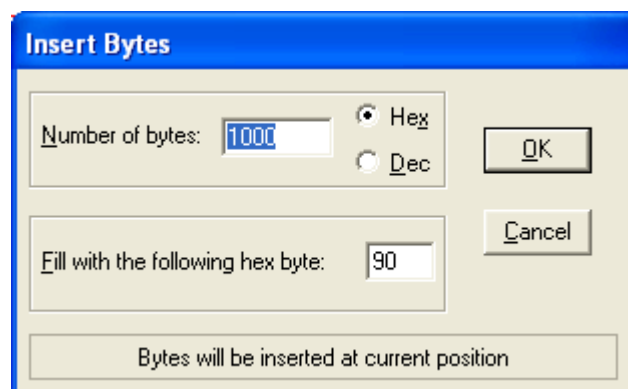
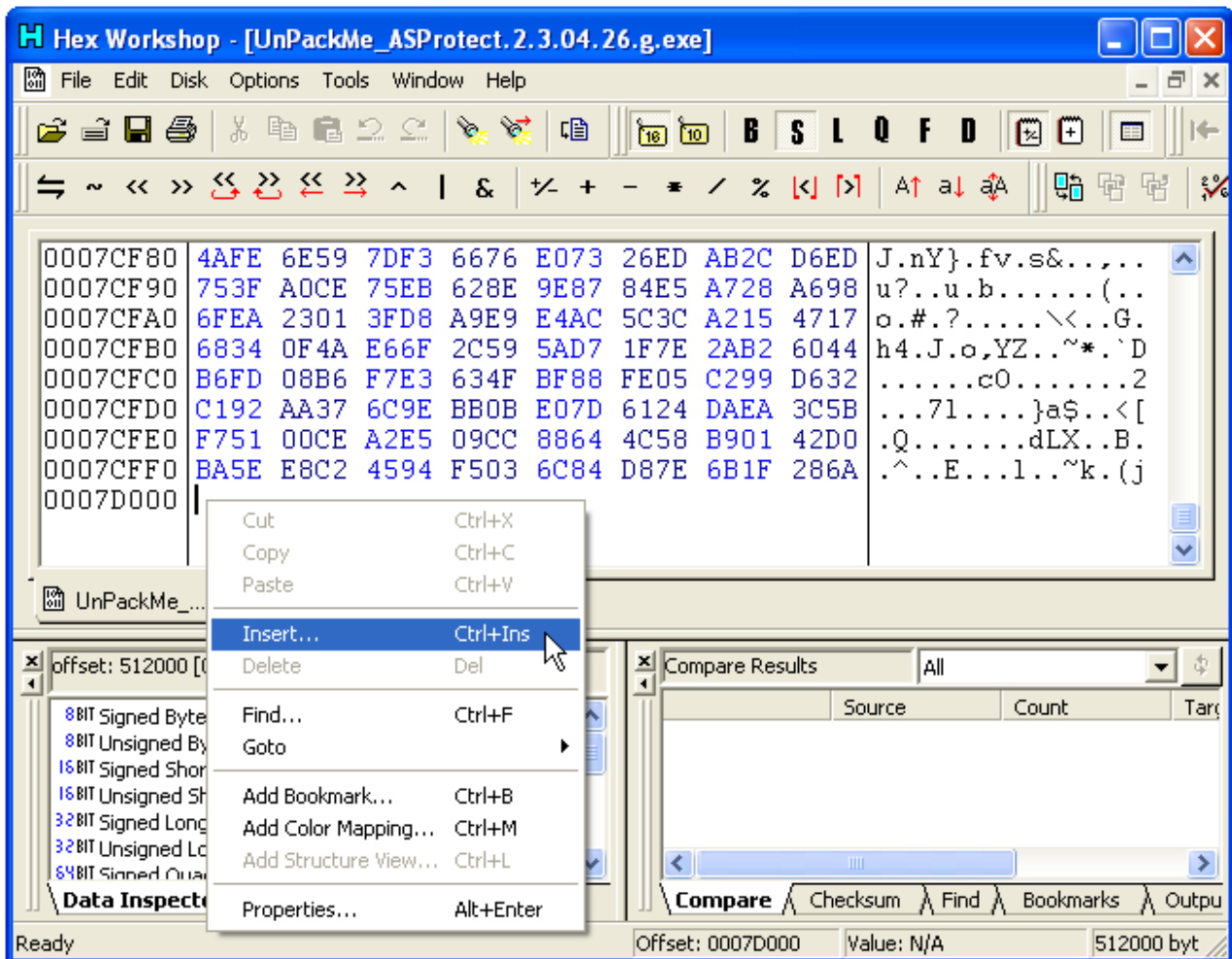
press the Sections button and select the last section **.adata** the right click and select the edit option then perform the RawSize change as show below:



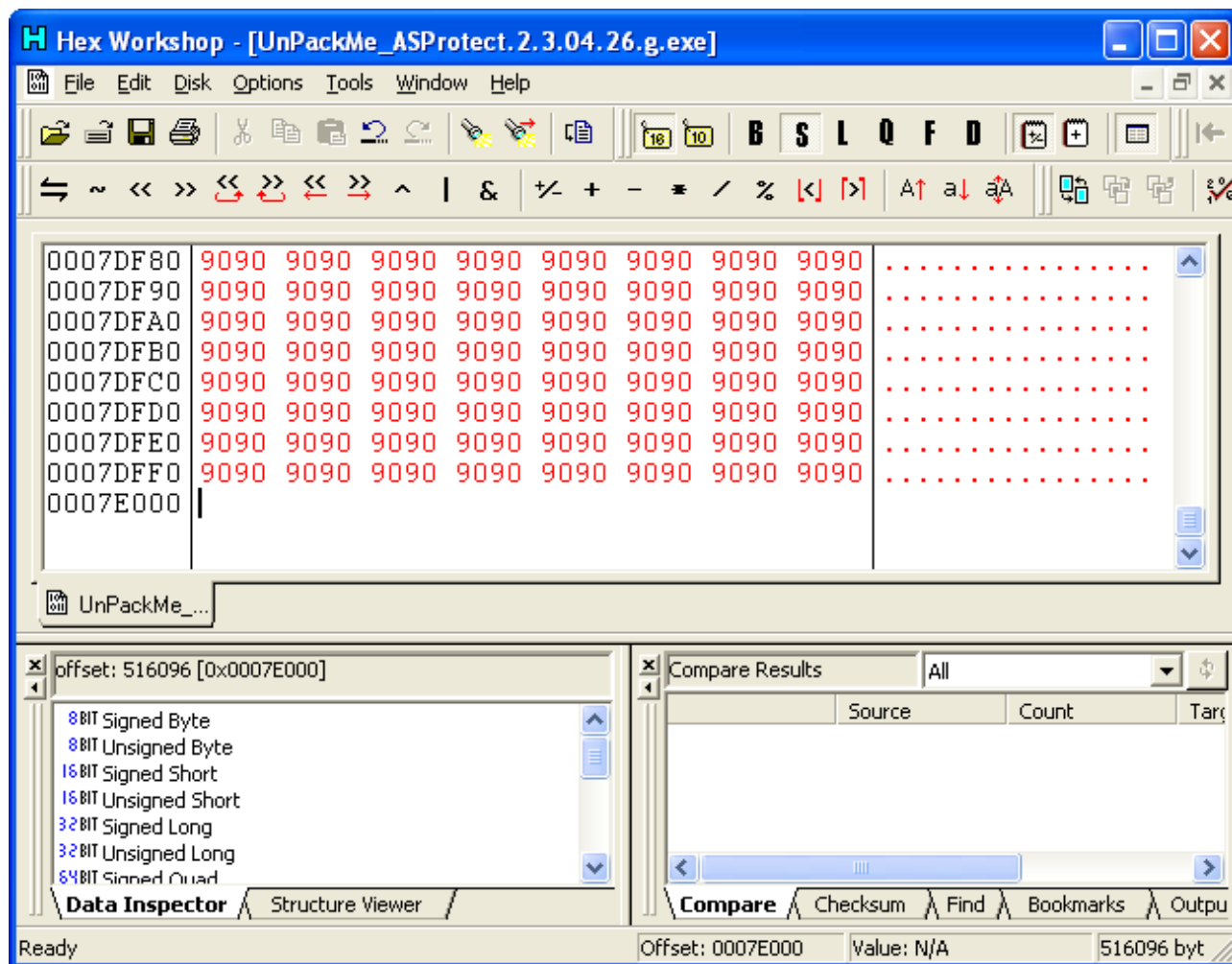
press OK and Save.



Now run Hex-Workshop, load the packed executable and go to the end of the file, now add 0x1000 byte with value set to 0x90.

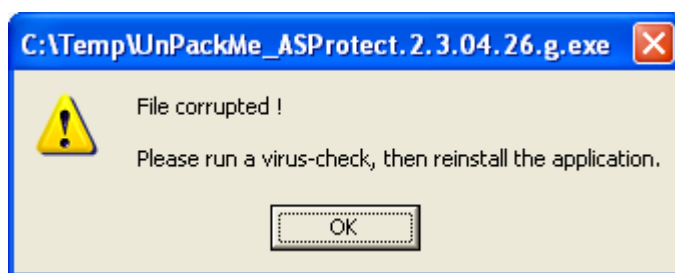


then press OK.



Save and close.

Now try run the modified packed exe outside OllyDbg:



this warning nag tell us about the integrity check and the modified executable will don't run until we make some fix related this CRC.

Well, now we have to reach again our keypoint and understand how ASProtect call the API's in order to perform the CRC check, we know what is the API involved, that is the **MapViewOfFileEx**, but at know we don't know from what point ASProtect perform this call.

Into the older ASProtect version found the call was easy because this one was called directly from the ASProtect code in a direct way (you can find easy the right point by a binary search), in this new version there is no direct call, but all will be performed through some bridge.

This idea was proof by our previous analysis, in fact after the first breakpoint into the **MapViewOfFileEx** we have some strange jump to a new section which start from address 14E0000 and from this drive the execution back to the ASProtect code.

Then in order to better know what crazy things was done by ASProtect is time for us to restart OllyDbg (CTRL+F2) and press Shift+F9. We should land into the settled hardware breakpoint:

00A4E5B0	55	PUSH EBP
00A4E5B1	8BEC	MOV EBP,ESP
00A4E5B3	83C4 B4	ADD ESP,-4C
00A4E5B6	B8 B8E2A400	MOV EAX,0A4E2B8
00A4E5B8	E8 C877FCFF	CALL 00A15D88
00A4E5C0	E8 FB4FFCFF	CALL 00A135C0
00A4E5C5	8D40 00	LEA EAX,DWORD PTR DS:[EAX]
00A4E5C8	0000	ADD BYTE PTR DS:[EAX],AL
00A4E5CA	0000	ADD BYTE PTR DS:[EAX],AL

Now we've to break into the **VirtualAlloc** API call, put a breakpoint into the RETN instruction:

7C809A81	8BFF	MOV EDI,EDI
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

Then press Shift+F9, look at the registry window and repeat with Shift+F9 until EAX keep the 14E0000 address, you've to repeat this step a lot of time, every time EAX keep a base address for some new allocated section.

Registers (FPU)	
EAX	014D0000
ECX	7C809AE9 kernel32.7C809AE9
EDX	7C91EB94 ntdll.KiFastSystemCallRet
EBX	0012EE08
ESP	0012E0DC
EBP	0012EE14
ESI	00000014
EDI	0012EE7F
EIP	7C809A9A kernel32.7C809A9A

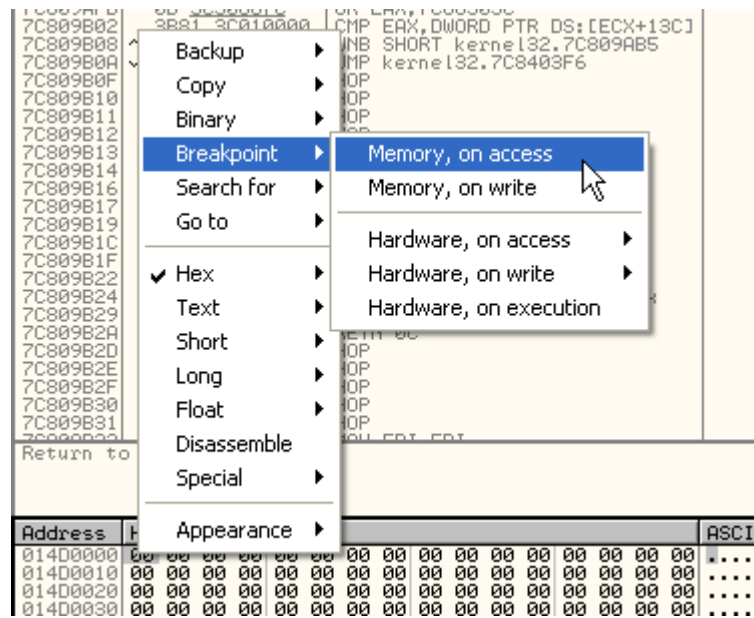
Now into the OllyDbg dump window press CTRL+G and write the address 014D0000 and put a memory breakpoint on access into the first byte, in this way we're able to see from what point into the ASProtect code this area will be written.

Then:



press OK, the new area was zeroes filled, that's right because we've simply allocated the area and no writing operation has been done.

Now put the memory breakpoint on access into the first byte.



Leave the memory breakpoint on the **VirtualAlloc** and put another memory breakpoint into the **MapViewOfFileEx** API and press Shift+F9, OllyDbg will be break on this piece of code.

00A3FB56	8BC0	MOV EAX,EAX
00A3FB58	55	PUSH EBP
00A3FB59	8BEC	MOV EBP,ESP
00A3FB5B	8B55 08	MOV EDX,DWORD PTR SS:[EBP+8]
00A3FB5E	8B52 FC	MOV EDX,DWORD PTR DS:[EDX-4]
00A3FB61	C602 68	MOV BYTE PTR DS:[EDX],68
00A3FB64	8B55 08	MOV EDX,DWORD PTR SS:[EBP+8]
00A3FB67	FF42 FC	INC DWORD PTR DS:[EDX-4]
00A3FB6A	8B55 08	MOV EDX,DWORD PTR SS:[EBP+8]
00A3FB6D	8B52 FC	MOV EDX,DWORD PTR DS:[EDX-4]
00A3FB70	8902	MOV DWORD PTR DS:[EDX],EAX
00A3FB72	8B45 08	MOV EAX,DWORD PTR SS:[EBP+8]
00A3FB75	8340 FC 04	ADD DWORD PTR DS:[EAX-4],4
00A3FB79	5D	POP EBP
00A3FB7A	C3	RETN

change the dump windows view mode in order to keep the code disassemble and reach the RETN instruction (press CTRL+F9), the first PUSH command will be write:

00A3FB56	8BC0	MOV EAX,EAX
00A3FB58	55	PUSH EBP
00A3FB59	8BEC	MOV EBP,ESP
00A3FB5B	8B55 08	MOV EDX,DWORD PTR SS:[EBP+8]
00A3FB5E	8B52 FC	MOV EDX,DWORD PTR DS:[EDX-4]
00A3FB61	C602 68	MOV BYTE PTR DS:[EDX],68
00A3FB64	8B55 08	MOV EDX,DWORD PTR SS:[EBP+8]
00A3FB67	FF42 FC	INC DWORD PTR DS:[EDX-4]
00A3FB6A	8B55 08	MOV EDX,DWORD PTR SS:[EBP+8]
00A3FB6D	8B52 FC	MOV EDX,DWORD PTR DS:[EDX-4]
00A3FB70	8902	MOV DWORD PTR DS:[EDX],EAX
00A3FB72	8B45 08	MOV EAX,DWORD PTR SS:[EBP+8]
00A3FB75	8340 FC 04	ADD DWORD PTR DS:[EAX-4],4
00A3FB79	5D	POP EBP
00A3FB7A	C3	RETN
00A3FB7B	90	NOP
00A3FB7C	55	PUSH EBP
Return to 00A3FBA7		
Address	Hex dump	Disassembly
01400000	68 00004D01	PUSH 1400000
01400005	0000	ADD BYTE PTR DS:[EAX],AL
01400007	0000	ADD BYTE PTR DS:[EAX],AL
01400009	0000	ADD BYTE PTR DS:[EAX],AL
0140000B	0000	ADD BYTE PTR DS:[EAX],AL
0140000D	0000	ADD BYTE PTR DS:[EAX],AL
0140000F	0000	ADD BYTE PTR DS:[EAX],AL
01400011	0000	ADD BYTE PTR DS:[EAX],AL

press F7 to execute the RETN instruction.

This nice piece of code write the three PUSH instruction and the final call instruction:

00A3FB7B	90	NOP	
00A3FB7C	55	PUSH EBP	Entry
00A3FB7D	8BEC	MOV EBP,ESP	
00A3FB7F	83C4 F4	ADD ESP,-0C	
00A3FB82	53	PUSH EBX	
00A3FB83	56	PUSH ESI	
00A3FB84	57	PUSH EDI	
00A3FB85	8BF1	MOV ESI,ECX	
00A3FB87	8955 F8	MOV DWORD PTR SS:[EBP-8],EDX	
00A3FB8A	8BD8	MOV EBX,EAX	
00A3FB8C	8BFE	MOV EDI,ESI	
00A3FB8E	8D45 F4	LEA EAX,DWORD PTR SS:[EBP-C]	
00A3FB91	BA 14000000	MOV EDX,14	
00A3FB96	E8 8D9FFFFF	CALL 00A39B28	
00A3FB98	8B45 F4	MOV EAX,DWORD PTR SS:[EBP-C]	
00A3FB9E	8945 FC	MOV DWORD PTR SS:[EBP-4],EAX	
00A3FBA1	55	PUSH EBP	
00A3FBA2	E8 B1FFFFFF	CALL 00A3FB58	Write the PUSH instruction #1
00A3FBA7	59	POP ECX	0012EE14
00A3FBA8	55	PUSH EBP	
00A3FBA9	8B47 44	MOV EAX,DWORD PTR DS:[EDI+44]	
00A3FBAC	E8 A7FFFFFF	CALL 00A3FB58	Write the PUSH instruction #2
00A3FBB1	59	POP ECX	
00A3FBB2	55	PUSH EBP	
00A3FBB3	8BC3	MOV EAX,EBX	
00A3FBB5	E8 9EFFFFFF	CALL 00A3FB58	Write the PUSH instruction #3
00A3FBB8	59	POP ECX	
00A3FBBB	8B45 FC	MOV EAX,DWORD PTR SS:[EBP-4]	
00A3FBBD	C600 E8	MOV BYTE PTR DS:[EAX],0E8	Write the call instruction
00A3FBC1	B8 702FA400	MOV EAX,0A42F70	
00A3FBC6	2B45 FC	SUB EAX,DWORD PTR SS:[EBP-4]	
00A3FBC9	83E8 05	SUB EAX,5	
00A3FBCC	8B55 FC	MOV EDX,DWORD PTR SS:[EBP-4]	
00A3FBCF	42	INC EDX	
00A3FBD0	8902	MOV DWORD PTR DS:[EDX],EAX	
00A3FBD2	8345 FC 05	ADD DWORD PTR SS:[EBP-4],5	
00A3FBD6	56	PUSH ESI	
00A3FBD7	B1 04	MOV CL,4	
00A3FBD9	8B55 F4	MOV EDX,DWORD PTR SS:[EBP-C]	
00A3FBD0C	8BC3	MOV EAX,EBX	
00A3FBDE	E8 95E8FFFF	CALL 00A3E478	
00A3FBE3	8B45 F8	MOV EAX,DWORD PTR SS:[EBP-8]	
00A3FBE6	8947 44	MOV DWORD PTR DS:[EDI+44],EAX	
00A3FBE9	8BD6	MOV EDX,ESI	
00A3FBE8	8BC3	MOV EAX,EBX	
00A3FBED	E8 EE300000	CALL 00A42CE0	Go into the writed section
00A3FBF2	5F	POP EDI	
00A3FBF3	5E	POP ESI	
00A3FBF4	5B	POP EBX	
00A3FBF5	8BE5	MOV ESP,EBP	
00A3FBF7	5D	POP EBP	
00A3FBF8	C3	RETN	
00A3FBF9	8D40 00	LEA EAX,DWORD PTR DS:[EAX]	

reach the last call before the RETN istruction, and step into by using F7.

