



# Writing a Loader to patch applications protected with AsProtect 2.0, theory and practice.

Shub-Nigurrath & ThunderPwr of ARTeam

Version 1.0 - June 2005

1. Abstract.....	1
2. Target description .....	2
3. Target analysis .....	2
4. Cracking stage part # 1 - Trial time fixing.....	2
5. Cracking stage part # 2 - Nag removing .....	12
6. Writing a debugger loader for the program, using Shub-Nigurrath's framework .....	13
6.1. Resemble the patches .....	13
6.2. Set the Victim Details .....	15
6.3. Write the GateCondition .....	15
6.4. Close the Loader and Hide the Debugger .....	19
6.5. Run the Loader.....	19
7. References.....	20
8. Conclusions.....	20
9. History.....	20
10. Greetings.....	20

## Keyword

Loader, Process, Thread, Breakpoint, Debugging

## 1. Abstract

This tutorial addresses a target packed with AsProtect 2.0 and shows you how to find where the application sets its trial information, how to defeat the nag, and finally write a generic loader for programs protected with AsProtect version 2.0.

The interesting thing is that the patch is done without manual unpacking the program, resulting in a lighter distribution of the patch and in a more reliable crack. The shown method (credits to ThunderPwr) works for all programs protected with version 2.0 of AsProtect. You will no more worry about stolen bytes because they are not useful for the applications patching and we really do not need them, AsProtect will continue managing them for us.

The real target used is indeed just an excuse used to show the concepts, also for this reason we will not completely patch it, but only the trial and the initial nag. Do your homework to complete the patch on your own.

Have phun,  
*Shub-Nigurrath & ThunderPwr*



## 2. Target description

Some notes about the target:

Deep Log Analyzer version 2.4 - <http://www.deep-software.com>

You might also use this link: [http://intechhosting.com/~access/ARTeam/tools/dlatrial\\_24.exe](http://intechhosting.com/~access/ARTeam/tools/dlatrial_24.exe)

Analyze web site visitors' behaviour and get complete website usage statistics. It's a completely new advanced and affordable web analytics solution for small and medium size websites. Know exactly where your customers come from, how they move through the site and where they leave it. Get reports about accessed site resources, visitors' activity, referral sites, search engines and keywords, browsers and OS, search spiders, server errors and more. Deep Log Analyzer makes it easy to view how statistics change over time or compare reports for different time intervals, dig deeper into the data with the unique hierarchical reports and reveal hidden patterns in your site stats. In addition, Deep Log Analyzer is a highly configurable program. Create own custom reports or tailor standard reports to meet your specific needs. Use data from company's sales or CRM database to create marketing reports for making smart business decisions. Deep Log Analyzer stores site stats data in the standard MS Access database for easy access in other applications.

## 3. Target analysis

After installation we will use PEiD to know something more about this target (if it is packed or about some crypto signature):

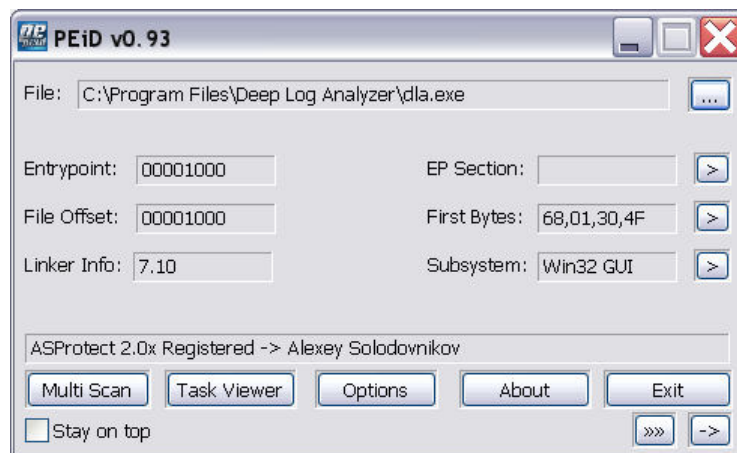


Figure 1 - PEiD response

Target was packed with the latest ASProtect release, this is really good!

## 4. Cracking stage part # 1 - Trial time fixing

As usual the following step is to understand how the protection looks from an external point of view, then just run it and see what happens.



## Writing a Loader to patch applications protected with AsProtect 2.0, theory and practice.

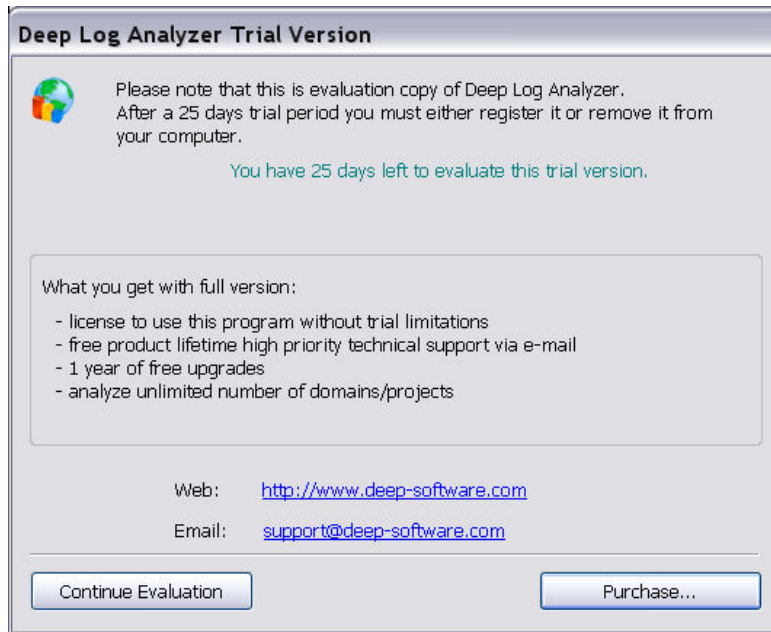


Figure 2 - Starting program's Nag screen

After you press the “Continue Evaluation” button the program’s main screen appears. When ready, look for the Help option into the program toolbar:



Figure 3 - Selection of the About menu in the main program



Figure 4 - Registration info



Close the target and load it into OllyDbg, you re now into the ASProtect loader entry point:

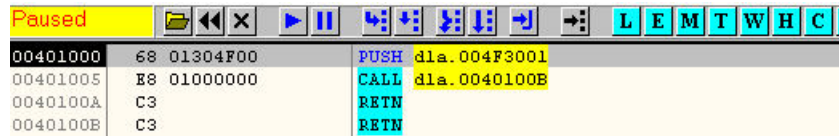


Figure 5 - Entry point of the program

Now uncheck all the exceptions into OllyDbg debugging options window as in Figure 6:

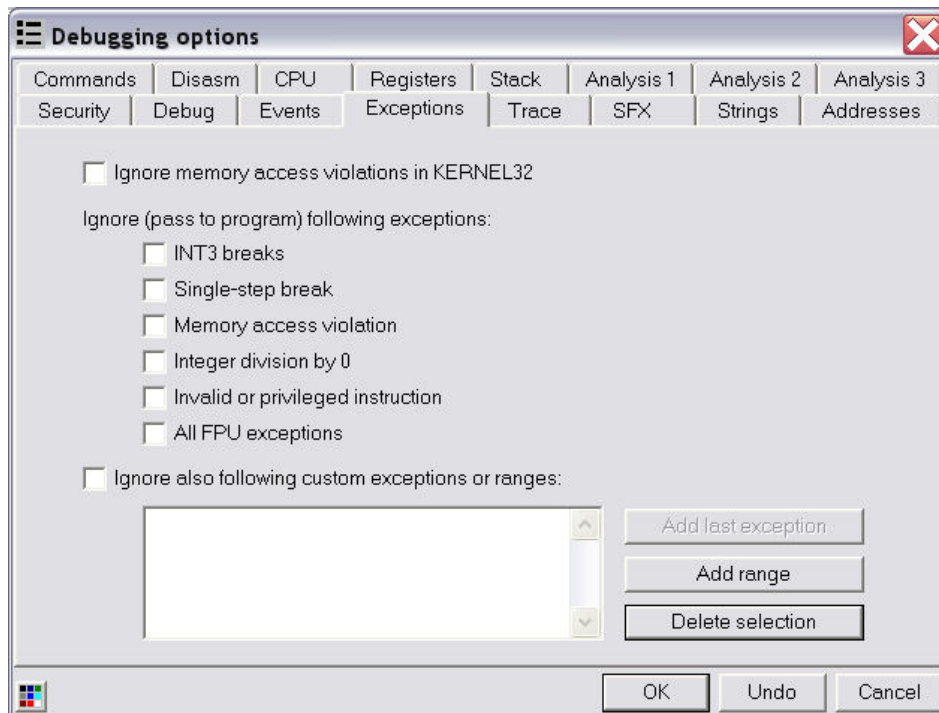


Figure 6 - Debugging option settings

You should also activate any of the available plugins to hide the Olly's presence to debugged programs, for this tutorial IsDebuggerPresent plugin is enough, but there are other as well:

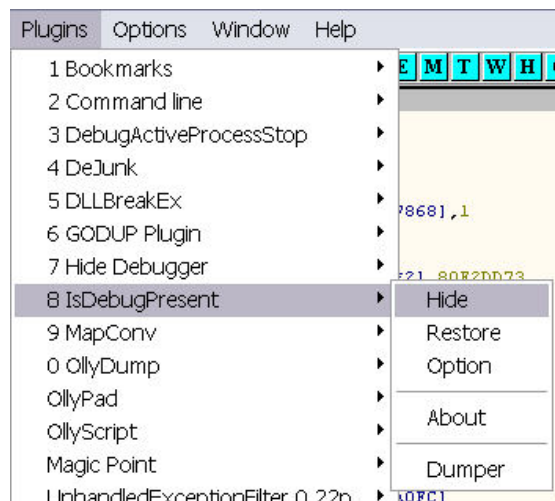


Figure 7 - IsDebuggerPresent activation



## Writing a Loader to patch applications protected with AsProtect 2.0, theory and practice.

Now we played around for too much, it is time to execute the target and see what happens! Press F9 and then SHIFT+F9 in order to pass all the exceptions to the program, until you are able to see the starting nag window (Figure 2, around 31 times on my PC). While you press SHIFT-F9 several times, try also to look at the code where OllyDbg stops, and especially try to take a look at the instruction just following the exception point: there are just two exceptions for which the code where stop contains an IRETD instruction<sup>1</sup>. The last exception makes you stopping just at the second IRETD; press SHIFT-F9 another time and you should see the nag screen.

00B39BF7	0156 00	ADD DWORD PTR DS:[ESI],EDX	
00B39BFA	CF	IRETD	
00B39BFB	67:3D D20AFDA7	CMP EAX,A7FD0AD2	Superfluous prefix
00B39C01	66:67:64:8F06 0000	POP WORD PTR FS:[0]	

Figure 8 - last exception. SHIFT-F9 here and the nag appears. Note the IRETD instruction, at EIP+2

Now, the nag screen is shown, and then you should press the pause button in OllyDbg and ... do not use the call stack, it's useless. Use instead the stack to find from where we are coming, who's the caller of the point where we landed.

0012F90C	00000001	
0012F910	003B7118	ASCII "mailto:support@deep-software.co
0012F914	0049B9DC	dla.0049B9DC
0012F918	280A0DEF	
0012F91C	0049B9DC	dla.0049B9DC

If you scroll down a bit the stack window you should see something like here beside shown. The string is one of the strings appearing in the nag, so we are browsing into the nag screen

building up stack. Well!

0012F910	003B7118	ASCII "mailto:support@deep-software.co
0012F914	0049B9DC	dla.0049B9DC
0012F918	280A0DEF	
0012F91C	0049B9DC	dla.0049B9DC
0012F920	960A0ED6	
0012F924	0075062B	
0012F928	7C165B24	mfc71.7C165B24

If you enlarge a little more the view of the stack, as beside, you should also understand that we are dealing with an application which was originally coded using MFC: there are several calls to MFC methods (exports of the

MFC library your system is using, version 7.1 on my PC).

0012FAF0	0012FEB8	Pointer to next SEH record
0012FAF4	0047F14B	SE handler
0012FAF8	00000000	
0012FAFC	00401733	RETURN to dla.00401733 from dla.0045C060

Good to know! Now scroll down again the stack window till you can see the returning address, stored of course on the stack. It's easily visible because OllyDbg is so kind to

mark it with red colour. The press enter on this value or use CTRL-G on the CPU view and go to address 00401733 (this an address which is into the .CODE section)

00401720	. E8 FB1B0000	CALL dla.00403320	
00401725	. 85C0	TEST EAX,EAX	
00401727	. 0F84 66030000	JE dla.00401A93	
0040172D	. FF15 88824B00	CALL DWORD PTR DS:[4B8288]	dla.0045C060
00401733	. 391D 34984B00	CMP DWORD PTR DS:[4B9834],EBX	
00401739	. 0F85 54030000	JNZ dla.00401A93	
0040173F	. E8 843E0700	CALL dla.004755C8	JMP to mfc71.Ordinal11126
00401744	. 85C0	TEST EAX,EAX	
00401746	. 75 0F	JNZ SHORT dla.00401757	
00401748	. 6A FF	PUSH -1	
0040174A	. 53	PUSH EBX	
0040174B	. 6A 64	PUSH 64	

Figure 9 - calling address of the nag routine

You should land at the point shown in Figure 9, if it doesn't look like press CTRL-A to re-analyze the code.

<sup>1</sup> The IRETD is simple to view, also because usually is shown different from the surrounding instructions by Olly..





## Writing a Loader to patch applications protected with AsProtect 2.0, theory and practice.

Now press CTRL+F2 to restart the process in OllyDbg (also remember to activate IsDebuggerpresent if you're using it). Press F9 and then SHIFT+F9 until you're able to reach the second exception followed by one IRETD instruction now press CTRL+G and write 00401733 and place a breakpoint (F2) into the 0040172D (instruction before) and then press SHIFT+F9 till you bang in the breakpoint. OllyDbg stops on our breakpoint: write down the CALL address contained at the address 004B8288, it is 0045C060, as shown in Figure 10.

