2008

ARTeam eZine Issue III rev.1

Editor: Shub-Nigurrath

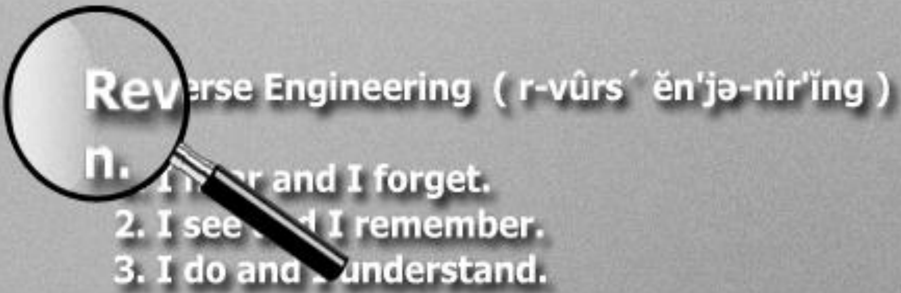
ARTeam

4/15/2008

ARTeam

REVERSE ENGINEERING MAGAZINE

Volume 1, Issue 3


Reverse Engineering (r-vûrs´ ĕn'jə-nîr'ĭng)
n.
1. I hear and I forget.
2. I see and I remember.
3. I do and I understand.

CODE INJECTION – 1CLICKDVDCOPYPRO BY CONDZERO

MUP ANYDVD V6.1.3.6 BY CONDZERO

PATCHING PRIMA EGUIDES (SINGLE BYTE PATCHING) BY SSLEVIN

EXAMDIFF 4.XXX REVERSING THE PROTECTION SCHEMA BY SHUB-NIGURRATH

REVERSING BUSINESS TRANSLATOR 9.00 BY KAIRA

EXECRYPTOR FOR DUMMIES OR HOW TO UNPACK EXECRYPTOR 2.4 WITHOUT HAVING A CLUE WHAT YOU ARE DOING BY HAGGAR

OCR TOOLS WALKTHROUGH OF KEY CHECK ROUTINE BY ANHS!RK

THE STRANGE CASE OF DBG_PRINTEXCEPTION_C & DBG_RIPEXCEPTION BY MOID

CRACKING FOR FUN BY ARJUNS

WRITING A SELF-KEYGENERATOR LOADER WITH ABEL BY M1SCH13F

TABLE OF CONTENTS

FOREWORDS 5

Disclaimer 6

Supplements 6

Verification 6

1 CODE INJECTION – 1CLICKDVDCOPYPRO BY CONDZERO 7

1.1 Introduction 7

1.2 Abstract 7

2 MUP ANYDVD V6.1.3.6 BY CONDZERO 13

2.1 Introduction 13

2.2 MUP AnyDVD v6.1.3.6 13

3 PATCHING PRIMA EGUIDES (SINGLE BYTE PATCHING) BY SSLEVIN 24

3.1 Introduction 24

3.2 Patching Prima eGuides 24

3.3 Inspecting the target 24

3.4 Finding patch(es) 26

3.5 Conclusions 28

4 EXAMDIFF 4.XXX REVERSING THE PROTECTION SCHEMA BY SHUB-NIGURRATH 29

4.1 Introduction 29

4.2 Approaching the enemy 29

4.3 Reversing the registration schema 30

4.4 Testing whole thing & Conclusions 36

5 REVERSING BUSINESS TRANSLATOR 9.00 BY KAIRA 37

5.1 Introduction 37

5.2 Approaching the Enemy 38

5.3 Conclusion 43

6 EXECRYPTOR FOR DUMMIES OR HOW TO UNPACK EXECRYPTOR 2.4 WITHOUT HAVING A CLUE WHAT YOU ARE DOING BY HAGGAR 44

6.1 Introduction 44

6.2 Content 44

6.3 [1] Requirements for this guide 44

6.4 [2] Preparations before loading target in Olly 45

6.5 [3] Loading target in Olly 46
6.6 [4] Using script to kill anti-debug tricks 46
6.7 [5] Finding OEP 47
6.8 [6] Using script to decrypt imports 50
6.9 [7] Dumping to hard disc 50
6.10 [9] Reference material 51
6.11 ScriptS 51
7 OCR TOOLS WALKTHROUGH OF KEY CHECK ROUTINE BY ANHS!RK 52
7.1 Introduction 52
7.2 Tool Required 52
7.3 Walkthrough 52
8 THE STRANGE CASE OF DBG_PRINTEXCEPTION_C & DBG_RIPEXCEPTION BY MOID 62
8.1 Introduction 62
8.2 Trick description 62
8.3 The cause 63
8.4 The solution 64
8.5 References 66
9 CRACKING FOR FUN BY ARJUNS 67
9.1 Some Theories 67
9.2 Diving into the scene 68
10 WRITING A SELF-KEYGENERATOR LOADER WITH ABEL BY M1SCH13F 71
10.1 Introduction 71
10.2 Examining the Application 72
10.3 Creating A Loader In ABEL 76
ARTEAM EZINE #4 CALL FOR PAPERS 80

FOREWORDS

Hi all finally we had time to complete the long awaited issue III of our eZine. It has been a long wait we know, but real life things kept me busy and far from editing this eZine. It's not an easy thing to collect, select and assembles all the contributions. Nevertheless I hope this issue will be at the same quality level of past issues.

As you can see there's a new look, thanks also to some graphics done by Gunther, which resembles more professional and clear.

This is a special issue, because most of its papers are tied to specific programs. I always try not to tie tutorials to specific targets, because the tutorial otherwise follows the destiny of the program, becoming rapidly obsolete or being too much specific, and also because is less "legal". Each time I do a tutorial the first thing I ask myself is which is the added value I am going to share with readers or, better, will I be Original?

I obviously can only speak for myself: to answer this question I often start from the reasons that bring me to reversing that specific program. It's not just for doing another crack or to release a program before others, because we are out of these businesses (which btw free us from the 0days logics and races). For my own purposes it's always easier to ask for a specific target or just access to 0day repositories: I always can find the programs I need. Then the reason lies behind some program's characteristics, some lessons I learnt that I want to share with someone, either developers or crackers.

This Issue has a lot of target specific tutorials which are often tied to program which have already been updated meanwhile; on the one hand this is a good thing because frees you from the urgency of reading the tutorial just as a longer way to have a crack, on the other hand you cannot directly test the steps, because the target isn't anymore available (but you can ask us). Anyway what I think is good is that you are free to understand the method and the underlying logic most reading the only thing you have, the tutorial.

The targets we used are then just an excuse to do something interesting by the reversers' point of view. The lessons learnt are underlined, but are not limited to what you are reading. Depending if you are a developer or a cracker you might draw your conclusions about how effective are some common behaviour when programming protections.

Anyway what you will do with this document is totally up to you; just remember that, within the trial period, you can do what you want with the programs, after trial expires you should consider removing or buying..

Editor's Note: Reason for this rev.1 is that previously I forgot to insert two contributions from MOID and M1SCH13F. Now they are restored..

Your Favourite Neighbourhood Shubby

DISCLAIMER

All code included with this tutorial is free to use and modify; we only ask that you mention where you found it. This eZine is also free to distribute in its current unaltered form, with all the included supplements.

All the commercial programs used within the different papers have been used only for the purpose of demonstrating the theories and methods described. No distribution of patched applications has been done under any media or host. The applications used were most of the times already been patched by other fellows, and cracked versions were available since a lot of time. ARTeam or the authors of the papers shouldn't be considered responsible for damages to the companies holding rights on those programs. The scope of this eZine as well as any other ARTeam tutorial is of sharing knowledge and teaching how to patch applications, how to bypass protections and generally speaking how to improve the RCE art. We are not releasing any cracked application.

SUPPLEMENTS

This eZine is distributed with Supplements for each paper; the supplements are stored in folders with the same title of the paper. Almost all the papers have supplements, check it.

VERIFICATION

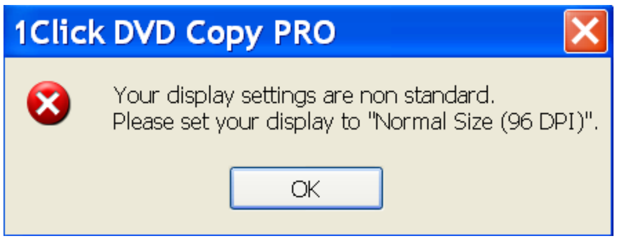
ARTeam.esfv can be opened in the ARTeamESFVChecker to verify all files have been released by ARTeam and are unaltered. The ARTeamESFVChecker can be obtained in the release section of the ARTeam site: http://releases.accessroot.com

1 CODE INJECTION – 1CLICKDVDCOPYPRO BY CONDZERO

1.1 INTRODUCTION

1 Click DVD Copy Technology Software Reviews 2007 reviewed the Top 10 DVD Copy Software Products that produce copies of DVD movies employing "1Click" technology and processing of DVD movie duplication and awarded 1Click DVD Copy Pro version 2.4.1.8 as clearly the more superior product. After our evaluation, we feel at the present time that it *sets the standard* for all DVD copy software applications. After careful examination one will discover that this is because it's feature set and proprietary CPRX technology that makes it possible to copy virtually any DVD in circulation. Something the other DVD movie copy software applications cannot confidently claim. With 1Click DVD Copy Pro, the user has more control over the finished DVD movie product.

In the ARTEAM EZine #2, I showed you how to inject code to mark the sister product application 1ClickDVDCopy 5 as registered. In this short tutorial, I will demonstrate the same principle for marking the pro version as registered. Note: This application is protected by AcProtect / UltraProtect. Don't bother with the AcStripper program on this application. It won't help you. This article demonstrates why a reverser sometimes needs to dig into their bag of tools and find another more convenient way to patch an application. Also note that this application has one annoying feature (see below):


1Click DVD Copy PRO

Your display settings are non standard.
Please set your display to "Normal Size (96 DPI)".

Figure 1

1Click DVD Copy - Pro requires that you have your display resolution set to the default (Normal 96 DPI) setting. Weird, but I guess the developer's had no time for a more robust solution. So let's reset our resolution (if necessary) and move on, shall we.

1.2 ABSTRACT

This Tutorial will introduce you to a method necessary in injecting code into an application so that you can further analyze the application beyond its trial limits. I show you one of the easiest methods in which to accomplish this. Remember, timing is key when dealing with packed / protected targets that employ code encryption and obfuscation.

1.2.1 TARGET

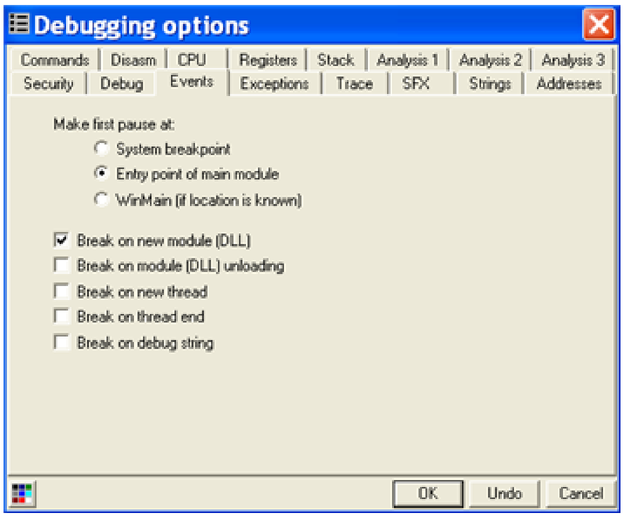
The target is an application called 1CLICK DVD COPY PRO. You can get it at the link below:

Download

1.2.2 PREPARATION

Tools used: OllyDbg v1.10, OllyAdvanced v1.26 Beta 10 for WinNT.

Since I have had some experience with a similar application before, I simply check all the exceptions and for Events I check off Break on new module (DLL) see below:



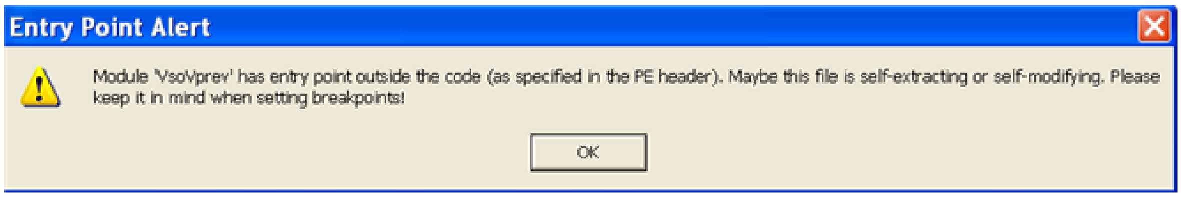
Make first pause at:
- System breakpoint
- [✓ Entry point of main module]
- WinMain (if location is known)

- [✓ Break on new module (DLL)]
- Break on module (DLL) unloading
- Break on new thread
- Break on thread end
- Break on debug string

Figure 2

1.2.2.1 *CHECKING OUT THE TARGET*

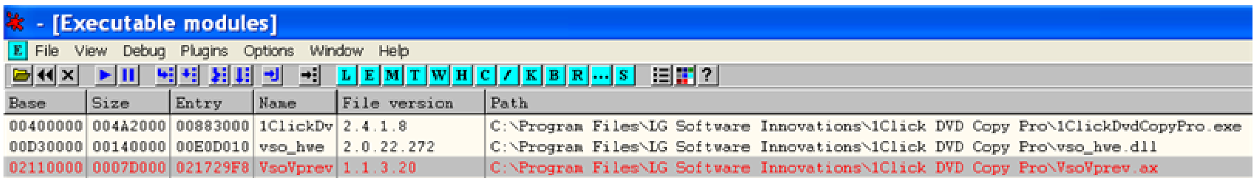
We first open our target in Olly. Don't bother analyzing it at this point. It's compressed and encrypted. We simply hit F9 and run our target and wait for the DLL load events to occur. In our previous Ezine, I reported a dll: vso_hwe.dll that could be used to inject our code. The pro version offers us another option (dll) which we are going to use.



Module 'VsoVprev' has entry point outside the code (as specified in the PE header). Maybe this file is self-extracting or self-modifying. Please keep it in mind when setting breakpoints!

Figure 3

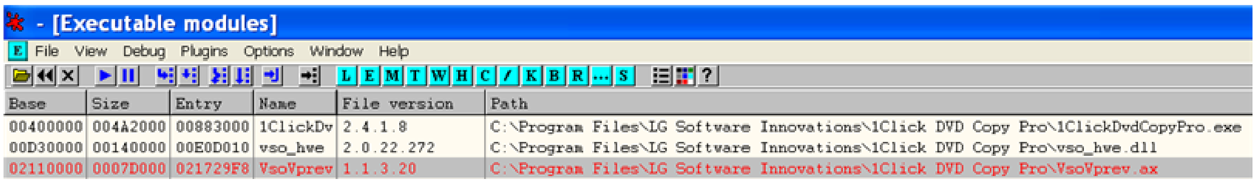
Press OK. Go into the executable modules screen in Olly as shown below:


- [Executable modules]
File View Debug Plugins Options Window Help
L E M T W H C / K B R ... S

```
Base     Size     Entry    Name     File version  Path
00400000 004A2000 00883000 1ClickDv 2.4.1.8       C:\Program Files\LG Software Innovations\1Click DVD Copy Pro\1ClickDvdCopyPro.exe
00D30000 00140000 00E0D010 vso_hwe  2.0.22.272    C:\Program Files\LG Software Innovations\1Click DVD Copy Pro\vso_hwe.dll
02110000 0007D000 021729F8 VsoVprev 1.1.3.20      C:\Program Files\LG Software Innovations\1Click DVD Copy Pro\VsoVprev.ax
```

Figure 4

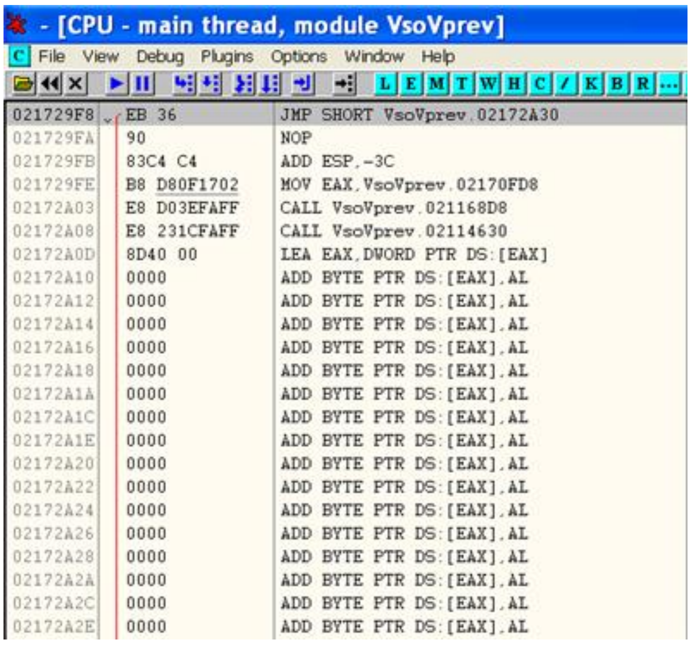
Notice our dll of choice: VsoVprev.ax is highlighted in Red. Notice also its extension .ax, not usual for a dll. We want to follow this entry, so right click on this line and select follow entry as shown below:


- [Executable modules]
File View Debug Plugins Options Window Help
L E M T W H C / K B R ... S

```
Base     Size     Entry    Name     File version  Path
00400000 004A2000 00883000 1ClickDv 2.4.1.8       C:\Program Files\LG Software Innovations\1Click DVD Copy Pro\1ClickDvdCopyPro.exe
00D30000 00140000 00E0D010 vso_hwe  2.0.22.272    C:\Program Files\LG Software Innovations\1Click DVD Copy Pro\vso_hwe.dll
02110000 0007D000 021729F8 VsoVprev 1.1.3.20      C:\Program Files\LG Software Innovations\1Click DVD Copy Pro\VsoVprev.ax
```

Figure 5

Notice we have plenty of “goose eggs” for injecting code via a code cave (see below):

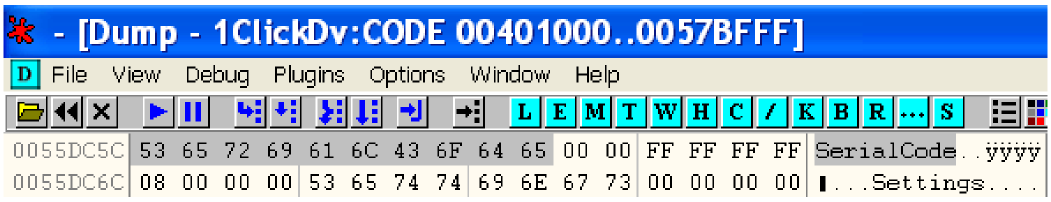

- [CPU - main thread, module VsoVprev]
File View Debug Plugins Options Window Help
L E M T W H C / K B R ...

```
021729F8   EB 36        JMP SHORT VsoVprev.02172A30
021729FA   90           NOP
021729FB   83C4 C4      ADD ESP,-3C
021729FE   B8 D80F1702  MOV EAX,VsoVprev.02170FD8
02172A03   E8 D03EFAFF  CALL VsoVprev.021168D8
02172A08   E8 231CFAFF  CALL VsoVprev.02114630
02172A0D   8D40 00      LEA EAX,DWORD PTR DS:[EAX]
02172A10   0000         ADD BYTE PTR DS:[EAX],AL
02172A12   0000         ADD BYTE PTR DS:[EAX],AL
02172A14   0000         ADD BYTE PTR DS:[EAX],AL
02172A16   0000         ADD BYTE PTR DS:[EAX],AL
02172A18   0000         ADD BYTE PTR DS:[EAX],AL
02172A1A   0000         ADD BYTE PTR DS:[EAX],AL
02172A1C   0000         ADD BYTE PTR DS:[EAX],AL
02172A1E   0000         ADD BYTE PTR DS:[EAX],AL
02172A20   0000         ADD BYTE PTR DS:[EAX],AL
02172A22   0000         ADD BYTE PTR DS:[EAX],AL
02172A24   0000         ADD BYTE PTR DS:[EAX],AL
02172A26   0000         ADD BYTE PTR DS:[EAX],AL
02172A28   0000         ADD BYTE PTR DS:[EAX],AL
02172A2A   0000         ADD BYTE PTR DS:[EAX],AL
02172A2C   0000         ADD BYTE PTR DS:[EAX],AL
02172A2E   0000         ADD BYTE PTR DS:[EAX],AL
```