00A42CE0	55	PUSH EBP	
00A42CE1	8BEC	MOV EBP,ESP	
00A42CE3	83C4 F4	ADD ESP,-0C	
00A42CE6	53	PUSH EBX	
00A42CE7	8B42 30	MOV EAX,DWORD PTR DS:[EDX+30]	
00A42CEA	83E8 20	SUB EAX,20	
00A42CED	83E8 04	SUB EAX,4	
00A42CF0	8945 F4	MOV DWORD PTR SS:[EBP-C],EAX	
00A42CF3	33C0	XOR EAX,EAX	
00A42CF5	8B4D F4	MOV ECX,DWORD PTR SS:[EBP-C]	
00A42CF8	83C1 20	ADD ECX,20	
00A42CFB	8BD8	MOV EBX,EAX	
00A42CFD	C1E3 02	SHL EBX,2	
00A42D00	2BCB	SUB ECX,EBX	
00A42D02	83E9 04	SUB ECX,4	
00A42D05	8B5C82 20	MOV EBX,DWORD PTR DS:[EDX+EAX*4+20]	
00A42D09	8919	MOV DWORD PTR DS:[ECX],EBX	
00A42D0B	40	INC EAX	
00A42D0C	83F8 08	CMP EAX,8	
00A42D0F	75 E4	JNZ SHORT 00A42CF5	
00A42D11	8B45 F4	MOV EAX,DWORD PTR SS:[EBP-C]	
00A42D14	83C0 20	ADD EAX,20	
00A42D17	8B4A 44	MOV ECX,DWORD PTR DS:[EDX+44]	
00A42D1A	8908	MOV DWORD PTR DS:[EAX],ECX	
00A42D1C	8B42 48	MOV EAX,DWORD PTR DS:[EDX+48]	
00A42D1F	8945 F8	MOV DWORD PTR SS:[EBP-8],EAX	
00A42D22	8B82 90000000	MOV EAX,DWORD PTR DS:[EDX+90]	
00A42D28	8945 FC	MOV DWORD PTR SS:[EBP-4],EAX	
00A42D2B	31C0	XOR EAX,EAX	
00A42D2D	8B55 FC	MOV EDX,DWORD PTR SS:[EBP-4]	
00A42D30	64:8910	MOV DWORD PTR FS:[EAX],EDX	
00A42D33	FF75 F8	PUSH DWORD PTR SS:[EBP-8]	
00A42D36	9D	POPAD	
00A42D37	8B65 F4	MOV ESP,DWORD PTR SS:[EBP-C]	
00A42D3A	61	POPAD	
00A42D3B	C3	RETN	
00A42D3C	5B	POP EBX	
00A42D3D	8BE5	MOV ESP,EBP	
00A42D3F	5D	POP EBP	
00A42D40	C3	RETN	
00A42D41	8D40 00	LEA EAX,DWORD PTR DS:[EAX]	

This last piece of code is the same which we've see during our first analysis, when we've try to stop into the **MapViewOfFileEx** for the first time, before to enlarge the executable last sections.

Press CTR+F9 and make some stepping, you should be able to reach the area at the address 014D0000 then step into the call address with F7, reach the 0x00A42F70 call and following the step brig us to review the call at 0x00A42E8C, step a little and you're again into the 0x00A42CE0 call.

Now the execution take from another place on the address 0x014E0000 and we've to trace again, at some time you return into the previous call and the API is charged into the stack, perfect, see the code below:

00A3FB7B	90	NOP	
00A3FB7C	55	PUSH EBP	Entry
00A3FB7D	8BEC	MOV EBP,ESP	
00A3FB7F	83C4 F4	ADD ESP,-0C	
00A3FB82	53	PUSH EBX	
00A3FB83	56	PUSH ESI	
00A3FB84	57	PUSH EDI	
00A3FB85	8BF1	MOV ESI,ECX	
00A3FB87	8955 F8	MOV DWORD PTR SS:[EBP-8],EDX	
00A3FB8A	8BD8	MOV EBX,EAX	
00A3FB8C	8BFE	MOV EDI,ESI	
00A3FB8E	8D45 F4	LEA EAX,DWORD PTR SS:[EBP-C]	
00A3FB91	BA 14000000	MOV EDX,14	
00A3FB96	E8 8D9FFFFF	CALL 00A39B28	
00A3FB9B	8B45 F4	MOV EAX,DWORD PTR SS:[EBP-C]	
00A3FB9E	8945 FC	MOV DWORD PTR SS:[EBP-4],EAX	
00A3FBA1	55	PUSH EBP	
00A3FBA2	E8 B1FFFFFF	CALL 00A3FB58	Write the PUSH instruction #1
00A3FBA7	59	POP ECX	
00A3FBA8	55	PUSH EBP	
00A3FBA9	8B47 44	MOV EAX,DWORD PTR DS:[EDI+44]	
00A3FBAC	E8 A7FFFFFF	CALL 00A3FB58	Write the PUSH instruction #2
00A3FBB1	59	POP ECX	
00A3FBB2	55	PUSH EBP	
00A3FBB3	8BC3	MOV EAX,EBX	
00A3FBB5	E8 9EFFFFFF	CALL 00A3FB58	Write the PUSH instruction #3
00A3FBB8	59	POP ECX	
00A3FBBB	8B45 FC	MOV EAX,DWORD PTR SS:[EBP-4]	
00A3FBBE	C600 E8	MOV BYTE PTR DS:[EAX],0E8	Write the call instruction
00A3FBC1	B8 702FA400	MOV EAX,0A42F70	
00A3FBC6	2B45 FC	SUB EAX,DWORD PTR SS:[EBP-4]	
00A3FBC9	83E8 05	SUB EAX,5	
00A3FBCC	8B55 FC	MOV EDX,DWORD PTR SS:[EBP-4]	
00A3FBCF	42	INC EDX	
00A3FBD0	8902	MOV DWORD PTR DS:[EDX],EAX	
00A3FBD2	8345 FC 05	ADD DWORD PTR SS:[EBP-4],5	
00A3FBD6	56	PUSH ESI	
00A3FBD7	B1 04	MOV CL,4	
00A3FBD9	8B55 F4	MOV EDX,DWORD PTR SS:[EBP-C]	
00A3FBDC	8BC3	MOV EAX,EBX	
00A3FBDE	E8 95E8FFFF	CALL 00A3E478	
00A3FBE3	8B45 F8	MOV EAX,DWORD PTR SS:[EBP-8]	
00A3FBE6	8947 44	MOV DWORD PTR DS:[EDI+44],EAX	EAX = API address
00A3FBE9	8BD6	MOV EDX,ESI	
00A3FBEB	8BC3	MOV EAX,EBX	
00A3FBED	E8 EE300000	CALL 00A42CE0	Go into the writed section
00A3FBF2	5F	POP EDI	
00A3FBF3	5E	POP ESI	
00A3FBF4	5B	POP EBX	
00A3FBF5	8BE5	MOV ESP,EBP	
00A3FBF7	5D	POP EBP	
00A3FBF8	C3	RETN	

Look into the EAX register you'll found the API address or simple code address that will be called at the subsequent POPAD / RETN instructions (which is inside the call 00A42CE0).



00A42CE0	55	PUSH EBP
00A42CE1	8BEC	MOV EBP,ESP
00A42CE3	83C4 F4	ADD ESP,-0C
00A42CE6	53	PUSH EBX
00A42CE7	8B42 30	MOV EAX,DWORD PTR DS:[EDX+30]
00A42CEA	83E8 20	SUB EAX,20
00A42CED	83E8 04	SUB EAX,4
00A42CF0	8945 F4	MOV DWORD PTR [EBP-C],EAX
00A42CF3	33C0	XOR EAX,EAX
00A42CF5	8B40 F4	MOV ECX,DWORD PTR SS:[EBP-C]
00A42CF8	83C1 20	ADD ECX,20
00A42CFB	8B08	MOV EBX,EAX
00A42CFD	C1E3 02	SHL EBX,2
00A42D00	2BC8	SUB ECX,EBX
00A42D02	83E9 04	SUB ECX,4
00A42D05	8B5C82 20	MOV EBX,DWORD PTR DS:[EDX+EAX*4+20]
00A42D09	8919	MOV DWORD PTR DS:[ECX],EBX
00A42D0B	40	INC EAX
00A42D0C	83F8 08	CMP EAX,8
00A42D0F	75 E4	JNZ SHORT 00A42CF5
00A42D11	8B45 F4	MOV EAX,DWORD PTR SS:[EBP-C]
00A42D14	83C0 20	ADD EAX,20
00A42D17	8B4A 44	MOV ECX,DWORD PTR DS:[EDX+44]
00A42D1A	8908	MOV DWORD PTR DS:[EAX],ECX
00A42D1C	8B42 48	MOV EAX,DWORD PTR DS:[EDX+48]
00A42D1F	8945 F8	MOV DWORD PTR SS:[EBP-8],EAX
00A42D22	8B82 90000000	MOV EAX,DWORD PTR DS:[EDX+90]
00A42D28	8945 FC	MOV DWORD PTR SS:[EBP-4],EAX
00A42D2B	31C0	XOR EAX,EAX
00A42D2D	8B55 FC	MOV EDX,DWORD PTR SS:[EBP-4]
00A42D30	64:8910	MOV DWORD PTR FS:[EAX],EDX
00A42D33	FF75 F8	PUSH DWORD PTR SS:[EBP-8]
00A42D36	9D	POPPD
00A42D37	8B65 F4	MOV ESP,DWORD PTR SS:[EBP-C]
00A42D38	61	POPAD
00A42D3B	C3	RETN
00A42D3C	5B	POP EBX
00A42D3D	8BE5	MOV ESP,EBP

0012F998	00A15F38	JMP to kernel32.GetModuleHandleA
0012F99C	014E0000	
0012F9A0	00A3470C	ASCII "kernel32.dll"
0012F9A4	86F189C7	

then we know what way ASProtect keep to perform the jump into the code and API. Referring only to the API sequence I'll report below what sequence I've found.

GetModuleHandleA  
 GetCurrentProcess  
 GetProcessAffinityMask  
 GetModuleFileNameA

After the **GetModuleFileNameA** execution when you're are going back into the POPAD / RETN instruction look at the EAX register, this tell us about the target executable full path:

00A42D2D	8B55 FC	MOV EDX,DWORD PTR SS:[EBP-4]
00A42D30	64:8910	MOV DWORD PTR FS:[EAX],EDX
00A42D33	FF75 F8	PUSH DWORD PTR SS:[EBP-8]
00A42D36	9D	POPPD
00A42D37	8B65 F4	MOV ESP,DWORD PTR SS:[EBP-C]
00A42D38	61	POPAD
00A42D3B	C3	RETN
00A42D3C	5B	POP EBX
00A42D3D	8BE5	MOV ESP,EBP
00A42D3F	5D	POP EBP
00A42D40	C3	RETN
00A42D41	8B42 30	MOV EAX,DWORD PTR DS:[EDX+30]

Registers (FPU)	
EAX	0012FA10 ASCII "C:\Temp\UnPackMe_ASProtect.2.3.04.26.g.exe"
ECX	014C0000
EDX	000553D4
EBX	00A92D0C
ESP	0012F9F4
EBP	0012FB44
ESI	00ACFFE8
EDI	86F189C7
EIP	00A42D3B

Now if you press Shift+F9 the next API will be the:

CreateFileA

With this stack structure:

0012F9B8	7C801A24	RETURN to kernel32.CreateFileA
0012F9BC	014E0000	
0012F9C0	0012FA10	ASCII "C:\Temp\UnPackMe_ASProtect.2.3.04.26.g.exe"
0012F9C4	80000000	
0012F9C8	00000001	
0012F9CC	00000000	
0012F9D0	00000003	
0012F9D4	08000000	
0012F9D8	00000000	
0012F9DC	86F189C7	

From MSDN we have:

### CreateFile / CreateFileA info

#### CreateFile

The **CreateFile** function creates or opens a file, file stream, directory, physical disk, volume, console buffer, tape drive, communications resource, mailslot, or named pipe. The function returns a handle that can be used to access an object.

**Windows Me/98/95:** The file system restricts **CreateFile** to creating or opening files, and opening COM ports. You cannot create or open the objects that are identified in the first paragraph of this topic.

```
HANDLE CreateFile(
    LPCTSTR lpFileName,
    DWORD dwDesiredAccess,
    DWORD dwShareMode,
    LPSECURITY_ATTRIBUTES lpSecurityAttributes,
    DWORD dwCreationDisposition,
    DWORD dwFlagsAndAttributes,
    HANDLE hTemplateFile
);
```

if the function succeed the returned value was the handle to access the object the ASProtect use the **CreateFileA** in order to keep a valid handle to access further the file on the disk and perform the check.

Well let's go straight, execute the RETN instruction.

0012F9BC	014E0000	CALL to CreateFileA
0012F9C0	0012FA1D	FileName = "C:\Temp\UnPackMe_ASProtect.2.3.04.26.g.exe"
0012F9C4	80000000	Access = GENERIC_READ
0012F9C8	00000001	ShareMode = FILE_SHARE_READ
0012F9CC	00000000	pSecurity = NULL
0012F9D0	00000003	Mode = OPEN_EXISTING
0012F9D4	00000000	Attributes = SEQUENTIAL_SCAN
0012F9D8	00000000	hTemplateFile = NULL
0012F9DC	86F189C7	

press CTRL+F9, now EAX keep your file handle and you're back into the bridge code:

014E0000	68 00004E01	PUSH 14E0000
014E0005	68 A2C2A400	PUSH 0A4C2A2
014E000A	68 747DAD00	PUSH 0AD7D74
014E000F	E8 5C2F56FF	CALL 00A42F70
014E0014	0000	ADD BYTE PTR DS:[EAX],AL
014E0016	0000	ADD BYTE PTR DS:[EAX],AL

Press again Shift+F9 and when OllyDbg break you've the API name in EAX now is the time of **CreateFileMappingA** API:

00A3FBE5	8947 44	MOV DWORD PTR DS:[EDI+44],EAX	EAX = API address	Registers (FPU) EAX 7C80946C kernel32.CreateFileMappingA ECX 014E0000 EDX 0012F9C3 EBX 00AD7D74 ESP 0012EC00 EBP 0012EC98 ESI 0012ED13 EDI 0012ED13 EIP 00A3FBE6
00A3FBE9	8B06	MOV EDX,ESI		
00A3FBEB	8BC3	MOV EAX,EBX		
00A3FBED	E8 EE300000	CALL 00A42CE0	Go into the writed section	
00A3FBF2	5F	POP EDI		
00A3FBF3	5E	POP ESI		
00A3FBF4	5B	POP EBX		
00A3FBF5	8BE5	MOV ESP,EBP		
00A3FBF7	5D	POP EBP		
00A3FBF8	C3	RETN		
00A3FBF9	8D40 00	LEA EAX,DWORD PTR DS:[EAX]		

execute the RETN instruction after the PUSHAD and we're into the API entry point:

0012F9C0	014E0000	CALL to CreateFileMappingA
0012F9C4	0000007C	hFile = 0000007C (window)
0012F9C8	00000000	pSecurity = NULL
0012F9CC	00000002	Protection = PAGE_READONLY
0012F9D0	00000000	MaximumSizeHigh = 0
0012F9D4	00000000	MaximumSizeLow = 0
0012F9D8	00000000	MapName = NULL
0012F9DC	86F189C7	

Well also in this case MSDN will help us (this is just for info not yet useful for our task):

## CreateFileMapping info

### CreateFileMapping

Creates or opens a named or unnamed file mapping object for a specified file.

To specify the NUMA node for the physical memory, see [CreateFileMappingNuma](#).

```
HANDLE CreateFileMapping(
    HANDLE hFile,
    LPSECURITY_ATTRIBUTES lpAttributes,
    DWORD flProtect,
    DWORD dwMaximumSizeHigh,
    DWORD dwMaximumSizeLow,
    LPCTSTR lpName
);
```

related the protection file flag we've:

#### *flProtect*

[in] The protection for the file view, when the file is mapped.

This parameter can be one of the following values.

Value	Meaning
PAGE_READONLY	Gives read-only access to a specific region of pages. An attempt to write to a specific region results in an access violation. The file that the <i>hFile</i> parameter specifies must be created with the GENERIC_READ access right.
PAGE_READWRITE	Gives read/write access to a specific region of pages. The file that <i>hFile</i> specifies must be created with the GENERIC_READ and GENERIC_WRITE access rights.
PAGE_WRITECOPY	Gives copy-on-write access to a specific region of pages. The files that the <i>hFile</i> parameter specifies must be created with the GENERIC_READ access right.
PAGE_EXECUTE_READ	Gives read and execute access to a specific region of pages. The file specified by <i>hFile</i> must be created with the GENERIC_READ and GENERIC_EXECUTE access rights. <b>Windows Server 2003 and Windows XP:</b> This feature is not available until Windows XP SP2 and Windows Server 2003 SP1.
PAGE_EXECUTE_READWRITE	Gives read, write, and execute access to a specific region of pages. The file that <i>hFile</i> specifies must be created with the GENERIC_READ, GENERIC_WRITE, and GENERIC_EXECUTE access rights. <b>Windows Server 2003 and Windows XP:</b> This feature is not available until Windows XP SP2 and Windows Server 2003 SP1.