00401725	85C0	TEST EAX,EAX	
00401727	0F84 66030000	JF dla.00401A93	
0040172D	FF15 88824B00	CALL DWORD PTR DS:[4B8288]	dla.0045C060
00401733	391D 34984B00	CMP DWORD PTR DS:[4B9834],EBX	
00401739	0F85 54030000	JNZ dla.00401A93	
0040173F	E8 843E0700	CALL dla.004755C8	JMP to mfc71.Ordinal1126
00401744	85C0	TEST EAX,EAX	
00401746	75 0F	JNZ SHORT dla.00401757	
00401748	6A FF	PUSH -1	

Figure 10 - The call which leads to the nag screen

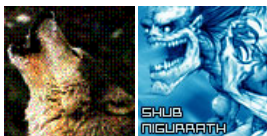
EAX	00000001
ECX	0012FE9C
EDX	003B0608
EBX	00000000
ESP	0012FAE0
EBP	0012FEAC
ESI	004B9868 dla.004B9868
EDI	004B9868 dla.004B9868
EIP	0040172D dla.0040172D

You should already have noticed that just one instruction above there is a conditional jump to the 00401A93 address, this jump is as a result of TEST EAX,EAX instruction which is in 00401725. And by observing the Registers you see EAX is equal to 1. A possible patch can be fixing the JMP here or the value of EAX, but this time we are going to do something different, patching at the deepest level possible.

Consider that this routine starts at 004015E0 (Figure 11) and the conditional JMP that lead me here is at 00401648 (Figure 12).

004015DF	CC	INT3	
004015E0	55	PUSH EBP	
004015E1	8BEC	MOV EBP,ESP	
004015E3	83E4 F8	AND ESP,FFFFFFF8	
004015E6	6A FF	PUSH -1	
004015E8	68 38584800	PUSH dla.00485838	SE handler installation

Figure 11 - where the routine starts



## Writing a Loader to patch applications protected with AsProtect 2.0, theory and practice.

00401648	OF84 A8000000	JE <b>dla.004016F6</b>	
0040164E	8D4C24 10	LEA ECX,DWORD PTR SS:[ESP+10]	
00401652	FF15 549B4800	CALL DWORD PTR DS:[489B54]	mfc71.Ordinal310
00401658	8D4C24 1C	LEA ECX,DWORD PTR SS:[ESP+1C]	
0040165C	FF15 549B4800	CALL DWORD PTR DS:[489B54]	mfc71.Ordinal310
00401662	6A 69	PUSH 69	
00401664	8D4C24 20	LEA ECX,DWORD PTR SS:[ESP+20]	
00401668	C68424 C8030000 0	MOV BYTE PTR SS:[ESP+3C8],2	
00401670	FF15 049B4800	CALL DWORD PTR DS:[489B04]	mfc71.Ordinal4035
00401676	68 FC000000	PUSH 0FC	
0040167B	8D4C24 14	LEA ECX,DWORD PTR SS:[ESP+14]	
0040167F	FF15 049B4800	CALL DWORD PTR DS:[489B04]	mfc71.Ordinal4035
00401685	8B4C24 1C	MOV ECX,DWORD PTR SS:[ESP+1C]	dla.004B9868
00401689	8B5424 10	MOV EDX,DWORD PTR SS:[ESP+10]	
0040168D	6A 21	PUSH 21	
0040168F	51	PUSH ECX	
00401690	52	PUSH EDX	
00401691	53	PUSH EBX	
00401692	E8 69E99800	CALL 00D90000	
00401697	92	XCHG EAX,EDX	
00401698	83F8 01	CMP EAX,1	
0040169B	75 40	JNZ SHORT <b>dla.004016DD</b>	
0040169D	8D4C24 14	LEA ECX,DWORD PTR SS:[ESP+14]	
004016A1	FF15 549B4800	CALL DWORD PTR DS:[489B54]	mfc71.Ordinal310
004016A7	68 B2000000	PUSH 0B2	
004016AC	8D4C24 18	LEA ECX,DWORD PTR SS:[ESP+18]	
004016B0	C68424 C8030000 0	MOV BYTE PTR SS:[ESP+3C8],3	
004016B8	FF15 049B4800	CALL DWORD PTR DS:[489B04]	mfc71.Ordinal4035
004016BE	8B4424 14	MOV EAX,DWORD PTR SS:[ESP+14]	
004016C2	6A 01	PUSH 1	
004016C4	53	PUSH EBX	
004016C5	53	PUSH EBX	
004016C6	50	PUSH EAX	
004016C7	68 640E4900	PUSH <b>dla.00490E64</b>	ASCII "open"
004016CC	53	PUSH EBX	
004016CD	E8 2EE99800	CALL 00D90000	
004016D2	CC	INT3	
004016D3	8D4C24 14	LEA ECX,DWORD PTR SS:[ESP+14]	
004016D7	FF15 5C9B4800	CALL DWORD PTR DS:[489B5C]	mfc71.Ordinal577
004016DD	8D4C24 1C	LEA ECX,DWORD PTR SS:[ESP+1C]	
004016E1	FF15 5C9B4800	CALL DWORD PTR DS:[489B5C]	mfc71.Ordinal577
004016E7	8D4C24 10	LEA ECX,DWORD PTR SS:[ESP+10]	
004016EB	FF15 5C9B4800	CALL DWORD PTR DS:[489B5C]	mfc71.Ordinal577
004016F1	E9 9D030000	JMP <b>dla.00401A93</b>	
004016F6	385C24 44	CMP BYTE PTR SS:[ESP+44],BL	
004016FA	74 17	JE SHORT <b>dla.00401713</b>	
004016FC	6A 01	PUSH 1	
004016FE	8D8E 20010000	LEA ECX,DWORD PTR DS:[ESI+120]	
00401704	C786 1C010000 010	MOV DWORD PTR DS:[ESI+11C],1	
0040170E	E8 3D420300	CALL <b>dla.00435950</b>	
00401713	FF15 848B4800	CALL DWORD PTR DS:[488B84]	[InitCommonControls
00401719	8BCE	MOV ECX,ESI	dla.004B9868
0040171B	E8 A23E0700	CALL <b>dla.004755C2</b>	JMP to mfc71.Ordinal3830
00401720	E8 FB1B0000	CALL <b>dla.00403320</b>	
00401725	85C0	TEST EAX,EAX	
00401727	OF84 66030000	JE <b>dla.00401A93</b>	
0040172D	FF15 88824B00	CALL DWORD PTR DS:[4B8288]	dla.0045C060

Figure 12 - The whole routine and the place for the following breakpoint

In order to get a little more accustomed to what the program does, hit CTRL+F2 (restart), reach the second exception (first instruction after one pointed by EIP is an IRETD) and place a breakpoint on 00401648, OllyDbg will stop here, now step over instructions using F8 and go into the 0045C060 call, using F7.