Figure 6

1.2.2.2 ANALYZING THE TARGET

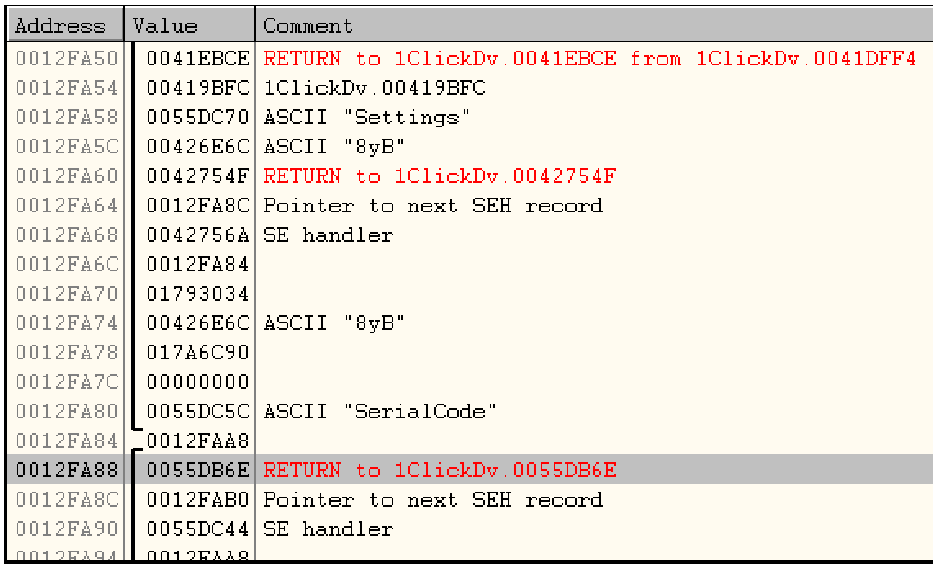
So why have we chosen a dll to inject our code? Timing of the dll entry in relation to the necessary decryption of the code section of interest to occur, and probably most important, the ease in which the process of patching can be implemented. From our previous EZine, the literal we are most interested in is shown below from Olly's memory map:


- [Dump - 1ClickDv:CODE 00401000..0057BFFF]
File View Debug Plugins Options Window Help
L E M T W H C / K B R ... S

```
0055DC5C 53 65 72 69 61 6C 43 6F 64 65 00 00 FF FF FF FF SerialCode..ÿÿÿÿ
0055DC6C 08 00 00 00 53 65 74 74 69 6E 67 73 00 00 00 00 ▮...Settings....
```

Figure 7

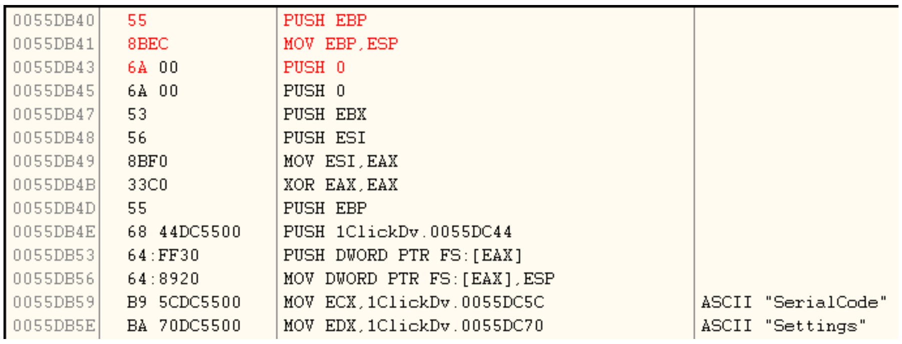
In some cases an application may push this value, but in this application, it moves this value to register ECX. There's a function that is interrogated (run) twice which checks for a valid registration. If a valid registration is found, the function returns EAX == 1, otherwise, invalid registration is EAX == 0. Our goal is to patch this function to always return EAX == 1. The function we are interested can be found by setting a HWBP on access (DWORD) on the address: 0055DC5C shown above. After doing so we can run our target (F9) and we will eventually break on our memory address. We must now look deep into the stack to find the function, or in this case, a return address within the function, shown below:



```
Address   Value     Comment
0012FA50  0041EBCE  RETURN to 1ClickDv.0041EBCE from 1ClickDv.0041DFF4
0012FA54  00419BFC  1ClickDv.00419BFC
0012FA58  0055DC70  ASCII "Settings"
0012FA5C  00426E6C  ASCII "8yB"
0012FA60  0042754F  RETURN to 1ClickDv.0042754F
0012FA64  0012FA8C  Pointer to next SEH record
0012FA68  0042756A  SE handler
0012FA6C  0012FA84
0012FA70  01793034
0012FA74  00426E6C  ASCII "8yB"
0012FA78  017A6C90
0012FA7C  00000000
0012FA80  0055DC5C  ASCII "SerialCode"
0012FA84  0012FAA8
0012FA88  0055DB6E  RETURN to 1ClickDv.0055DB6E
0012FA8C  0012FAB0  Pointer to next SEH record
0012FA90  0055DC44  SE handler
0012FA94  0012FAA8
```

Figure 8

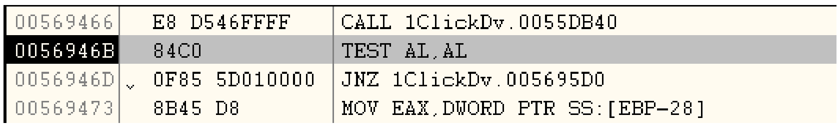
Our reference to "SerialCode" address: 0055DC5C is displayed more than once in the stack, but we are interested in the first instance. If we follow return address: 0055DB6E above in the disassembler we are in the main registration function. Scroll to the top of this procedure as shown below:



```
0055DB40  55              PUSH EBP
0055DB41  8BEC            MOV EBP,ESP
0055DB43  6A 00           PUSH 0
0055DB45  6A 00           PUSH 0
0055DB47  53              PUSH EBX
0055DB48  56              PUSH ESI
0055DB49  8BF0            MOV ESI,EAX
0055DB4B  33C0            XOR EAX,EAX
0055DB4D  55              PUSH EBP
0055DB4E  68 44DC5500     PUSH 1ClickDv.0055DC44
0055DB53  64:FF30         PUSH DWORD PTR FS:[EAX]
0055DB56  64:8920         MOV DWORD PTR FS:[EAX],ESP
0055DB59  B9 5CDC5500     MOV ECX,1ClickDv.0055DC5C              ASCII "SerialCode"
0055DB5E  BA 70DC5500     MOV EDX,1ClickDv.0055DC70              ASCII "Settings"
```

Figure 9

The Red highlighted instructions above are those which will be changed by our code injection. When this procedure returns (after our patch), register EAX will equal ‘1” and the subsequent test for validation shown below:



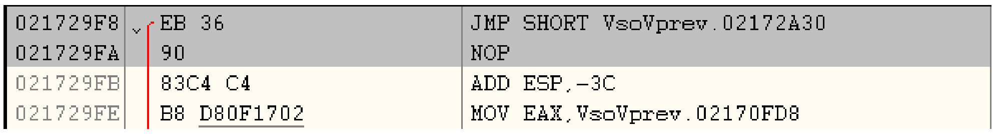
```
00569466    E8 D546FFFF    CALL 1ClickDv.0055DB40
0056946B    84C0           TEST AL,AL
0056946D  ⌄ 0F85 5D010000  JNZ 1ClickDv.005695D0
00569473    8B45 D8        MOV EAX,DWORD PTR SS:[EBP-28]
```

Figure 10

EAX should equal 1 at address 0056946B shown above. So on to our task to inject some code.

1.2.2.3 INJECTING OUR CODE

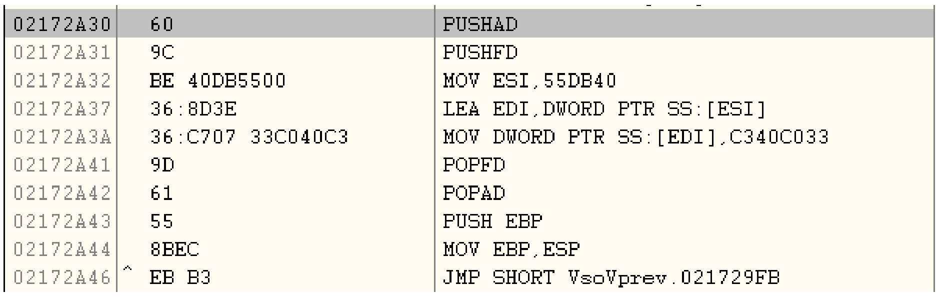
We know the function to be patched. We also know by what means to patch it (inject our code). We will rerun the application again, wait for our dll of interest (VsoVprev.ax) and make the following changes. Note: Refer to figure 6 for the “Before” image.



```
021729F8  EB 36        JMP SHORT VsoVprev.02172A30
021729FA  90           NOP
021729FB  83C4 C4      ADD ESP,-3C
021729FE  B8 D80F1702  MOV EAX,VsoVprev.02170FD8
```

Figure 11

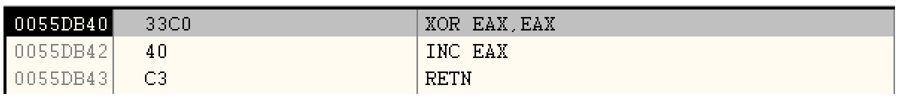
Changes highlighted in grey, above. We now jump to our cave, noting the instructions we have altered which must be saved and reexecuted upon return:



```
02172A30    60                 PUSHAD
02172A31    9C                 PUSHFD
02172A32    BE 40DB5500        MOV ESI,55DB40
02172A37    36:8D3E            LEA EDI,DWORD PTR SS:[ESI]
02172A3A    36:C707 33C040C3   MOV DWORD PTR SS:[EDI],C340C033
02172A41    9D                 POPFD
02172A42    61                 POPAD
02172A43    55                 PUSH EBP
02172A44    8BEC               MOV EBP,ESP
02172A46  ^ EB B3              JMP SHORT VsoVprev.021729FB
```

Figure 12

Here we save our registers and flags. We move the beginning address of our registration function from figure 9 to register ESI so we can perform some changes. We load this address to register EDI and move the following sequence of instructions:



```
0055DB40    33C0    XOR EAX,EAX
0055DB42    40      INC EAX
0055DB43    C3      RETN
```

Figure 13

Note: That the value of register EAX upon return will equal '1'. We then execute our saved instructions and return to the next instruction of the dll. Remember to right click and save to our executable the (2) selections noted in figures 11 and 12 above. We save these changes to a new executable (dll in this case: VsoVprev_new.ax). We can then rename the original dll and rename our saved dll to the original. We are now ready to test our code injection.

Restart the target with no HWBP's or any Events checked in Olly. I won't show the screen, but it runs good, doesn't it.

1.2.2.4 CONCLUSIONS

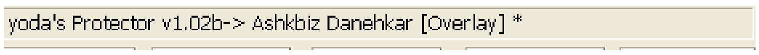
Well, we showed you one way in which to inject code into a packed / encrypted executable to accomplish our goals. Dll's can be your friend in many cases. I hope you learned something new.

2 MUP ANYDVD V6.1.3.6 BY CONDZERO

2.1 INTRODUCTION

AnyDVD is a Windows-based driver that works automatically in the background to unprotect encrypted movie DVDs. This popular software is the subject of many RCE forums. Most of the “cracks” you will see posted revolve around a “Well Known” serial code or a “patch + keygen” fix. I thought it might be interesting to produce a Tutorial that analyzes the limitations and walks through some of the methods to obtain “serial code” functionality without external help.

Peid shows us the type of protection below:


yoda's Protector v1.02b-> Ashkbiz Danehkar [Overlay] *

There’s a fairly straightforward method to finding the OEP, dumping and fixing the IAT, analyzing the application and finally to patch which will be shown in this tutorial. I am not going in-depth on the protection system. Also note that an additional upgrade is available for this application with additional features for full HD-DVD (High Definition DVD) and Blu-Ray support, including decryption of HD-DVD & Blu-Ray movie discs. I’ll point out some steps needed to gain access to this functionality. If you are well versed with the tools described in Setup (below), than you should have no problem following along. With this in mind, let’s move on.

2.2 MUP ANYDVD V6.1.3.6

2.2.1 TARGET

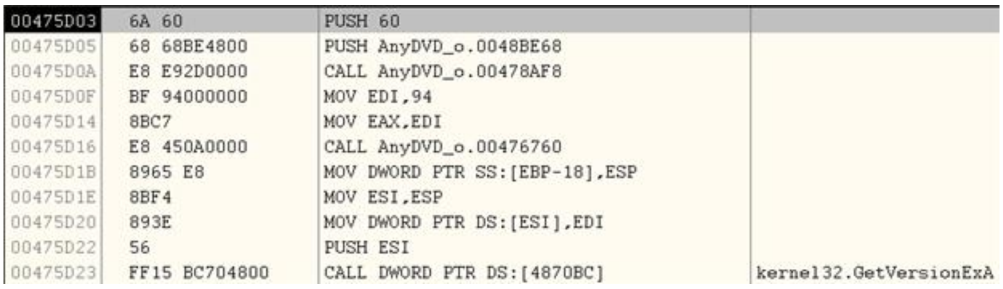
You can get it at this link: SlySoft Products | Copy Movie DVDs with AnyDVD and CloneDVD

2.2.2 SETUP

Tools used: OllyDbg v1.10, PETOOLS v.1.5.800.2006 RC7, ImpRec v1.6.

2.2.3 CHECKING OUT THE TARGET AND FINDING THE OEP

We first open our target in Olly. Olly warns us immediately that our entry point is outside the code and that our application maybe compressed. You don’t need to analyze at this point. Simply go to Olly’s [Executable modules] window and right click on kernel32 > view names. We are looking for the GetModuleHandleA API. Find it and follow it in the disassembler and either set a HWBP on execute or F2 breakpoint on the beginning of this API. Now you can run [F9] the target in Olly. You will break on this API after a very short while. Now we can go into Olly’s [Memory Map] window. Find the code section beginning at address: 00401000 and set a memory breakpoint on access. Run [F9] the target. We will break again on the GetModuleHandleA API. Run [F9] again and we break on our OEP, see below:



```
00475D03  6A 60          PUSH 60
00475D05  68 68BE4800    PUSH AnyDVD_o.0048BE68
00475D0A  E8 E92D0000    CALL AnyDVD_o.00478AF8
00475D0F  BF 94000000    MOV EDI,94
00475D14  8BC7           MOV EAX,EDI
00475D16  E8 450A0000    CALL AnyDVD_o.00476760
00475D1B  8965 E8        MOV DWORD PTR SS:[EBP-18],ESP
00475D1E  8BF4           MOV ESI,ESP
00475D20  893E           MOV DWORD PTR DS:[ESI],EDI
00475D22  56             PUSH ESI
00475D23  FF15 BC704800  CALL DWORD PTR DS:[4870BC]          kernel32.GetVersionExA
```

Figure 14

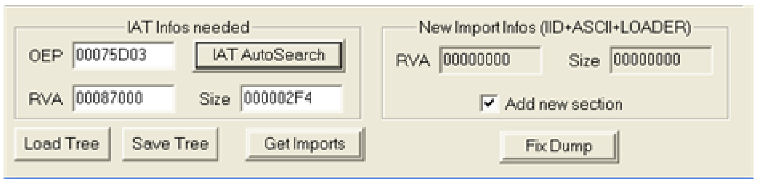
2.2.4 DUMPING THE TARGET

I chose PETools for dumping the target. If we were to dump normally with LordPE, we would end up with a small dump. Why? The sections are protected. With PETools, we can simply find our process and let it dump it for us and we will get a complete dump. After dumping the target, we can fix the EP in the header to our OEP from figure 1. Do not close the target at this point.

2.2.5 FIXING THE IAT

Fortunately with this version of Yoda's protector, there are no emulated API's or anything too strange. We can simply fire up ImpRec on our target and find the process. For options, we are going to choose Create new IAT. Enter the OEP as shown below and press IAT AutoSearch. ImpRec will return with the imports found message. Press okay on this messagebox.

We can now press Get Imports. You will get back a screen that looks similar to the following:



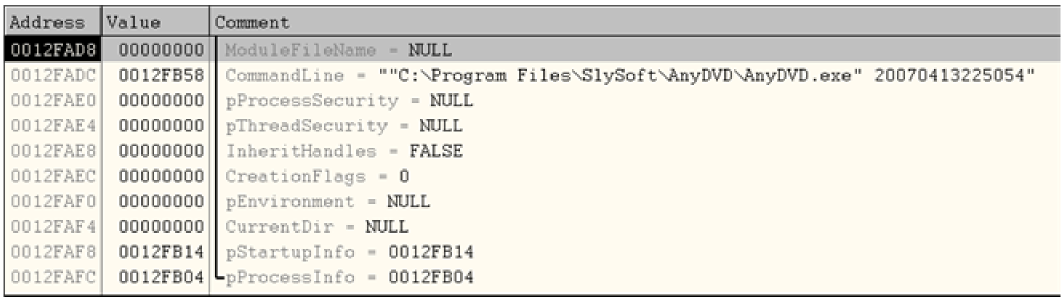
OEP 00075D03
RVA 00087000 Size 000002F4
RVA 00000000 Size 00000000
[✓] Add new section

Figure 15

Now we can simply press Show Invalid and a bunch of highlighted entries will appear. Right click on the highlighted portion of Import Functions Found window and select Trace Level I (Disasm). You will be rewarded with the Congratulatory message. Now we can select Fix Dump and point to our dumped file.

2.2.6 ANALYZING & PATCHING THE TARGET

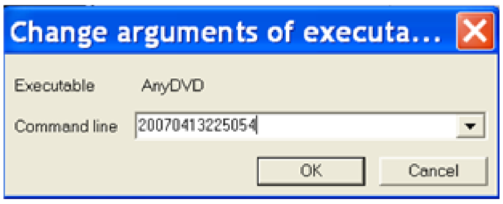
Before we do anything else, it is important that we rename the original target name to something else, and rename our new fixed dump to the original. The important considerations are the time limitation, the nag screen, and the inability to save settings. One more thing. The application behaves similar to Armadillo's father / child protection in that it sets an event, checks for its existence and starts a new process with an argument string. We can temporarily disable this mechanism so as to analyze a single process without the need to attach to a new one. In order to do this we can let Olly analyze our new dump (if not already done) and in Olly's code window, right click and select > Search for > All intermodular calls. Find the CreateProcessA Call and double click on it and set a breakpoint [F2]. Run [F9] our target to this BP. Look at the stack window and notice the argument passed on the CommandLine below:



```
Address   Value     Comment
0012FAD8  00000000  ModuleFileName = NULL
0012FADC  0012FB58  CommandLine = ""C:\Program Files\SlySoft\AnyDVD\AnyDVD.exe" 20070413225054"
0012FAE0  00000000  pProcessSecurity = NULL
0012FAE4  00000000  pThreadSecurity = NULL
0012FAE8  00000000  InheritHandles = FALSE
0012FAEC  00000000  CreationFlags = 0
0012FAF0  00000000  pEnvironment = NULL
0012FAF4  00000000  CurrentDir = NULL
0012FAF8  0012FB14  pStartupInfo = 0012FB14
0012FAFC  0012FB04  └pProcessInfo = 0012FB04
```

Figure 16

Without a great deal of effort, we can see a date string (release date) and perhaps a productid appended to it. The application wants this argument "stamp" in order to run. We can simply copy this string to our clipboard and paste it into Olly's > Debug (menu item) > Arguments as shown below:



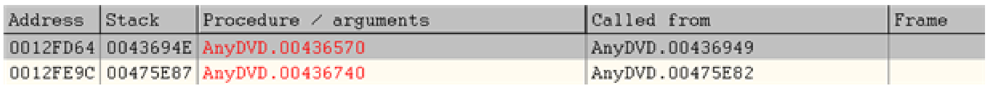
Change arguments of executa...