With this value:

PAGE_NOACCESS	0x01
PAGE_READONLY	0x02
PAGE_READWRITE	0x04
PAGE_WRITECOPY	0x08
PAGE_EXECUTE_READ	0x20
PAGE_EXECUTE_READWRITE	0x40

This API open a file mapping object that we can use to perform a file mapping by using the **MapViewOfFile** API call, to proof this sentence simply press again Shift+F9, we've a first:

CloseHandle

Again press Shift+F9 and finally we're able to reach our API:

00A3FBE6	8947 44	MOV DWORD PTR DS:[EDI+44],EAX	EAX = API address	Registers (FPU) EAX 7C80B780 kernel32.MapViewOfFile ECX 014E0000 EDX 0012F9C7 EBX 00A07D74 ESP 0012EC98 EBP 0012EC98 ESI 0012ED13 EDI 0012ED13 EIP 00A3FBE6
00A3FBE9	8BD6	MOV EDX,ESI		
00A3FBE8	8BC3	MOV EAX,EBX		
00A3FBE0	E8 EE300000	CALL 00A42CE0	Go into the writed section	
00A3FBF2	5F	POP EDI		
00A3FBF3	5E	POP ESI		
00A3FBF4	5B	POP EBX		
00A3FBF5	8BE5	MOV ESP,EBP		
00A3FBF7	5D	POP EBP		
00A3FBF8	C3	RETN		
00A3FBF9	8040 00	LEA EAX,DWORD PTR DS:[EAX]		

again Shift+F9 to reach the POPAD / RETN instructions:

00A42D28	8945 FC	MOV DWORD PTR SS:[EBP-4],EAX	Registers (FPU) EAX 7C80B780 kernel32.MapViewOfFile ECX 0012F9C4 EDX 7C91EB94 ntdll.KiFastSystemCallRet EBX 0000007C ESP 0012F9C0 EBP 0012F9F4 ESI FFFFFFFF EDI 014C0000 EIP 00A42D3B
00A42D2B	31C0	XOR EAX,EAX	
00A42D2D	8B55 FC	MOV EDX,DWORD PTR SS:[EBP-4]	
00A42D30	64:8910	MOV DWORD PTR FS:[EAX],EDX	
00A42D33	FF75 F8	PUSH DWORD PTR SS:[EBP-8]	
00A42D36	9D	POPF	
00A42D37	8B65 F4	MOV ESP,DWORD PTR SS:[EBP-C]	
00A42D3A	61	POPAD	
00A42D3E	C3	RETN	
00A42D3C	5B	POP EBX	
00A42D3D	8BE5	MOV ESP,EBP	

0012F9C0	7C80B780	kernel32.MapViewOfFile
0012F9C4	014E0000	
0012F9C8	00000080	
0012F9CC	00000004	
0012F9D0	00000000	
0012F9D4	00000000	
0012F9D8	00000000	
0012F9DC	86F189C7	
0012F9E0	00ACFFE8	

or to better understanding:

\$ ==>	7C80B780	kernel32.MapViewOfFile
\$+4	014E0000	
\$+8	00000080	
\$+C	00000004	
\$+10	00000000	
\$+14	00000000	
\$+18	00000000	
\$+1C	86F189C7	

press once F7 to reach the API entry point:

7C80B780	8BFF	MOV EDI,EDI
7C80B78F	55	PUSH EBP
7C80B790	8BEC	MOV EBP,ESP
7C80B792	6A 00	PUSH 0
7C80B794	FF75 18	PUSH DWORD PTR SS:[EBP+18]
7C80B797	FF75 14	PUSH DWORD PTR SS:[EBP+14]
7C80B79A	FF75 10	PUSH DWORD PTR SS:[EBP+10]
7C80B79D	FF75 0C	PUSH DWORD PTR SS:[EBP+C]
7C80B7A0	FF75 08	PUSH DWORD PTR SS:[EBP+8]
7C80B7A3	E8 76FFFFFF	CALL kernel32.MapViewOfFileEx
7C80B7A8	5D	POP EBP
7C80B7A9	C2 1400	RETN 14
7C80B7AC	81FE 00000040	CMP ESI,40000000

\$+4	014E0000	CALL to MapViewOfFile
\$+8	00000080	hMapObject = 00000080 (window)
\$+C	00000004	AccessMode = FILE_MAP_READ
\$+10	00000000	OffsetHigh = 0
\$+14	00000000	OffsetLow = 0
\$+18	00000000	MapSize = 0
\$+1C	86F189C7	

ok from this we have the first difference from the previous version, now with the new ASProtect release we can simply break into the MapViewOfFile entry point, no need for break in **MapViewOfFileEx** because this time the API will be directly called.

Well until this point the API's sequence was:

GetModuleHandleA  
GetCurrentProcess  
GetProcessAffinityMask  
GetModuleFileNameA  
CreateFileA  
CreateFileMappingA  
CloseHandle  
MapViewOfFile

Press CTRL+F9 to reach the **MapViewOfFile** API RETN instruction, EAX keep the file image base address.

7C80B785	C2 1800	RETN 18			
7C80B788	90	NOP			
7C80B789	90	NOP			
7C80B78A	90	NOP			
7C80B78B	90	NOP			
7C80B78C	90	NOP			
7C80B78D	8BFF	MOV EDI,EDI			
7C80B78F	55	PUSH EBP			
7C80B790	8BEC	MOV EBP,ESP			
7C80B792	6A 00	PUSH 0			
7C80B794	FF75 18	PUSH DWORD PTR SS:[EBP+18]			
7C80B797	FF75 14	PUSH DWORD PTR SS:[EBP+14]			
7C80B79A	FF75 10	PUSH DWORD PTR SS:[EBP+10]			
7C80B79D	FF75 0C	PUSH DWORD PTR SS:[EBP+0C]			
7C80B7A0	FF75 08	PUSH DWORD PTR SS:[EBP+08]			
7C80B7A3	E8 76FFFFFF	CALL kernel32.MapViewOfFileEx			
7C80B7A8	5D	POP EBP	0012F9F4		
7C80B7A9	C2 1400	RETN 14			
7C80B7AC	81FE 00000040	CMF ESI,40000000			
7C80B7B2	7F84 D9430100	JE kernel32.7C81FB91			
7C80B7B8	6A 00	PUSH 0			

Registers (MMX)			
EAX	014F0000		
ECX	0012F968		
EDX	7C91EB94	ntdll.KiFastSystemCallRet	
EBX	0000007C		
ESP	0012F9C0		
EBP	0012F9C0		
ESI	FFFFFFFF		
EDI	014C0000		
EIP	7C80B7A8	kernel32.7C80B7A8	
C 0	ES 0023 32bit 0(FFFFFFFF)		
P 1	CS 001B 32bit 0(FFFFFFFF)		
A 0	SS 0023 32bit 0(FFFFFFFF)		
Z 1	DS 0023 32bit 0(FFFFFFFF)		
S 0	FS 003B 32bit 7FFDE000(FFF)		
T 0	GS 0000 NULL		
D 0			
O 0	LastErr ERROR_SUCCESS (00000000)		
EFL	00000246 (NO,NB,E,BE,NS,PE,GE,LE)		

take a look into the dump window at the EAX pointed address.

014F0000	4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00	MZÉ.♥...♦... ..
014F0010	B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00	@.....@.....
014F0020	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
014F0030	00 00 00 00 00 00 00 00 00 00 00 00 80 00 00 00	.....C.....
014F0040	0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68	PE  .!.!.*@0L=+Th
014F0050	69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F	is program canno
014F0060	74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20	t be run in DOS
014F0070	6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00	mode.....\$.....
014F0080	50 45 00 00 4C 01 08 00 C5 91 FC 41 00 00 00 00 00	PE..L6.+.z²A....
014F0090	00 00 00 00 E0 00 0F 01 0B 01 05 00 00 92 04 00	...0.*000+..E♦
014F00A0	00 7A 01 00 00 00 00 00 10 00 00 00 10 00 00 00	.z0.....>...>..
014F00B0	00 B0 04 00 00 00 40 00 00 10 00 00 00 02 00 00	...♦...@...>...>..
014F00C0	04 00 00 00 00 00 00 00 04 00 00 00 00 00 00 00	♦...♦...♦...@...>...>..
014F00D0	00 F0 0B 00 00 04 00 00 00 00 00 00 10 00 00 00	..-♦...♦...@...>...>..
014F00E0	00 00 10 00 00 10 00 00 00 10 00 00 10 00 00 00	...>...>...>...>...>..
014F00F0	00 00 00 10 00 00 00 00 00 00 00 00 00 00 00 00	.....
014F0100	54 CA 06 00 DC 02 00 00 00 30 06 00 F4 78 00 00	T±.±.0...0±.¶x..
014F0110	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
014F0120	E0 C9 06 00 08 00 00 00 00 00 00 00 00 00 00 00	0F±.0.....

yep 🤪 this look exactly for the file mapped image.

Execute the RETN instruction and we're redirect to the ASProtect bridge, then let's trace a little:

014E0000	68 00004E01	PUSH 14E0000
014E0005	68 07C3A400	PUSH 0A4C307
014E000A	68 747DAD00	PUSH 0AD7D74
014E000F	E8 5C2F56FF	CALL 00A42F70
014E0014	0000	ADD BYTE PTR DS:[EAX],AL
014E0016	0000	ADD BYTE PTR DS:[EAX],AL

Press Shift+F9 twice and we're into our POPAD / RETN instruction, now we're jumping into this piece of ASProtect code (offset 0x00A24B2C - 0x00A10000 = 0x0014B2C):

00A24B2C	55	PUSH EBP
00A24B2D	8BEC	MOV EBP,ESP
00A24B2F	83C4 F0	ADD ESP,-10
00A24B32	53	PUSH EBX
00A24B33	56	PUSH ESI
00A24B34	894D F4	MOV DWORD PTR SS:[EBP-C],ECX
00A24B37	8955 F8	MOV DWORD PTR SS:[EBP-8],EDX
00A24B39	8945 FC	MOV DWORD PTR SS:[EBP-4],EAX
00A24B3D	8B75 08	MOV ESI,DWORD PTR SS:[EBP+8]
00A24B40	8BC6	MOV EAX,ESI
00A24B42	B9 32000000	MOV ECX,32
00A24B47	33D2	XOR EDX,EDX
00A24B49	F7F1	DIV ECX
00A24B4B	8945 F0	MOV DWORD PTR SS:[EBP-10],EAX
00A24B4E	BB 31000000	MOV EBX,31
00A24B53	8BC3	MOV EAX,EBX
00A24B55	F76D F0	IMUL DWORD PTR SS:[EBP-10]
00A24B58	8BD6	MOV EDX,ESI
00A24B5A	2BD0	SUB EDX,EAX
00A24B5C	52	PUSH EDX
00A24B5D	8B4D F4	MOV ECX,DWORD PTR SS:[EBP-C]
00A24B60	03C8	ADD ECX,EAX
00A24B62	8B55 F8	MOV EDX,DWORD PTR SS:[EBP-8]
00A24B65	8B45 FC	MOV EAX,DWORD PTR SS:[EBP-4]
00A24B68	E8 ABFFFFFF	CALL 00A24B18
00A24B6D	4B	DEC EBX
00A24B6E	83FB FF	CMF EBX,-1
00A24B71	75 E0	JNZ SHORT 00A24B53
00A24B73	5E	POP ESI
00A24B74	5B	POP EBX
00A24B75	8BE5	MOV ESP,EBP
00A24B77	5D	POP EBP
00A24B78	C2 0400	RETN 4
00A24B7B	90	NOP

look also at the register window:

EAX	014C0000
ECX	014F0000
EDX	00004248
EBX	014F0000
ESP	0012F9D4
EBP	0012F9F4
ESI	014F0000
EDI	014C0000
EIP	00A24B2C

ECX keep the base address for the mapped file, then we can remind some offset which came from the area which was allocated at address 0x0046C50F by means the first **VirtuaAlloc** API call. A good redirection point may be at the address 0x00A24B34 or to better say at offset **0x0014B34**.

#### Note

Let me say from a general point of view to take care of this offset because we really don't know now if this piece of code was executed for the first time or if this can be executed more time. At this stage we can just remind this offset for future (hope useful 😊) reference.

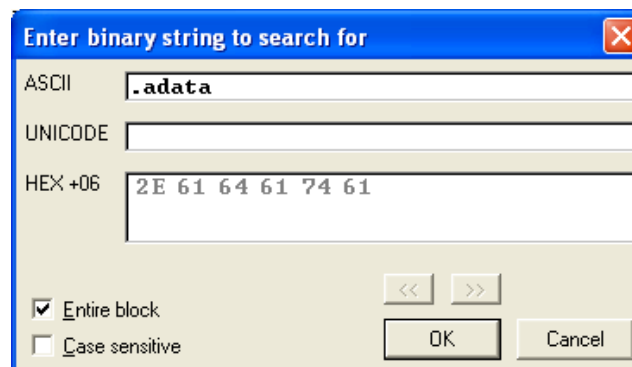
Then to speed up our finding we have to reach the ASPACK code and the perform a binary search with this pattern:

55 8B EC 83 C4 F0 53 56 89 4D F4 89 55 F8 89 45 FC 8B 75 08 8B C6 B9 32 00 00 00 33 D2  
F7 F1 89 45 F0 BB 31 00 00 00 8B C3 F7 6D F0 8B D6 2B D0 52 8B 4D F4 03 C8 8B 55 F8 8B  
45 FC

this gave us the function, the first execution keep in EAX our file image mapped base address.



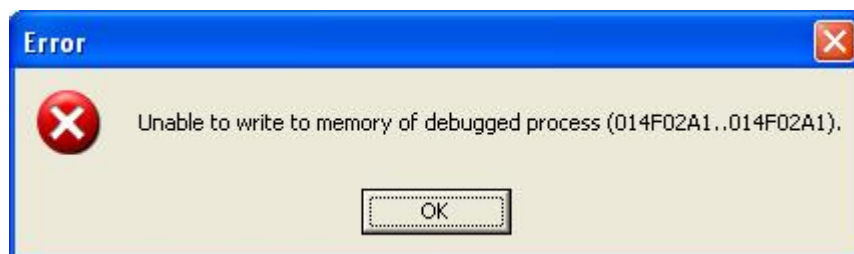
Also pause a second, go into the dump window, press CTRL+G and type in the file image mapping base address, in my case 014F0000 (generally speaking this value may be different because we deal about a run-time area), then press CTRL+B and perform a string search for the text **.adata**:



press OK and you should see:



the red squared parameter was the RAW-SIZE value that we've change to 0x1000 (offset from the base address 0x2A1), then simply change it back to the original value hence set it to 0, but... hey what happen, OllyDbg tell us that we can't change anything from this area:



that's right bcz we don't have the right access to write into this area which is committed from the process, to write in we've to access direct from the process area (by our patching code for example) and also we have to change the file image mapping access attribute before call the **MapViewOfFile**. Well we've to restart now and try some other things before finish.

### Hint

If you don't like this way to found the RAW-SIZE value I'll tell you another way. Go into the dump window and press right click then select **Special -> PE header**, now the dump window is able to show the area contents as this is a PE header section, scroll down a bit and search for the **.adata** section (this is the last one), here you've the RAW-SIZE parameter in a more readable form.

Now simply press Shift+F9, OllyDbg should break into the first placed hardware breakpoint:

00A4E5B0	55	PUSH EBP
00A4E5B1	8BEC	MOV EBP,ESP
00A4E5B3	83C4 B4	ADD ESP,-4C
00A4E5B6	B8 B8E2A400	MOV EAX,0A4E2B8
00A4E5B8	E8 C877FCFF	CALL 00A15D88
00A4E5C0	E8 FB4FFCFF	CALL 00A135C0
00A4E5C5	8D40 00	LEA EAX,DWORD PTR DS:[EAX]
00A4E5C8	0000	ADD BYTE PTR DS:[EAX],AL
00A4E5CA	0000	ADD BYTE PTR DS:[EAX],AL

Place a memory breakpoint (F2) into the **MapViewOfFile**, press Shift+F9, we've a first break, press again and we've another break. Look into the stack window and change the access attribute from 4 to 1 (you can edit it directly into the stack window, right click and modify).

Now press Shift+F9, we have a break into the code which save in ECX the base address, write this code:

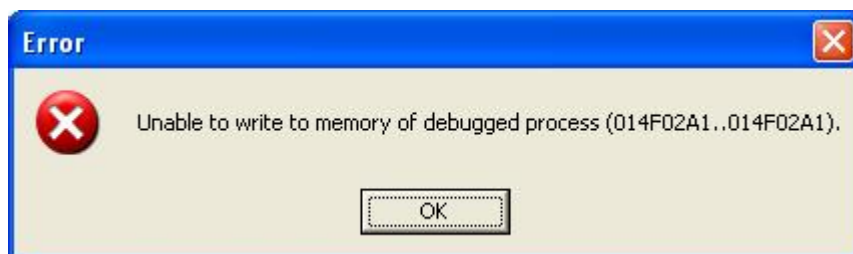
00A24B27	C2 0400	RETN 4
00A24B2A	8BC0	MOV EAX,EAX
00A24B2C	55	PUSH EBP
00A24B2D	8BEC	MOV EBP,ESP
00A24B2F	83C4 F0	ADD ESP,-10
00A24B32	53	PUSH EBX
00A24B33	56	PUSH ESI
00A24B34	C681 A1020000	MOV BYTE PTR DS:[ECX+2A1],0
00A24B38	90	NOP
00A24B3C	90	NOP
00A24B3D	8B75 08	MOV ESI,DWORD PTR SS:[EBP+8]
00A24B40	8BC6	MOV EAX,ESI
00A24B42	B9 32000000	MOV ECX,32
00A24B47	33D2	XOR EDX,EDX
00A24B49	F7F1	DIV ECX
00A24B4B	8945 F0	MOV DWORD PTR SS:[EBP-10],EAX
00A24B4E	BB 31000000	MOV EBX,31
00A24B53	8BC3	MOV EAX,EBX
00A24B55	F76D F0	IMUL DWORD PTR SS:[EBP-10]
00A24B58	8BD6	MOV EDI,ESI
00A24B5A	2BD0	SUB EDI,EAX
00A24B5C	52	PUSH EDI
00A24B5D	8B4D F4	MOV ECX,DWORD PTR SS:[EBP-C]
00A24B60	03C8	ADD ECX,EAX
00A24B62	8B55 F8	MOV EDI,DWORD PTR SS:[EBP-8]
00A24B65	8B45 FC	MOV EAX,DWORD PTR SS:[EBP-4]
00A24B68	E8 ABFFFFFF	CALL 00A24B18
00A24B6D	4B	DEC EBX
00A24B6E	83FB FF	CMP EBX,-1
00A24B71	75 E0	JNZ SHORT 00A24B53
00A24B73	5E	POP ESI
00A24B74	5B	POP EBX
00A24B75	8BE5	MOV ESP,EBP
00A24B77	5D	POP EBP
00A24B78	C2 0400	RETN 4

and execute it by press F7 once, look at the dump window which was focused to the mapped image area:

014F0290	2E 61 64 61 74 61 00 00 00 10 00 00 00 E0 0B 00	.adata...P...00.
014F02A0	00 00 00 00 00 00 07 00 00 00 00 00 00 00 00 00	.....\$*.....
014F02B0	00 00 00 00 40 00 00 E0 00 00 00 00 00 00 00 00	....@..0.....
014F02C0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....

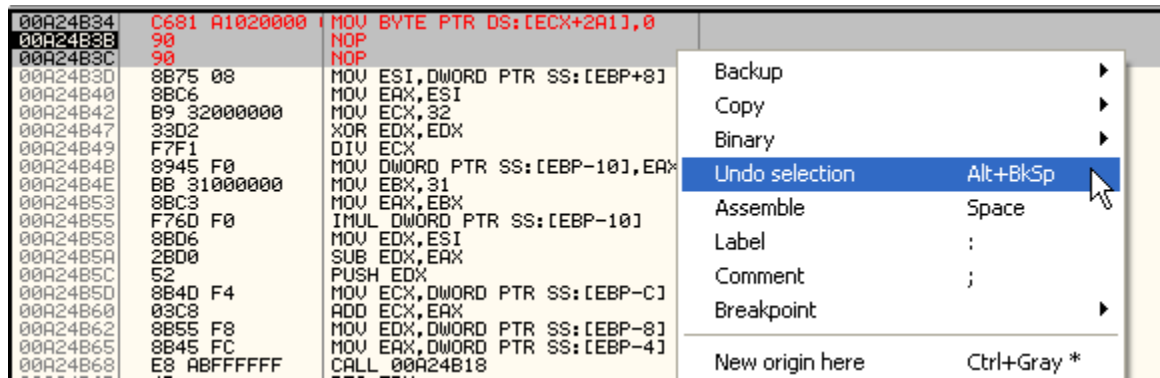
ok, now we're able to change the data because all the condition was satisfied:

1. we've the right access (READ and WRITE) to the mapped image area;
2. the patching instruction has to be executed inside the process space or better say inside the space of the process which have created the mapped image; if you try to change this byte directly from OllyDbg (regard you have set the right access flag) you will reach again another write error like the previous one.