## Writing a Loader to patch applications protected with AsProtect 2.0, theory and practice.

0045C060	6A FF	PUSH -1	SE handler installation
0045C062	68 4BF14700	PUSH d1a.0047F14B	
0045C067	64:A1 00000000	MOV EAX,DWORD PTR FS:[0]	ntdll.RtlRaiseException
0045C06D	50	PUSH EAX	
0045C06E	64:8925 00000000	MOV DWORD PTR FS:[0],ESP	
0045C075	81EC C8020000	SUB ESP,2C8	
0045C07B	A1 84824B00	MOV EAX,DWORD PTR DS:[4B8284]	DS:[4B8284]=00000018 -> 24 days before trial expiration
0045C080	6A 00	PUSH 0	EAX hold the total days count
0045C082	50	PUSH EAX	ntdll.RtlRaiseException
0045C083	8D4C24 08	LEA ECX,DWORD PTR SS:[ESP+8]	
0045C087	51	PUSH ECX	
0045C088	E8 73000000	CALL d1a.0045C100	
0045C08D	8D0C24	LEA ECX,DWORD PTR SS:[ESP]	
0045C090	C78424 D0020000 000	MOV DWORD PTR SS:[ESP+2D0],0	
0045C09B	E8 94950100	CALL d1a.00475634	JMP to mfc71.Ordinal2020 which shows the nag window
0045C0A0	83F8 02	CMP EAX,2	continue evaluation -> EAX=1
0045C0A3	75 0A	JNZ SHORT d1a.0045C0AF	jmp is taken
0045C0A5	C705 34984B00 01000	MOV DWORD PTR DS:[4B9834],1	
0045C0AF	8D0C24	LEA ECX,DWORD PTR SS:[ESP]	
0045C0B2	C78424 D0020000 FFF	MOV DWORD PTR SS:[ESP+2D0],-1	
0045C0BD	E8 DE000000	CALL d1a.0045C1A0	
0045C0C2	8B8C24 C8020000	MOV ECX,DWORD PTR SS:[ESP+2C8]	
0045C0C9	64:890D 00000000	MOV DWORD PTR FS:[0],ECX	
0045C0D0	81C4 D4020000	ADD ESP,2D4	
0045C0D6	C3	RETN	

Figure 13 - Call body at 0045C060 with comments

Step the call, Figure 13 has also some comments. Note that the value at [4B8284] is equal to 0x18 (I'm my case, because I started to patch the same day of installation), if you move the clock this value will change. So, 0x18 means 24 days before trial expiration, oh nice stuff, we have found the days counter!

Ok, so we know that the information about the number of residual days is stored at [4B8284], so we will restart the program using CTRL-F2 and at the entrypoint will just set a memory breakpoint on write at location [4B8284] (Figure 14).

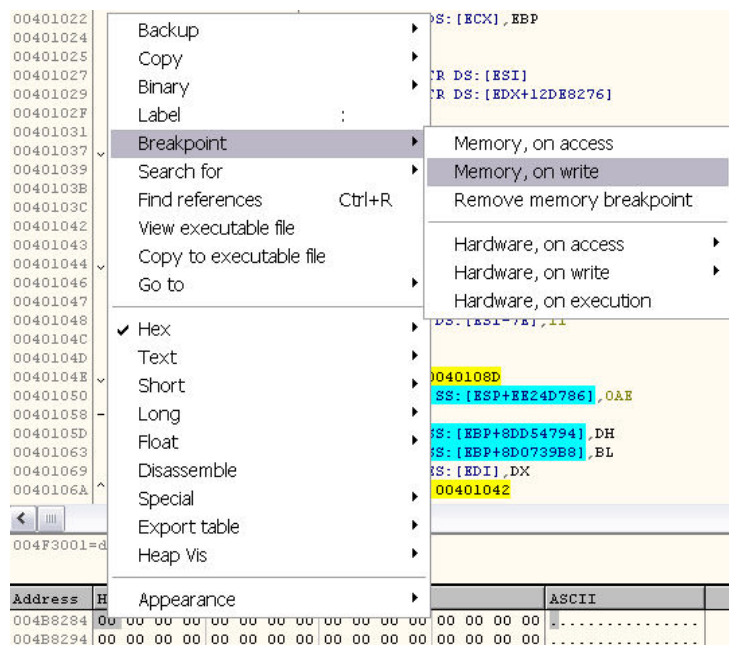


Figure 14 - How to set a memory breakpoint on write

Then press F9 and also SHIFT+F9 until OllyDbg breaks on our memory write address. Note that there is a first breakpoint, our address is filled by FFFFFFFF, skip it and press again SHIFT+F9:

004B8284	FF FF FF FF	F0 BF 45 00	40 FC 48 00	F0 FA 48 00	FF FF FF FF	E. GÜH. GüH.
004B8294	01 01 00 00	1E 01 00 00	0F 00 00 00	C0 00 49 00	FF FF FF FF	rr...r...W...A.I.
004B82A4	68 FB 48 00	00 00 00 00	1E 00 00 00	0F 00 00 00	hÜH.....W...	
004B82B4	00 00 00 00	E0 FE 48 00	00 00 00 00	13 00 00 00	....àÜH.....!!...	

Now we land in the real place (Figure 15):





0045BFCF	CC	INT3	
0045BFD0	8B4424 04	MOV EAX,DWORD PTR SS:[ESP+4]	
0045BFD4	8B4C24 08	MOV ECX,DWORD PTR SS:[ESP+8]	
0045BFD8	A3 80824B00	MOV DWORD PTR DS:[4B8280],EAX	
0045BFDD	890D 84824B00	MOV DWORD PTR DS:[4B8284],ECX	

Registers (FPU)  
EAX 00000019  
ECX 00000018  
EDX 0012FE64  
EBX 00BB0F75

Figure 15 - Place where address [4B8284] is filled with left trial days. Note register's value

The code where we landed is crystal clear! EAX is equal to 0x19 (25 decimal) which is the total days of our trial and ECX is equal to 0x18 (24 decimal) hence our residual days before trial expiration. To keep the residual days equal to the total full trial days we have just to make these changes:

- MOV to [ESP+8] the value in EAX
- MOV to [4B8284] the value in EAX

To do this you must just apply patch of Figure 16:

0045BFCF	CC	INT3	
0045BFD0	8B4424 04	MOV EAX,DWORD PTR SS:[ESP+4]	
0045BFD4	894424 08	MOV DWORD PTR SS:[ESP+8],EAX	
0045BFD8	A3 80824B00	MOV DWORD PTR DS:[4B8280],EAX	
0045BFDD	A3 84824B00	MOV DWORD PTR DS:[4B8284],EAX	
0045BFE2	90	NOP	
0045BFE3	C2 0800	RETN 8	

Figure 16 - Patch to fix left days equal to trial days

#### NOTE

Figure 15 and Figure 16 shows a red code, as Olly does for modified code. This is because the code has been written here by AsProtect and Olly has been driven to believe that this is a patch, because were not here at the program's beginning. It's just cosmetics anyway.

To check our patching without the need for restart OllyDbg just change the EIP to 0045BFD0 (point with your mouse the 0045BFD0 instruction, right-click and choose the "New origin here" option).

Summarizing so these are the patches we did till now (included in this archive as patch1.txt):

Address	Original	Patched
0045BFD4	8B	89
0045BFD5	4C	44
0045BFDD	89	A3
0045BFDE	0D	84
0045BFDF	84	82
0045BFE2	C2	90
0045BFE3		C2
0045BFE4		08
0045BFE5		00

Table 1 – Patch 1 data



To save all the patches you did there's an alternative (faster) way:

1. select in the CPU view the patched code
2. right click and select copy-> to file and save the file
3. use OllyDumpTranslator later on when we will build the loader (see [2], included in this archive as patch1\_translated.txt)

Figure 17 shows the result of the patch we just did. To see it just press SHIFT-F9 to run the program freely. We already taken note of all the required patches we have to do.