Executable: AnyDVD

Command line: 20070413225054

Figure 17

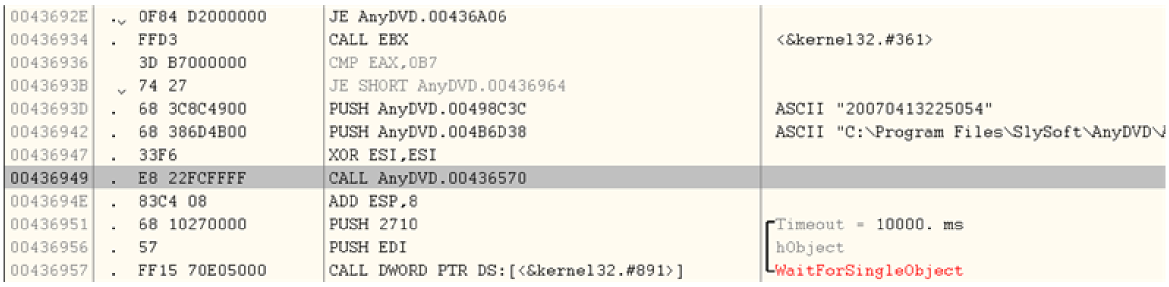
Press OK. Olly will ask you to restart the application. Hold off on this for a moment. We should still be at the BP. Go into Olly's [Call stack of main thread] window and select the first CALL highlighted below:



```
Address   Stack     Procedure / arguments   Called from       Frame
0012FD64  0043694E  AnyDVD.00436570         AnyDVD.00436949
0012FE9C  00475E87  AnyDVD.00436740         AnyDVD.00475E82
```

Figure 18

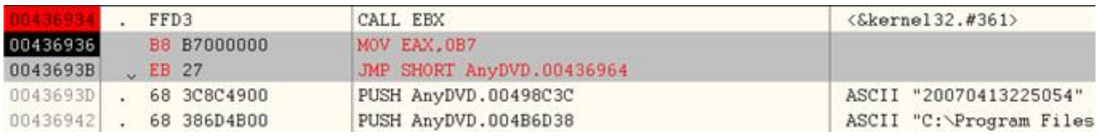
Double click on this line and you will be here:



```
0043692E  .v 0F84 D2000000   JE AnyDVD.00436A06
00436934  .  FFD3            CALL EBX                          <&kernel32.#361>
00436936     3D B7000000     CMP EAX,0B7
0043693B   v 74 27           JE SHORT AnyDVD.00436964
0043693D  .  68 3C8C4900     PUSH AnyDVD.00498C3C              ASCII "20070413225054"
00436942  .  68 386D4B00     PUSH AnyDVD.004B6D38              ASCII "C:\Program Files\SlySoft\AnyDVD\
00436947  .  33F6            XOR ESI,ESI
00436949  .  E8 22FCFFFF     CALL AnyDVD.00436570
0043694E  .  83C4 08         ADD ESP,8
00436951  .  68 10270000     PUSH 2710                         rTimeout = 10000. ms
00436956  .  57              PUSH EDI                          |hObject
00436957  .  FF15 70E05000   CALL DWORD PTR DS:[<&kernel32.#891>]  LWaitForSingleObject
```

Figure 19

The CALL highlighted above at address: 00436949 is for the CreateProcessA API. More importantly is the instruction at address: 00436936 above. There is a call immediately above it for (ntdll.RtlGetLastWin32Error) and is looking to compare the result with “B7” or message “Already exists”. Set a BP on this address. Now we can go ahead and Restart [Ctrl + F2] the application. When we come to our BP we can modify the code as follows:



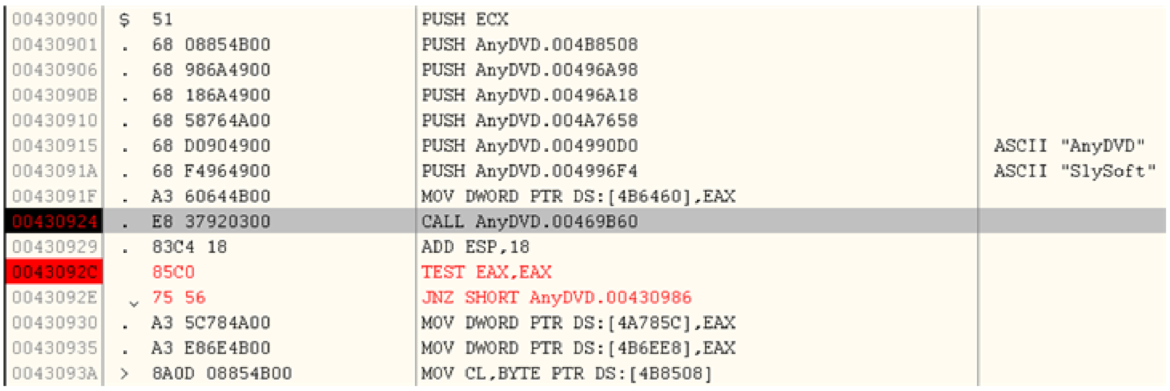
```
00436934  .  FFD3            CALL EBX                          <&kernel32.#361>
00436936     B8 B7000000     MOV EAX,0B7
0043693B   v EB 27           JMP SHORT AnyDVD.00436964
0043693D  .  68 3C8C4900     PUSH AnyDVD.00498C3C              ASCII "20070413225054"
00436942  .  68 386D4B00     PUSH AnyDVD.004B6D38              ASCII "C:\Program Files
```

Figure 20

Now we don’t have to be bothered with attaching to a new process. We will need to apply the code above each and every time we restart the application. We also will need the argument string in Olly.

2.2.6.1 TIME LIMITATION - EXPIRATION

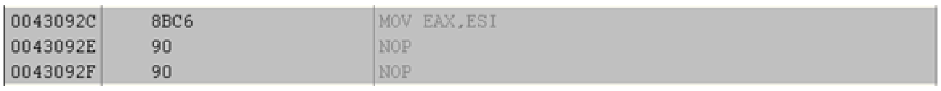
The next order of business is the trial limitation nag screen. The application checks if we have a valid “Key” and Serial code. The CALL address highlighted below leads to the functionality that determines if we have a valid registration key. Step into this function for all the details.



```
00430900  $  51              PUSH ECX
00430901  .  68 08854B00     PUSH AnyDVD.004B8508
00430906  .  68 986A4900     PUSH AnyDVD.00496A98
0043090B  .  68 186A4900     PUSH AnyDVD.00496A18
00430910  .  68 58764A00     PUSH AnyDVD.004A7658
00430915  .  68 D0904900     PUSH AnyDVD.004990D0              ASCII "AnyDVD"
0043091A  .  68 F4964900     PUSH AnyDVD.004996F4              ASCII "SlySoft"
0043091F  .  A3 60644B00     MOV DWORD PTR DS:[4B6460],EAX
00430924  .  E8 37920300     CALL AnyDVD.00469B60
00430929  .  83C4 18         ADD ESP,18
0043092C     85C0            TEST EAX,EAX
0043092E   v 75 56           JNZ SHORT AnyDVD.00430986
00430930  .  A3 5C784A00     MOV DWORD PTR DS:[4A785C],EAX
00430935  .  A3 E86E4B00     MOV DWORD PTR DS:[4B6EE8],EAX
0043093A  >  8A0D 08854B00   MOV CL,BYTE PTR DS:[4B8508]
```

Figure 21

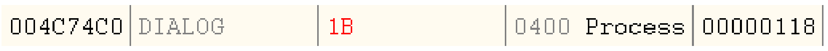
After returning from the CALL above is a TEST EAX,EAX condition which will normally be equal to '1'. The application will determine how much time is left of the 21 day trial. There are (3) key areas here. (1) is a variable that stores the amount of time allowed. Another variable stores the time left. Notice above at addresses: 00430930 and 00430935 are (2) global variables for this purpose. The first address stores the trial period time. The 2nd stores time left. Knowing this in advance, we can influence the decision mechanism by moving some value to EAX that is greater than 0. We could simply NOP the TEST EAX,EAX condition and let EAX == 1. Or, you could move some other register value > 0 to EAX, perhaps, as in my case, the value in register ESI as shown below:


```
0043092C    8BC6    MOV EAX,ESI
0043092E    90      NOP
0043092F    90      NOP
```

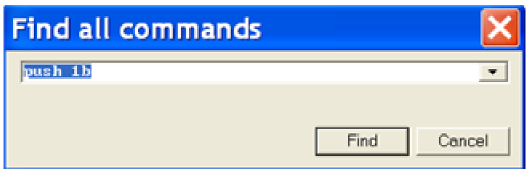
On my machine this was equal to hex '0x44' or decimal 68 "days" left of a 68 day trial period because both values will be the same.

2.2.6.2 NAG SCREEN

We can find the function for the nag screen. I cheated a little bit. I previewed the resources using PE Explorer. The resources are normally stored in the AnyDialog.dll, but we will find the nag screen dialog in the main module. In Olly's [Memory map] window, view the resources section beginning at address: 004BC000, right click on this line and choose > View all resources. Scan down to what maybe the last line shown below:


```
004C74C0  DIALOG  1B  0400 Process  00000118
```

In Olly's code window we can search for > All commands the value "1B" being pushed in the program.



Olly will return a bunch. Select BP on every command for all found commands. Run the target. You will break in the "nag screen" function. Scroll up a few lines and you'll see the ascii literal warning about "…days left …", etc.

Simply scroll to the top of this function and if you want, you can set a BP shown below:

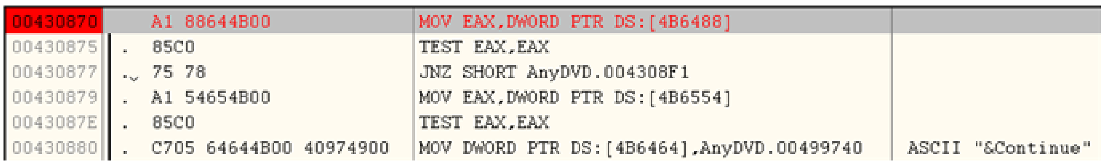

```
00430870    A1 88644B00            MOV EAX,DWORD PTR DS:[4B6488]
00430875  . 85C0                   TEST EAX,EAX
00430877  .v 75 78                 JNZ SHORT AnyDVD.004308F1
00430879  . A1 54654B00            MOV EAX,DWORD PTR DS:[4B6554]
0043087E  . 85C0                   TEST EAX,EAX
00430880  . C705 64644B00 40974900 MOV DWORD PTR DS:[4B6464],AnyDVD.00499740   ASCII "&Continue"
```

Figure 22

But to be more expedient, [and there is more than one way to do this], We can simply NOP the MOV instruction above. The value in EAX will remain unchanged at '1'. This will force the test condition to take the JNZ and move past the nag screen.

2.2.6.3 GENERAL SETTINGS & REGISTRATION INFORMATION

The "trial" version of the application claims to not save any settings after the application exits implying the settings are only saved while the application remains active. This statement is only partially true. The settings have to be stored somewhere and retrieved for interrogation. The "trial" version simply skips the process of retrieving this information and only displays the "default" options on application launch. You can easily verify how the application is storing the settings. The absence of any *.ini file suggests the registry is the data warehouse for this activity. Simply scan for the RegSetValueExA API and set BP's on all of them and you'll find the addresses where your settings get saved. The restoration of settings is a bit trickier. The application doesn't directly CALL the API, but moves the API to a register variable to mask it's use:

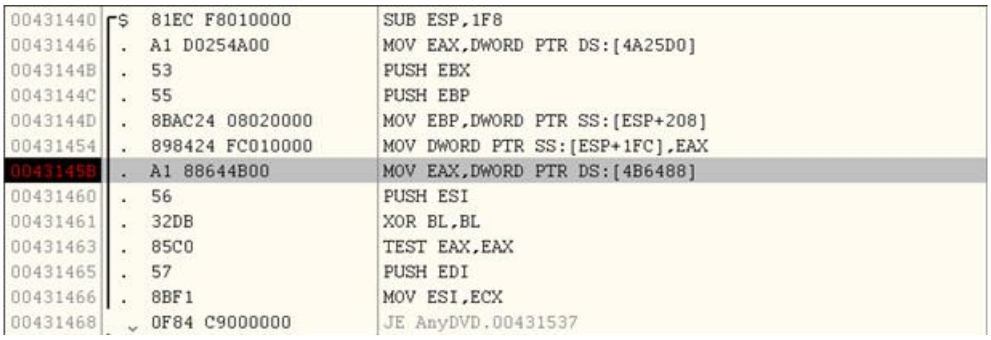
```
0041D376  |.  8B2D 00E05000          MOV EBP,DWORD PTR
DS:[<&advapi32.#494>]  ;  advapi32.RegQueryValueExA
```

and then simply CALLS the register variable.

For those concerned about what information is displayed, a certain function is called once before displaying the application in the system tray (WM_ACTIVATE) and then each time you right click (WM_LBUTTONDOWN) on the application for "Settings".

Next is determining the registered to: " " and serial code in the information window. The application looks to retrieve this information in another function. If you have a valid registration, then that info would appear otherwise "Nobody" appears with the trial time left. What is important is that register BL is set to '1' registered, or remains '0' unregistered and moved to register AL at the end of the function. In turn, register AL is moved to variable and used for determining "saved settings", display of the "order" button and "Registered to:" in the "Settings" dialog.

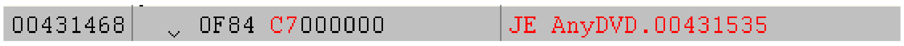
The beginning of what I'll call the "Registration" function appears below:



```
00431440  r$  81EC F8010000    SUB ESP,1F8
00431446  .   A1 D0254A00      MOV EAX,DWORD PTR DS:[4A25D0]
0043144B  .   53               PUSH EBX
0043144C  .   55               PUSH EBP
0043144D  .   8BAC24 08020000  MOV EBP,DWORD PTR SS:[ESP+208]
00431454  .   898424 FC010000  MOV DWORD PTR SS:[ESP+1FC],EAX
0043145B  .   A1 88644B00      MOV EAX,DWORD PTR DS:[4B6488]
00431460  .   56               PUSH ESI
00431461  .   32DB             XOR BL,BL
00431463  .   85C0             TEST EAX,EAX
00431465  .   57               PUSH EDI
00431466  .   8BF1             MOV ESI,ECX
00431468  .   0F84 C9000000    JE AnyDVD.00431537
```

Figure 23

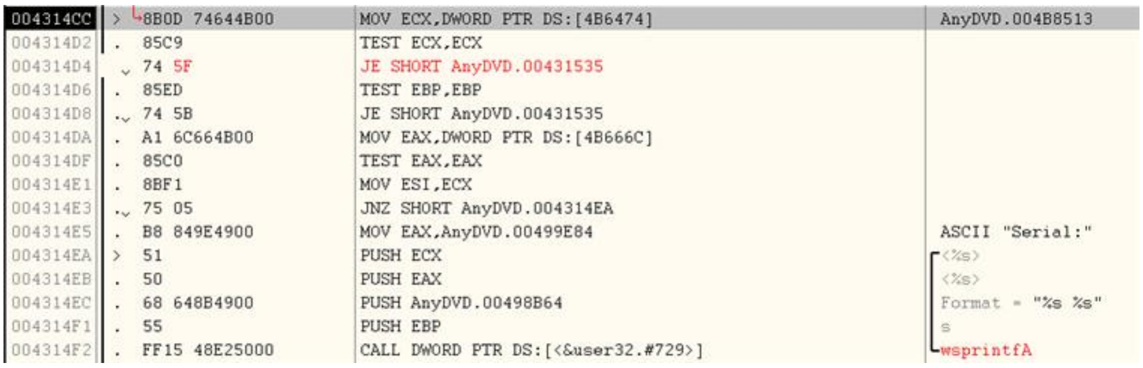
Notice that register BL is cleared at address 00431461. I have set a BP on address 0043145B. The DWORD pointer [004B6488] points to a DWORD that contains the address "value" of whom the application is "Registered to". Note: if not registered, this DWORD would contain '0's (i.e. no valid DWORD address). Find some '0's in the same section referenced and hexedit your ascii literal into this area and simply reference that DWORD value in DWORD pointer 004B6488 above. Register EAX is tested for a valid DWORD address and if none found the function bypasses the remaining registration details so '1' is never moved to register BL. If you are going this route you will need to consider one additional change. For those that don't want to be bothered with this, you can simply make the following change and skip the rest of this section.



```
00431468   ⌄ 0F84 C7000000        JE AnyDVD.00431535
```

2.2.6.4 REGISTERED NAME OPTION (CONT'D):

You have a registration name, but no serial number. The easiest method here is to make the following change:



```
004314CC  > 8B0D 74644B00   MOV ECX,DWORD PTR DS:[4B6474]        AnyDVD.004B8513
004314D2  . 85C9            TEST ECX,ECX
004314D4  ⌄ 74 5F           JE SHORT AnyDVD.00431535
004314D6  . 85ED            TEST EBP,EBP
004314D8  .⌄ 74 5B          JE SHORT AnyDVD.00431535
004314DA  . A1 6C664B00     MOV EAX,DWORD PTR DS:[4B666C]
004314DF  . 85C0            TEST EAX,EAX
004314E1  . 8BF1            MOV ESI,ECX
004314E3  .⌄ 75 05          JNZ SHORT AnyDVD.004314EA
004314E5  . B8 849E4900     MOV EAX,AnyDVD.00499E84              ASCII "Serial:"
004314EA  > 51              PUSH ECX                             <%s>
004314EB  . 50              PUSH EAX                             <%s>
004314EC  . 68 648B4900     PUSH AnyDVD.00498B64                 Format = "%s %s"
004314F1  . 55              PUSH EBP                             s
004314F2  . FF15 48E25000   CALL DWORD PTR DS:[<&user32.#729>]   wsprintfA
```

Figure 24

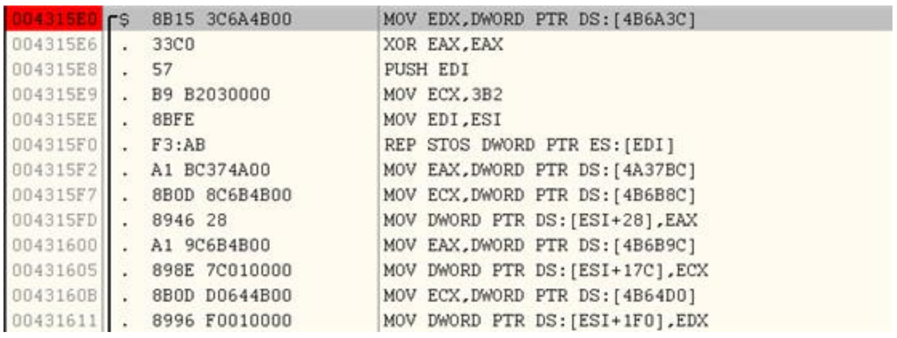
The highlighted line at address 004314D4 above takes you to the all important line below:

```
00431535   B3 01                    MOV BL,1
```

After stepping through this function a few times, you will see what is going on.