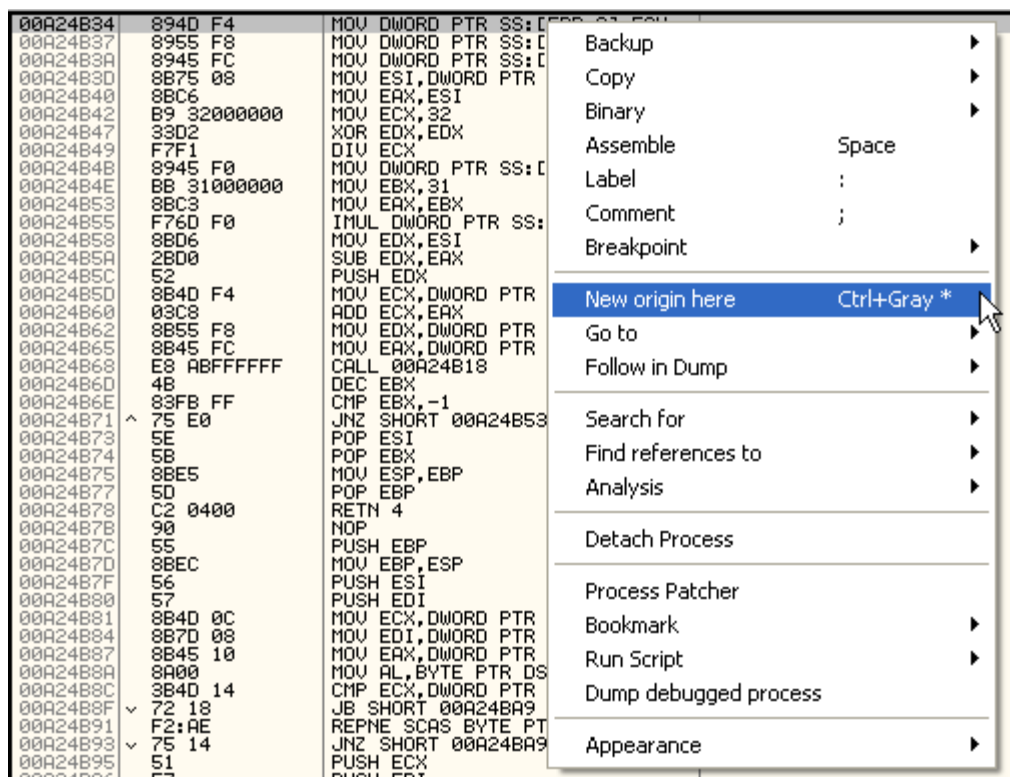
ok after this little secret 😊 we've to restore the code and go on in our analysis.

Then highlight the modified code, right click and choose the **Undo selection** option or simply use the shortcut Alt+BkSp as you like.



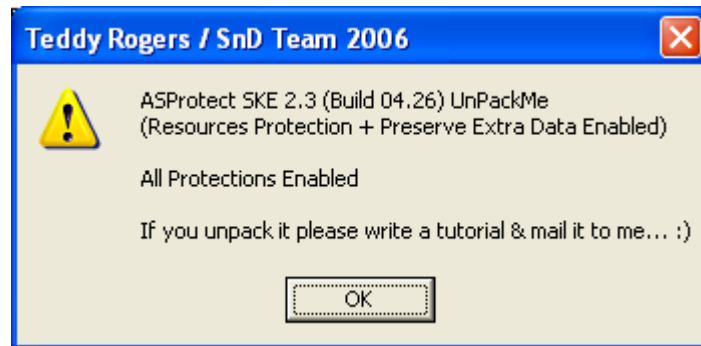
Now the code was restored to the original form (what we are doing now in manual way must be performed by our inline code, I'll just show you how we've to proceed, but also remind that our inline code have to restore also other byte than the single RAW-SIZE data into the PE header, in fact we have also to restore the byte related the first redirection at the end of the first decryption loop, address 0x0046C1AC).

Now we've also to reset the instruction pointer (EIP) to the restored instruction, put your mouse into the MOV restored instruction and do a right click then choose the **New origin here** option like I show below:



Now after pressing Shift+F9 we experienced other break in this point then we've to set a counter because the patch must be applied one and one time.

Continue with Shift+F9, you keep other breakpoint into the **MapViewOfFile**, remove this breakpoint and again with Shift+F9 until...



yep it runnnn then we've successfully defeated the CRC now 😎.

Stop again yourself from singing and think about how we can change the access properties byte before run the **MapViewOfFile** API, we've proof what we can defeat the CRC stuff but we've another point unsolved, a suitable hook point was at the famous POPAD / RETN point, here we've all the API calling the we can also set another counter and break only when we've a match from our counter, we're sure for the **MapViewOfFile** API and then go to change the access mode for the file mapping area.

Ok then now is time to found a easy way to recover the redirection point at the POPAD / RETN instruction, then you've basically two way, one follow the Madman binary search as him show in [1] or simply after you reach the ASPACK code perform a binary search for:

```
8B C0 55 8B EC 83 C4 F4 53 8B 42 30 83 E8 20 83 E8 04 89 45 F4 33 C0 8B 4D F4 83 C1 20
8B D8 C1 E3 02 2B CB 83 E9 04 8B 5C 82 20 89 19 40 83 F8 08
```

Place a breakpoint into the RETN instruction after the POPAD and press Shift+F9, now you've to count the number of time you reach this breakpoint and stop counting when you see into the stack the **MapViewOfFile** text string, my count for this unpackMe was equal to 46.

Well in order to reassuming the step that we've to do in order to fix the CRC we have:

1. Reach the ASPACK sequence
2. To identify the first redirection perform a binary search for:

```
8B C0 55 8B EC 83 C4 F4 53 8B 42 30 83 E8 20 83 E8 04 89 45 F4 33 C0 8B 4D F4 83
C1 20 8B D8 C1 E3 02 2B CB 83 E9 04 8B 5C 82 20 89 19 40 83 F8 08
```

3. Place a breakpoint into the RETN instruction.
4. To identify the second redirection perform a binary search for:

```
55 8B EC 83 C4 F0 53 56 89 4D F4 89 55 F8 89 45 FC 8B 75 08 8B C6 B9 32 00 00 00
33 D2 F7 F1 89 45 F0 BB 31 00 00 00 8B C3 F7 6D F0 8B D6 2B D0 52 8B 4D F4 03 C8
8B 55 F8 8B 45 FC
```

5. Place a breakpoint into the MOV DWORD PTR SS:[EBP-C],ECX instruction.
6. Press Shift+F9 and increment the counter#1 every time you reach the breakpoint until you can see into the stack the **MapViewOfFile** API name, when you reach the counter number change the access right by perform a MOV BYTE [ESP+0x0C],0x01 instruction.



7. When you break into the second redirection (see point 4), check if the counter#2 was equal to 0, if yes keep ECX and perform the byte restoring into the PE header image (offset 0x2A1) and into the file image section (offset 0x002B1AC), increment the counter to prevent further access to this patching point.
8. Open the memory window into the .adata section and see if some byte will be set to zero from ASProtect, found the first byte to 0x90 this is the place how we've to place the subsequent cave code.
9. that's enough for the file integrity check.

After this big in-deep analysis we can return to the last things the memory integrity check, to found this simply put a memory breakpoint into the first byte of the OEP (if this isn't obfuscated) or into the fake OEP (you can found it by using the exception method) and press Shift+F9 to run the program, we should break into the decompression loop:

00B6C5FE	A1 6CFAB600	MOV EAX,DWORD PTR DS:[B6FA6C]	
00B6C603	C600 CF	MOV BYTE PTR DS:[EAX],0CF	
00B6C606	EB 7E	JMP SHORT 00B6C686	
00B6C608	E8 3F5FFCFF	CALL 00B3254C	Decompression loop - START
00B6C60D	8BD8	MOV EBX,EAX	
00B6C60F	8B45 EC	MOV EAX,DWORD PTR SS:[EBP-14]	
00B6C612	8B00	MOV EAX,DWORD PTR DS:[EAX]	
00B6C614	0345 E8	ADD EAX,DWORD PTR SS:[EBP-18]	
00B6C617	8945 FC	MOV DWORD PTR SS:[EBP-4],EAX	
00B6C61A	8B45 EC	MOV EAX,DWORD PTR SS:[EBP-14]	
00B6C61D	8B48 04	MOV ECX,DWORD PTR DS:[EAX+4]	
00B6C620	8BD3	MOV EDX,EBX	
00B6C622	8B45 FC	MOV EAX,DWORD PTR SS:[EBP-4]	
00B6C625	E8 96FAFFFF	CALL 00B6C0C0	
00B6C62A	8B45 EC	MOV EAX,DWORD PTR SS:[EBP-14]	
00B6C62D	8B70 04	MOV ESI,DWORD PTR DS:[EAX+4]	
00B6C630	8A45 FC	MOV AL,BYTE PTR SS:[EBP-4]	
00B6C633	8B15 14FAB600	MOV EDX,DWORD PTR DS:[B6FA14]	
00B6C639	8B02	MOV BYTE PTR DS:[EDX],AL	
00B6C63B	8BC6	MOV EAX,ESI	
00B6C63D	8B15 28FAB600	MOV EDX,DWORD PTR DS:[B6FA28]	
00B6C643	8B02	MOV BYTE PTR DS:[EDX],AL	
00B6C645	8B7D FB 00	CMP BYTE PTR SS:[EBP-5],0	
00B6C649	75 1E	JNZ SHORT 00B6C669	
00B6C64B	C645 FB 01	MOV BYTE PTR SS:[EBP-5],1	
00B6C64F	56	PUSH ESI	
00B6C650	8B75 FC	MOV ESI,DWORD PTR SS:[EBP-4]	
00B6C653	83C6 14	ADD ESI,14	
00B6C656	FF36	PUSH DWORD PTR DS:[ESI]	
00B6C658	C606 C3	MOV BYTE PTR DS:[ESI],0C3	
00B6C65B	FFD6	CALL NEAR ESI	
00B6C65D	8F06	POP DWORD PTR DS:[ESI]	
00B6C65F	5E	POP ESI	
00B6C660	8BD6	MOV EDX,ESI	
00B6C662	8BC3	MOV EAX,EBX	
00B6C664	E8 0BFBFFFF	CALL 00B6C174	
00B6C669	8BCE	MOV ECX,ESI	
00B6C66B	8BD3	MOV EDX,EBX	
00B6C66D	8B45 FC	MOV EAX,DWORD PTR SS:[EBP-4]	
00B6C670	E8 839AFCFF	CALL 00B360F8	
00B6C675	8B45 EC	MOV EAX,DWORD PTR SS:[EBP-14]	
00B6C678	8B50 04	MOV EDX,DWORD PTR DS:[EAX+4]	
00B6C67B	8BC3	MOV EAX,EBX	
00B6C67D	E8 E25EFCFF	CALL 00B32564	
00B6C682	8345 EC 0C	ADD DWORD PTR SS:[EBP-14],0C	
00B6C686	8B45 EC	MOV EAX,DWORD PTR SS:[EBP-14]	
00B6C689	8B40 04	MOV EAX,DWORD PTR DS:[EAX+4]	
00B6C68C	85C0	TEST EAX,EAX	
00B6C68E	0F87 74FFFFFF	JA 00B6C608	Decompression loop - END
00B6C694	837D F4 00	CMP DWORD PTR SS:[EBP-C],0	
00B6C698	74 08	JE SHORT 00B6C6A2	
00B6C69A	8B45 F4	MOV EAX,DWORD PTR SS:[EBP-C]	
00B6C69D	8B55 F0	MOV EDX,DWORD PTR SS:[EBP-10]	
00B6C6A0	8910	MOV DWORD PTR DS:[EAX],EDX	
00B6C6A2	5F	POP EDI	
00B6C6A3	5E	POP ESI	
00B6C6A4	5B	POP EBX	
00B6C6A5	8BE5	MOV ESP,EBP	
00B6C6A7	5D	POP EBP	
00B6C6A8	C3	RETN	



Some step away from this code we're into the memory CRC check routine:

00B3FA20	33C0	XOR EAX,EAX	
00B3FA22	C3	RETN	
00B3FA23	90	NOP	
00B3FA24	53	PUSH EBX	CRC Checking routine
00B3FA25	56	PUSH ESI	
00B3FA26	57	PUSH EDI	
00B3FA27	55	PUSH EBP	
00B3FA28	51	PUSH ECX	
00B3FA29	890424	MOV DWORD PTR SS:[ESP],EAX	
00B3FA2C	8B0424	MOV EAX,DWORD PTR SS:[ESP]	
00B3FA2F	8B48 48	MOV ECX,DWORD PTR DS:[EAX+48]	
00B3FA32	8B0424	MOV EAX,DWORD PTR SS:[ESP]	
00B3FA35	8B58 4C	MOV EBX,DWORD PTR DS:[EAX+4C]	
00B3FA38	8B0424	MOV EAX,DWORD PTR SS:[ESP]	
00B3FA3B	8B70 50	MOV ESI,DWORD PTR DS:[EAX+50]	
00B3FA3E	8B0424	MOV EAX,DWORD PTR SS:[ESP]	
00B3FA41	8B40 54	MOV EAX,DWORD PTR DS:[EAX+54]	
00B3FA44	8BFE	MOV EDI,ESI	
00B3FA46	23FB	AND EDI,EBX	
00B3FA48	8BEB	MOV EBP,EBX	
00B3FA4A	F7D5	NOT EBP	
00B3FA4C	23E8	AND EBP,EAX	
00B3FA4E	0BFD	OR EDI,EBP	
00B3FA50	033A	ADD EDI,DWORD PTR DS:[EDX]	
00B3FA52	03CF	ADD ECX,EDI	
00B3FA54	8BF9	MOV EDI,ECX	
00B3FA56	C1E7 03	SHL EDI,3	
00B3FA59	C1E9 10	SHR ECX,10	
00B3FA5C	0BF9	OR EDI,ECX	
00B3FA5E	8BCF	MOV ECX,EDI	
00B3FA60	8BFB	MOV EDI,EBX	
00B3FA62	23F9	AND EDI,ECX	
00B3FA64	8BE9	MOV EBP,ECX	
00B3FA66	F7D5	NOT EBP	
00B3FA68	23EE	AND EBP,ESI	

This code will be executed some time, after all the executions we land into this routine:

00B44A0A	8BC0	MOV EAX,EAX	
00B44A0C	53	PUSH EBX	
00B44A0D	56	PUSH ESI	
00B44A0E	57	PUSH EDI	
00B44A0F	8BFA	MOV EDI,EDX	
00B44A11	8BF0	MOV ESI,EAX	
00B44A13	B2 01	MOV DL,1	
00B44A15	A1 14F7B300	MOV EAX,DWORD PTR DS:[B3F714]	
00B44A1A	E8 C5E1FEFF	CALL 00B32BE4	
00B44A1F	8BD8	MOV EBX,EAX	
00B44A21	8BC3	MOV EAX,EBX	
00B44A23	8B10	MOV EDX,DWORD PTR DS:[EAX]	
00B44A25	FF12	CALL NEAR DWORD PTR DS:[EDX]	
00B44A27	8BD6	MOV EDX,ESI	
00B44A29	8BCF	MOV ECX,EDI	
00B44A2B	8BC3	MOV EAX,EBX	
00B44A2D	8B30	MOV ESI,DWORD PTR DS:[EAX]	
00B44A2F	FF56 04	CALL NEAR DWORD PTR DS:[ESI+4]	
00B44A32	8BC3	MOV EAX,EBX	
00B44A34	8B10	MOV EDX,DWORD PTR DS:[EAX]	
00B44A36	FF52 08	CALL NEAR DWORD PTR DS:[EDX+8]	
00B44A39	8BC3	MOV EAX,EBX	
00B44A3B	8B10	MOV EDX,DWORD PTR DS:[EAX]	
00B44A3D	FF52 0C	CALL NEAR DWORD PTR DS:[EDX+C]	
00B44A40	8B30	MOV ESI,DWORD PTR DS:[EAX]	DS:[00BE804C]=BF39AFFD (Calculated CRC value)
00B44A42	8BC3	MOV EAX,EBX	
00B44A44	E8 CBE1FEFF	CALL 00B32C14	
00B44A49	8BC6	MOV EAX,ESI	
00B44A4B	5F	POP EDI	
00B44A4C	5E	POP ESI	
00B44A4D	5B	POP EBX	
00B44A4E	C3	RETN	

reach the RETN instruction and we're into the MOV EBP,EAX instruction just before the check 45, patching delivery can be do just after this checking, for example into the MOV EAX instruction.

00B6CD55	03C7	ADD EAX,EDI	
00B6CD57	8B5424 0C	MOV EDX,DWORD PTR SS:[ESP+C]	
00B6CD5B	E8 AC7CFDFF	CALL 00B44A0C	
00B6CD60	8BE8	MOV EBP,EAX	EAX current CRC
00B6CD62	892D 5882B700	MOV DWORD PTR DS:[B78258],EBP	
00B6CD68	3B6C24 20	CMP EBP,DWORD PTR SS:[ESP+20]	[ESP+20] = right CRC (BF39AFFD)
00B6CD6C	74 0C	JE SHORT 00B6CD7A	
00B6CD6E	68 04CEB600	PUSH 0B6CE04	ASCII "45J0"
00B6CD73	E8 D889FDFF	CALL 00B45750	
00B6CD78	EB 12	JMP SHORT 00B6CD8C	
00B6CD7A	8B4424 0C	MOV EAX,DWORD PTR SS:[ESP+C]	
00B6CD7E	A3 4482B700	MOV DWORD PTR DS:[B78244],EAX	

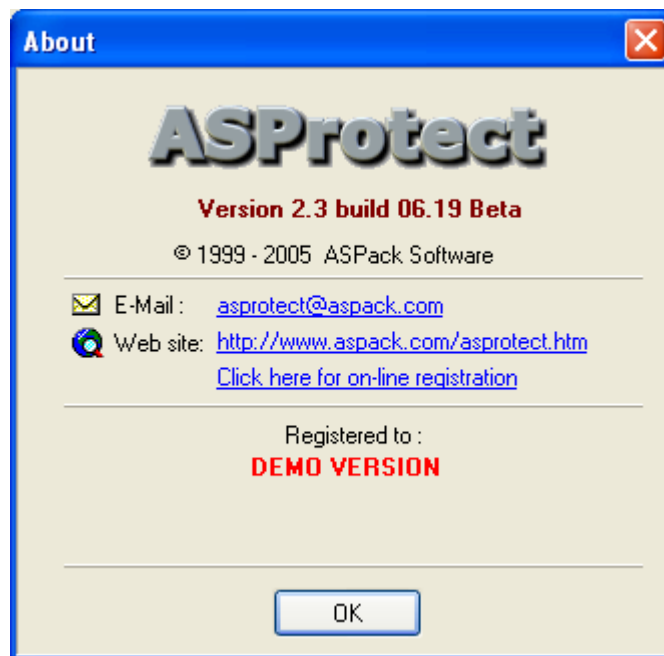
Stack:

0012FF34	00000000	
0012FF38	00400000	UnPackMe.00400000
0012FF3C	00001000	
0012FF40	00049200	
0012FF44	00000000	
0012FF48	00000000	
0012FF4C	00000000	
0012FF50	00000000	
0012FF54	BF39AFFD	CRC
0012FF58	00000000	
0012FF5C	00000000	
0012FF60	00000000	
0012FF64	0012FF98	

From this point of view the patching delivery stage can be done in the same way as we do into the old ASProtect version.