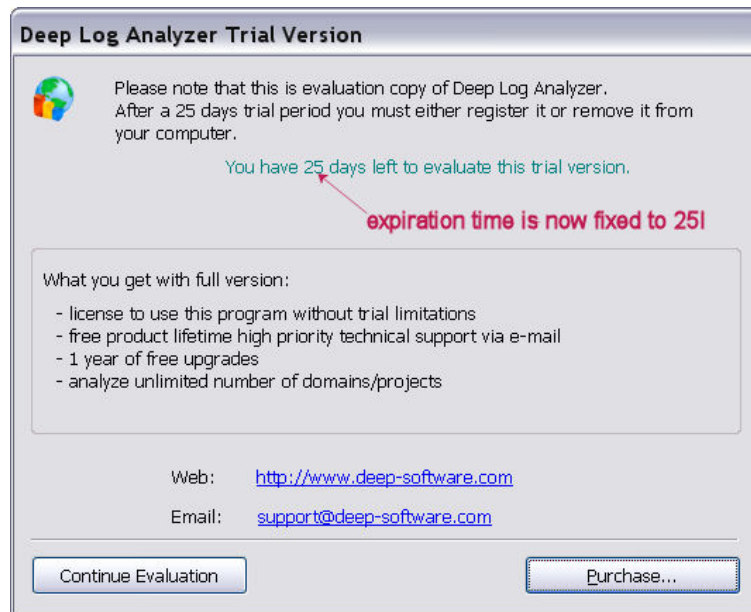


Figure 17 - Result of patching the left trial days

At this point we have successfully found the patching point but we have another important task to do, relate to trial time expiration: the program once the trial time is over shows another nag, which we have to defeat also.

So change the PC clock to the following year or to a date distant more than 25 days from now.

Do you remember what we found in Figure 10? Proceed with the following steps:

1. restart the process, using CTRL-F2
2. press F9 one time and then SHIFT-F9 till you reach the second exception where the instruction just after the one pointed by EIP is IRETD
3. go to the address 0040172D and press F2 to set a breakpoint.
4. press SHIFT-F9 one time more and you should land at the breakpoint

This time the call of Figure 10 points to another call; now points to 0045BFF0 (Figure 18).

This time the simplest patch to do is to just modify the PUSH -1 at 0045BFF0 into a JMP to the previous nag handling routine, we were jumping when the application was not expired.



## Writing a Loader to patch applications protected with AsProtect 2.0, theory and practice.

0045BFF0	6A FF	PUSH -1	
0045BFF2	68 4BF14700	PUSH <a href="#">dla.0047F14B</a>	SE handler installation
0045BFF7	64:A1 00000000	MOV EAX,DWORD PTR FS:[0]	
0045BFFD	50	PUSH EAX	
0045BFFE	64:8925 00000000	MOV DWORD PTR FS:[0],ESP	
0045C005	81EC C8020000	SUB ESP,2C8	
0045C00B	6A 00	PUSH 0	
0045C00D	6A 00	PUSH 0	
0045C00F	8D4424 08	LEA EAX,DWORD PTR SS:[ESP+8]	
0045C013	50	PUSH EAX	
0045C014	E8 E7000000	CALL <a href="#">dla.0045C100</a>	
0045C019	8D0C24	LEA ECX,DWORD PTR SS:[ESP]	
0045C01C	C78424 D0020000 0	MOV DWORD PTR SS:[ESP+2D0],0	
0045C027	E8 08960100	CALL <a href="#">dla.00475634</a>	JMP to mfc71.Ordinal2020
0045C02C	8D0C24	LEA ECX,DWORD PTR SS:[ESP]	
0045C02F	C705 34984B00 010	MOV DWORD PTR DS:[4B9834],1	
0045C039	C78424 D0020000 F	MOV DWORD PTR SS:[ESP+2D0],-1	
0045C044	E8 57010000	CALL <a href="#">dla.0045C1A0</a>	
0045C049	8B8C24 C8020000	MOV ECX,DWORD PTR SS:[ESP+2C8]	
0045C050	64:890D 00000000	MOV DWORD PTR FS:[0],ECX	
0045C057	81C4 D4020000	ADD ESP,2D4	
0045C05D	C3	RETN	

Figure 18 - Routine handling the nag when the application is expired

So the patch looks like in Figure 19.

0045BFF0	EB 6E	JMP SHORT <a href="#">dla.0045C060</a>	
0045BFF2	68 4BF14700	PUSH <a href="#">dla.0047F14B</a>	SE handler installation
0045BFF7	64:A1 00000000	MOV EAX,DWORD PTR FS:[0]	
0045BFFD	50	PUSH EAX	
0045BFFE	64:8925 00000000	MOV DWORD PTR FS:[0],ESP	
0045C005	81EC C8020000	SUB ESP,2C8	
0045C00B	6A 00	PUSH 0	
0045C00D	6A 00	PUSH 0	
0045C00F	8D4424 08	LEA EAX,DWORD PTR SS:[ESP+8]	
0045C013	50	PUSH EAX	
0045C014	E8 E7000000	CALL <a href="#">dla.0045C100</a>	
0045C019	8D0C24	LEA ECX,DWORD PTR SS:[ESP]	
0045C01C	C78424 D0020000 0	MOV DWORD PTR SS:[ESP+2D0],0	
0045C027	E8 08960100	CALL <a href="#">dla.00475634</a>	JMP to mfc71.Ordinal2020
0045C02C	8D0C24	LEA ECX,DWORD PTR SS:[ESP]	
0045C02F	C705 34984B00 010	MOV DWORD PTR DS:[4B9834],1	
0045C039	C78424 D0020000 F	MOV DWORD PTR SS:[ESP+2D0],-1	
0045C044	E8 57010000	CALL <a href="#">dla.0045C1A0</a>	
0045C049	8B8C24 C8020000	MOV ECX,DWORD PTR SS:[ESP+2C8]	
0045C050	64:890D 00000000	MOV DWORD PTR FS:[0],ECX	
0045C057	81C4 D4020000	ADD ESP,2D4	
0045C05D	C3	RETN	
0045C05E	CC	INT3	
0045C05F	CC	INT3	
0045C060	6A FF	PUSH -1	
0045C062	68 4BF14700	PUSH <a href="#">dla.0047F14B</a>	SE handler installation

Figure 19 - Patched routine handling the nag when the application is expired

Summarizing so these are the other patches we did till now:

Address	Original	Patched
0045BFF0	6A	EB
0045BFF1	FF	6E

Table 2 - Patch 2 data

Again you can use the OllyDumpTranslator to save the patches (included in this archive as patch2.txt)



## 5. Cracking stage part # 2 - Nag removing

What is left is to remove the starting nag screen. At this stage is quite simple, restart the process in OllyDbg and then stop at the last exception where there's the second IRETD.

Now place a breakpoint on 0045C09B (see Figure 13) and let the target run. When OllyDbg stops in our breakpoint just apply the patch as in Figure 20: skip the nag call, store the right value into EAX register, avoid jmp.