2.2.6.5 HD DVD & BLU-RAY SETTINGS

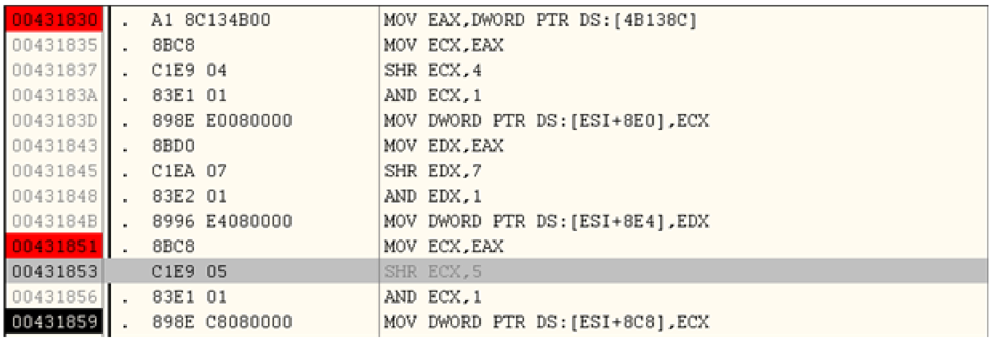
Unless you have purchased the upgrade, this functionality is disabled. You cannot change any of the settings. There is a huge function, you can't miss it because there are literally a hundred moves that is moving literals and values to variables before the "Settings" dialog is displayed. See beginning of the function below:



```
004315E0  ┌$  8B15 3C6A4B00   MOV EDX,DWORD PTR DS:[4B6A3C]
004315E6  │.  33C0            XOR EAX,EAX
004315E8  │.  57              PUSH EDI
004315E9  │.  B9 B2030000     MOV ECX,3B2
004315EE  │.  8BFE            MOV EDI,ESI
004315F0  │.  F3:AB           REP STOS DWORD PTR ES:[EDI]
004315F2  │.  A1 BC374A00     MOV EAX,DWORD PTR DS:[4A37BC]
004315F7  │.  8B0D 8C6B4B00   MOV ECX,DWORD PTR DS:[4B6B8C]
004315FD  │.  8946 28         MOV DWORD PTR DS:[ESI+28],EAX
00431600  │.  A1 9C6B4B00     MOV EAX,DWORD PTR DS:[4B6B9C]
00431605  │.  898E 7C010000   MOV DWORD PTR DS:[ESI+17C],ECX
0043160B  │.  8B0D D0644B00   MOV ECX,DWORD PTR DS:[4B64D0]
00431611  │.  8996 F0010000   MOV DWORD PTR DS:[ESI+1F0],EDX
```

Figure 25

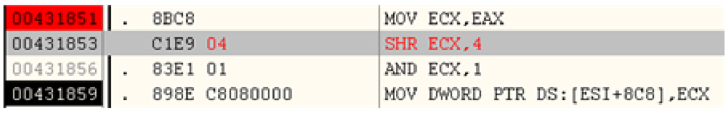
In this function is a test for the HD DVD & Blu-Ray upgrade. Step through this function and you will see the details. If we scroll down to the following section shown below, set (2) BP's as shown:



```
00431830  .  A1 8C134B00     MOV EAX,DWORD PTR DS:[4B138C]
00431835  .  8BC8            MOV ECX,EAX
00431837  .  C1E9 04         SHR ECX,4
0043183A  .  83E1 01         AND ECX,1
0043183D  .  898E E0080000   MOV DWORD PTR DS:[ESI+8E0],ECX
00431843  .  8BD0            MOV EDX,EAX
00431845  .  C1EA 07         SHR EDX,7
00431848  .  83E2 01         AND EDX,1
0043184B  .  8996 E4080000   MOV DWORD PTR DS:[ESI+8E4],EDX
00431851  .  8BC8            MOV ECX,EAX
00431853     C1E9 05         SHR ECX,5
00431856  .  83E1 01         AND ECX,1
00431859  .  898E C8080000   MOV DWORD PTR DS:[ESI+8C8],ECX
```

Figure 26

I have set (2) BP's, one at address 00431830 and 00431851. The first BP is for reference for the change that will be required to enable changing the options for HD DVD and Blu-Ray within the Settings dialog. Address 00431859 moves the good '1' value or bad '0' value to a DWORD address that is later conditionally tested in the ANYDIALOG.DLL module. We want the good value '1' in this address so we make the following change to the grey highlighted line above as shown below:

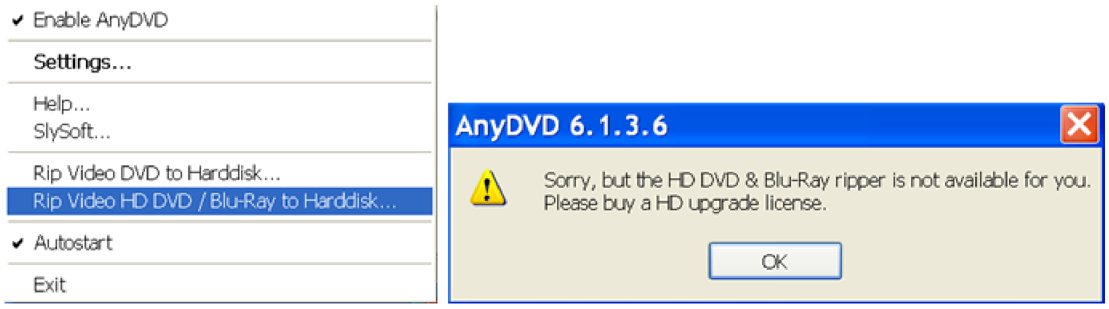


```
00431851  .  8BC8            MOV ECX,EAX
00431853     C1E9 04         SHR ECX,4
00431856  .  83E1 01         AND ECX,1
00431859  .  898E C8080000   MOV DWORD PTR DS:[ESI+8C8],ECX
```

Now register ECX will contain the good value '1'. This takes care of enabling the settings.

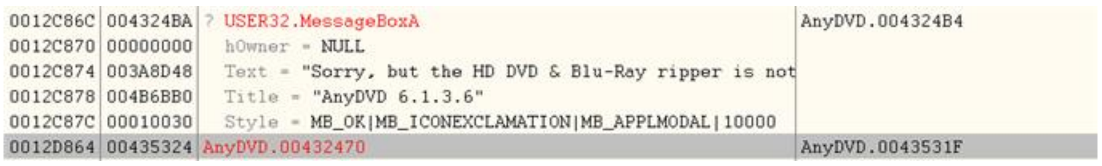
2.2.6.6 HD DVD & BLU-RAY FUNCTIONALITY

The Context menu below highlighted section in blue is ordinarily disabled and a message box appears indicating you need to purchase the upgrade to use:



Sorry, but the HD DVD & Blu-Ray ripper is not available for you.
Please buy a HD upgrade license.

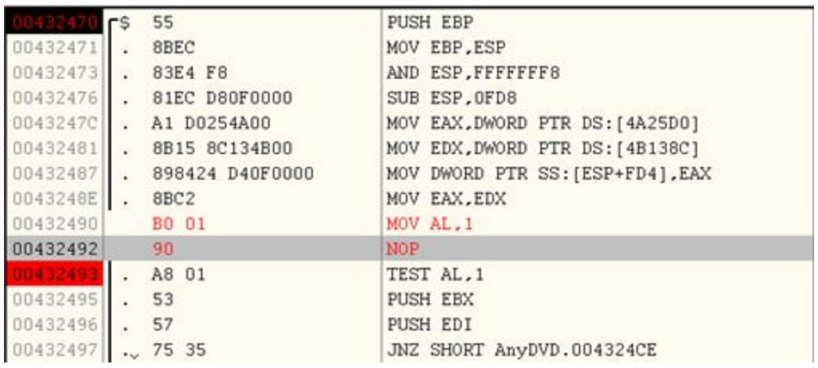
To enable this feature pause the debugger "Ollydbg" at the messagebox above and look in the Call Stack of Main Thread window. Double click on the grey highlighted line below:



```
0012C86C 004324BA ? USER32.MessageBoxA                          AnyDVD.004324B4
0012C870 00000000   hOwner = NULL
0012C874 003A8D48   Text = "Sorry, but the HD DVD & Blu-Ray ripper is not
0012C878 004B6BB0   Title = "AnyDVD 6.1.3.6"
0012C87C 00010030   Style = MB_OK|MB_ICONEXCLAMATION|MB_APPLMODAL|10000
0012D864 00435324 AnyDVD.00432470                               AnyDVD.0043531F
```

Figure 27

This takes us to a function that checks if we have the upgrade option. Note I have already made the necessary change to reflect the upgrade option shown below:



```
00432470 ┌$  55              PUSH EBP
00432471 │.  8BEC            MOV EBP,ESP
00432473 │.  83E4 F8         AND ESP,FFFFFFF8
00432476 │.  81EC D80F0000   SUB ESP,0FD8
0043247C │.  A1 D0254A00     MOV EAX,DWORD PTR DS:[4A25D0]
00432481 │.  8B15 8C134B00   MOV EDX,DWORD PTR DS:[4B138C]
00432487 │.  898424 D40F0000 MOV DWORD PTR SS:[ESP+FD4],EAX
0043248E │.  8BC2            MOV EAX,EDX
00432490     B0 01           MOV AL,1
00432492     90              NOP
00432493 │.  A8 01           TEST AL,1
00432495 │.  53              PUSH EBX
00432496 │.  57              PUSH EDI
00432497 │.v 75 35           JNZ SHORT AnyDVD.004324CE
```

Figure 28

The application will now take the JNZ at address 00432497 and bypass the "Sorry" message which you can see by scrolling further down. This change is too easy and I suspect there may be another condition lurking in the background.

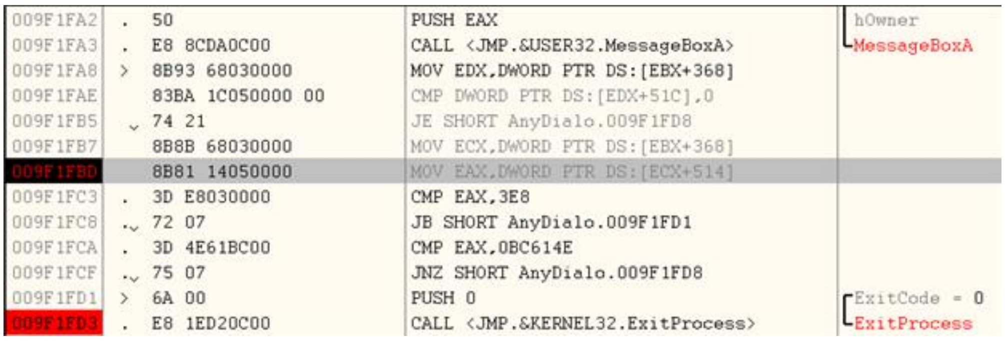
2.2.6.7 GOTCHA'S

While doing some preliminary testing of the patched version of this application, I noticed an immediate problem when viewing the Settings >> Information window. As soon as I pressed the OK button the application exited. Upon further review, I discovered an address pointer value which is tested in the ANYDIALOG.DLL module after checking to see if you are registered and whether to display the message:

"This is a trial version of AnyDVD. Settings change will be lost after program exit.

The registered version will save settings and restore them on program start."

The application then has a few conditional tests prior to an ExitProcess API. See below:

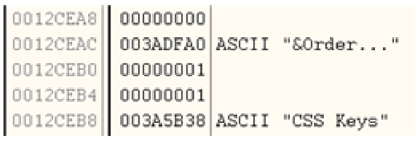


```
009F1FA2  .  50              PUSH EAX                                  hOwner
009F1FA3  .  E8 8CDA0C00     CALL <JMP.&USER32.MessageBoxA>            MessageBoxA
009F1FA8  >  8B93 68030000   MOV EDX,DWORD PTR DS:[EBX+368]
009F1FAE     83BA 1C050000 00 CMP DWORD PTR DS:[EDX+51C],0
009F1FB5  ., 74 21           JE SHORT AnyDialo.009F1FD8
009F1FB7     8B8B 68030000   MOV ECX,DWORD PTR DS:[EBX+368]
009F1FBD     8B81 14050000   MOV EAX,DWORD PTR DS:[ECX+514]
009F1FC3  .  3D E8030000     CMP EAX,3E8
009F1FC8  ., 72 07           JB SHORT AnyDialo.009F1FD1
009F1FCA  .  3D 4E61BC00     CMP EAX,0BC614E
009F1FCF  ., 75 07           JNZ SHORT AnyDialo.009F1FD8
009F1FD1  >  6A 00           PUSH 0                                    ExitCode = 0
009F1FD3  .  E8 1ED20C00     CALL <JMP.&KERNEL32.ExitProcess>          ExitProcess
```

Figure 29

The value I showed for the DWORD PTR DS:[ECX+514], Address: 009F1FBD appears in the pane window below:

```
Stack DS:[0012CEA8]=00000000

EAX=00000001
```



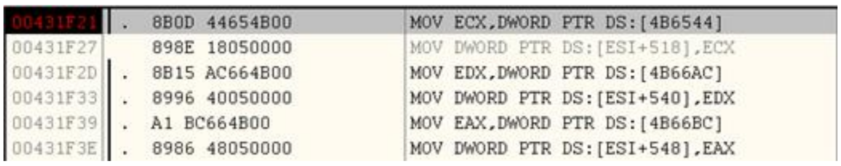
```
0012CEA8  00000000
0012CEAC  003ADFA0  ASCII "&Order..."
0012CEB0  00000001
0012CEB4  00000001
0012CEB8  003A5B38  ASCII "CSS Keys"
```

Since our value == 0, when compared with the value of 3E8, the application would take the JB and jump to ExitProcess. The stack for this address range shows the following.

Interesting, an "&Order…" literal immediately below our address of concern. More investigation reveals that this value is tied to whether or not you have a valid serial number. It gets more interesting after this. If you have entered a serial number beware of a file called AnyDVD.chk. On my machine this file is located in the following folder:

```
C:\Documents and Settings\Administrator\Application Data\SlySoft\AnyDVD
```

Yours may be different. The existence of this file in combination with perhaps an invalid serial number ?? or old serial number ?? Can cause the application to issue the "Your trial period has expired" messagebox. Just delete this file and continue. Getting back to the ExitProcess problem, I traced the ASCII "&Order…" move to the following routine:



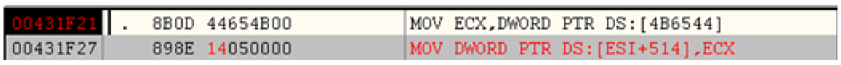
```
00431F21  .  8B0D 44654B00   MOV ECX,DWORD PTR DS:[4B6544]
00431F27     898E 18050000   MOV DWORD PTR DS:[ESI+518],ECX
00431F2D  .  8B15 AC664B00   MOV EDX,DWORD PTR DS:[4B66AC]
00431F33  .  8996 40050000   MOV DWORD PTR DS:[ESI+540],EDX
00431F39  .  A1 BC664B00     MOV EAX,DWORD PTR DS:[4B66BC]
00431F3E  .  8986 48050000   MOV DWORD PTR DS:[ESI+548],EAX
```

Code pane:

```
DS:[004B6544]=003ADFA0, (ASCII "&Order...")
```

```
ECX=003AFF20, (ASCII "About")
```

The recipient address of [ESI+518] is +4 our concerned address. We can simply change this move to the following:



```
00431F21  .  8B0D 44654B00   MOV ECX,DWORD PTR DS:[4B6544]
00431F27     898E 14050000   MOV DWORD PTR DS:[ESI+514],ECX
```

Our aim is to get a value in this address > 3E8. Also, we don't want to patch the ANYDIALOG.DLL if we don't have to. At this point, you should save all your patches and copy them to the executable. Remember that the saved executable name must be the same as the original. Also, we do not want to save the patch that prohibits the CreateProcess from executing. This patch was only to analyze the application.

2.2.7 CONCLUSIONS

There could possibly be more mysteries uncovered in this application and certainly better ways to patch. I freely admit that I don't really use this application, but just wanted to get a better understanding of its process flow. My goal was to analyze the major limitations and point out some options from a precursory perspective. I hope you enjoyed reading this Tutorial and perhaps learned a few things about it.

3 PATCHING PRIMA EGUIDES (SINGLE BYTE PATCHING) BY SSLEVIN

3.1 INTRODUCTION

After a lot of wondering thru the Internet I finally found something interesting to reverse and at the same time to have a tight connection with my favorite file protector, you guess, ActiveMark.

But the scope of this tutorial won't be unpacking and patching ActiveMark although targets **are** protected with it, since this matter is well explained in previous tutorials written by **condzero**.

Namely, guys from PrimaGames got an idea how to earn some money by making e-Guides for different software (mostly games).

Price of this stuff is fair, but hey, who is talking about the price? Reversing is the topic, right? So let see how to make this stuff work without limitations.

3.2 PATCHING PRIMA EGUIDES

This tutorial will show you how to patch exe file used to browse a pdf document which is password protected and its trial use is limited to first ten pages.

3.2.1 TARGET

Target used in this tutorial can be downloaded here:

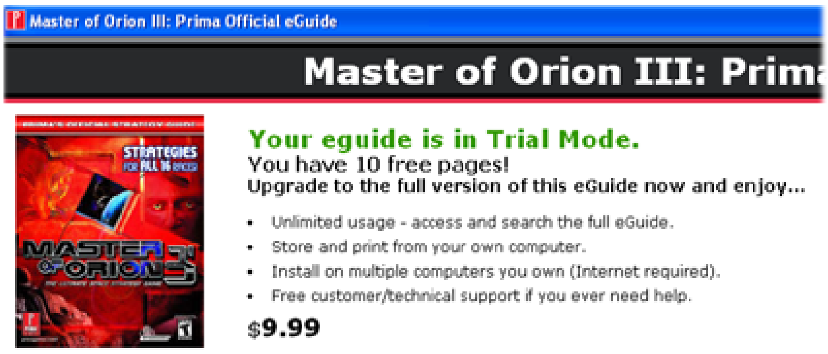
http://d.trymedia.com/dm/primag/0761550259_1/trygames/MasterofOrionIIIPrimaOffici.exe

3.2.2 TOOLS

- OllyDbg v1.10
- Some brain

3.3 INSPECTING THE TARGET

Initial nag screen which informs us about the fact that we are using trial version of software is classic form used by Macrovision (see Figure 30.) The only difference is that there is no time limitation but number of pages you can read for free (10 pages).


Master of Orion III: Prima Official eGuide

Master of Orion III: Prim

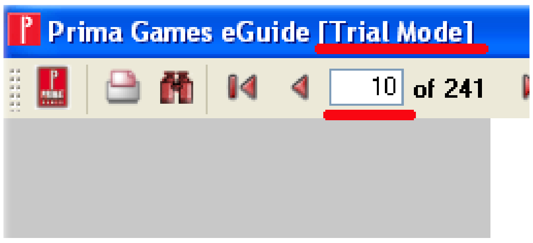
Your eguide is in Trial Mode.
You have 10 free pages!
Upgrade to the full version of this eGuide now and enjoy...

- Unlimited usage - access and search the full eGuide.
- Store and print from your own computer.
- Install on multiple computers you own (Internet required).
- Free customer/technical support if you ever need help.

$9.99

Figure 30

If you think that after successful removing of ActiveMark 5 you also solved trial limitations you'll be disappointed. Trial limit of ten pages is still there (see Figure 31.)

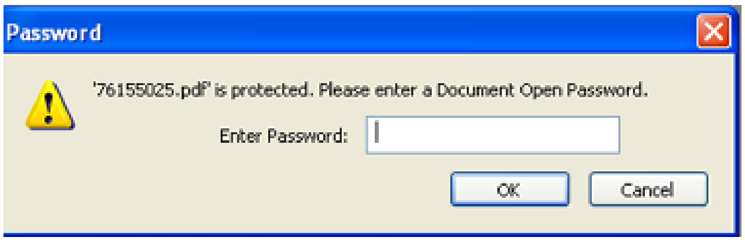

Prima Games eGuide [Trial Mode]

10 of 241

Figure 31

But this also give us a good hint on where to look later for patches, right? (String: Trial Mode).

Last, but not the least, if you try to open pdf document itself which is located in C:\Program Files\Prima Games\Master of Orion III Prima Official eGuide\pdf you'll find that it is password protected (see Figure 32.) Bruteforceing is an option here, but where to find a readymade bruteforcer? (If you know how to make one, hat down, you don't need to read this further).


Password

'76155025.pdf' is protected. Please enter a Document Open Password.