## Extension to the ASProtect 2.3 Build 06.19 Beta

Now we will try our approach with the newest ASProtect release, download the latest demo from the [www.aspack.com](http://www.aspack.com) web site:



Load in OllyDbg and trace to walk through all the decryption loop:

00610121	81E2 45149E12	AND EDX,129E1445	
00610127	8939	MOV DWORD PTR DS:[ECX],EDI	
00610129	B3 BF	MOV BL,0BF	
0061012B	83E9 04	SUB ECX,4	
0061012E	81EB B598C52F	SUB EBX,2FC598B5	
00610134	81EE 01000000	SUB ESI,1	
0061013A	0F85 10000000	JNZ ASProtect.00610150	
00610140	81E3 33A6776E	AND EBX,6E77A633	
00610146	E9 14000000	JMP ASProtect.0061015F	LOOP#1 exit
0061014B	90	NOP	
0061014C	90	NOP	
0061014D	90	NOP	
0061014E	90	NOP	
0061014F	90	NOP	
00610150	51	PUSH ECX	
00610151	80F7 DD	XOR BH,0DD	
00610154	5A	POP EDX	
00610155	E9 A6FFFFFF	JMP ASProtect.00610100	
0061015A	D99E 7F4C95E8	FSTP QWORD PTR DS:[ESI+E8954C7F]	
00610160	05 00000039	ADD EAX,39000000	
00610165	7E DF	JLE SHORT ASProtect.00610146	

**LOOP#1 EXIT POINT (Redirection #1 - hardcoded):** 0x00610146

**ORIGINAL CODE:** JMP 61015F (E9 14 00 00 00)

Trace and we've another loop (number 2):

00610184	66:BF 8C90	MOV DI,908C	
00610188	5F	POP EDI	
00610189	81C3 E7090000	ADD EBX,9E7	
0061018F	66:8BFE	MOV DI,SI	
00610192	BA 00000000	MOV EDX,0	
00610197	66:BF 8EB8	MOV DI,0B88E	
0061019B	FF341A	PUSH DWORD PTR DS:[EDX+EBX]	Loop#2 start
0061019E	81D7 C1075856	ADC EDI,565807C1	
006101A4	58	POP EAX	
006101A5	81C0 DAB8EB5D	ADD EAX,5DEBB8DA	
006101AB	81F7 9F7BEB30	XOR EDI,30EB7B9F	
006101B1	81C0 0B514C22	ADD EAX,224C510B	
006101B7	81C6 D8C38A24	ADD ESI,248AC308	
006101BD	81F0 E80BB254	XOR EAX,54B20BE8	
006101C3	890413	MOV DWORD PTR DS:[EBX+EDX],EAX	
006101C6	66:81C7 25F1	ADD DI,0F125	
006101CB	83EA 04	SUB EDX,4	
006101CE	B9 873C0515	MOV ECX,15053C87	
006101D3	81FA 90F6FFFF	CMP EDX,-970	
006101D9	0F85 BCFFFFFF	JNZ ASProtect.0061019B	Loop#2 end
006101DF	1B11	SBB EDX,DWORD PTR DS:[ECX]	
006101E1	7A D4	JPE SHORT ASProtect.006101B7	
006101E3	032A	ADD EBP,DWORD PTR DS:[EDX]	
006101E5	6BA2 BAE5C7D6	IMUL ESP,DWORD PTR DS:[EDX+D6C7E5BA],16	
006101EC	52	PUSH EDX	

we've the first difference with the other ASProtect version, if you try to put a memory breakpoint in 0x006101DF ASProtect crash because the loop at the end read the code at the breakpoint address, perform some task and then take the execution of the modified code. But if you put a breakpoint, then a 0xCC instruction, the decrypted instruction will be not valid and perform the crash, to see what is the right instruction simply put one hardware breakpoint and press Shift+F9 to run all the loop, the final code will be:

00610184	66:BF 8C90	MOV DI,908C	
00610188	5F	POP EDI	
00610189	81C3 E7090000	ADD EBX,9E7	
0061018F	66:8BFE	MOV DI,SI	
00610192	BA 00000000	MOV EDX,0	
00610197	66:BF 8EB8	MOV DI,0B88E	
0061019B	FF341A	PUSH DWORD PTR DS:[EDX+EBX]	Loop#2 start
0061019E	81D7 C1075856	ADC EDI,565807C1	
006101A4	58	POP EAX	
006101A5	81C0 DAB8EB5D	ADD EAX,5DEBB8DA	
006101AB	81F7 9F7BEB30	XOR EDI,30EB7B9F	
006101B1	81C0 0B514C22	ADD EAX,224C510B	
006101B7	81C6 D8C38A24	ADD ESI,248AC308	
006101BD	81F0 E80BB254	XOR EAX,54B20BE8	
006101C3	890413	MOV DWORD PTR DS:[EBX+EDX],EAX	
006101C6	66:81C7 25F1	ADD DI,0F125	
006101CB	83EA 04	SUB EDX,4	
006101CE	B9 873C0515	MOV ECX,15053C87	
006101D3	81FA 90F6FFFF	CMP EDX,-970	
006101D9	0F85 BCFFFFFF	JNZ ASProtect.0061019B	Loop#2 end
006101DF	E8 10000000	CALL ASProtect.006101F4	
006101E4	3B11	CMP BYTE PTR DS:[ECX],DL	
006101E6	76 77	JBE SHORT ASProtect.0061025F	
006101E8	E4 4D	IN AL,4D	I/O command
006101EA	0213	ADD DL,BYTE PTR DS:[EBX]	

ok, this is a anti-inline countermeasure and we've to defeat it in order to trace the decompression sequence, then we've to found a way able to write our inline code or at least put a scrambled code that will be decrypted in a right way.

To know how look at the code, the decryption loop start from:

DS:[00610B4B]=2DEF6A6A

At the next cycle we have:

DS:[00610B47]=154BAEE5

DS:[00610B43]=A996EC15

Total length 00610B4B - 6101DF = 96C

If you look into the code the loop end condition is equal to 970, this is because the first four address was fixed before entering the loop.

Ok the the loop scan the subsequent code with step of 4 byte (SUB EDX,4) and rebuild the code after some manipulation, now we've to reverse the algorithm In order to obtain at the end the decryption code a JMP instruction to our patching cave instead a call to the ASProtect code.

To do this we need to know the redirection address for our patching cave #2 then we can evaluate it if we know the starting cave address and the size of the cave#1.

This can be done easily because the size of the cave#1 is the same which we've see into the previous ASProtect 2.0 inline patching tutorial, we have to write the first byte of the cave#2 15 byte far from the starting cave address.

If we assume as starting cave address the 0x0066B100 the cave#2 starting address will be from 0x0050D10F then our redirection code should look as:

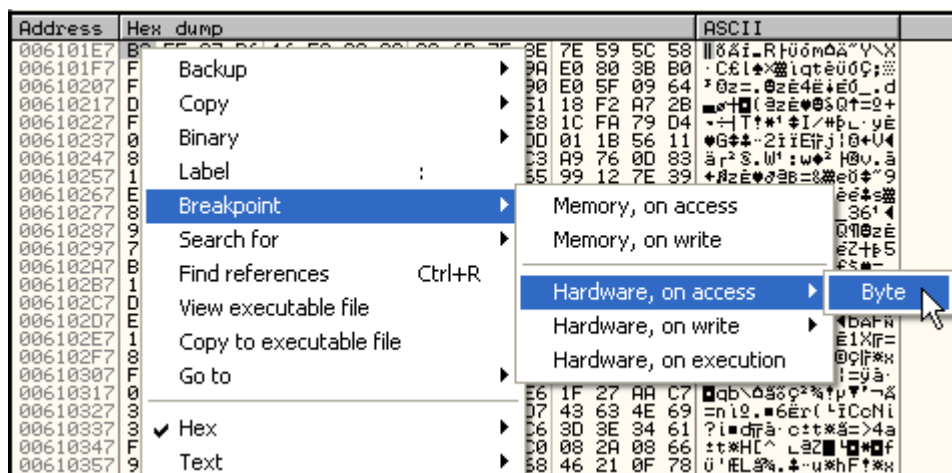
JMP 0x0050D10F

Then exactly 5 byte

E9 A3 1E 02 00

Because the loop scan the code with a step of 4 byte we need to keep 8 byte in length we've to replace 8 byte then we can run freely the loop and stop the execution at the 0x006101E7 address, look at the value into the EAX register and make our things.

Let's go, place a hardware breakpoint on access in 0x006101E7:



when OllyDbg stop we have to step until the address 006101C3 in order to read the next byte (our byte):

Address	Hex dump	ASCII
00610184	66:BF 8C90	MOV DI,909C
00610188	5F	POP EDI
00610189	81C3 E7090000	ADD EBX,9E7
0061018F	66:8BFE	MOV DI,SI
00610192	BA 00000000	MOV EDX,0
00610197	66:BF 8EB8	MOV DI,0B88E
0061019B	FF341A	PUSH DWORD PTR DS:[EDX+EBX]
0061019E	81D7 C1075856	ADC EDI,565807C1
006101A4	58	POP EAX
006101A5	81C0 DAB8EB5D	ADD EAX,5DEBB8DA
006101AB	81F7 9F7BEB30	XOR EDI,30EB7B9F
006101B1	81C0 0B514C22	ADD EAX,224C510B
006101B7	81C6 D8C38A24	ADD ESI,248AC3D8
006101BD	81F0 E80BB254	XOR EAX,54B20BE8
006101C3	890413	MOV DWORD PTR DS:[EBX+EDX],EAX
006101C6	66:81C7 25F1	ADD DI,0F125
006101CB	83EA 04	SUB EDX,4
006101CE	B9 873C0515	MOV ECX,15053C87
006101D3	81FA 90F6FFFF	CMP EDX,-970
006101D9	0F85 BCFFFFFF	JNZ ASProtect.0061019B
006101DF	1B11	SBB EDX,DWORD PTR DS:[ECX]
006101E1	7A D4	JPE SHORT ASProtect.006101B7
006101E3	032A	ADD EBP,DWORD PTR DS:[EDX]
006101E5	6BA2 BA5C7D6	IMUL ESP,DWORD PTR DS:[EDX+D6C7E5BA],13
006101EC	50	PUSH EAX
006101ED	49	DEC ECX
006101F0	4F	DEC ECX

Registers (FPU)

EAX 4E495013  
ECX 15053C87  
EDX FFFFF69C  
EBX 00610B48 ASProtect.00610B48  
ESP 0012FF90  
EBP 00610013 ASProtect.00610013  
ESI C9C5C618  
EDI 6E7349F5  
EIP 0061019E ASProtect.0061019E  
C 0 ES 0023 32bit 0(FFFFFFFF)  
P 1 CS 001B 32bit 0(FFFFFFFF)  
A 0 SS 0023 32bit 0(FFFFFFFF)  
Z 0 DS 0023 32bit 0(FFFFFFFF)  
S 0 FS 003B 32bit 7FFDD000(FFF)  
T 0 GS 0000 NULL  
D 0  
O 0  
I 0 LastErr ERROR\_ACCESS\_DENIED (00000005)  
EFL 00000206 (NO,NB,NE,A,NS,PE,GE,G)  
ST0 empty -UNORM BBB0 01050104 00000000  
ST1 empty 0.0  
ST2 empty 0.0  
ST3 empty 0.0  
ST4 empty 0.0  
ST5 empty 0.0  
ST6 empty 1.00000000000000000000  
ST7 empty 1.00000000000000000000  
3 2 1 0 E S P U O Z D I

Address	Hex dump	ASCII
00539000	1C 43 B2 B8 19 A9 99 80 2A 79 BA 11 1B 2D D1 D6	LC#0400C*yl 4+-0i
00539010	58 B0 81 EA FC 3F 16 A7 5F B8 DB E8 DA C0 05 87	%000?=?_0_00br'4q
00539020	46 E7 85 A8 AE B0 66 05 29 80 78 2D 78 F6 C6 C9	F&000f*)Cw-x+8f
00539030	0A 6C 52 A4 79 4E 3A CE 85 D9 7D CF 93 18 BC 6F	.lRkyN:frj)00t'o

now we have to replace the byte: 66b119

1B 11 7A D4 03 2A 6B A2

with another one able to give us this result:

E9 35 AF 05 00 90 00 00 = X1 | X2

Where we define for sake of clarity:

X1 = E9 35 AF 05

X2 = 00 90 00 00

Ok now take a look at the algo it's time to reverse it:

EDI = F4201B4E

EAX = 024DE477

ESI = EE5089F0

```

0061019B    FF341A    PUSH DWORD PTR DS:[EDX+EBX] ; Loop#2 start
0061019E    81D7 C1075856 ADC EDI,565807C1
006101A4    58        POP EAX
006101A5    81C0 DAB8EB5D ADD EAX,5DEBB8DA
006101AB    81F7 9F7BEB30 XOR EDI,30EB7B9F
006101B1    81C0 0B514C22 ADD EAX,224C510B
006101B7    81C6 D8C38A24 ADD ESI,248AC3D8
006101BD    81F0 E80BB254 XOR EAX,54B20BE8
006101C3    890413    MOV DWORD PTR DS:[EBX+EDX],EAX

```

The major register that we've to keep in consideration was EAX, don't care about other registers.

We have at first in EAX the current dword that ASProtect will decrypt:

EAX = A2 6B 2A 03 -> new value to found, name it generically as X

New value to write = **00 00 90 00** = EAX = (X + 5DEBB8DA + 224C510B) XOR 54B20BE8

Well now look at this boolean expression:

$$\text{Value} = (\text{X} + \text{CONST1}) \text{ XOR } \text{CONST2}$$

But this can be rewritten in this way:

$$\text{Value XOR CONST2} = \text{X} + \text{CONST1}$$

And finally:

$$\text{X} = (\text{Value XOR CONST2}) - \text{CONST1}$$

It give us (X1 and X2 are instruction code then take care to write the byte in reverse order, calculation with Widows calc in Qword / Hex mode and result rounded to Dword):

$$\text{X2} = (\text{00 00 90 00 XOR 54B20BE8}) - 803809E5 = \text{D47A9203 (Rounded to Dword)}$$

Now we have to repeat the step for the last dword:

$$\text{X1} = (\text{05 AF 35 E9 XOR 54B20BE8}) - 803809E5 = \text{D0E5341C (Rounded to Dword)}$$

Then we're ready to build our first two patch cave:

0066B0FE	90	NOP	
0066B0FF	90	NOP	
0066B100	C705 DF016100	MOV DWORD PTR DS:[6101DF],D0E5341C	Cave #1
0066B10A	C705 E3016100	MOV DWORD PTR DS:[6101E3],D47A9203	
0066B114	E9 4650FAFF	JMP ASProtect.0061015F	
0066B119	90	NOP	Cave #2
0066B11A	90	NOP	
0066B11B	90	NOP	
0066B11C	90	NOP	

With this we have finally the jump address to the cave#2 decrypted by ASProtect itself:

00610184	66:BF 8C90	MOV DI,908C	
00610188	5F	POP EDI	
00610189	81C3 E7090000	ADD EBX,9E7	
0061018F	66:8BFE	MOV DI,SI	
00610192	BA 00000000	MOV EDX,0	
00610197	66:BF 8EB8	MOV DI,0B8E	
0061019B	FF341A	PUSH DWORD PTR DS:[EDX+EBX]	Loop start -> read the current instruction
0061019E	81D7 C1075856	ADC EDI,565807C1	
006101A4	58	POP EAX	
006101A5	81C0 DAB8EB5D	ADD EAX,5DEBB8DA	
006101AB	81F7 9F7BEB30	XOR EDI,30EB7B9F	
006101B1	81C0 0B514C22	ADD EAX,224C510B	
006101B7	81C6 D8C38A24	ADD ESI,248AC3D8	
006101BD	81F0 E80BB254	XOR EAX,54B20BE8	
006101C3	890413	MOV DWORD PTR DS:[EBX+EDX],EAX	Write back the decrypted code
006101C6	66:81C7 25F1	ADD DI,0F125	
006101CB	83EA 04	SUB EDX,4	
006101CE	B9 873C0515	MOV ECX,15053C87	
006101D3	81FA 90F6FFFF	CMP EDX,-970	
006101D9	^ 0F85 BCF7FFFF	JNZ ASProtect.0061019B	Loop end
006101DF	- E9 35AF0500	JMP ASProtect.0066B119	Redirection to cave #2 (decrypted by ASProtect)
006101E4	90	NOP	
006101E5	0000	ADD BYTE PTR DS:[EAX],AL	
006101E7	^ 77 E4	JA SHORT ASProtect.006101CD	

Now into the cave#2 we've first to restore the original code and return to the 0x006101DF address, here the partial code (we have to found the next redirection now).



0066B0FF	90	NOP	
0066B100	C705 DF016100	MOV DWORD PTR DS:[6101DF],D0E5341C	Cave #1
0066B10A	C705 E3016100	MOV DWORD PTR DS:[6101E3],D47A9203	
0066B114	E9 4650FAFF	JMP ASProtect.0061015F	
0066B119	C705 DF016100	MOV DWORD PTR DS:[6101DF],10E8	Cave #2
0066B123	C705 E3016100	MOV DWORD PTR DS:[6101E3],76113800	
0066B12D	E9 AD50FAFF	JMP ASProtect.006101DF	
0066B132	90	NOP	
0066B133	90	NOP	

Then redirection#2 will be decrypted on the fly by ASProtect itself.

Step until the next decryption loop, there is some obfuscated code.