0045C07B	. A1 84824B00	MOV EAX,DWORD PTR DS:[4B8284]	DS:[4B8284]=00000018 -> 24 days before trial expiration
0045C080	. 6A 00	PUSH 0	EAX hold the total days count
0045C082	. 50	PUSH EAX	
0045C083	. 8D4C24 08	LEA ECX,DWORD PTR SS:[ESP+8]	
0045C087	. 51	PUSH ECX	
0045C088	. E8 73000000	CALL <a href="#">dia.0045C100</a>	
0045C08D	. 8D0C24	LEA ECX,DWORD PTR SS:[ESP]	
0045C090	. C78424 D0020000 00000000	MOV DWORD PTR SS:[ESP+2D0],0	
0045C09B	. 90	NOP	JMP to mfc71.Ordinal2020 which shows the nag window
0045C09C	. 90	NOP	
0045C09D	. 90	NOP	
0045C09E	. 90	NOP	
0045C09F	. 90	NOP	
0045C0A0	. B8 01000000	MOV EAX,1	continue evaluation -> EAX=1
0045C0A5	. 90	NOP	
0045C0A6	. 90	NOP	
0045C0A7	. 90	NOP	
0045C0A8	. 90	NOP	
0045C0A9	. 90	NOP	
0045C0AA	. 90	NOP	
0045C0AB	. 90	NOP	
0045C0AC	. 90	NOP	
0045C0AD	. 90	NOP	
0045C0AE	. 90	NOP	
0045C0AF	. > 8D0C24	LEA ECX,DWORD PTR SS:[ESP]	
0045C0B2	. C78424 D0020000 FFFFFFFF	MOV DWORD PTR SS:[ESP+2D0],-1	

Figure 20 - Nag routine patched to not show the nag

Summarizing so these are the new patches to add to the previous (patch3.txt):

Address	Original	Patched
0045C09B	E8	90
0045C09C	94	90
0045C09D	95	90
0045C09E	01	90
0045C09F	00	90
0045C0A0	83	B8
0045C0A1	F8	01
0045C0A2	02	00
0045C0A3	75	00
0045C0A4	0A	00
0045C0A5	C7	90
0045C0A6	05	90
0045C0A7	34	90
0045C0A8	98	90
0045C0A9	4B	90
0045C0AA	00	90
0045C0AB	01	90
0045C0AC	00	90
0045C0AD	00	90
0045C0AE	00	90
0045C0AF	00	90

Table 3 - Patch 3 data





## 6. Writing a debugger loader for the program, using Shub-Nigurrath's framework

Well, we learned how to patch the application inside Olly, but indeed we haven't unpacked it yet, so there's a need of a way to do these patches automatically and eventually distribute the patched version. The faster instrument for this purpose is a debug loader (see [1]) which, thanks to the framework we developed is really fast to write. Otherwise you would have to manually unpack the program and then distribute the unpacked exe, which usually might be hard and for sure enlarge your distribution.

### 6.1. Resemble the patches

In previous sections we already wrote into tables the patches we did, but as told there's a simpler instrument we developed. OllyDumpTranslator (see [1], [2]) automatically passes from txt dump files (produced by Olly using the contextual menu Copy -> To File) to C code snippets you can use to write a loader.

The situation for the patches we did is the following one:

Patches for Patch 1 (see Table 1)

```
0045BFD4  89 44 24 08 A3 80 82 4B 00 A3 84 82 4B 00 90 C2  %D$□,€,K.,.,K. ¨ á
0045BFE4  08 00
```

```
<-----Code Snippet----->
// =====
// Olly File Translator 1.0 by ThunderPwr
// 08/06/2005 1.04.59
// Translating file utility
// =====
```

```
#define IMAXINDEXINJ 18// Patch size
```

```
// -----
// Definition about the addresses where to apply the patches.
// -----
```

```
DWORD dwPatchaddrInj[IMAXINDEXINJ] = { 0x0045BFD4, 0x0045BFD5, 0x0045BFD6, 0x0045BFD7,
                                         0x0045BFD8, 0x0045BFD9, 0x0045BFDA, 0x0045BFDB,
                                         0x0045BFDC, 0x0045BFDD, 0x0045BFDE, 0x0045BFDF,
                                         0x0045BFE0, 0x0045BFE1, 0x0045BFE2, 0x0045BFE3,
                                         0x0045BFE4, 0x0045BFE5 };
```

```
// -----
// Definition about the patching value
// -----
```

```
int iPatchDataInj[IMAXINDEXINJ] = { 0x89, 0x44, 0x24, 0x08,
                                     0xA3, 0x80, 0x82, 0x4B,
                                     0x00, 0xA3, 0x84, 0x82,
                                     0x4B, 0x00, 0x90, 0xC2,
                                     0x08, 0x00 };
```

```
<-----End Code Snippet----->
```

Patches for Patch 2 (see Table 2)

```
0045BFF0  EB 6E                                     ěn
```

```
<-----Code Snippet----->
#define IMAXINDEXINJ 2// Patch size
```

```
// -----
// Definition about the addresses where to apply the patches.
// -----
```

```
DWORD dwPatchaddrInj[IMAXINDEXINJ] = { 0x0045BFF0, 0x0045BFF1 };
```



## Writing a Loader to patch applications protected with AsProtect 2.0, theory and practice.

```
// -----  
// Definition about the patching value  
// -----  
int iPatchDataInj[IMAXINDEXINJ] = { 0xEB, 0x6E };  
<-----End Code Snippet----->
```

### Patches for Patch 3 (see Table 3)

```
0045C09B  90 90 90 90 90 B8 01 00 00 00 90 90 90 90 90 90  ÉÉÉÉÉ+Γ...ÉÉÉÉÉÉ  
0045C0AB  90 90 90 90  ÉÉÉÉ
```

```
<-----Code Snippet----->  
#define IMAXINDEXINJ 20// Patch size
```

```
// -----  
// Definition about the addresses where to apply the patches.  
// -----
```

```
DWORD dwPatchaddrInj[IMAXINDEXINJ] = { 0x0045C09B, 0x0045C09C, 0x0045C09D, 0x0045C09E,  
                                         0x0045C09F, 0x0045C0A0, 0x0045C0A1, 0x0045C0A2,  
                                         0x0045C0A3, 0x0045C0A4, 0x0045C0A5, 0x0045C0A6,  
                                         0x0045C0A7, 0x0045C0A8, 0x0045C0A9, 0x0045C0AA,  
                                         0x0045C0AB, 0x0045C0AC, 0x0045C0AD, 0x0045C0AE  
                                         };
```

```
// -----  
// Definition about the patching value  
// -----
```

```
int iPatchDataInj[IMAXINDEXINJ] = { 0x90, 0x90, 0x90, 0x90,  
                                     0x90, 0xB8, 0x01, 0x00,  
                                     0x00, 0x00, 0x90, 0x90,  
                                     0x90, 0x90, 0x90, 0x90,  
                                     0x90, 0x90, 0x90, 0x90  
                                     };
```

```
<-----End Code Snippet----->
```

Sources are included into this tutorial's archive, so I'm going to comment the LoaderActions.cpp file, which according to what I explained in [1] is the only file I have modified to create the loader for this particular program.