Enter Password:

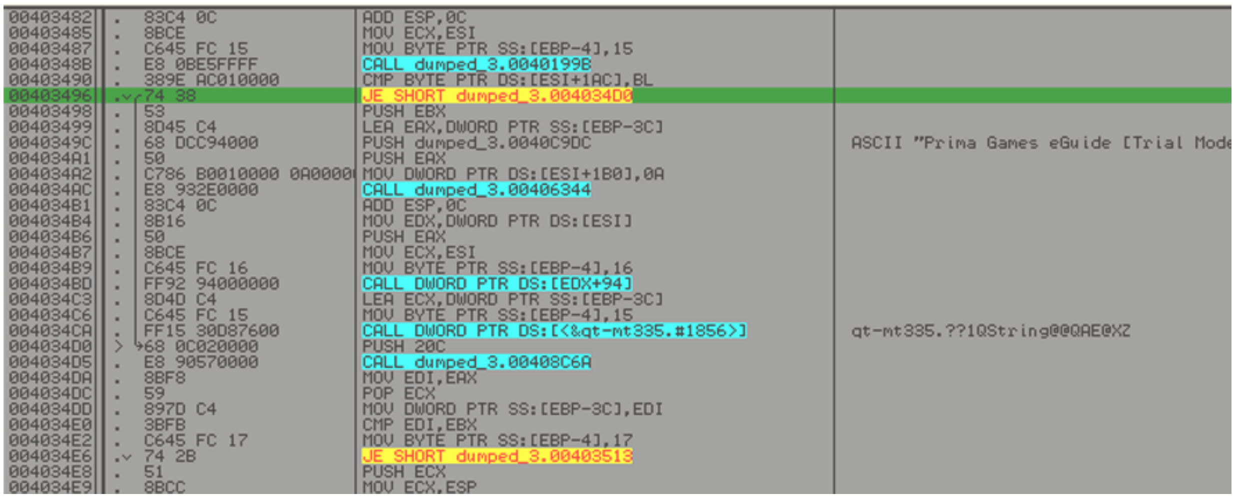
Figure 32

3.4 FINDING PATCH(ES)

3.4.1 PIECE OF CAKE

It is literally piece of cake to find patches for this stuff. At first I was amazed that there is no other protection than sole byte comparing. No anti-debug tricks, no checksum, simply nothing.

So, let start. Load your rebuilt target in Olly. Then search for all referenced text strings. After Olly finishes search, scroll to the top of the list, mark first line, rightclick and search for text: Trial Mode. (I already said that target is giving us a good hint on where to search ☺). Follow in disassembler (press Enter). You'll find yourself in a spot like in Figure 33.



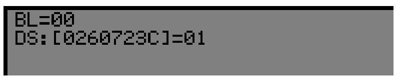
```
00403482  .   83C4 0C              ADD ESP,0C
00403485  .   8BCE                 MOV ECX,ESI
00403487  .   C645 FC 15           MOV BYTE PTR SS:[EBP-4],15
0040348B  .   E8 0BE5FFFF          CALL dumped_3.0040199B
00403490  .   389E AC010000        CMP BYTE PTR DS:[ESI+1AC],BL
00403496  .v,74 38                 JE SHORT dumped_3.004034D0
00403498  .  |53                   PUSH EBX
00403499  .  |8D45 C4              LEA EAX,DWORD PTR SS:[EBP-3C]
0040349C  .  |68 DCC94000          PUSH dumped_3.0040C9DC                    ASCII "Prima Games eGuide [Trial Mode
004034A1  .  |50                   PUSH EAX
004034A2  .  |C786 B0010000 0A0000 MOV DWORD PTR DS:[ESI+1B0],0A
004034AC  .  |E8 932E0000          CALL dumped_3.00406344
004034B1  .  |83C4 0C              ADD ESP,0C
004034B4  .  |8B16                 MOV EDX,DWORD PTR DS:[ESI]
004034B6  .  |50                   PUSH EAX
004034B7  .  |8BCE                 MOV ECX,ESI
004034B9  .  |C645 FC 16           MOV BYTE PTR SS:[EBP-4],16
004034BD  .  |FF92 94000000        CALL DWORD PTR DS:[EDX+94]
004034C3  .  |8D4D C4              LEA ECX,DWORD PTR SS:[EBP-3C]
004034C6  .  |C645 FC 15           MOV BYTE PTR SS:[EBP-4],15
004034CA  .  |FF15 30D87600        CALL DWORD PTR DS:[<&qt-mt335.#1856>]     qt-mt335.??1QString@@QAE@XZ
004034D0  >  `>68 0C020000         PUSH 20C
004034D5  .   E8 90570000          CALL dumped_3.00408C6A
004034DA  .   8BF8                 MOV EDI,EAX
004034DC  .   59                   POP ECX
004034DD  .   897D C4              MOV DWORD PTR SS:[EBP-3C],EDI
004034E0  .   3BFB                 CMP EDI,EBX
004034E2  .   C645 FC 17           MOV BYTE PTR SS:[EBP-4],17
004034E6  .v  74 2B                JE SHORT dumped_3.00403513
004034E8  .   51                   PUSH ECX
004034E9  .   8BCC                 MOV ECX,ESP
```

Figure 33

Remark the conditional jump (JE) which jumps over Trial Mode if taken. Plain stupid method of patching would be to change conditional to unconditional jump (JE to JMP), problem solved, right?

Wroong … Try to do this, and you ll find yourself in lalaland. ☺

In a reversers way, you will try to find which value is set at memory location pointed by value of ESI register plus 1AC h and then compared with value of less significant byte of EBX register – BL (see the comparison above conditional jump).



```
BL=00
DS:[0260723C]=01
```

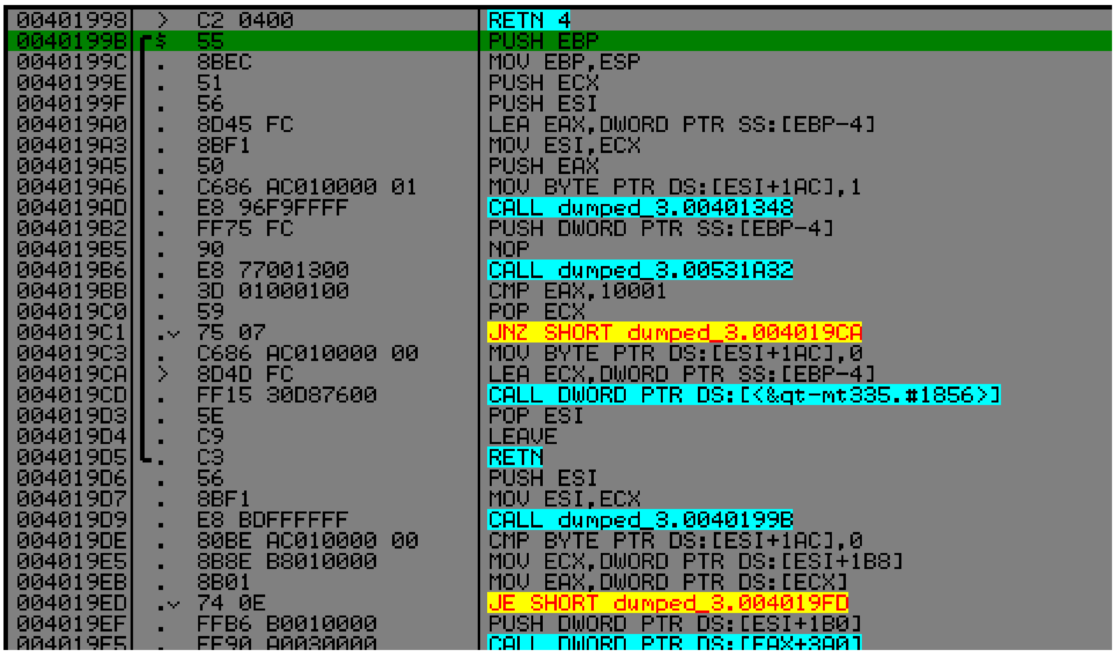
How to achieve this? Simply, set breakpoint at 403490 (F2) and run application (F9). After Olly breaks check the content of its pane window and you will see that BL is 00 and is compared to 0260723C (ESI+1AC h) which is 01.

After comparison is done, conditional jump won't be taken and we are in trial mode.

Now, we have to find where the value of 0260723C is set to 01. Common sense is telling us that it happens in some CALL above comparison. So set breakpoint at this line:

```
0040348B    E8 0BE5FFFF           CALL dumped_3.0040199B
```

Restart application and follow your new breakpoint in disassembler. Press Enter to get into the call without executing code. You ll see something similar to Figure 34.



```
00401998  >  C2 0400           RETN 4
0040199B  r$ 55                PUSH EBP
0040199C  .  8BEC              MOV EBP,ESP
0040199E  .  51                PUSH ECX
0040199F  .  56                PUSH ESI
004019A0  .  8D45 FC           LEA EAX,DWORD PTR SS:[EBP-4]
004019A3  .  8BF1              MOV ESI,ECX
004019A5  .  50                PUSH EAX
004019A6  .  C686 AC010000 01  MOV BYTE PTR DS:[ESI+1AC],1
004019AD  .  E8 96F9FFFF       CALL dumped_3.00401348
004019B2  .  FF75 FC           PUSH DWORD PTR SS:[EBP-4]
004019B5  .  90                NOP
004019B6  .  E8 77001300       CALL dumped_3.00531A32
004019BB  .  3D 01000100       CMP EAX,10001
004019C0  .  59                POP ECX
004019C1  .v 75 07             JNZ SHORT dumped_3.004019CA
004019C3  .  C686 AC010000 00  MOV BYTE PTR DS:[ESI+1AC],0
004019CA  >  8D4D FC           LEA ECX,DWORD PTR SS:[EBP-4]
004019CD  .  FF15 30D87600     CALL DWORD PTR DS:[<&qt-mt335.#1856>]
004019D3  .  5E                POP ESI
004019D4  .  C9                LEAVE
004019D5  L. C3                RETN
004019D6  .  56                PUSH ESI
004019D7  .  8BF1              MOV ESI,ECX
004019D9  .  E8 BDFFFFFF       CALL dumped_3.0040199B
004019DE  .  80BE AC010000 00  CMP BYTE PTR DS:[ESI+1AC],0
004019E5  .  8B8E B8010000     MOV ECX,DWORD PTR DS:[ESI+1B8]
004019EB  .  8B01              MOV EAX,DWORD PTR DS:[ECX]
004019ED  .v 74 0E             JE SHORT dumped_3.004019FD
004019EF  .  FFB6 B0010000     PUSH DWORD PTR DS:[ESI+1B0]
004019F5  .  FF90 A0030000     CALL DWORD PTR DS:[EAX+3A0]
```

Figure 34

See, there are two lines of code which are setting value of ESI+1ACh:

```
004019A6           MOV BYTE PTR DS:[ESI+1AC],1      ← to 0

[…omissis…]

004019C1   |. /75 07                  JNZ SHORT dumped_3.004019CA

004019C3   |.  C686 AC010000 00       MOV BYTE PTR DS:[ESI+1AC],0

004019CA   |> \8D4D FC                LEA ECX,DWORD PTR SS:[EBP-4]
```

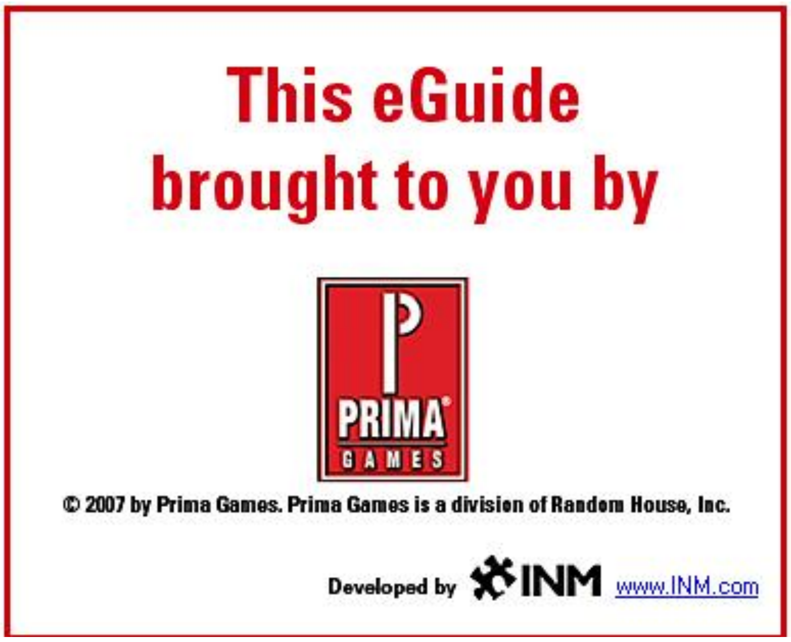
So, you have two options for patching: either you will change 01 to 00 at line 004019A6 or you will NOP conditional jump at line 004019C1 (75 07 to 90 90).

After assembling any of this changes copy them to executable, save under another name and you are done!!!

3.4.2 AESTHETIC PATCH

To those of you who do not like any kind of nag screens showing up. You can remove nag screen shown in Figure 35 by simply NOP–ing a call to this procedure at line:

```
004012D4  |.  FF90 B8000000         CALL DWORD PTR DS:[EAX+B8]
```



This eGuide brought to you by

© 2007 by Prima Games. Prima Games is a division of Random House, Inc.

Developed by INM www.INM.com

Figure 35

And, yes, how to find it? Simply, trace thru the code with F8, and after this nag appears set breakpoint one line above EIP. Since steping into (F7) this call and assembling RET at the beginning of nag subroutine doesn't work simply NOP this call and nag is history.

3.5 CONCLUSIONS

As you could see, this was incredibly easy to do. There is a whole bunch of these on trygames.com and if you want to practice be my guest. Personally, I don't need this stuff, (uuuhhh, I can't even remember when I played some game ☺). Weird kind of limitation was my motivation to reverse this.

4 EXAMDIFF 4.XXX REVERSING THE PROTECTION SCHEMA BY SHUB-NIGURRATH

4.1 INTRODUCTION

The program is a really interesting and powerful comparison program. I will not list all the advantages of this tool, because I am not interested in them for this tutorial. Also according to the Foreword section of this issue I decided to write this contribution just because the tutorial contains some "lessons". The previous version of this program has been keygenned since a lot of time, cracked and abused in all the possible ways. The new version 4.0 has been worked for a long time then I was expecting some type of improvement in protection. I started to reverse this last version just because I was curious to verify which improvements were running.

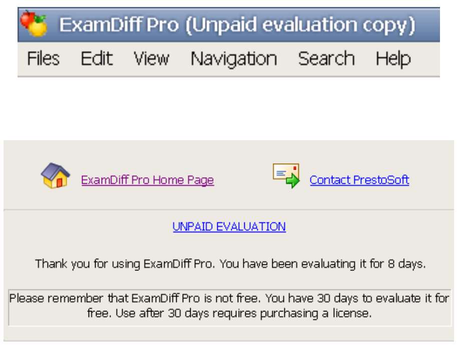
You can download ExamDiff Pro (www.prestosoft.com), I tested the described approach on versions up to 4.0.23.

4.2 APPROACHING THE ENEMY

The approach I will show works for a lot of versions, from 3.4.2, 3.5.1.5 up to 4.0.0.xx (I tested it up to latest build too).

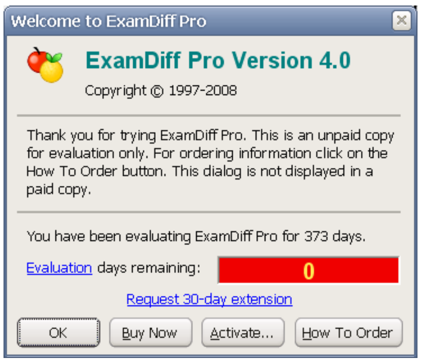
The program is a normal MFC program, not compressed, which is using an encryption-based licensing scheme and has a limited trial time (30 days). Previous versions of the program have been keygenned and keygens are out on the net. I will not teach where to find one of them but one of the team which made them is CORE...I will not even try to understand the keygen algo..

The program, once installed, has an initial nag reminding you that your copy is a trial and, after 30 days, several functions are disabled (and also the title bar reports it).


ExamDiff Pro (Unpaid evaluation copy)
Files Edit View Navigation Search Help
ExamDiff Pro Home Page
Contact PrestoSoft
UNPAID EVALUATION

Thank you for using ExamDiff Pro. You have been evaluating it for 8 days.

Please remember that ExamDiff Pro is not free. You have 30 days to evaluate it for free. Use after 30 days requires purchasing a license.



Thank you for trying ExamDiff Pro. This is an unpaid copy for evaluation only. For ordering information click on the How To Order button. This dialog is not displayed in a paid copy.

You have been evaluating ExamDiff Pro for 373 days.

Evaluation days remaining: 0

The interesting result you get reversing this program is that you realize how developers rarely learn from their past errors, this new version has been worked for a long time, but the protection is as lame as before. It didn't take long time to realize how to generally patch it.

Note

•After having patched the program I realized that there was an existing patch already on the net, which was using almost the same approach (it is by CW2K), damn ^_^. Anyway I decided to align this tutorial to that patch, integrating it with my comments.

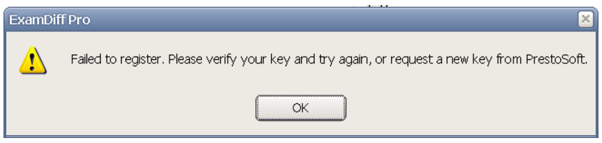
The first thing to do (after installation of course) is to run the main program within Olly and to analyze it using IDA, for a higher and graphical view of functions and branches. I usually use both tools simultaneously, for different reasons I will not explain here: the main reason is that IDA gives you a better view of the program structures and branches, as well as library part of code (using the signatures), but the debugger is not as usable as OllyDbg, which remains the top class win32 Ring3 debugger.

4.3 REVERSING THE REGISTRATION SCHEMA

The first thing I tried was to test the older serial numbers just to see if the program was accepting them. I used this serial, which was good for version 3.1:

```
H+Jgcces3ANbNh8mC+ldHgpALFNelaAEdneZ1R/3h9n2huUEL0Hr+K

/PT8FU2/ZK1slC74IsZgu4FbdtfYjz0njjISAvyKbpifrZfpe0ESzt

OIhfA5paKXtIGbJu5JYX6VRq+6JzlStfR6puDK2Q5iWI8sB/EHkxia

1rH2WfqrnkfbBrrrdylnEvSu4E8JC6YPHG2F5JqJ/3bN/+k3J1BbXP

TJmghW70dwiuX32XZGtSL5iCrIC0TtFwrT/ZJP8pkGpT17SkrOG1TE

1gykc+sR9avpl3zMkwz/ijSwVQlUVlLXJOs6W6oBjBBC5ifvqrLE4s

pOmKLcfla9YHI0rEgeb5y2XQhhrdJ2dryiJC474=
```

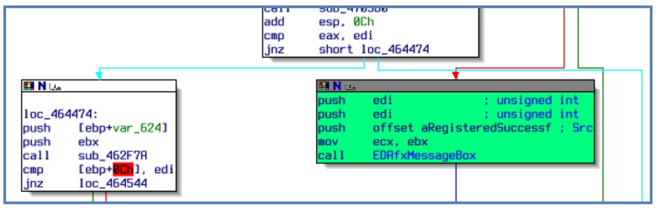
If you enter this serial the program warns you that the serial is old and that you must ask for a new one to PrestoSoft.


ExamDiff Pro

Failed to register. Please verify your key and try again, or request a new key from PrestoSoft.

```
0046465A  |.  68 38376200              PUSH ExamDiff.00623738
;  UNICODE "Failed to register. Please verify your key and try again, or request a
new key from PrestoSoft."
```

this message is into the function sub_464164 (using the name IDA assign to it).