00610284	4B	DEC EBX	
00610285	81C6 94EFE560	ADD ESI,60E5EF94	
0061028B	81FB 88F7FFFF	CMP EBX,-878	
00610291	0F85 11000000	JNZ ASProtect.006102A8	
00610297	8BF1	MOV ESI,ECX	
00610299	E9 36000000	JMP ASProtect.006102D4	Loop #3 exit point
0061029E	8AFB	MOV BH,BL	
006102A0	1871 56	SBB BYTE PTR DS:[ECX+56],DH	
006102A3	D7	XLAT BYTE PTR DS:[EBX+AL]	
006102A4	90	NOP	
006102A5	90	NOP	
006102A6	90	NOP	
006102A7	90	NOP	
006102A8	E9 13000000	JMP ASProtect.006102C0	
006102AD	CF	IRETD	
006102AE	5C	POP ESP	
006102AF	65:3AEB	CMP CH,BL	Superfluous prefix
006102B2	48	DEC EAX	
006102B3	E1 06	LOOPE SHORT ASProtect.006102BB	
006102B5	C7	???	Unknown command
006102B6	F4	HLT	Privileged command
006102B7	1D 92636019	SBB EAX,19606392	
006102BC	90	NOP	
006102BD	90	NOP	
006102BE	90	NOP	
006102BF	90	NOP	
006102C0	E9 61FFFFFF	JMP ASProtect.00610226	Loop end
006102C5	DB78 51	FSTP TBYTE PTR DS:[EAX+51]	
006102C8	B6 B7	MOV DH,0B7	
006102CA	24 8D	AND AL,8D	
006102CC	42	INC EDX	
006102CD	53	PUSH EBX	
006102CE	90	NOP	
006102CF	898E AFBC4584	MOV DWORD PTR DS:[ESI+8445BCAF],ECX	
006102D5	B4 76	MOV AH,76	
006102D7	27	DAA	

**LOOP#3 EXIT POINT (Redirection #3): 0x00610299**

**ORIGINAL CODE:** JMP 6102D4 (E9 36 00 00 00)

Reach the exit point:

00610285	81C6 94EFE560	ADD ESI,60E5EF94	
0061028B	81FB 88F7FFFF	CMP EBX,-878	
00610291	0F85 11000000	JNZ ASProtect.006102A8	
00610297	8BF1	MOV ESI,ECX	
00610299	E9 36000000	JMP ASProtect.006102D4	Loop #3 exit point
0061029E	8AFB	MOV BH,BL	
006102A0	1871 56	SBB BYTE PTR DS:[ECX+56],DH	
006102A3	D7	XLAT BYTE PTR DS:[EBX+AL]	
006102A4	90	NOP	
006102A5	90	NOP	
006102A6	90	NOP	
006102A7	90	NOP	
006102A8	E9 13000000	JMP ASProtect.006102C0	
006102AD	CF	IRETD	
006102AE	5C	POP ESP	
006102AF	65:3AEB	CMP CH,BL	Superfluous prefix
006102B2	48	DEC EAX	
006102B3	E1 06	LOOPE SHORT ASProtect.006102BB	
006102B5	C7	???	Unknown command
006102B6	F4	HLT	Privileged command
006102B7	1D 92636019	SBB EAX,19606392	
006102BC	90	NOP	
006102BD	90	NOP	
006102BE	90	NOP	
006102BF	90	NOP	
006102C0	E9 61FFFFFF	JMP ASProtect.00610226	Loop end
006102C5	DB78 51	FSTP TBYTE PTR DS:[EAX+51]	
006102C8	B6 B7	MOV DH,0B7	
006102CA	24 8D	AND AL,8D	
006102CC	42	INC EDX	
006102CD	53	PUSH EBX	
006102CE	90	NOP	
006102CF	898E AFBC4584	MOV DWORD PTR DS:[ESI+F45BCAF],ECX	
006102D5	BF C9E80E00	MOV EDI,0EE8C9	
006102DA	0000	ADD BYTE PTR DS:[EAX],AL	
006102DC	7E DF	JLE SHORT ASProtect.006102BD	
006102DE	2C F5	SUB AL,0F5	

then we can finish our patching code for the cave #2.

0066B0FE	90	NOP	
0066B0FF	90	NOP	
0066B100	C705 DF016100 1C34E5D0	MOV DWORD PTR DS:[6101DF],D0E5341C	Cave #1
0066B10A	C705 E3016100 03927AD4	MOV DWORD PTR DS:[6101E3],D47A9203	
0066B114	E9 4650FAFF	JMP ASProtect.0061015F	
0066B119	C705 DF016100 E8100000	MOV DWORD PTR DS:[6101DF],10E8	Cave #2
0066B123	C705 E3016100 00381176	MOV DWORD PTR DS:[6101E3],76113800	
0066B12D	C705 99026100 E99EAE05	MOV DWORD PTR DS:[610299],5AE9EE9	
0066B137	E9 A350FAFF	JMP ASProtect.006101DF	
0066B13A	90	NOP	Cave #3
0066B13D	90	NOP	
0066B13E	90	NOP	
0066B13F	90	NOP	

ok trace until we reach another loop (number 4), this loop is similar to the loop #2 then we've to reverse it:

0061030F	66:B8 B7FC	MOV AX,0FCB7	
00610313	8B341A	MOV ESI,DWORD PTR DS:[EDX+EBX]	Loop #4 start
00610316	81C6 FC32DB4D	ADD ESI,4DD832FC	
0061031C	66:B9 AFCB	MOV CX,0CBAF	
00610320	81F6 85ADC55A	XOR ESI,5AC5AD85	
00610326	66:8BFE	MOV DI,SI	
00610329	81C6 DA97E75A	ADD ESI,5AE797DA	
0061032F	80C4 54	ADD AH,54	
00610332	56	PUSH ESI	
00610333	81F1 3E108E03	XOR ECX,38E103E	
00610339	8F041A	POP DWORD PTR DS:[EDX+EBX]	
0061033C	0F8B 03000000	JPO ASProtect.00610345	
00610342	66:8BCE	MOV CX,SI	
00610345	E8 08000000	CALL ASProtect.00610352	
0061034A	F0:69EE 8F1C25FA	LOCK IMUL EBP,ESI,FA251C8F	LOCK prefix is not allowed
00610351	AB	STOS DWORD PTR ES:[EDI]	
00610352	68 C6C3FC05	PUSH 5FCC3C6	
00610357	59	POP ECX	
00610358	5F	POP EDI	
00610359	83EB 04	SUB EBX,4	
0061035C	81FB 24F8FFFF	CMP EBX,-7DC	
00610362	0F85 ABFFFFFF	JNZ ASProtect.00610313	Loop #4 end
00610368	52	PUSH EDX	
00610369	0F88 03000000	JS ASProtect.00610372	

now the major registry is ESI, as above we've to altered the original encrypted instruction in order to make ASProtect able to decrypt them in a JMP to our cave #4.  
Place one hardware breakpoint on execution in 0x0061030F and one other to 0x00610368 in order to see what byte is changed by the loop.

0061030F	66:B8 B7FC	MOV AX,0FCB7	
00610313	8B341A	MOV ESI,DWORD PTR DS:[EDX+EBX]	Loop #4 start
00610316	81C6 FC32DB4D	ADD ESI,4DD832FC	
0061031C	66:B9 AFCB	MOV CX,0CBAF	
00610320	81F6 85ADC55A	XOR ESI,5AC5AD85	
00610326	66:8BFE	MOV DI,SI	
00610329	81C6 DA97E75A	ADD ESI,5AE797DA	
0061032F	80C4 54	ADD AH,54	
00610332	56	PUSH ESI	
00610333	81F1 3E108E03	XOR ECX,38E103E	
00610339	8F041A	POP DWORD PTR DS:[EDX+EBX]	
0061033C	0F8B 03000000	JPO ASProtect.00610345	
00610342	66:8BCE	MOV CX,SI	
00610345	E8 08000000	CALL ASProtect.00610352	
0061034A	F0:69EE 8F1C25FA	LOCK IMUL EBP,ESI,FA251C8F	LOCK prefix is not allowed
00610351	AB	STOS DWORD PTR ES:[EDI]	
00610352	68 C6C3FC05	PUSH 5FCC3C6	
00610357	59	POP ECX	
00610358	5F	POP EDI	
00610359	83EB 04	SUB EBX,4	
0061035C	81FB 24F8FFFF	CMP EBX,-7DC	
00610362	0F85 ABFFFFFF	JNZ ASProtect.00610313	Loop #4 end
00610368	52	PUSH EDX	ASProtect.0061034B
00610369	0F88 03000000	JS ASProtect.00610372	
0061036F	66:8BC7	MOV AX,01	
00610372	5F	POP EDI	
00610373	56	PUSH ESI	
00610374	68 89C8C160	PUSH 60C1C889	

cool the last 7 byte was unchanged, I don't know if this may be a bug but for sure I'll use it and we can place our redirection address on 0x00610368.

**LOOP#4 EXIT POINT (Redirection #4):** 0x00610368

**ORIGINAL CODE:** PUSH EDX / JS ....

We can also write our partial code for the cave #4.

0066B0FE	90	NOP	
0066B0FF	90	NOP	
0066B100	C705 DF016100 1C34E5D0	MOV DWORD PTR DS:[6101DF],D0E5341C	Cave #1
0066B10A	C705 E3016100 03927AD4	MOV DWORD PTR DS:[6101E3],D47A9203	
0066B114	E9 4650FAFF	JMP ASProtect.0061015F	
0066B119	C705 DF016100 E8100000	MOV DWORD PTR DS:[6101DF],10E8	Cave #2
0066B123	C705 E3016100 00381176	MOV DWORD PTR DS:[6101E3],76113800	
0066B12D	C705 99026100 E99EAE05	MOV DWORD PTR DS:[610299],5AE9EE9	
0066B137	E9 A350FAFF	JMP ASProtect.006101DF	
0066B13C	C705 68036100 E9DEAD05	MOV DWORD PTR DS:[610368],5ADDEE9	Cave #3
0066B146	E9 8951FAFF	JMP ASProtect.006102D4	
0066B14B	C705 68036100 520F8803	MOV DWORD PTR DS:[610368],3880F52	Cave #4
0066B155	E9 0E52FAFF	JMP ASProtect.00610368	
0066B15A	90	NOP	
0066B15B	90	NOP	
0066B15C	90	NOP	

Now start again with the step, there is another loop:

006103A2	66:BB C6B3	MOV BX,0B3C6	
006103A6	FF30	PUSH DWORD PTR DS:[EAX]	Loop #5 start
006103A8	E8 07000000	CALL ASProtect.006103B4	
006103AD	2320	AND ESP,DWORD PTR DS:[EAX]	
006103AF	90	NOP	
006103B0	90	NOP	
006103B1	90	NOP	
006103B2	90	NOP	
006103B3	90	NOP	
006103B4	✓ E9 0D000000	JMP ASProtect.006103C6	
006103B9	1176 77	ADC DWORD PTR DS:[ESI+77],ESI	
006103BC	E4 4D	IN AL,4D	I/O command
006103BE	0213	ADD DL,BYTE PTR DS:[EBX]	
006103C0	50	PUSH EAX	
006103C1	49	DEC ECX	
006103C2	4E	DEC ESI	
006103C3	6F	OUTS DX,DWORD PTR ES:[EDI]	I/O command
006103C4	✓ 7C 05	JL SHORT ASProtect.006103CB	
006103C6	5B	POP EBX	
006103C7	5F	POP EDI	
006103C8	66:BE 80DB	MOV SI,0DBD0	
006103CC	81C7 D54EFF3B	ADD EDI,3BFF4ED5	
006103D2	✓ 0F80 00000000	J0 ASProtect.006103D8	
006103D8	81EF EAA67763	SUB EDI,6377A6EA	
006103DE	BA 98AFA152	MOV EDX,52A1AF98	
006103E3	81EF DB8DC951	SUB EDI,51C98D08	
006103E9	8938	MOV DWORD PTR DS:[EAX],EDI	
006103EB	81EB 4F517574	SUB EBX,7475514F	
006103F1	81E8 E5F5E17C	SUB EAX,7CE1F5E5	
006103F7	66:81CE 8696	OR SI,9686	
006103FC	81C0 E1F5E17C	ADD EAX,7CE1F5E1	
00610402	BB 1295DE50	MOV EBX,50DE9512	
00610407	83E9 01	SUB ECX,1	
0061040A	^ 0F85 96FFFFFF	JNZ ASProtect.006103A6	Loop #5 end
00610410	E8 14000000	CALL ASProtect.00610429	
00610415	0C 55	OR AL,55	
00610417	6A 5B	PUSH 5B	
00610419	F8	CLC	
0061041A	D136	SAL DWORD PTR DS:[ESI],1	
0061041C	37	AAA	
0061041D	A4	MOVS BYTE PTR ES:[EDI],BYTE PTR DS:[	

place a software breakpoint (F2) at the address 0x00610410 and press Shift+F9 in order to skip all the loop.

006103A2	66:BB C6B3	MOV BX,0B3C6	
006103A6	FF30	PUSH DWORD PTR DS:[EAX]	Loop #5 start
006103A8	E8 07000000	CALL ASProtect.006103B4	
006103AD	2320	AND ESP,DWORD PTR DS:[EAX]	
006103AF	90	NOP	
006103B0	90	NOP	
006103B1	90	NOP	
006103B2	90	NOP	
006103B3	90	NOP	
006103B4	✓ E9 0D000000	JMP ASProtect.006103C6	
006103B9	1176 77	ADC DWORD PTR DS:[ESI+77],ESI	I/O command
006103BC	E4 40	IN AL,40	
006103BE	0213	ADD DL,BYTE PTR DS:[EBX]	
006103C0	50	PUSH EAX	
006103C1	49	DEC ECX	
006103C2	4E	DEC ESI	
006103C3	6F	OUTS DX,DWORD PTR ES:[EDI]	I/O command
006103C4	✓ 7C 05	JL SHORT ASProtect.006103CB	
006103C6	5B	POP EBX	
006103C7	5F	POP EDI	
006103C8	66:BE B0DB	MOV SI,0DBD	
006103CC	81C7 D54EFF3B	ADD EDI,3BFF4ED5	
006103D2	✓ 0F80 00000000	J0 ASProtect.006103D8	
006103D8	81EF EAA67763	SUB EDI,6377A6EA	
006103DE	BA 90AFA152	MOV EDX,52A1AF98	
006103E3	81EF DB8DC951	SUB EDI,51C98DDB	
006103E9	8938	MOV DWORD PTR DS:[EAX],EDI	
006103EB	81EB 4F517574	SUB EBX,7475514F	
006103F1	81E8 E5F5E17C	SUB EAX,7CE1F5E5	
006103F7	66:81CE 8696	OR SI,9686	
006103FC	81C0 E1F5E17C	ADD EAX,7CE1F5E1	
00610402	BB 1295DE50	MOV EBX,50DE9512	
00610407	83E9 01	SUB ECX,1	
0061040A	^ 0F85 96FFFFFF	JNZ ASProtect.006103A6	Loop #5 end
00610410	E8 14000000	CALL ASProtect.00610429	
00610415	0C 55	OR AL,55	
00610417	6A 5B	PUSH 5B	
00610419	F8	CLC	
0061041A	D136	SAL DWORD PTR DS:[ESI],1	
0061041C	37	AAA	
0061041D	A4	MOVS BYTE PTR ES:[EDI],BYTE PTR DS:[EAX]	
0061041E	0D C2D31009	OR EAX,910D3C2	
00610423	0E	PUSH CS	
00610424	2F	DAS	
00610425	3C C5	CMP AL,0C5	
00610427	1A48 0F	SBB CL,BYTE PTR DS:[EBX+F]	
0061042A	B7 F2	MOV BH,0F2	
0061042C	5A	POP EDX	
0061042D	E8 00000000	CALL ASProtect.00610432	
00610432	5D	POP EBP	
00610433	5B	POP EBX	
00610434	895D 5B	MOV DWORD PTR SS:[EBP+5B],EBX	
00610437	5B	POP EBX	
00610438	895D 5F	MOV DWORD PTR SS:[EBP+5F],EBX	

we can set our redirection to the cave #5 at the address 0x00610410 then we have:

**LOOP#5 EXIT POINT (Redirection #5):** 0x00610410

**ORIGINAL CODE:** CALL 610429 (E8 14 00 00 00)

We can complete the patching code for the cave #4 and write the first code for the cave #5.

0066B0FE	90	NOP	
0066B0FF	90	NOP	
0066B100	C705 DF016100 1C34E5D0	MOV DWORD PTR DS:[6101DF],D0E5341C	Cave #1
0066B10A	C705 E3016100 03927AD4	MOV DWORD PTR DS:[6101E3],D47A9203	
0066B114	- E9 4650FAFF	JMP ASProtect.0061015F	
0066B119	C705 DF016100 E8100000	MOV DWORD PTR DS:[6101DF],10E8	Cave #2
0066B123	C705 E3016100 00381176	MOV DWORD PTR DS:[6101E3],76113800	
0066B12D	C705 99026100 E99EAE05	MOV DWORD PTR DS:[610299],5AE9EE9	
0066B137	- E9 A350FAFF	JMP ASProtect.006101DF	
0066B13C	C705 68036100 E9DEAD05	MOV DWORD PTR DS:[610368],5ADDEE9	Cave #3
0066B146	- E9 8951FAFF	JMP ASProtect.006102D4	
0066B14B	C705 68036100 520F8803	MOV DWORD PTR DS:[610368],3880F52	Cave #4
0066B155	C705 10046100 E94FAD05	MOV DWORD PTR DS:[610410],5AD4FE9	
0066B15F	- E9 0452FAFF	JMP ASProtect.00610368	
0066B164	C705 10046100 E8140000	MOV DWORD PTR DS:[610410],14E8	Cave #5
0066B16E	- E9 9D52FAFF	JMP ASProtect.00610410	
0066B173	90	NOP	
0066B174	90	NOP	
0066B175	90	NOP	

Start to step from the 0x00610410 address, we can reach on address 0x006104E3 the first **VirtualAlloc** API call:

00610404	6A 40	PUSH 40	
00610406	68 00100000	PUSH 1000	
00610408	FFB5 00040000	PUSH DWORD PTR SS:[EBP+408]	
006104E1	6A 00	PUSH 0	
006104E3	FF95 F0030000	CALL DWORD PTR SS:[EBP+3F0]	VirtualAlloc #1
006104E9	8985 CC010000	MOV DWORD PTR SS:[EBP+1CC],EAX	EAX = 00C90000
006104EF	8B9D 00040000	MOV EBX,DWORD PTR SS:[EBP+400]	
006104F5	039D 00040000	ADD EBX,DWORD PTR SS:[EBP+400]	
006104FB	50	PUSH EAX	
006104FC	53	PUSH EBX	
006104FD	E8 04010000	CALL ASProtect.00610606	
00610502	6A 40	PUSH 40	
00610504	68 00100000	PUSH 1000	
00610509	FFB5 00040000	PUSH DWORD PTR SS:[EBP+408]	
0061050F	6A 00	PUSH 0	
00610511	FF95 F0030000	CALL DWORD PTR SS:[EBP+3F0]	VirtualAlloc #2
00610517	8985 31040000	MOV DWORD PTR SS:[EBP+431],EAX	EAX = 00CF0000
0061051D	8985 D0010000	MOV DWORD PTR SS:[EBP+1D0],EAX	
00610523	64:67:R1 0000	MOV EAX,DWORD PTR FS:[0]	
00610528	8985 2D040000	MOV DWORD PTR SS:[EBP+42D],EAX	
0061052E	8B55 5B	MOV EDX,DWORD PTR SS:[EBP+5B]	
00610531	8B85 D0010000	MOV EAX,DWORD PTR SS:[EBP+1D0]	
00610537	8902	MOV DWORD PTR DS:[EDX],EAX	
00610539	8B85 00040000	MOV EAX,DWORD PTR SS:[EBP+408]	
0061053F	8942 04	MOV DWORD PTR DS:[EDX+4],EAX	
00610542	8085 9F030000	LEA EAX,DWORD PTR SS:[EBP+39F]	
00610548	8B40 55	MOV EAX,DWORD PTR DS:[EAX+55]	
0061054E	8942 08	MOV DWORD PTR DS:[EDX+8],EAX	
0061054E	8B85 EC030000	MOV EAX,DWORD PTR SS:[EBP+3EC]	
00610554	8942 10	MOV DWORD PTR DS:[EDX+10],EAX	
00610557	8B85 E8030000	MOV EAX,DWORD PTR SS:[EBP+3E8]	
0061055D	8942 14	MOV DWORD PTR DS:[EDX+14],EAX	
00610560	8B95 CC010000	MOV EDX,DWORD PTR SS:[EBP+1CC]	
00610566	BB F8010000	MOV EBX,1F8	
0061056B	8B7C1A 0C	MOV EDI,DWORD PTR DS:[EDX+EBX+C]	
0061056F	0BF	OR EDI,EDI	
00610571	74 1E	JE SHORT ASProtect.00610591	
00610573	8B4C1A 10	MOV ECX,DWORD PTR DS:[EDX+EBX+10]	
00610577	0BC9	OR ECX,ECX	
00610579	74 11	JE SHORT ASProtect.0061059C	
0061057B	03BD D0010000	ADD EDI,DWORD PTR SS:[EBP+1D0]	
00610581	8B741A 14	MOV ESI,DWORD PTR DS:[EDX+EBX+14]	
00610585	03F2	ADD ESI,EDX	
00610587	C1F9 02	SAR ECX,2	
0061058A	F3:A5	REP MOVS DWORD PTR ES:[EDI],DWORD PTR DS:[ESI]	Copy from first allocated area to the second area
0061058C	83C3 28	ADD EBX,28	
0061058F	EB DA	JMP SHORT ASProtect.0061056B	
00610591	8B85 CC010000	MOV EAX,DWORD PTR SS:[EBP+1CC]	
00610597	50	PUSH EAX	
00610598	8B95 D0010000	MOV EDX,DWORD PTR SS:[EBP+1D0]	
0061059E	52	PUSH EDX	
0061059F	8B18	MOV EBX,DWORD PTR DS:[EAX]	
006105A1	03DA	ADD EBX,EDX	

note the two areas keep the same size.