First of all you have to insert the patch data into the InitializePatchStack method.

```
<-----Code Snippet----->
```

```
BOOL Loader::InitializePatchStack(growing_arraystack<Patch> &stkPatches)  
{
```

```
    //NB 0x00 must explicitly converted to BYTE because otherwise the compiler confuses  
    //it with a NULL pointer and doesn't know which constructor of class Patch to use.
```

```
    DWORD dwPatchaddrInj1[IMAXINDEXINJ1] = { 0x0045BFD4, 0x0045BFD5, 0x0045BFD6, 0x0045BFD7,  
                                                0x0045BFD8, 0x0045BFD9, 0x0045BFDA, 0x0045BFDB,  
                                                0x0045BFDC, 0x0045BFDD, 0x0045BFDE, 0x0045BFDF,  
                                                0x0045BFE0, 0x0045BFE1, 0x0045BFE2, 0x0045BFE3,  
                                                0x0045BFE4, 0x0045BFE5 };
```

```
    int iPatchDataInj1[IMAXINDEXINJ1] = { 0x89, 0x44, 0x24, 0x08,  
                                           0xA3, 0x80, 0x82, 0x4B, 0x00, 0xA3, 0x84, 0x82,  
                                           0x4B, 0x00, 0x90, 0xC2, 0x08, 0x00  
                                           };
```

```
    DWORD dwPatchaddrInj2[IMAXINDEXINJ2] = { 0x0045C09B, 0x0045C09C, 0x0045C09D, 0x0045C09E,  
                                                0x0045C09F, 0x0045C0A0, 0x0045C0A1, 0x0045C0A2,  
                                                0x0045C0A3, 0x0045C0A4, 0x0045C0A5, 0x0045C0A6,  
                                                0x0045C0A7, 0x0045C0A8, 0x0045C0A9, 0x0045C0AA,  
                                                0x0045C0AB, 0x0045C0AC, 0x0045C0AD, 0x0045C0AE  
                                                };
```

```
    int iPatchDataInj2[IMAXINDEXINJ2] = { 0x90, 0x90, 0x90, 0x90,  
                                           0x90, 0xB8, 0x01, 0x00,  
                                           0x00, 0x00, 0x90, 0x90,  
                                           0x90, 0x90, 0x90, 0x90,  
                                           0x90, 0x90, 0x90, 0x90  
                                           };
```

```
    DWORD dwPatchaddrInj3[IMAXINDEXINJ3] = { 0x0045BFF0, 0x0045BFF1 };  
    int iPatchDataInj3[IMAXINDEXINJ3] = { 0xEB, 0x6E };
```



## Writing a Loader to patch applications protected with AsProtect 2.0, theory and practice.

```
int idx=0;
for (idx=0; idx<IMAXINDEXINJ1; idx++)
    stkPatches.push(Patch(dwPatchaddrInj1[idx], (BYTE)iPatchDataInj1[idx]));

for (idx=0; idx<IMAXINDEXINJ2; idx++)
    stkPatches.push(Patch(dwPatchaddrInj2[idx], (BYTE)iPatchDataInj2[idx]));

for (idx=0; idx<IMAXINDEXINJ3; idx++)
    stkPatches.push(Patch(dwPatchaddrInj3[idx], (BYTE)iPatchDataInj3[idx]));

return TRUE;
}

<-----End Code Snippet----->
```

What we are creating is then a Blind Debugger Loader, according to definitions given in [1].

### 6.2. Set the Victim Details

We are going to set the details of the victim and its CRC value (we used the utility distributed with [1]).

```
<-----Code Snippet----->
BOOL Loader::SetVictimDetails(TextString &victimFileName)
{
#ifdef _DEBUG
    victimFileName=TextString(".\\_dla.exe");
#else
    victimFileName=TextString("C:\\Program Files\\Deep Log Analyzer\\_dla.exe");
#endif

    //Set this parameter to true when you want the loader to check the CRC of the file!
    SetVictimCRC(0x34684ef4);

    SetCreateProcessFlags(DEBUG_PROCESS | DEBUG_ONLY_THIS_PROCESS | CREATE_NEW_CONSOLE);

    return TRUE;
}

<-----End Code Snippet----->
```

#### NOTE

As you can see here, the victim's exe name in release mode is \_dla.exe in the same folder where the loader is copied. So you will have to rename the original exe file according, before installing the loader. Doing this way the shortcuts the target's installation process built will work calling the loader instead of the original exe file.

### 6.3. Write the GateCondition

The GateCondition is the most complex, but it has nothing special with the one described in [1], the only think is what we said about the IRETD instruction that is used by the loader as the test to start the patching.

```
<-----Code Snippet----->
BOOL Loader::GateProcedure()
{
    BOOL bRet=FALSE;

    DEBUG_EVENT DebugEv; // debugging event information
    DWORD dwContinueStatus = DBG_CONTINUE; // exception continuation
    // Define the CONTEXT structure used to load the victim process context
    // when debugged process break due to exception event
```



## Writing a Loader to patch applications protected with AsProtect 2.0, theory and practice.

---

```
CONTEXT victimContext;
int iExceptionCounter = 0;

//will hold the code pointed by the EIP, plus 4 bytes forward.
BYTE OridataRead[4];
int OridataExceptionCount=0;

//reset the buffer
memset(OridataRead,0,4*sizeof(BYTE));

try {
    for(;;)
    {
        // Wait for a debugging event to occur. The second parameter indicates
        // that the function does not return until a debugging event occurs.
        // We are waiting for infinite time, then wait for each Debug Event.
        WaitForDebugEvent(&DebugEv, INFINITE);

        // Process the debugging event code.
        switch (DebugEv.dwDebugEventCode)
        {
            case EXCEPTION_DEBUG_EVENT: {
                // Process the exception code. When handling
                // exceptions, remember to set the continuation
                // status parameter (dwContinueStatus). This value
                // is used by the ContinueDebugEvent function.

                // Increment exception counter (not used)
                iExceptionCounter++;

                #ifdef _DEBUG
                // Show the current exception number
                char str[256];
                sprintf(str,"Exception number %d", iExceptionCounter);
                ::MessageBox(NULL, str, DEFAULT_MSG_CAPTION, MB_OK);
                #endif

                // Check if this is the right exception by reading the context
                // structure for the victim process. Before to do it set the
                // ContextFlags to READ_ALL
                victimContext.ContextFlags = 0x1003F;

                // Fill the process CONTEXT with the process information
                GetThreadContext(GetPI()->hThread , &victimContext);

                // Now I've to scan the process memory in order to see
                // if I can find the IRETD instruction just after EIP.
                ReadProcessMemory(GetPI()->hProcess,
                    (LPVOID)((victimContext.Eip)), &OridataRead, 4, NULL);

                // We must stop at the second exception with an IRETD
                // instruction near EIP (see tutorial)
                // 0156 00  ADD DWORD PTR DS:[ESI], EDX
                // CF      IRETD
                if (OridataRead[0]==0x01 && OridataRead[1]==0x56 &&
                    OridataRead[2]==0x00 && OridataRead[3]==0xCF)
                {
                    OridataExceptionCount++;

                    if(OridataExceptionCount==NUMBER_OF_IRETD_TO_WAIT) {
                        // Key location found, now we can apply the
                        // patch
                        #ifdef _DEBUG
                        char str[256];
                        sprintf(str,"Found right IRETD location at %X",
                            (LPVOID)((victimContext.Eip)));
                        MessageBox(NULL, str, DEFAULT_MSG_CAPTION,
                            MB_OK);
                        #endif

                        throw TRUE;
                    }
                }

                // Exception handler
                switch(DebugEv.u.Exception.ExceptionRecord.ExceptionCode)
```