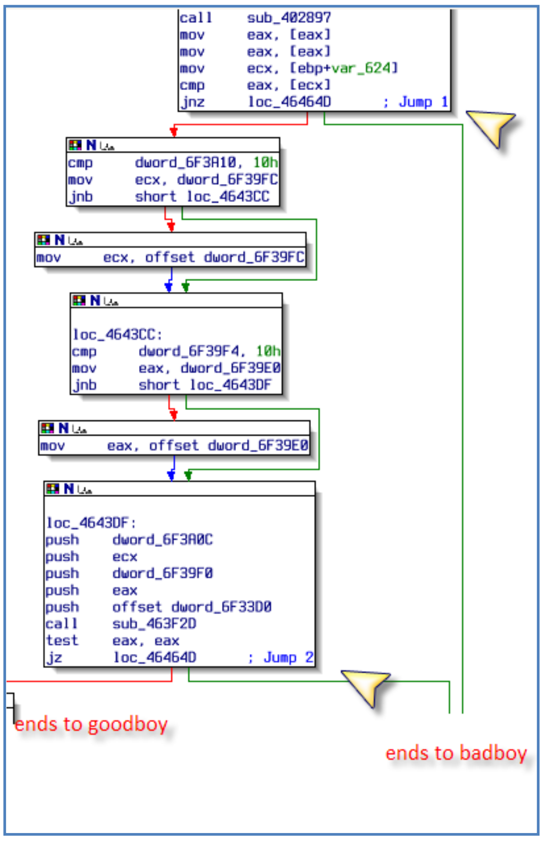


```
add     esp, 0Ch
cmp     eax, edi
jnz     short loc_464474

loc_464474:
push    [ebp+var_624]
push    ebx
call    sub_462F7A
cmp     [ebp+0Ch], edi
jnz     loc_464544

push    edi                 ; unsigned int
push    edi                 ; unsigned int
push    offset aRegisteredSuccessf ; Src
mov     ecx, ebx
call    EDAfxMessageBox
```

Thanks to the string analysis done by IDA, you can see that, a little above this code, there the following PUSH, which pushes (and IDA tells you clearly) the string "registered successfully" (see figure beside).

```
00464633  |.  68 00376200     PUSH ExamDiff.00623700
; |Arg1 = 00623700
```

Needless to say that this is the correct place where to start your journey.



```
call    sub_402897
mov     eax, [eax]
mov     eax, [eax]
mov     ecx, [ebp+var_624]
cmp     eax, [ecx]
jnz     loc_46464D        ; Jump 1

cmp     dword_6F3A10, 10h
mov     ecx, dword_6F39FC
jnb     short loc_4643CC

mov     ecx, offset dword_6F39FC

loc_4643CC:
cmp     dword_6F39F4, 10h
mov     eax, dword_6F39E0
jnb     short loc_4643DF

mov     eax, offset dword_6F39E0

loc_4643DF:
push    dword_6F3A0C
push    ecx
push    dword_6F39F0
push    eax
push    offset dword_6F33D0
call    sub_463F2D
test    eax, eax
jz      loc_46464D        ; Jump 2
```

The two branches of this function, the goodboy and the badboy, are starting from two important jumps:

jump 1:

```
.text:004643B0  cmp     eax, [ecx]

.text:004643B2  jnz     loc_46464D ; good
boys don't jump
```

jump 2:

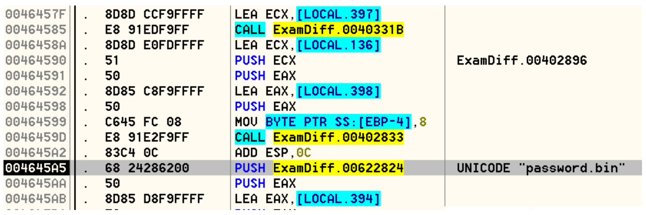
```
.text:004643F2  call    sub_463F2D

.text:004643F7  test    eax, eax

.text:004643F9  jz      loc_46464D  ; good
boys don't jump here
```

If you debug the program you'll realize that not jumping there, directly brings you to the "registered successfully" message.

If you force these two jumps you will directly go to the section of the function which is responsible of coding and creating the registration number file, the password.bin file, holding the registration number.

Note

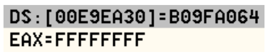
• NOPing these two jumps will not resolve the problem. It will force the program to accept any serial you enter but the program wil still show the initial nag and will remain unregistered. There's an additional check I will explain later..



```
0046457F  .  8D8D CCF9FFFF   LEA ECX,[LOCAL.397]
00464585  .  E8 91EDF9FF     CALL ExamDiff.0040331B
0046458A  .  8D8D E0FDFFFF   LEA ECX,[LOCAL.136]
00464590  .  51              PUSH ECX                       ExamDiff.00402896
00464591  .  50              PUSH EAX
00464592  .  8D85 C8F9FFFF   LEA EAX,[LOCAL.398]
00464598  .  50              PUSH EAX
00464599  .  C645 FC 08      MOV BYTE PTR SS:[EBP-4],8
0046459D  .  E8 91E2F9FF     CALL ExamDiff.00402833
004645A2  .  83C4 0C         ADD ESP,0C
004645A5  .  68 24286200     PUSH ExamDiff.00622824         UNICODE "password.bin"
004645AA  .  50              PUSH EAX
004645AB  .  8D85 D8F9FFFF   LEA EAX,[LOCAL.394]
```

4.3.1 STUDYING THE JUMP 1

Then let we approach the two jumps one by one. The first one (004643B2) depends by the CMP instruction just above (004643B0) where (debugging) you have:

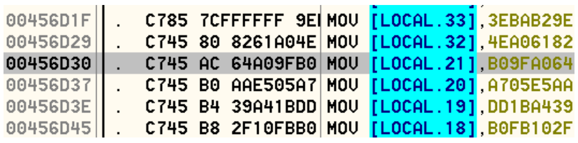


```
DS:[00E9EA30]=B09FA064
EAX=FFFFFFFF
```

EAX=FFFFFFFF and [ECX]= B09FA064

EAX is always equal to FFFFFFFF, but what about the constant B09FA064 pointed by ECX? What I did was to do a "search for all constants" on Olly (or similarly with IDA) using the value B09FA064.

The result is that the constant is moved to an array at the function sub_456BAE.



```
00456D1F  .  C785 7CFFFFFF 9EI MOV [LOCAL.33],3EBAB29E
00456D29  .  C745 80 8261A04E  MOV [LOCAL.32],4EA06182
00456D30  .  C745 AC 64A09FB0  MOV [LOCAL.21],B09FA064
00456D37  .  C745 B0 AAE505A7  MOV [LOCAL.20],A705E5AA
00456D3E  .  C745 B4 39A41BDD  MOV [LOCAL.19],DD1BA439
00456D45  .  C745 B8 2F10FBB0  MOV [LOCAL.18],B0FB102F
```

This function is really interesting. I will talk about it a little later.

Anyway at the moment what I did is to modify the value pointed by ECX to FFFFFFFF -> [ECX]=FFFFFFFF.

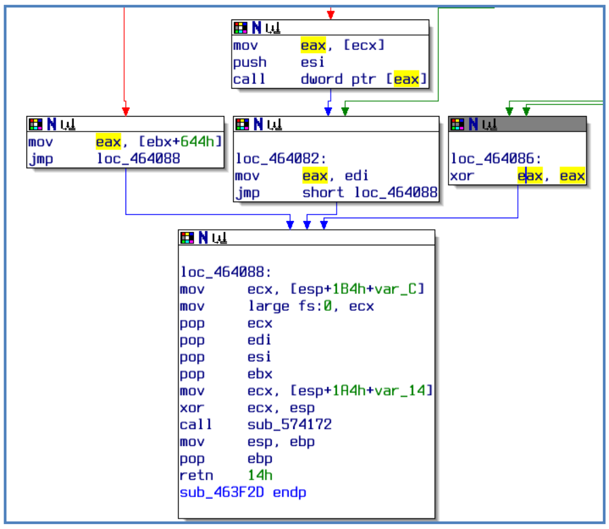
This allows skipping the first jump correctly, but I will handle the function sub_456BAE separately into one of the following sections

4.3.2 STUDYING THE JUMP 2

Now I am worrying of the second jump at the address 004643F9, which depends by the value of EAX (which shouldn't be 0 to be a goodboy), which in turn depends by the call above the cmp:

```
.text:004643F2                  call    sub_463F2D
```

The deeper you patch the more robust is the patch, then I patched the call where it zeroed out the EAX registry.



```
mov     eax, [ecx]
push    esi
call    dword ptr [eax]
```

```
mov     eax, [ebx+644h]
jmp     loc_464088
```

```
loc_464082:
mov     eax, edi
jmp     short loc_464088
```

```
loc_464086:
xor     eax, eax
```

```
loc_464088:
mov     ecx, [esp+1B4h+var_C]
mov     large fs:0, ecx
pop     ecx
pop     edi
pop     esi
pop     ebx
mov     ecx, [esp+1A4h+var_14]
xor     ecx, esp
call    sub_574172
mov     esp, ebp
pop     ebp
retn    14h
sub_463F20 endp
```

Figure 36

Figure 36 clearly shows that there's only one exit path which set to 0 the value of EAX. Than the correct place where to patch the program is here:

Original code:

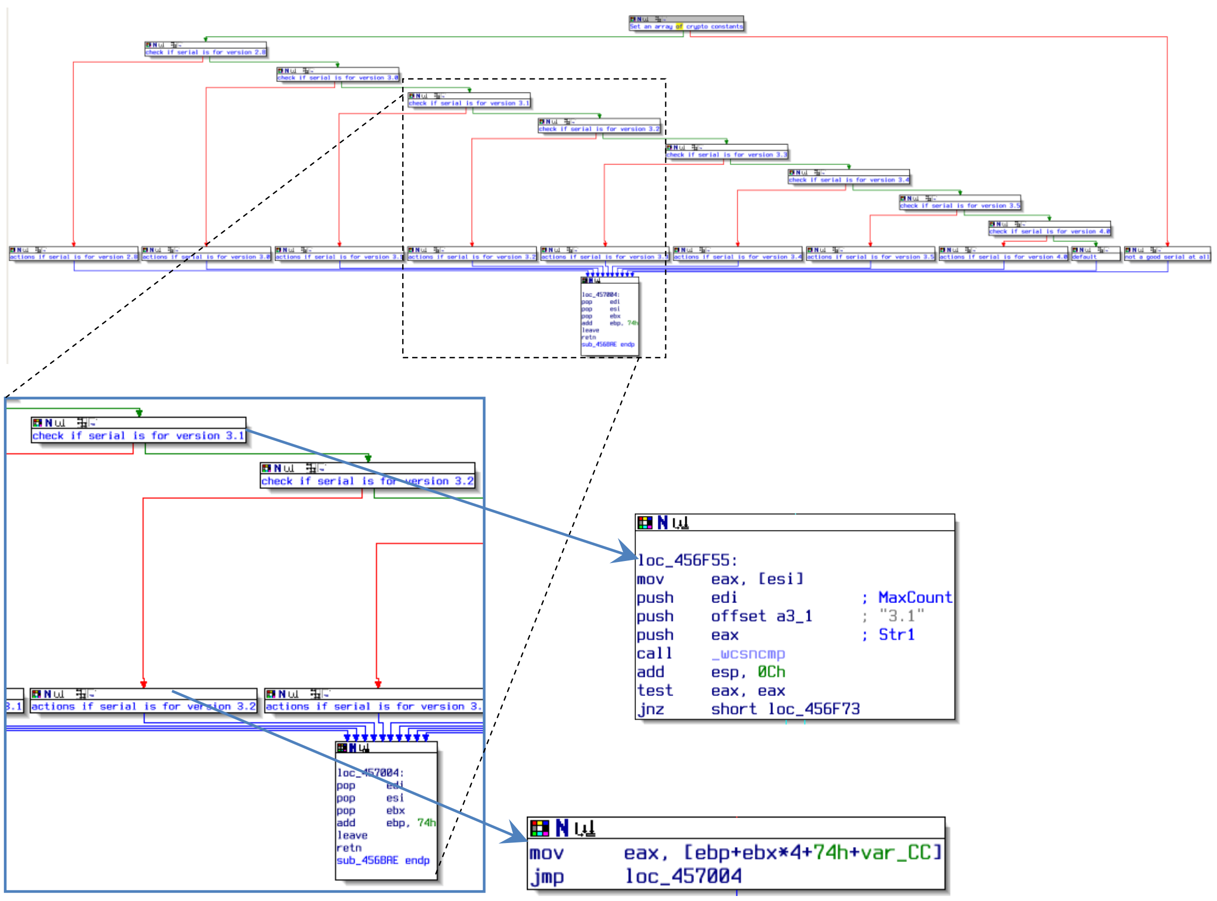
```
00464086       33C0                XOR EAX,EAX
```

Patched code: I used an opcode with same weight of the original one, using the same number of bytes (important to not overwrite surrounding instructions), that forces EAX!=0

```
00464086       B0 01               MOV AL,1
```

4.3.3 HANDLING OF FUNCTION SUB_456BAE WHERE SERIALS VERSION ARE CHECKED

As said in section 0, the function sub_456BAE is crucial for the skipping of the first jump. Its structure is very well disassembled by IDA: it's a waterfall of tests and actions (see Figure 37).


check if serial is for version 3.1
check if serial is for version 3.2
actions if serial is for version 3.2
actions if serial is for version 3.

```
loc_456F55:
mov     eax, [esi]
push    edi             ; MaxCount
push    offset a3_1     ; "3.1"
push    eax             ; Str1
call    _wcsncmp
add     esp, 0Ch
test    eax, eax
jnz     short loc_456F73
```

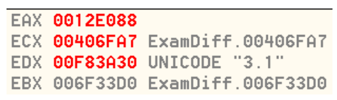
```
mov     eax, [ebp+ebx*4+74h+var_CC]
jmp     loc_457004
```

```
loc_457004:
pop     edi
pop     esi
pop     ebx
add     ebp, 74h
leave
retn
sub_456BAE endp
```

Figure 37

What this function does?

The function controls that the serial number, read from password.bin file or manually entered, is one generated for version 4.0.x of ExamDiff Pro. To answer this question, the function receives through the registry a string, holding the version number, which has been found inside the serial number.



```
EAX 0012E088
ECX 00406FA7 ExamDiff.00406FA7
EDX 00F83A30 UNICODE "3.1"
EBX 006F33D0 ExamDiff.006F33D0
```

The obvious thing to do is then, to place a BP at the beginning and see what it actually the function receives. The function receive as parameter the string "3.1" which matches with the serial I used (that was good for version 3.1)..

You can easily find this placing a BreakPoint here:

```
00456BAE   /$  55           PUSH EBP
```

EDX is equal to "3.1" in unicode

A bird-fly overview of the function tells that there's a nested sequence of jumps, checking against the given string: the branch corresponding to the string "3.1" is the following:

```
loc_456F55:
 mov     eax, [esi]
 push    edi               ; MaxCount
 push    offset a3_1       ; "3.1"
 push    eax               ; Str1
 call    _wcsncmp          ; this call is identified by IDA only..
 add     esp, 0Ch
 test    eax, eax
 jnz     short loc_456F73
```

The exit points are all MOVs of this type (here I wrote just the instructions executed when the parameter is "3.1"):

```
.text:00456F6A         mov     eax, [ebp+ebx*4+74h+var_CC]
.text:00456F6E         jmp     loc_457004  ;jump to the function exit
```

After execution of the instruction at 00456F6A we have EAX=B09FA064. This value is one of those handled at the beginning of this same function!

Reassuming then, we have a function that initializes a map of crypto constants, which are used to place a return value into EAX, starting from the version of the program found inside the license code. According to what I found also on section 0 this same constants are used to handle the first jump I identified before. These constants are compared against a value of FFFFFFFF at 004643B0.

The patch I did is an obvious one: force the function to always return FFFFFFFF so the compare at 004643B0 whatever constant will be used will be always true.

```
00456BAE      33C0              XOR EAX,EAX
00456BB0      B8 FFFFFFFF       MOV EAX,-1
00456BB5      C3                RETN
00456BB6      90                NOP
00456BB7      90                NOP
00456BB8      90                NOP
```

which is equivalent to this code:

```
sub_456BAE() {
       return -1;
}
```

This is the resulting second patch:

Original Code:

```
00456BAE   /$   55                 PUSH EBP
00456BAF   |.   8D6C24 8C          LEA EBP,DWORD PTR SS:[ESP-74]
00456BB3   |.   81EC 90010000      SUB ESP,190
```

Patched Code:

```
00456BAE        33C0               XOR EAX,EAX
00456BB0        B8 FFFFFFFF        MOV EAX,-1
00456BB5        C3                 RETN
00456BB6        90                 NOP
00456BB7        90                 NOP
00456BB8        90                 NOP
```

Note

• The function sub_456BAE is directly called only by the function sub_45F597 which is directly connected to the interpretation of password.bin file (already read from disk). This function is called a few instruction above the first of the two jumps I already described (in this case receives the buffer coming from the Activate dialogbox). These constants are then connected to the decryption of password.bin file into the resulting serial number. KANAL for PEiD doesn't reveal anything here, probably then the developers team used a proprietary crypto system, or a customized one.

4.4 TESTING WHOLE THING & CONCLUSIONS

Once you patched the program use one of the mentioned keygenerators available for older versions of ExamDiff Pro, generate a key and copy the generated passowrd.bin file, or paste the generated key in the activation window (depending on the keygenerator you'll use). Close and restart and you'll have a fully working program.

I will leave to you to properly create an efficient search&replace patcher, using an unique invariant byte pattern common to all the versions of this program (it's not that difficult), what I can tell is that this patch works for all the mentioned versions.

5 REVERSING BUSINESS TRANSLATOR 9.00 BY KAIRA

5.1 INTRODUCTION

The Target	Business Translator 9.00
Available url	http://www.zhangduo.com
The Tools	OllyDbg 1.10, PEiD
The Protection	No Protection
Category	Serial Fishing
Level	Beginner
Recommended Tools	CrackersKit 2.0

All the tools can be available on these following sites :-

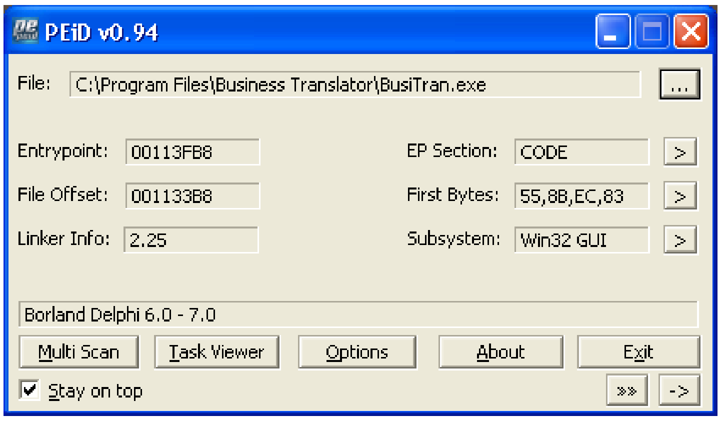
- http://home.t-online.de/home/Ollydbg --> Ollydbg and Plug-ins
- http://www.teamicu.org --> CrackersKit 2.0 which have almost all the cracking tools.

Note

•A little classical tutorial from Kaira, just to remember everybody how simple are sometimes modern programs which should apparently be well protected, or at least, not so badly unprotected. Years of reversing didn't teach anything?