After the writing loop we have the classical PUSH 8000 instruction, we've reach our first jump into the run-time allocated area, as usual Edi keep the base address.

0061057B	03BD D0010000	ADD EDI,DWORD PTR SS:[EBP+1D0]	
00610581	8B741A 14	MOV ESI,DWORD PTR DS:[EDX+EBX+14]	
00610585	03F2	ADD ESI,EDX	
00610587	C1F9 02	SAR ECX,2	
0061058A	F3:A5	REP MOVS DWORD PTR ES:[EDI],DWORD PTR DS:[ESI]	Copy from first allocated area to the second area
0061058C	83C3 28	ADD EBX,28	
0061058F	EB DA	JMP SHORT ASProtect.0061056B	
00610591	8B85 CC010000	MOV EAX,DWORD PTR SS:[EBP+1CC]	
00610597	50	PUSH EAX	
00610598	8B95 D0010000	MOV EDX,DWORD PTR SS:[EBP+1D0]	
0061059E	52	PUSH EDX	
0061059F	8B18	MOV EBX,DWORD PTR DS:[EAX]	
006105A1	03DA	ADD EBX,EDX	
006105A3	8B85 E4030000	MOV EAX,DWORD PTR SS:[EBP+3E4]	
006105A9	8903	MOV DWORD PTR DS:[EBX],EAX	
006105AB	8B85 E8030000	MOV EAX,DWORD PTR SS:[EBP+3E8]	
006105B1	8943 04	MOV DWORD PTR DS:[EBX+4],EAX	
006105B4	8B85 EC030000	MOV EAX,DWORD PTR SS:[EBP+3EC]	
006105BA	8943 08	MOV DWORD PTR DS:[EBX+8],EAX	
006105BD	5F	POP EDI	
006105BE	5E	POP ESI	
006105BF	8B46 04	MOV EAX,DWORD PTR DS:[ESI+4]	
006105C2	03C7	ADD EAX,EDI	
006105C4	8985 C7010000	MOV DWORD PTR SS:[EBP+1C7],EAX	
006105CA	8B55 5B	MOV EDX,DWORD PTR SS:[EBP+5B]	
006105CD	8B85 C7010000	MOV EAX,DWORD PTR SS:[EBP+1C7]	
006105D3	8942 0C	MOV DWORD PTR DS:[EDX+C],EAX	
006105D6	8D9D 00040000	LEA EBX,DWORD PTR SS:[EBP+40D]	
006105DC	53	PUSH EBX	
006105DD	6A 00	PUSH 0	
006105DF	6A 00	PUSH 0	
006105E1	6A 01	PUSH 1	
006105E3	57	PUSH EDI	
006105E4	8B5E 08	MOV EBX,DWORD PTR DS:[ESI+8]	
006105E7	03DF	ADD EBX,EDI	
006105E9	53	PUSH EBX	
006105EA	68 00800000	PUSH 8000	Redirection to the cave #6 / EDI base address
006105EF	6A 00	PUSH 0	
006105F2	56	PUSH ESI	
006105F2	FF95 F4030000	CALL DWORD PTR SS:[EBP+3F4]	
006105F8	68 00200400	PUSH 00420000	
006105FD	C3	RET	
006105FE	0000	ADD BYTE PTR DS:[EAX],AL	

**ADDRESS FOR REDIRECTION #6:** 0x006105EA

**ORIGINAL CODE:** PUSH 8000 (68 00 80 00 00)

Then we can finish the patching code for the cave #5 and start the code for the cave#6:

0066B0FE	90	NOP	
0066B0FF	90	NOP	
0066B100	C705 DF016100 10	MOV DWORD PTR DS:[6101DF],D0E5341C	Cave #1
0066B10A	C705 E3016100 03	MOV DWORD PTR DS:[6101E3],D47A9203	
0066B114	- E9 4650FAFF	JMP ASProtec.0061015F	
0066B119	C705 DF016100 E8	MOV DWORD PTR DS:[6101DF],10E8	Cave #2
0066B123	C705 E3016100 00	MOV DWORD PTR DS:[6101E3],76113800	
0066B12D	C705 99026100 E8	MOV DWORD PTR DS:[610299],5AE9EE9	
0066B137	- E9 A350FAFF	JMP ASProtec.006101DF	
0066B13C	C705 68036100 E8	MOV DWORD PTR DS:[610368],5ADDEE9	Cave #3
0066B146	- E9 8951FAFF	JMP ASProtec.006102D4	
0066B14B	C705 68036100 52	MOV DWORD PTR DS:[610368],3880F52	Cave #4
0066B155	C705 10046100 E8	MOV DWORD PTR DS:[610410],5AD4FE9	
0066B15F	- E9 0452FAFF	JMP ASProtec.00610368	
0066B164	C705 10046100 E8	MOV DWORD PTR DS:[610410],14E8	Cave #5
0066B16E	C705 EA056100 E8	MOV DWORD PTR DS:[6105EA],5AB8EE9	
0066B178	- E9 9352FAFF	JMP ASProtec.00610410	
0066B17D	893D FCBF6600	MOV DWORD PTR DS:[66BFFC],EDI	cave 6 (save the base address)
0066B183	C705 EA056100 68	MOV DWORD PTR DS:[6105EA],800068	
0066B18D	- E9 5854FAFF	JMP ASProtec.006105EA	
0066B192	90	NOP	
0066B193	90	NOP	
0066B194	90	NOP	

From this point we're ready to jump into the newly allocated memory area, press F7 once.

00D42000	90	NOP	
00D42001	60	PUSHAD	
00D42002	E8 40060000	CALL 00D42647	
00D42007	✓ EB 44	JMP SHORT 00D4204D	
00D42009	0000	ADD BYTE PTR DS:[EAX],AL	
00D4200B	0000	ADD BYTE PTR DS:[EAX],AL	
00D4200D	0000	ADD BYTE PTR DS:[EAX],AL	
00D4200F	0000	ADD BYTE PTR DS:[EAX],AL	
00D42011	87DB	XCHG EBX,EBX	
00D42013	90	NOP	
00D42014	0000	ADD BYTE PTR DS:[EAX],AL	
00D42016	0000	ADD BYTE PTR DS:[EAX],AL	
00D42018	0000	ADD BYTE PTR DS:[EAX],AL	
00D4201A	0000	ADD BYTE PTR DS:[EAX],AL	
00D4201C	0000	ADD BYTE PTR DS:[EAX],AL	
00D4201E	0000	ADD BYTE PTR DS:[EAX],AL	
00D42020	0000	ADD BYTE PTR DS:[EAX],AL	
00D42022	0000	ADD BYTE PTR DS:[EAX],AL	
00D42024	0000	ADD BYTE PTR DS:[EAX],AL	
00D42026	0000	ADD BYTE PTR DS:[EAX],AL	
00D42028	0000	ADD BYTE PTR DS:[EAX],AL	
00D4202A	0000	ADD BYTE PTR DS:[EAX],AL	
00D4202C	0000	ADD BYTE PTR DS:[EAX],AL	
00D4202E	2005 00000000	AND BYTE PTR DS:[0],AL	
00D42034	0000	ADD BYTE PTR DS:[EAX],AL	
00D42036	0000	ADD BYTE PTR DS:[EAX],AL	
00D42038	0000	ADD BYTE PTR DS:[EAX],AL	
00D4203A	0000	ADD BYTE PTR DS:[EAX],AL	
00D4203C	0000	ADD BYTE PTR DS:[EAX],AL	
00D4203E	0000	ADD BYTE PTR DS:[EAX],AL	
00D42040	0000	ADD BYTE PTR DS:[EAX],AL	
00D42042	0000	ADD BYTE PTR DS:[EAX],AL	
00D42044	0000	ADD BYTE PTR DS:[EAX],AL	
00D42046	0000	ADD BYTE PTR DS:[EAX],AL	
00D42048	0000	ADD BYTE PTR DS:[EAX],AL	
00D4204A	0000	ADD BYTE PTR DS:[EAX],AL	
00D4204C	00BB 44294400	ADD BYTE PTR DS:[EBX+442944],BH	
00D42052	03DD	ADD EBX,EBP	
00D42054	2B9D 71294400	SUB EBX,DWORD PTR SS:[EBP+442971]	
00D4205A	83BD 08304400 00	CMP DWORD PTR SS:[EBP+4430D8],0	
00D42061	899D 2F2E4400	MOV DWORD PTR SS:[EBP+442E2F],EBX	
00D42067	✓ 0F85 3E050000	JNZ 00D425AB	
00D4206D	8D85 E0304400	LEA EAX,DWORD PTR SS:[EBP+4430E0]	
00D42073	50	PUSH EAX	
00D42074	FF95 EC314400	CALL DWORD PTR SS:[EBP+4431EC]	
00D4207A	8985 DC304400	MOV DWORD PTR SS:[EBP+4430DC],EAX	
00D42080	8BF8	MOV EDI,EAX	
00D42082	8D9D ED304400	LEA EBX,DWORD PTR SS:[EBP+4430ED]	

this structure is the same as we've see into the previous version then scroll down a little until you reach another PUSH 8000 instruction, this is the our next redirection:

00D420DD	8BC8	MOV ECX,EAX	
00D420DF	8DBD 452A4400	LEA EDI,DWORD PTR SS:[EBP+442A45]	
00D420E5	8BB5 75294400	MOV ESI,DWORD PTR SS:[EBP+442975]	
00D420EB	F3:A4	REP MOVS BYTE PTR ES:[EDI],BYTE PTR	
00D420ED	8B85 75294400	MOV EAX,DWORD PTR SS:[EBP+442975]	
00D420F3	68 00800000	PUSH 8000	Place for redirection to cave #7
00D420F8	6A 00	PUSH 0	
00D420FA	50	PUSH EAX	
00D420FB	FF95 7D294400	CALL DWORD PTR SS:[EBP+44297D]	
00D42101	8D85 512C4400	LEA EAX,DWORD PTR SS:[EBP+442C51]	
00D42107	50	PUSH EAX	
00D42108	C3	RETN	
00D42109	0000	ADD BYTE PTR DS:[EAX],AL	



**OFFSET FOR REDIRECTION #7:** 0x00520F3 (base 0x00CF0000)

**ORIGINAL CODE:** PUSH 8000 (68 00 80 00 00)

Here the final code for the cave #6 and the partial code for the cave #7:

0066B0FE	90	NOP	
0066B0FF	90	NOP	
0066B100	C705 DF016100	10 MOV DWORD PTR DS:[6101DF],D0E5341C	Cave #1
0066B10A	C705 E3016100	03 MOV DWORD PTR DS:[6101E3],D47A9203	
0066B114	- E9 4650FAFF	JMP ASProtect.0061015F	
0066B119	C705 DF016100	E8 MOV DWORD PTR DS:[6101DF],10E8	Cave #2
0066B123	C705 E3016100	00 MOV DWORD PTR DS:[6101E3],76113800	
0066B12D	C705 99026100	E9 MOV DWORD PTR DS:[610299],5AE9EE9	
0066B137	- E9 A350FAFF	JMP ASProtect.006101DF	
0066B13C	C705 68036100	E9 MOV DWORD PTR DS:[610368],5ADDEE9	Cave #3
0066B146	- E9 8951FAFF	JMP ASProtect.006102D4	
0066B14B	C705 68036100	52 MOV DWORD PTR DS:[610368],3880F52	Cave #4
0066B155	C705 10046100	E9 MOV DWORD PTR DS:[610410],5AD4FE9	
0066B15F	- E9 0452FAFF	JMP ASProtect.00610368	
0066B164	C705 10046100	E8 MOV DWORD PTR DS:[610410],14E8	Cave #5
0066B16E	C705 EA056100	E9 MOV DWORD PTR DS:[6105EA],5AB8EE9	
0066B178	- E9 9352FAFF	JMP ASProtect.00610410	
0066B17D	893D FCBF6600	MOV DWORD PTR DS:[66BFFC],EDI	cave 6 (save the base address)
0066B183	C705 EA056100	68 MOV DWORD PTR DS:[6105EA],800068	Restore the PUSH 8000 instruction
0066B18D	C787 F3200500	68 MOV DWORD PTR DS:[EDI+520F3],66B1A568	Redirection to cave #7
0066B197	66:C787 F7200500	MOV WORD PTR DS:[EDI+520F7],0C300	
0066B1A0	- E9 4554FAFF	JMP ASProtect.006105EA	
0066B1A5	60	PUSHAD	
0066B1A6	9C	PUSHFD	cave 7
0066B1A7	A1 FCBF6600	MOV EAX,DWORD PTR DS:[66BFFC]	Load the base address
0066B1AC	C780 F3200500	68 MOV DWORD PTR DS:[EAX+520F3],800068	Restore the push 8000 instruction
0066B1B6	C680 F7200500	00 MOV BYTE PTR DS:[EAX+520F7],0	
0066B1BD	C680 F8200500	6A MOV BYTE PTR DS:[EAX+520F8],6A	
0066B1C4	05 F3200500	ADD EAX,520F3	make the return address
0066B1C9	A3 D1B16600	MOV DWORD PTR DS:[66B1D1],EAX	
0066B1CE	9D	POPAD	
0066B1CF	61	POPAD	
0066B1D0	68 00000000	PUSH 0	
0066B1D5	C3	RETN	
0066B1D6	90	NOP	
0066B1D7	90	NOP	
0066B1D8	90	NOP	
0066B1D9	90	NOP	

Now start tracing from offset 0x00520F3, reach the RETN instruction and jump into the next area.

00042300	8B9D 552A4400	MOV EBX,DWORD PTR SS:[EBP+442A55]
00042313	0BD8	OR EBX,EBX
00042315	74 0A	JE SHORT 00042321
00042317	8B03	MOV EAX,DWORD PTR DS:[EBX]
00042319	8785 592A4400	XCHG DWORD PTR SS:[EBP+442A59],EAX
0004231F	8903	MOV DWORD PTR DS:[EBX],EAX
00042321	8DB5 712A4400	LEA ESI,DWORD PTR SS:[EBP+442A71]
00042327	833E 00	CMPL DWORD PTR DS:[ESI],0
0004232A	0F84 D3000000	JE 00042403
00042330	8DB5 712A4400	LEA ESI,DWORD PTR SS:[EBP+442A71]

Looks equal to previous ASProtect release, scroll down a little you should see the ASPACK sequence:

000425B8	5B	POP EBX
000425B9	0BD8	OR EBX,EBX
000425BB	8985 112F4400	MOV DWORD PTR SS:[EBP+442F11],EAX
000425C1	61	POPAD
000425C2	75 08	JNZ SHORT 000425CC
000425C4	B8 01000000	MOV EAX,1
000425C9	C2 0C00	RETN 0C
000425CC	68 00000000	PUSH 0
000425D1	C3	RETN
000425D2	8B85 DC304400	MOV EAX,DWORD PTR SS:[EBP+4430DC]

reach the POPAD instruction at offset 0x00525C1, this is our next redirection point.

**OFFSET FOR REDIRECTION #8:** 0x00525C1 (base 0x00CF0000)

**ORIGINAL CODE:** POPAD / JNZ



Here you've the full code for the cave#7 ad the partial code for the cave #8:

0066B0FF	90	NOP	
0066B100	C705 DF016100	10 MOV DWORD PTR DS:[6101DF],D0E5341C	Cave #1
0066B10A	C705 E3016100	03 MOV DWORD PTR DS:[6101E3],D47A9203	
0066B114	E9 4650FAFF	JMP ASProtect.0061015F	
0066B119	C705 DF016100	E8 MOV DWORD PTR DS:[6101DF],10E8	Cave #2
0066B123	C705 E3016100	00 MOV DWORD PTR DS:[6101E3],76113800	
0066B12D	C705 99026100	E9 MOV DWORD PTR DS:[610299],5AE9EE9	
0066B137	E9 A350FAFF	JMP ASProtect.006101DF	
0066B13C	C705 68036100	E9 MOV DWORD PTR DS:[610368],5ADDEE9	Cave #3
0066B146	E9 8951FAFF	JMP ASProtect.006102D4	
0066B148	C705 68036100	52 MOV DWORD PTR DS:[610368],3880F52	Cave #4
0066B155	C705 10046100	E9 MOV DWORD PTR DS:[610410],5AD4FE9	
0066B15F	E9 0452FAFF	JMP ASProtect.00610368	
0066B164	C705 10046100	E8 MOV DWORD PTR DS:[610410],14E8	Cave #5
0066B16E	C705 EA056100	E9 MOV DWORD PTR DS:[6105EA],5AB8EE9	
0066B178	E9 9352FAFF	JMP ASProtect.00610410	
0066B17D	893D FCBF6600	MOV DWORD PTR DS:[66BFFC],EDI	cave 6 (save the base address)
0066B183	C705 EA056100	68 MOV DWORD PTR DS:[6105EA],800068	Restore the PUSH 8000 instruction
0066B18D	C787 F3200500	68 MOV DWORD PTR DS:[EDI+520F3],66B1A568	Redirection to cave #7
0066B197	66:C787 F7200500	MOV WORD PTR DS:[EDI+520F7],0C300	
0066B1A0	E9 4554FAFF	JMP ASProtect.006105EA	
0066B1A5	60	PUSHAD	cave 7
0066B1A6	9C	PUSHFD	
0066B1A7	A1 FCBF6600	MOV EAX,DWORD PTR DS:[66BFFC]	Load the base address
0066B1AC	C780 F3200500	68 MOV DWORD PTR DS:[EAX+520F3],800068	Restore the push 8000 instruction
0066B1B6	C680 F7200500	00 MOV BYTE PTR DS:[EAX+520F7],0	
0066B1BD	C680 F8200500	6A MOV BYTE PTR DS:[EAX+520F8],6A	
0066B1C4	C780 C1250500	68 MOV DWORD PTR DS:[EAX+525C1],66B1EA68	Redirection from the POPAD/JNZ instructions
0066B1CE	C780 C5250500	00 MOV DWORD PTR DS:[EAX+525C5],0C300	
0066B1D8	05 F3200500	ADD EAX,520F3	Make the return address
0066B1DD	A3 E5B16600	MOV DWORD PTR DS:[66B1E5],EAX	
0066B1E2	9D	POPF	
0066B1E3	61	POPAD	
0066B1E4	68 00000000	PUSH 0	
0066B1E9	C3	RETN	
0066B1EA	60	PUSHAD	cave 8
0066B1EB	9C	PUSHFD	
0066B1EC	A1 FCBF6600	MOV EAX,DWORD PTR DS:[66BFFC]	
0066B1F1	C780 C1250500	68 MOV DWORD PTR DS:[EAX+525C1],B8087561	Restore the POPAD/JNZ instructions
0066B1FB	66:C780 C5250500	MOV WORD PTR DS:[EAX+525C5],1	
0066B204	05 C1250500	ADD EAX,525C1	Make the return address
0066B209	A3 11B26600	MOV DWORD PTR DS:[66B211],EAX	
0066B20E	9D	POPF	
0066B20F	61	POPAD	
0066B210	68 00000000	PUSH 0	
0066B215	C3	RETN	
0066B216	90	NOP	

Now execute the POPAD / JNZ instruction and walk into to the ASPR.DLL:

00030BC4	55	PUSH EBP
00030BC5	8BEC	MOV EBP,ESP
00030BC7	83C4 B4	ADD ESP,-4C
00030BCA	B8 A408D300	MOV EAX,0D308A4
00030BCF	E8 B451FCFF	CALL 00CF5D88
00030BD4	E8 E729FCFF	CALL 00CF35C0
00030BD9	8D40 00	LEA EAX,DWORD PTR DS:[EAX]
00030BDC	0000	ADD BYTE PTR DS:[EAX],AL
00030BDE	0000	ADD BYTE PTR DS:[EAX],AL
00030BE0	0000	ADD BYTE PTR DS:[EAX],AL

Place one hardware breakpoint on this offset 0x0040BC4 for future reference. Now we can try to perform the byte search step carried for the previous ASProtect release and see if this is able to work also with this newest release.