## Writing a Loader to patch applications protected with AsProtect 2.0, theory and practice.

---

```
{
    case EXCEPTION_ACCESS_VIOLATION: {
        // First chance: Pass this on to the system.
        // Last chance: Display an appropriate error.
        dwContinueStatus = DBG_EXCEPTION_NOT_HANDLED;
    }
    break;

    case EXCEPTION_BREAKPOINT: {
        // First chance: Display the current
        // instruction and register values.
    }
    break;

    case EXCEPTION_DATATYPE_MISALIGNMENT: {
        // First chance: Pass this on to the system.
        // Last chance: Display an appropriate error.
    }
    break;

    case EXCEPTION_SINGLE_STEP: {
        // First chance: Update the display of the
        // current instruction and register values.
    }
    break;

    case DBG_CONTROL_C: {
        // First chance: Pass this on to the system.
        // Last chance: Display an appropriate error.
    }
    break;

    default: {
        // Handle other exceptions.
    }
    break;
}

case CREATE_THREAD_DEBUG_EVENT: {
    // As needed, examine or change the thread's registers
    // with the GetThreadContext and SetThreadContext functions;
    // and suspend and resume thread execution with the
    // SuspendThread and ResumeThread functions.
}
break;

case CREATE_PROCESS_DEBUG_EVENT: {
    // As needed, examine or change the registers of the
    // process's initial thread with the GetThreadContext and
    // SetThreadContext functions; read from and write to the
    // process's virtual memory with the ReadProcessMemory and
    // WriteProcessMemory functions; and suspend and resume
    // thread execution with the SuspendThread and ResumeThread
    // functions. Be sure to close the handle to the process image
    // file with CloseHandle.

    dwContinueStatus = DBG_CONTINUE;
}
break;

case EXIT_THREAD_DEBUG_EVENT: {
    // Display the thread's exit code.
}
break;

case EXIT_PROCESS_DEBUG_EVENT: {
    // Target Process is closed from user, then we have
    // to stop the debugger work and exit from loader
    // Exit form loader
    ContinueDebugEvent(DebugEv.dwProcessId, DebugEv.dwThreadId,
        DBG_CONTINUE);
    throw FALSE;
}
break;
```



## Writing a Loader to patch applications protected with AsProtect 2.0, theory and practice.

```

        case LOAD_DLL_DEBUG_EVENT: {
            // Read the debugging information included in the newly
            // loaded DLL. Be sure to close the handle to the loaded DLL
            // with CloseHandle.
        }
        break;

        case UNLOAD_DLL_DEBUG_EVENT: {
            // Display a message that the DLL has been unloaded.
        }
        break;

        case OUTPUT_DEBUG_STRING_EVENT: {
            // Display the output debugging string.
        }
        break;
    }

    // Resume executing the thread that reported the debugging event.
    ContinueDebugEvent(DebugEv.dwProcessId, DebugEv.dwThreadId, dwContinueStatus);

} //end for(;;)

} //end try

catch(BOOL bRet) {
    return bRet;
}
}
<-----End Code Snippet----->

```

The most important part is the following one:

```

<-----Code Snippet----->
// Now I've to scan the process memory in order to see
// if I can find the IRETD instruction just after EIP.
ReadProcessMemory(GetPI()->hProcess, (LPVOID)((victimContext.Eip)), &OridataRead, 4, NULL);

// We must stop at the second exception with an IRETD
// instruction near EIP (see tutorial)
// 0156 00  ADD DWORD PTR DS:[ESI], EDX
// CF      IRETD
if (OridataRead[0]==0x01 && OridataRead[1]==0x56 && OridataRead[2]==0x00 && OridataRead[3]==0xCF)
{
    OridataExceptionCount++;

    if(OridataExceptionCount==NUMBER_OF_IRETD_TO_WAIT) {
        // Key location found, now we can apply the patch
#ifdef _DEBUG
        char str[256];
        sprintf(str, "Found right IRETD location at %X", (LPVOID)((victimContext.Eip)));
        MessageBox(NULL, str, DEFAULT_MSG_CAPTION, MB_OK);
#endif

        throw TRUE;
    }
}
<-----End Code Snippet----->

```

The define `NUMBER_OF_IRETD_TO_WAIT` is equal to 2, but after some experiments we found the first IRETD instruction is also good to do the patches (just to complete this we have also found this behaviour for other ASProtect 2.0x registered targets: IRETD can be used as suitable trigger point to apply your patch, only check about first IRETD and last IRETD before applying it). Actually I still use 2, but the define allows you to change this behaviour.

Another interesting thing is that the `ReadProcessMemory` and `WriteProcessMemory` APIs are not the real system APIs, but the methods exposed by the class `ShubLoaderCore` of the framework, which take care of all the controls to ensure reading and writing is allowed. This allows the writer of the



class Loader and of the method GateCondition, to skip all the controls to check if the reading/writing went fine.

We think that above code should be quite clear now..

## 6.4. Close the Loader and Hide the Debugger

As also explained in [1] we also worried to hide the debugger using the framework's method and to detach it from the program once the patch has been performed.

```
<-----Code Snippet----->
BOOL Loader::ActionsAfterGateProcedure()
{
    //Stop debugger action and let program run freely
    DWORD dwProcessId = GetProcessId(GetPI()->hProcess);
    BOOL bDbgStopFlag = DebugActiveProcessStop(dwProcessId);

    return TRUE;
}

//This function is called just before the process has been created but it is still in waiting mode
BOOL Loader::ActionsAfterCreateProc()
{
    HideDebugger(GetPI()->hThread, GetPI()->hProcess);
    return TRUE;
}
<-----End Code Snippet----->
```

Note that the `DebugActiveProcessStop` called by the function above is not the real Windows API (which is available only since Windows XP), but rather the method exposed by the class `NTInternals` (see [1]) which also ensure compatibility with Windows 9x/NT/2000. In those cases it will simply do nothing at all.

## 6.5. Run the Loader

If you compile the loader in Debug Mode the framework behaves differently being more interactive and telling you what's happening. Try to run on the target program and you would see:

1. an exception counter dialog for each exception matched
2. a messagebox telling where the program matched the right exception (you should note that the address is the same as in Figure 8).
3. a final messagebox reporting the applied patches, the address, the final error status (OK if all went fine).

If you compile the same thing in Release Mode the patcher will run silently<sup>2</sup> and detaches itself at the end, using `DebugActiveProcessStop`.

---

<sup>2</sup> Note that in release mode the Visual Studio project included has a post-build action that you might want to tune. The post built actions are commands issues after the build terminates. The command is:

"upx.exe" "\$(TargetPath)" -9 --force -q --compress-icons=1 --all-methods

which means that the program just after compiled is packed with UPX at it best.



## 7. References

- [1] “Cracking with Loaders, theory, methods and a framework”, Shub-Nigurrath, ThunderPwr of ARTeam, <http://tutorials.accessroot.com>
- [2] “OllyDumpTranslator”, ThunderPwr of ARTeam, <http://releases.accessroot.com>

## 8. Conclusions

Well this is the end of the story. AsProtected programs can in most cases be cracked simply using a smart loader which will wait for a specific condition before applying patches. This happens because most of the applications protected with AsProtect simply doesn't use its advanced anti-tampering techniques and limits the AsProtect's protection to the unpacking of the application. We hope you learnt something new and as usual have phun!

**All the code provided with this tutorial is free for public use, just make a greetz to the authors and the ARTeam if you find it useful to use. Don't use these concepts for making illegal operation, all the info here reported are only meant for studying and to help having a better knowledge of application code security techniques.**

## 9. History

- Version 1.0 – First public release!

## 10. Greetings

We wish to tanks all the ARTeam members of course and who read the beta versions of this tutorial,.. and of course you, who are still alive at the end of this quite long and complex document!



<http://cracking.accessroot.com>