5.2 APPROACHING THE ENEMY

Open PEiD. in PEiD go to Open which is "**. . .**" and load the program which is on Program Files, look at this picture below


PEiD v0.94

File: C:\Program Files\Business Translator\BusiTran.exe

Entrypoint: 00113FB8
EP Section: CODE
File Offset: 001133B8
First Bytes: 55,8B,EC,83
Linker Info: 2.25
Subsystem: Win32 GUI

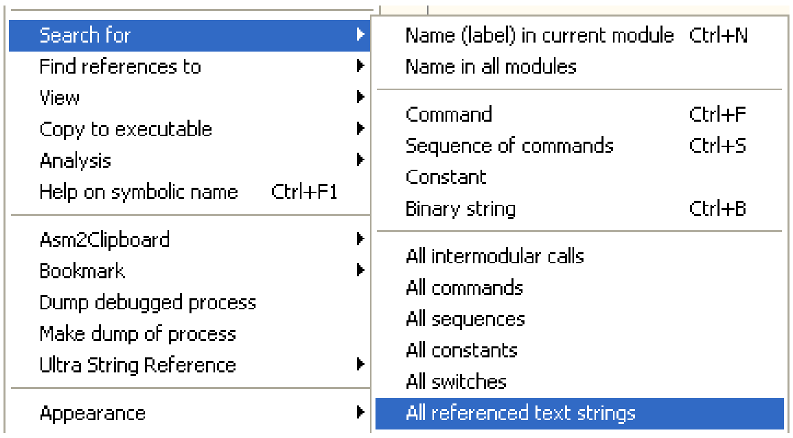
Borland Delphi 6.0 - 7.0

5.2.1 BODY

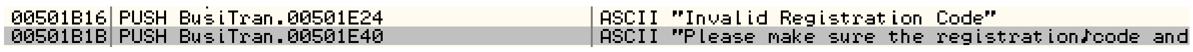
As you can see from the picture above that - the program is not protected only coded in "**Borland Delphi 6.0 – 7.0**". Lets us do the little eximanation scheme. Open up **BusiTran.exe**. It goes directly to the nag screen. I will assume that we all know what is checking for? the registration key. We are presented with a new window asking for Registration Name, and a box asking for Registration Code. Enter anything for the box of reg. name, maybe your name etc, and for the registration code, enter any numbers & press Register Now. We get a message box saying "Please make sure the registration code and the registration name are correct ". Remeber to write this message down as we will be searching for it later. Press OK and the program returns to the window asking for our Registration Code.

5.2.2 SEARCHING FOR A SERIAL

Close **BusiTran.exe** and open Ollydbg. In Olly, go to File, and Open **BusiTran.exe**, it will take few seconds to analyzes the code. We are going to start by looking for the string from the error message we recieved. To do this, right click on the code window and go to Search For, select All Referenced Text Strings. Look at this picture below.

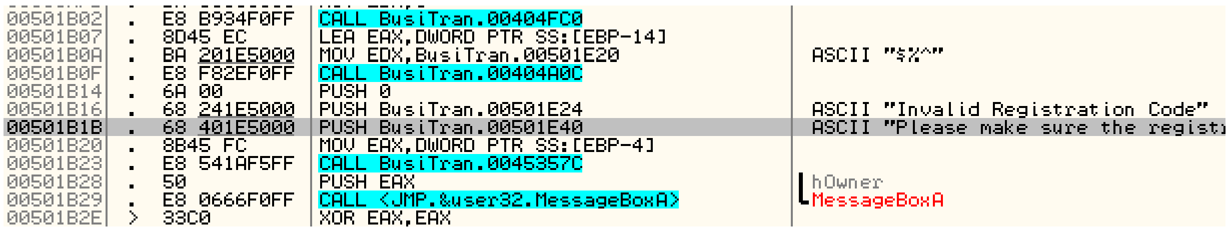


A new window will open up with all the referenced strings in the executable. Scroll to the top of the page and select the first string. Right click and choose Search For Text. In the dialog box enter "**Please make sure the registration code and the registration name are correct**" without the quotes. Make sure Case Sensitive is unchecked. You should end up here.



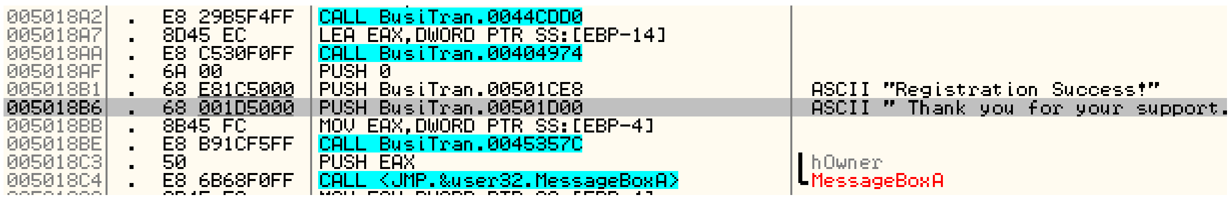
```
00501B16 PUSH BusiTran.00501E24          ASCII "Invalid Registration Code"
00501B1B PUSH BusiTran.00501E40          ASCII "Please make sure the registration♪code and
```

Double click this string. You should be here.



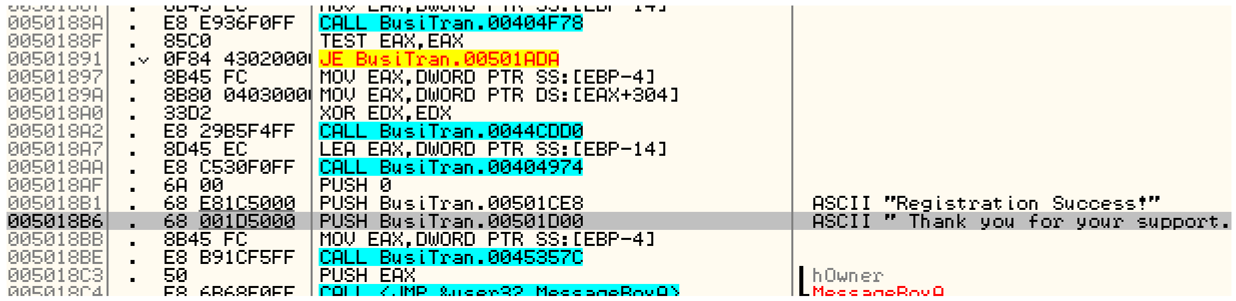
```
00501B02  .  E8 B934F0FF   CALL BusiTran.00404FC0
00501B07  .  8D45 EC       LEA EAX,DWORD PTR SS:[EBP-14]
00501B0A  .  BA 201E5000   MOV EDX,BusiTran.00501E20          ASCII "$%^"
00501B0F  .  E8 F82EF0FF   CALL BusiTran.00404A0C
00501B14  .  6A 00         PUSH 0
00501B16  .  68 241E5000   PUSH BusiTran.00501E24             ASCII "Invalid Registration Code"
00501B1B  .  68 401E5000   PUSH BusiTran.00501E40             ASCII "Please make sure the registr
00501B20  .  8B45 FC       MOV EAX,DWORD PTR SS:[EBP-4]
00501B23  .  E8 541AF5FF   CALL BusiTran.0045357C
00501B28  .  50            PUSH EAX                           hOwner
00501B29  .  E8 0666F0FF   CALL <JMP.&user32.MessageBoxA>     MessageBoxA
00501B2E  >  33C0          XOR EAX,EAX
```

This string "**Please make sure the registration code and the registration name are correct**" is the Bad Boy, Scroll up to see the Good Guy or press **Ctrl + G** on the Keyboard to go straight to the Good Guy, and type **005018B6** and press OK. You should land here.



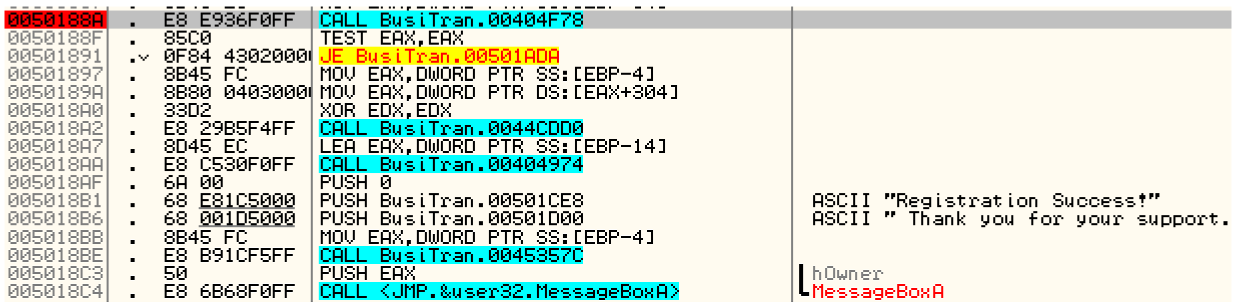
```
005018A2  .  E8 29B5F4FF   CALL BusiTran.0044CDD0
005018A7  .  8D45 EC       LEA EAX,DWORD PTR SS:[EBP-14]
005018AA  .  E8 C530F0FF   CALL BusiTran.00404974
005018AF  .  6A 00         PUSH 0
005018B1  .  68 E81C5000   PUSH BusiTran.00501CE8             ASCII "Registration Success!"
005018B6  .  68 001D5000   PUSH BusiTran.00501D00             ASCII " Thank you for your support.
005018BB  .  8B45 FC       MOV EAX,DWORD PTR SS:[EBP-4]
005018BE  .  E8 B91CF5FF   CALL BusiTran.0045357C
005018C3  .  50            PUSH EAX                           hOwner
005018C4  .  E8 6B68F0FF   CALL <JMP.&user32.MessageBoxA>     MessageBoxA
```

Scoll up a little bit to see whats happening above the Good Guy, check this picture below.



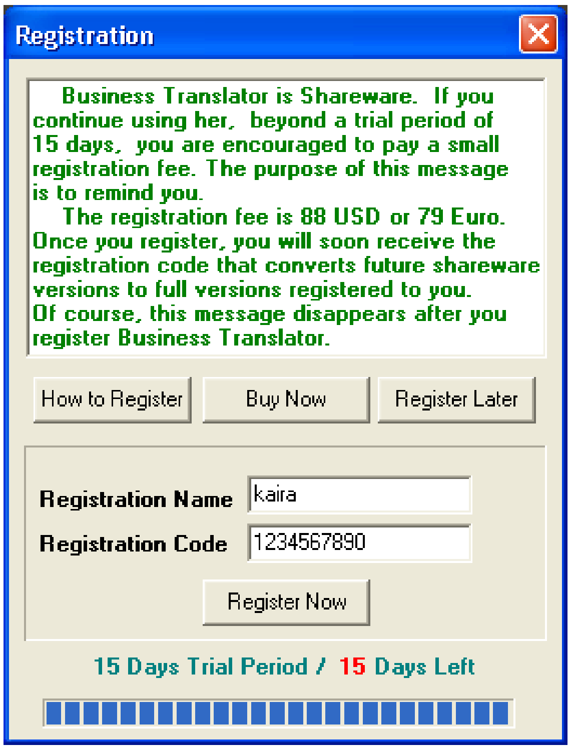
```
0050188A  .  E8 E936F0FF   CALL BusiTran.00404F78
0050188F  .  85C0          TEST EAX,EAX
00501891  .v 0F84 4302000  JE BusiTran.00501ADA
00501897  .  8B45 FC       MOV EAX,DWORD PTR SS:[EBP-4]
0050189A  .  8B80 0403000  MOV EAX,DWORD PTR DS:[EAX+304]
005018A0  .  33D2          XOR EDX,EDX
005018A2  .  E8 29B5F4FF   CALL BusiTran.0044CDD0
005018A7  .  8D45 EC       LEA EAX,DWORD PTR SS:[EBP-14]
005018AA  .  E8 C530F0FF   CALL BusiTran.00404974
005018AF  .  6A 00         PUSH 0
005018B1  .  68 E81C5000   PUSH BusiTran.00501CE8          ASCII "Registration Success!"
005018B6  .  68 001D5000   PUSH BusiTran.00501D00          ASCII " Thank you for your support.
005018BB  .  8B45 FC       MOV EAX,DWORD PTR SS:[EBP-4]
005018BE  .  E8 B91CF5FF   CALL BusiTran.0045357C
005018C3  .  50            PUSH EAX                        hOwner
```

Above the Good Guy there is **JE BusiTran.00501ADA** which Jump straight to the side of a Bad Boy, so our valid serial number is stored on this CALL BusiTran.00404F78 above TEST EAX, EAX, so lets set a BreakPoint (F2) on this Call, check this picture below.



```
0050188A  .  E8 E936F0FF   CALL BusiTran.00404F78
0050188F  .  85C0          TEST EAX,EAX
00501891  .v 0F84 4302000  JE BusiTran.00501ADA
00501897  .  8B45 FC       MOV EAX,DWORD PTR SS:[EBP-4]
0050189A  .  8B80 0403000  MOV EAX,DWORD PTR DS:[EAX+304]
005018A0  .  33D2          XOR EDX,EDX
005018A2  .  E8 29B5F4FF   CALL BusiTran.0044CDD0
005018A7  .  8D45 EC       LEA EAX,DWORD PTR SS:[EBP-14]
005018AA  .  E8 C530F0FF   CALL BusiTran.00404974
005018AF  .  6A 00         PUSH 0
005018B1  .  68 E81C5000   PUSH BusiTran.00501CE8          ASCII "Registration Success!"
005018B6  .  68 001D5000   PUSH BusiTran.00501D00          ASCII " Thank you for your support.
005018BB  .  8B45 FC       MOV EAX,DWORD PTR SS:[EBP-4]
005018BE  .  E8 B91CF5FF   CALL BusiTran.0045357C
005018C3  .  50            PUSH EAX                        hOwner
005018C4  .  E8 6B68F0FF   CALL <JMP.&user32.MessageBoxA>  MessageBoxA
```

Press **(F9)** on the keyboard to run the Program and the nagscreen that will appear - Enter your Registration Name & Registration Code, and click Register Now button look at this picture below.



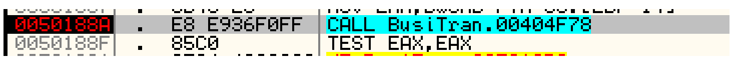
Registration

Business Translator is Shareware. If you continue using her, beyond a trial period of 15 days, you are encouraged to pay a small registration fee. The purpose of this message is to remind you.
The registration fee is 88 USD or 79 Euro. Once you register, you will soon receive the registration code that converts future shareware versions to full versions registered to you.
Of course, this message disappears after you register Business Translator.

Registration Name: kaira
Registration Code: 1234567890

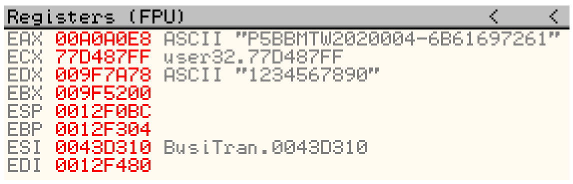
15 Days Trial Period / 15 Days Left

Move back to Olly, after you have pressed **Register Now** button it breaks at the breakpoint we set, look at this picture below.



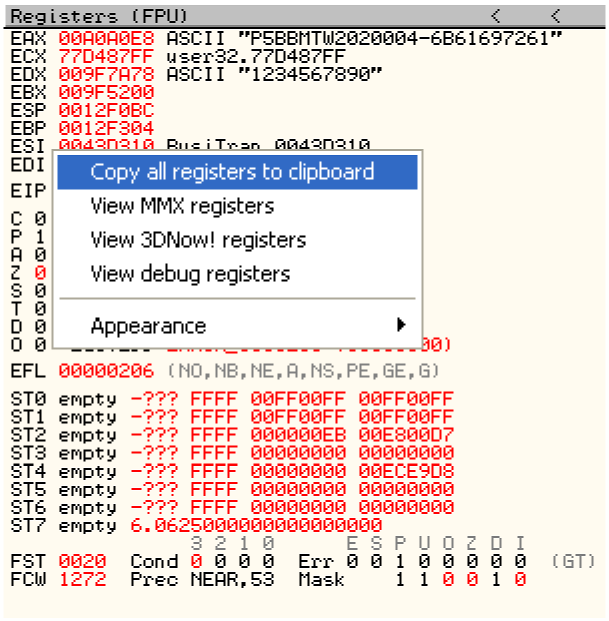
```
0050188A  .  E8 E936F0FF  CALL BusiTran.00404F78
0050188F  .  85C0         TEST EAX,EAX
```

Look also at the **Registers (FPU)** - EAX 00A0A0E8 ASCII "P5BBMTW2020004-6B61697261" --> my valid serial number!!!



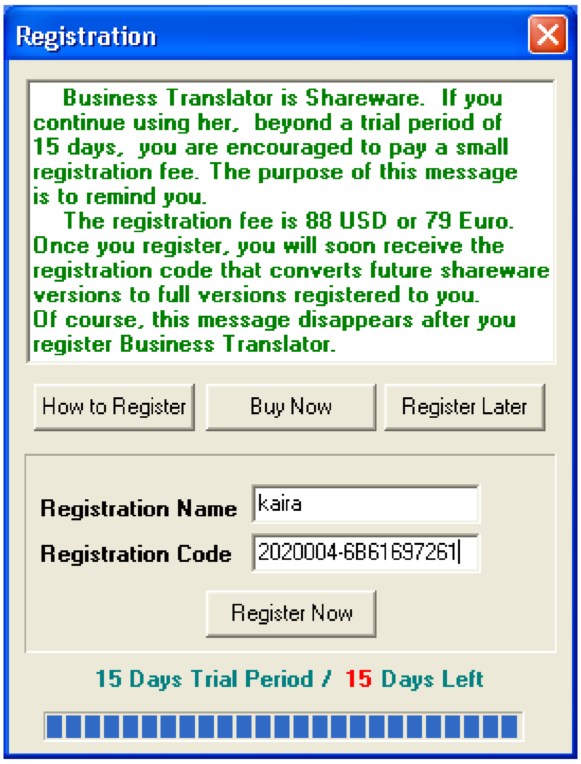
```
Registers (FPU)
EAX 00A0A0E8 ASCII "P5BBMTW2020004-6B61697261"
ECX 77D487FF user32.77D487FF
EDX 009F7A78 ASCII "1234567890"
EBX 009F5200
ESP 0012F0BC
EBP 0012F304
ESI 0043D310 BusiTran.0043D310
EDI 0012F480
```

Right Click on **Registers (FPU)** and the menu that appears after you've right clicked - click Copy all registers to clipboard, and paste it with **Notepad.exe.** Look at this picture below.



```
Registers (FPU)
EAX 00A0A0E8 ASCII "P5BBMTW2020004-6B61697261"
ECX 77D487FF user32.77D487FF
EDX 009F7A78 ASCII "1234567890"
EBX 009F5200
ESP 0012F0BC
EBP 0012F304
ESI
EDI

EIP

C 0
P 1
A 0
Z 0
S 0
T 0
D 0
O 0

EFL 00000206 (NO,NB,NE,A,NS,PE,GE,G)

ST0 empty -??? FFFF 00FF00FF 00FF00FF
ST1 empty -??? FFFF 00FF00FF 00FF00FF
ST2 empty -??? FFFF 000000EB 00E800D7
ST3 empty -??? FFFF 00000000 00000000
ST4 empty -??? FFFF 00000000 00ECE9D8
ST5 empty -??? FFFF 00000000 00000000
ST6 empty -??? FFFF 00000000 00000000
ST7 empty 6.0625000000000000000
               3 2 1 0      E S P U O Z D I
FST 0020  Cond 0 0 0 0  Err 0 0 1 0 0 0 0 0  (GT)
FCW 1272  Prec NEAR,53  Mask    1 1 0 0 1 0
```

Restart Ollydbg Ctrl + F2 on the keyboard or press Restart button on Ollydbg & remove the Breakpoint and press (F9) on the keyboard to run the program or Play button on Ollydbg, when the nagscreen appears - Enter your Registration Name & Registration Code, look at the picture below.



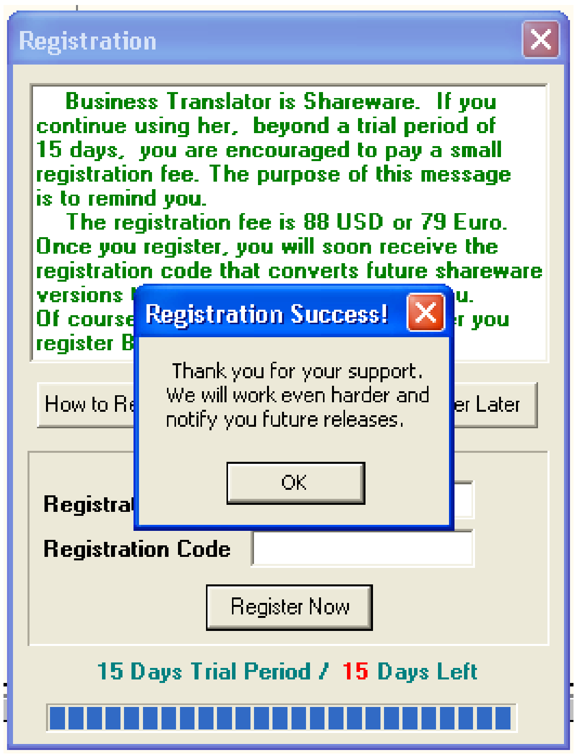
Registration

Business Translator is Shareware. If you continue using her, beyond a trial period of 15 days, you are encouraged to pay a small registration fee. The purpose of this message is to remind you.
The registration fee is 88 USD or 79 Euro. Once you register, you will soon receive the registration code that converts future shareware versions to full versions registered to you.
Of course, this message disappears after you register Business Translator.