Perform the first byte searching give us right result, yup we've found the POPAD / RETN routine:

00D2A682	8BC0	MOV EAX,EAX
00D2A684	55	PUSH EBP
00D2A685	8BEC	MOV EBP,ESP
00D2A687	83C4 F4	ADD ESP,-0C
00D2A68A	53	PUSH EBX
00D2A68B	8B42 30	MOV EAX,DWORD PTR DS:[EDX+30]
00D2A68E	83E8 20	SUB EAX,20
00D2A691	83E8 04	SUB EAX,4
00D2A694	8945 F4	MOV DWORD PTR SS:[EBP-C],EAX
00D2A697	33C0	XOR EAX,EAX
00D2A699	8B4D F4	MOV ECX,DWORD PTR SS:[EBP-C]
00D2A69C	83C1 20	ADD ECX,20
00D2A69F	8BD8	MOV EBX,EAX
00D2A6A1	C1E3 02	SHL EBX,2
00D2A6A4	2BC8	SUB ECX,EBX
00D2A6A6	83E9 04	SUB ECX,4
00D2A6A9	8B5C82 20	MOV EBX,DWORD PTR DS:[EDX+EAX*4+20]
00D2A6AD	8919	MOV DWORD PTR DS:[ECX],EBX
00D2A6AF	40	INC EAX
00D2A6B0	83F8 08	CMP EAX,8
00D2A6B3	75 E4	JNZ SHORT 00D2A699
00D2A6B5	8B45 F4	MOV EAX,DWORD PTR SS:[EBP-C]
00D2A6B8	83C0 20	ADD EAX,20
00D2A6BB	8B4A 44	MOV ECX,DWORD PTR DS:[EDX+44]
00D2A6BE	8908	MOV DWORD PTR DS:[EAX],ECX
00D2A6C0	8B42 48	MOV EAX,DWORD PTR DS:[EDX+48]
00D2A6C3	8945 F8	MOV DWORD PTR SS:[EBP-8],EAX
00D2A6C6	8B82 90000000	MOV EAX,DWORD PTR DS:[EDX+90]
00D2A6CC	8945 FC	MOV DWORD PTR SS:[EBP-4],EAX
00D2A6CF	31C0	XOR EAX,EAX
00D2A6D1	8B55 FC	MOV EDX,DWORD PTR SS:[EBP-4]
00D2A6D4	64:8910	MOV DWORD PTR FS:[EAX],EDX
00D2A6D7	FF75 F8	PUSH DWORD PTR SS:[EBP-8]
00D2A6DA	9D	POPF0
00D2A6DB	8B65 F4	MOV ESP,DWORD PTR SS:[EBP-C]
00D2A6DE	61	POPAD
00D2A6DF	C3	RETN
00D2A6E0	5B	POP EBX

place a software breakpoint into the RETN instruction and perform the second byte search:

00D05ACC	55	PUSH EBP
00D05ACD	8BEC	MOV EBP,ESP
00D05ACF	83C4 F0	ADD ESP,-10
00D05AD2	53	PUSH EBX
00D05AD3	56	PUSH ESI
00D05AD4	894D F4	MOV DWORD PTR SS:[EBP-C],ECX
00D05AD7	8955 F8	MOV DWORD PTR SS:[EBP-8],EDX
00D05ADA	8945 FC	MOV DWORD PTR SS:[EBP-4],EAX
00D05ADD	8B75 08	MOV ESI,DWORD PTR SS:[EBP+8]
00D05AE0	8BC6	MOV EAX,ESI
00D05AE2	B9 32000000	MOV ECX,32
00D05AE7	33D2	XOR EDX,EDX
00D05AE9	F7F1	DIV ECX
00D05AEB	8945 F0	MOV DWORD PTR SS:[EBP-10],EAX
00D05AEE	BB 31000000	MOV EBX,31
00D05AF3	8BC3	MOV EAX,EBX
00D05AF5	F76D F0	IMUL DWORD PTR SS:[EBP-10]
00D05AF8	8BD6	MOV EDX,ESI
00D05AFA	2BD0	SUB EDX,EAX
00D05AFC	52	PUSH EDX
00D05AFD	8B4D F4	MOV ECX,DWORD PTR SS:[EBP-C]
00D05B00	03C8	ADD ECX,EAX
00D05B02	8B55 F8	MOV EDX,DWORD PTR SS:[EBP-8]
00D05B05	8B45 FC	MOV EAX,DWORD PTR SS:[EBP-4]
00D05B08	E8 ABFFFFFF	CALL 00D05AB8
00D05B0D	48	DEC EBX
00D05B0E	83FB FF	CMP EBX,-1
00D05B11	75 E0	JNZ SHORT 00D05AF3
00D05B13	5E	POP ESI
00D05B14	5B	POP EBX
00D05B15	8BE5	MOV ESP,EBP
00D05B17	5D	POP EBP
00D05B18	C2 0400	RETN 4
00D05B18	90	NOP
00D05B1C	55	PUSH EBP
00D05B1D	8BEC	MOV EBP,ESP
00D05B1F	56	PUSH ESI
00D05B20	57	PUSH EDI

ohh so good also the second patter was found.

Well, press Shift+F9 and wait, OllyDbg will be break into POPAD routine, count 53 and we keep the first API:

GetModuleFileNameA	count 53
CreateFileA	count 55
CreateFileMappingA	count 56
CloseHandle	count 57
MapViewOfFile	count 58

Look into the stack window:

0012F9BC	77E56105	kernel32.MapViewOfFile
0012F9C0	01F00000	
0012F9C4	00000074	
0012F9C8	00000004	
0012F9CC	00000000	
0012F9D0	00000000	
0012F9D4	00000000	
0012F9D8	0012FF34	

\$ ==>	77E56105	kernel32.MapViewOfFile
\$+4	01F00000	
\$+8	00000074	
\$+C	00000004	
\$+10	00000000	
\$+14	00000000	
\$+18	00000000	
\$+1C	0012FF34	

press F7 once:

\$+4	01F00000	CALL to MapViewOfFile
\$+8	00000074	hMapObject = 00000074
\$+C	00000004	AccessMode = FILE_MAP_READ
\$+10	00000000	OffsetHigh = 0
\$+14	00000000	OffsetLow = 0
\$+18	00000000	MapSize = 0
\$+1C	0012FF34	

Change the access mode from 0x04 to 0x01, leave the breakpoint into the RETN instruction and press Shift+F9.

OllyDbg will break into the ECX routine, now we can grab from ECX the image mapping base:

00005AD4	894D F4	MOV DWORD PTR SS:[EBP-C],ECX	
00005AD7	8955 F8	MOV DWORD PTR SS:[EBP-8],EDX	
00005ADA	8945 FC	MOV DWORD PTR SS:[EBP-4],EAX	
00005ADD	8B75 08	MOV ESI,DWORD PTR SS:[EBP+8]	
00005AE0	8BC6	MOV EAX,ESI	
00005AE2	B9 32000000	MOV ECX,32	
00005AE7	33D2	XOR EDX,EDX	
00005AE9	F7F1	DIV ECX	
00005AEB	8945 F0	MOV DWORD PTR SS:[EBP-10],EAX	
00005AEE	BB 31000000	MOV EBX,31	
00005AF3	8BC3	MOV EAX,EBX	

Registers (3DNow!)	
EAX	01EE0000
ECX	01F10000 ASCII "MZP"
EDX	000020CC
EBX	01F10000 ASCII "MZP"
ESP	0012F9B4
EBP	0012F9CC
ESI	01F10000 ASCII "MZP"
EDI	01EE0000
EIP	00005AD4

Address	Hex dump	ASCII
01F10000	4D 5A 50 00 02 00 00 00 04 00 0F 00 FF FF 00 00	MZP.0...*. ..
01F10010	B8 00 00 00 00 00 00 00 00 40 00 1A 00 00 00 00	@.....@.+.....
01F10020	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....0..
01F10030	00 00 00 00 00 00 00 00 00 00 00 00 00 01 00 00	.....0..
01F10040	BA 10 00 0E 1F B4 09 CD 21 B8 01 4C CD 21 90 90	.A? =!00L=!EE
01F10050	54 68 69 73 20 70 72 6F 67 72 61 6D 20 6D 75 73	This program mus
01F10060	74 20 62 65 20 72 75 6E 20 75 6E 64 65 72 20 57	t be run under W
01F10070	69 6E 33 32 00 0A 24 37 00 00 00 00 00 00 00 00	in32..\$7.....
01F10080	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
01F10090	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
01F100A0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....

now we've to restore:

1. The raw-size of the section .adata at offset ECX+0x399
2. The first hardcoded jump (set to 0x00610146) at offset ECX+0xD7146

well I'll try by hand just to see if this is able to work:

00005A04	C681 99030000	MOV BYTE PTR DS:[ECX+399],0
00005A08	C781 46710D00	MOV DWORD PTR DS:[ECX+07146],14E9
00005AE5	90	NOP
00005AE6	90	NOP
00005AE7	33D2	XOR EDX,EDX
00005AE9	F7F1	DIV ECX

press F7 twice then restore the code and set the new origin to 0x00D05AD4 then press Shift+F9 and.....yeppppp.....it runnnnn 🤪.

Well now just for fun we've to finish the in-line patching.

At this point we've to inject the redirection in two different point, the first one was at the end of the POPAD / RETN routine and the last one into the ECX routine.

Offset for the two redirection may be set as show below:

1. redirection from the POPAD/RETN function at offset 0x003A6DA  
(Original byte 9D 8B 65 F4 61 C3)  
Offset for counter#1 : 0x0066BFF8
2. redirection from the MOV ...,ECX function at offset 0x0015AD4  
(Original byte 89 4D F4 89 55 F8)  
Offset for counter#2 : 0x0066BFF4

Now for the first redirection we've to set the initial counter value into the 0x0066BFF8 buffer (we've a total count of 58 then 0x3A):

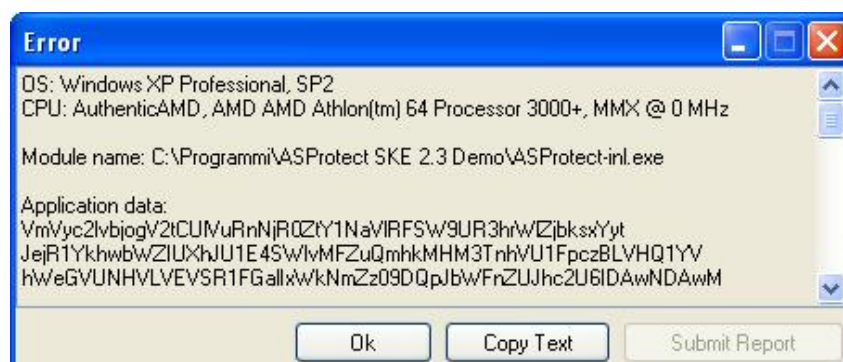
Address	Hex dump
0066BFF8	3A 00 00 00 90 90 90 90

see when this one reach the trigger value, if the trigger point meet the count we can change the access right into the stack, if not simply execute the original instruction. For better coding the counter value will be decreased every time until this one reach zero.

For the second redirection we've to set the initial counter#2 value at offset 0x0066BFF4 to 0x01:

Address	Hex dump
0066BFF4	01 00 00 00 3A 00 00 00 90 90 90 90

Remember also one important things, after all the patching work we've also to think about ASProtect code integrity checking, then we've also to restore the original code in order to avoid this error:



**FULL PATCHING CODE (part 1 of 2)**

```

0066B100  MOV DWORD PTR DS:[6101DF],D0E5341C      ; Cave 1
0066B10A  MOV DWORD PTR DS:[6101E3],D47A9203
0066B114  JMP ASProtec.0061015F

0066B119  MOV DWORD PTR DS:[6101DF],10E8          ; Cave 2
0066B123  MOV DWORD PTR DS:[6101E3],76113800
0066B12D  MOV DWORD PTR DS:[610299],5AE9EE9
0066B137  JMP ASProtec.006101DF

0066B13C  MOV DWORD PTR DS:[610368],5ADDEE9      ; Cave 3
0066B146  JMP ASProtec.006102D4

0066B14B  MOV DWORD PTR DS:[610368],3880F52      ; Cave 4
0066B155  MOV DWORD PTR DS:[610410],5AD4FE9
0066B15F  JMP ASProtec.00610368

0066B164  MOV DWORD PTR DS:[610410],14E8        ; Cave 5
0066B16E  MOV DWORD PTR DS:[6105EA],5AB8EE9
0066B178  JMP ASProtec.00610410

0066B17D  MOV DWORD PTR DS:[66BFFC],EDI          ; Cave 6
0066B183  MOV DWORD PTR DS:[6105EA],800068
0066B18D  MOV DWORD PTR DS:[EDI+520F3],66B1A568
0066B197  MOV WORD PTR DS:[EDI+520F7],0C300
0066B1A0  JMP ASProtec.006105EA

0066B1A5  PUSHAD                                ; Cave 7
0066B1A6  PUSHFD
0066B1A7  MOV EAX,DWORD PTR DS:[66BFFC]
0066B1AC  MOV DWORD PTR DS:[EAX+520F3],800068
0066B1B6  MOV BYTE PTR DS:[EAX+520F7],0
0066B1BD  MOV BYTE PTR DS:[EAX+520F8],6A
0066B1C4  MOV DWORD PTR DS:[EAX+525C1],66B1EA68
0066B1CE  MOV DWORD PTR DS:[EAX+525C5],0C300
0066B1D8  ADD EAX,520F3
0066B1DD  MOV DWORD PTR DS:[66B1E5],EAX
0066B1E2  POPFD
0066B1E3  POPAD
0066B1E4  PUSH 0
0066B1E9  RETN

0066B1EA  PUSHAD                                ; Cave 8
0066B1EB  PUSHFD
0066B1EC  MOV EAX,DWORD PTR DS:[66BFFC]
0066B1F1  MOV DWORD PTR DS:[EAX+525C1],B8087561  ; Restore the POPAD/JNZ instructions
0066B1FB  MOV WORD PTR DS:[EAX+525C5],1
0066B204  MOV DWORD PTR DS:[EAX+3A6DA],66B23C68  ; Write the redirection one (POPAD/RETN)
0066B20E  MOV WORD PTR DS:[EAX+3A6DE],0C300
0066B217  MOV DWORD PTR DS:[EAX+15AD4],66B27868  ; Write the redirection two (MOV ...,ECX)
0066B221  MOV WORD PTR DS:[EAX+15AD8],0C300
0066B22A  ADD EAX,525C1                          ; Make the return address
0066B22F  MOV DWORD PTR DS:[66B237],EAX
0066B234  POPFD
0066B235  POPAD
0066B236  PUSH 0
0066B23B  RETN

```

**FULL PATCHING CODE (part 2 of 2)**

```

0066B23C  PUSHAD                                ; Cave 9 (Redirection for the POPAD/RETN routine)
0066B23D  PUSHFD
0066B23E  MOV EAX,DWORD PTR DS:[66BFF8]         ; Load the counter#1 value in EAX
0066B243  DEC EAX                               ; Decrement the counter
0066B244  MOV DWORD PTR DS:[66BFF8],EAX         ; Write back the new counter#1 value
0066B249  JNZ SHORT ASProtec.0066B270          ; Jump if count isn't reached
0066B24B  MOV EAX,DWORD PTR DS:[66BFFC]
0066B250  MOV DWORD PTR DS:[EAX+3A6DA],F4658B9D ; Restore the original instruction at POPAD/RETN
0066B25A  MOV WORD PTR DS:[EAX+3A6DE],0C361
0066B263  POPFD                                ; Restore the process context
0066B264  POPAD                                ; Restore the process context
0066B265  POPFD                                ; Execute the original POPFD instruction
0066B266  MOV ESP,DWORD PTR SS:[EBP-C]         ; Execute the original MOV ESP,... instruction
0066B269  MOV BYTE PTR SS:[ESP+2C],1           ; Apply the patch to keep the right access
0066B26E  POPAD                                ; Execute the original POPAD instruction
0066B26F  RETN                                 ; Execute the original RETN (go into the API)
0066B270  POPFD                                ; Restore the process context
0066B271  POPAD                                ; Restore the process context
0066B272  POPFD                                ; Execute the original POPFD instruction
0066B273  MOV ESP,DWORD PTR SS:[EBP-C]         ; Execute the original MOV ESP,... instruction
0066B276  POPAD                                ; Execute the original POPAD instruction
0066B277  RETN                                 ; Execute the original RETN (go back)

0066B278  PUSHAD                                ; Cave 10 (Redirection for the MOV ...,ECX)
0066B279  PUSHFD
0066B27A  MOV EAX,DWORD PTR DS:[66BFF4]
0066B27F  DEC EAX
0066B280  MOV DWORD PTR DS:[66BFF4],EAX
0066B285  JNZ SHORT ASProtec.0066B298
0066B287  MOV BYTE PTR DS:[ECX+399],0           ; Patch the file mapping: fix the raw-size
0066B28E  MOV DWORD PTR DS:[ECX+D7146],14E9     ; Patch the file mapping: fix the first jump
0066B298  MOV EAX,DWORD PTR DS:[66BFFC]
0066B29D  MOV DWORD PTR DS:[EAX+15AD4],89F44D89
0066B2A7  MOV WORD PTR DS:[EAX+15AD8],0F855
0066B2B0  ADD EAX,15AD4
0066B2B5  MOV DWORD PTR DS:[66B2BD],EAX
0066B2BA  POPFD
0066B2BB  POPAD
0066B2BC  PUSH 0
0066B2C1  RETN

```

Ok, now I think that's enough for now, try your way and let me know what is your idea, if you've some target packed with the latest ASProtect release of let me know I'll carry on some study about it...



# ThunderPwr

**I would just say thanks to Madman He3cul3s, to all ARTeam friends,  
and to Ricardo Narvaja / Cracks Latinos.  
Of course thanks to you that are still alive at the end of this tutorial.**