Registration Name: kaira
Registration Code: 2020004-6B61697261

15 Days Trial Period / 15 Days Left

Click Register Now button, look at this picture below.

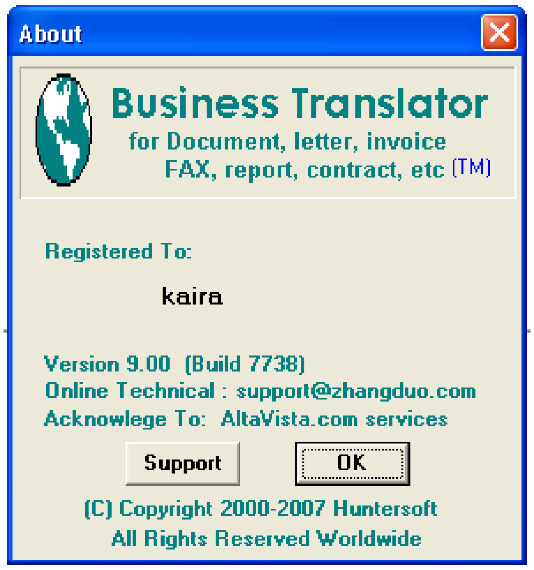


Registration Success!

Thank you for your support. We will work even harder and notify you future releases.

Registeration Success! Click **OK** button.

Your can close Ollydbg now because we've finished serial fishing Business Translator 9.00. Open **BusiTran.exe** and click **Help -> About...** button from the menu, look at this picture below.



Business Translator
for Document, letter, invoice
FAX, report, contract, etc (TM)

Registered To:

kaira

Version 9.00 (Build 7738)
Online Technical : support@zhangduo.com
Acknowlege To: AltaVista.com services

(C) Copyright 2000-2007 Huntersoft
All Rights Reserved Worldwide

Job Well Done!!!

5.3 CONCLUSION

If you like the program and going to use it please purchase it, because programmers rely on the income in-order to continue updating their softwares for us!

6 EXECRYPTOR FOR DUMMIES OR HOW TO UNPACK EXECRYPTOR 2.4 WITHOUT HAVING A CLUE WHAT YOU ARE DOING BY HAGGAR

6.1 INTRODUCTION

Hi and welcome to this small ExeCryptor unpacking guide!

You may wonder why another ExeCryptor paper? To tell you the truth, I'm not interested in writing another ExeCryptor paper (I have already wrote four of them), but Shub-Nigurrath asked me to contribute new edition of ARTEAM ezine. So why not contribute.

This guide will not explain ExeCryptor in details. Actually, it will not exaplain ExeCryptor at all. This guide will exaplain how to unpack ExeCryptor with as less possible knowledge about unpacking.

Happy unpacking!

6.2 CONTENT

1. Requirements for this guide

2. Preparations before loading target in Olly

3. Loading target in Olly

4. Using script to kill anti-debug tricks

5. Finding OEP

6. Using script to decrypt imports

7. Dumping to hard disc

9. Reference material

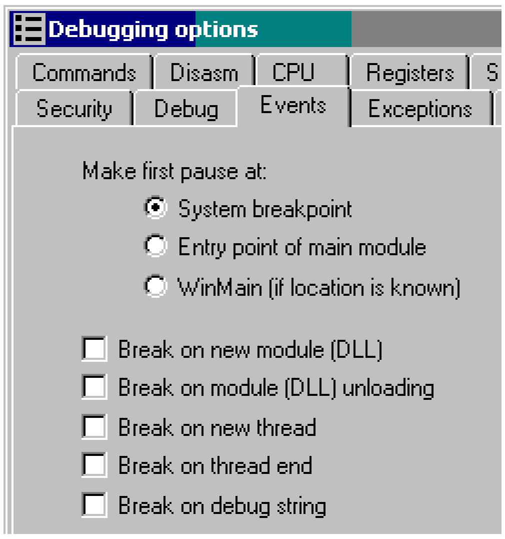
6.3 [1] REQUIREMENTS FOR THIS GUIDE

- OllyDbg 1.10
- Windows XP (or some other NT based windows)
- OllyScript or ODbgScript plugin
- LordPE
- ImpREC
- Some hex editor

You will also need some target application protected with ExeCryptor. Any target should do, it also doesn't mathers exact version of ExeCryptor. I will use "**mp3tag 5.4 lite**" that is protected with ExeCryptor 2.4.

6.4 [2] PREPARATIONS BEFORE LOADING TARGET IN OLLY

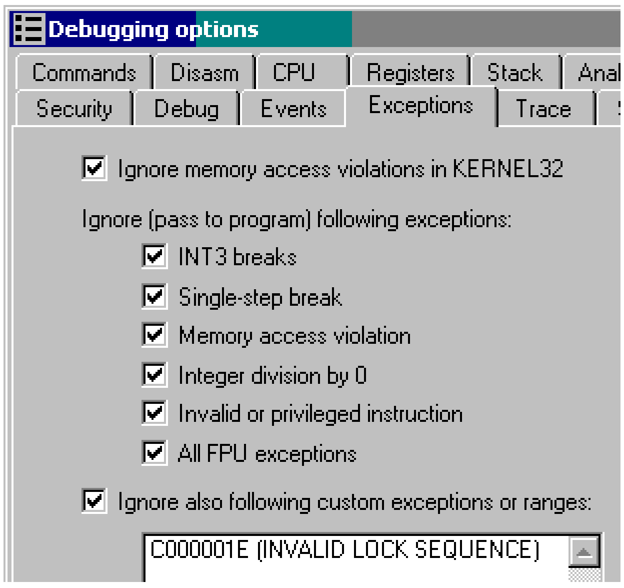
- Disable all plugins that's purpose is to hide Olly from various anti-debug tricks. Plugins such as HideOlly, OllyAdvanced, etc... can be detected by ExeCryptor.
- Set Olly to break at system breakpoint (check image below):



Make first pause at:
(●) System breakpoint
( ) Entry point of main module
( ) WinMain (if location is known)

☐ Break on new module (DLL)
☐ Break on module (DLL) unloading
☐ Break on new thread
☐ Break on thread end
☐ Break on debug string

- Ignore all exceptions and add C000001E to custom ones (check image below):



☑ Ignore memory access violations in KERNEL32

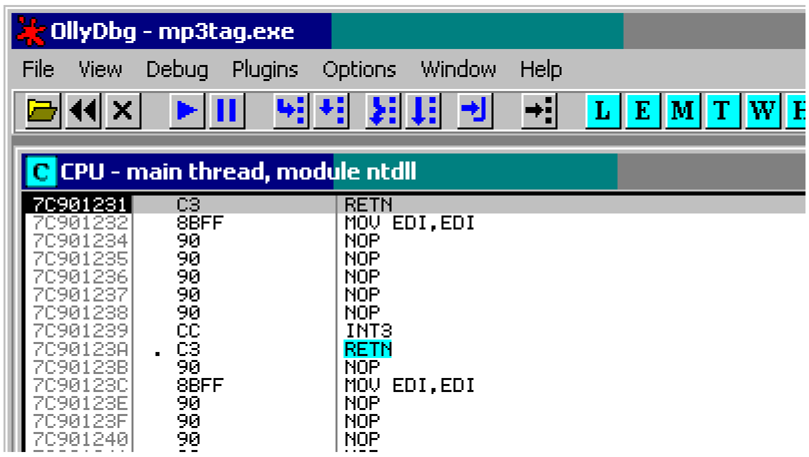
Ignore (pass to program) following exceptions:
☑ INT3 breaks
☑ Single-step break
☑ Memory access violation
☑ Integer division by 0
☑ Invalid or privileged instruction
☑ All FPU exceptions

☑ Ignore also following custom exceptions or ranges:

C000001E (INVALID LOCK SEQUENCE)

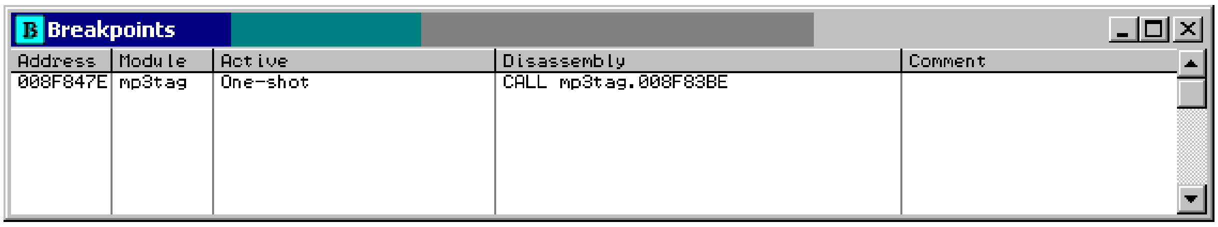
6.5 [3] LOADING TARGET IN OLLY

Now, after we properly configured Olly, we can load target. Since we set Olly to break at system breakpoint, we will be in ntdll.dll:


OllyDbg - mp3tag.exe
File View Debug Plugins Options Window Help
L E M T W H
CPU - main thread, module ntdll

```
7C901231    C3       RETN
7C901232    8BFF     MOV EDI,EDI
7C901234    90       NOP
7C901235    90       NOP
7C901236    90       NOP
7C901237    90       NOP
7C901238    90       NOP
7C901239    CC       INT3
7C90123A  . C3       RETN
7C90123B    90       NOP
7C90123C    8BFF     MOV EDI,EDI
7C90123E    90       NOP
7C90123F    90       NOP
7C901240    90       NOP
```

That is OK. But Olly is not yet ready to debug target. Hit **Alt+B** to open breakpoint window:



```
Address   Module  Active    Disassembly            Comment
008F847E  mp3tag  One-shot  CALL mp3tag.008F83BE
```

In this window you can see all breakpoints that are present. OllyDbg automaticly places one temporary breakpoint to the EntryPoint of debugged program. Delete this bp by selecting it with right-click and chosing "Remove Del".

As a final step, delete all possible hardware or memory breakpoints too.

6.6 [4] USING SCRIPT TO KILL ANTI-DEBUG TRICKS

Now we are ready to use first script that will kill all ExeCryptor anti-Olly tricks. First script is:

ExeCryptor 2.x OEP.txt

Although this script is called OEP script, it will not find OEP. That name is left from the beggining when I thought that I will write script for finding OEP on all 2.x ExeCryptor versions. But that looks impossible. ExeCryptor code is different even in same versions of protection.

But this script will kill all tricks that ExeCryptor uses to detect or kill Olly. After using it, you can hit **Shift+F9** and your target should run fine under Olly.

So, for the slow ones: Don't cry at forums that script doesn't work! It will not find OEP! It will kill all anti-debug traps and tricks! You have to find OEP manually.

6.7 [5] FINDING OEP

Ha! Now is time for you to use your skills and experience. There are no generic way to find OEP and ExeCryptor code is impossible to trace. For this step you need to have some experience.

Do you know how OEP of different compilers look? Any of these looks familiar:

Visual Basic

Deplhi

Borland 1999 C++

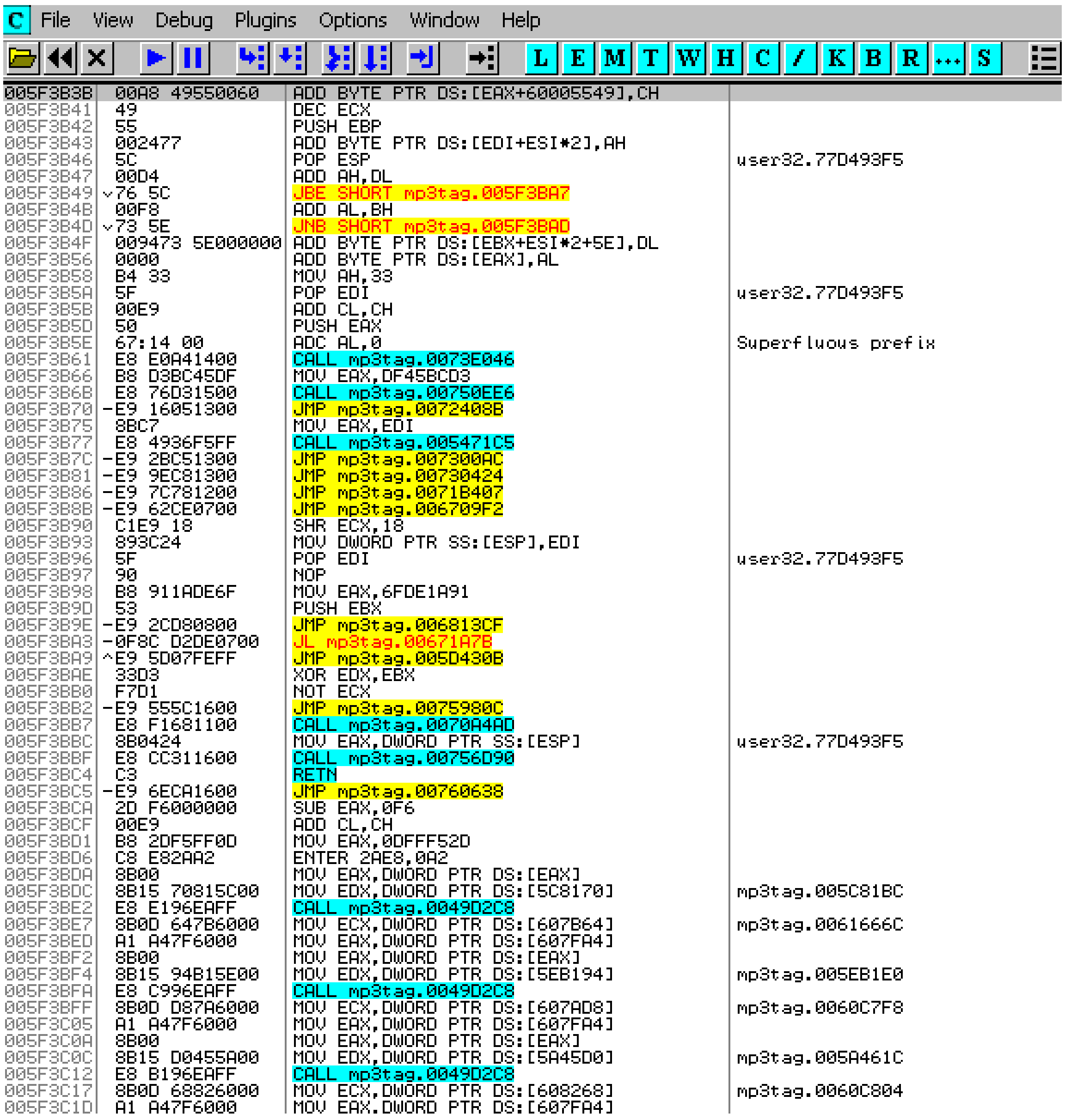
MSVC++ versions

etc...

If you have experience and you could recognize these OEP's, then you will have no problem finding OEP in ExeCryptor protected applications. If you don't have idea what I'm talking about, than you are lost. Sorry, no help for you.

Since it there is no generic way to find OEP, we will try to find OEP by using our experience. First script has killed all anti-debug tricks and we can run application under Olly. Run it, then pause it (**F12**), and let's have look at it. If we go to first section, which is at 401000 in my target, we can see that app in my example is compiled in Deplhi.

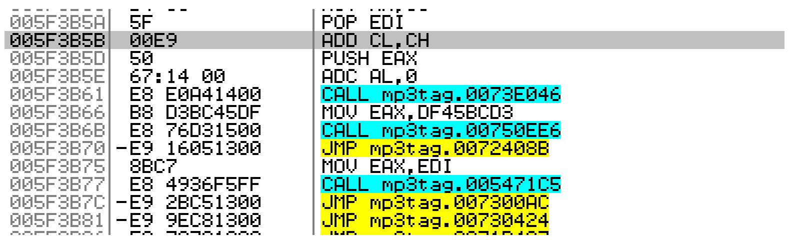
Where Deplhi applications have OEP? Always at the end of code section. So I scroll down to the end of code section in order to find OEP. But instead of finding OEP, I see some weird jumps and calls:



```
005F3B3B  00A8 49550060    ADD BYTE PTR DS:[EAX+60005549],CH
005F3B41  49               DEC ECX
005F3B42  55               PUSH EBP
005F3B43  002477           ADD BYTE PTR DS:[EDI+ESI*2],AH
005F3B46  5C               POP ESP                                  user32.77D493F5
005F3B47  00D4             ADD AH,DL
005F3B49 v76 5C            JBE SHORT mp3tag.005F3BA7
005F3B4B  00F8             ADD AL,BH
005F3B4D v73 5E            JNB SHORT mp3tag.005F3BAD
005F3B4F  009473 5E000000  ADD BYTE PTR DS:[EBX+ESI*2+5E],DL
005F3B56  0000             ADD BYTE PTR DS:[EAX],AL
005F3B58  B4 33            MOV AH,33
005F3B5A  5F               POP EDI                                  user32.77D493F5
005F3B5B  00E9             ADD CL,CH
005F3B5D  50               PUSH EAX
005F3B5E  67:14 00         ADC AL,0                                 Superfluous prefix
005F3B61  E8 E0A41400      CALL mp3tag.0073E046
005F3B66  B8 D3BC45DF      MOV EAX,DF45BCD3
005F3B6B  E8 76D31500      CALL mp3tag.00750EE6
005F3B70 -E9 16051300      JMP mp3tag.0072408B
005F3B75  8BC7             MOV EAX,EDI
005F3B77  E8 4936F5FF      CALL mp3tag.005471C5
005F3B7C -E9 2BC51300      JMP mp3tag.007300AC
005F3B81 -E9 9EC81300      JMP mp3tag.00730424
005F3B86 -E9 7C781200      JMP mp3tag.0071B407
005F3B8B -E9 62CE0700      JMP mp3tag.006709F2
005F3B90  C1E9 18          SHR ECX,18
005F3B93  893C24           MOV DWORD PTR SS:[ESP],EDI
005F3B96  5F               POP EDI                                  user32.77D493F5
005F3B97  90               NOP
005F3B98  B8 911ADE6F      MOV EAX,6FDE1A91
005F3B9D  53               PUSH EBX
005F3B9E -E9 2CD80800      JMP mp3tag.006813CF
005F3BA3 -0F8C D2DE0700    JL mp3tag.00671A7B
005F3BA9 ^E9 5D07FEFF      JMP mp3tag.005D430B
005F3BAE  33D3             XOR EDX,EBX
005F3BB0  F7D1             NOT ECX
005F3BB2 -E9 555C1600      JMP mp3tag.0075980C
005F3BB7  E8 F1681100      CALL mp3tag.0070A4AD
005F3BBC  8B0424           MOV EAX,DWORD PTR SS:[ESP]               user32.77D493F5
005F3BBF  E8 CC311600      CALL mp3tag.00756D90
005F3BC4  C3               RETN
005F3BC5 -E9 6ECA1600      JMP mp3tag.00760638
005F3BCA  2D F6000000      SUB EAX,0F6
005F3BCF  00E9             ADD CL,CH
005F3BD1  B8 2DF5FF0D      MOV EAX,0DFFF52D
005F3BD6  C8 E82AA2        ENTER 2AE8,0A2
005F3BDA  8B00             MOV EAX,DWORD PTR DS:[EAX]
005F3BDC  8B15 70815C00    MOV EDX,DWORD PTR DS:[5C8170]            mp3tag.005C81BC
005F3BE2  E8 E196EAFF      CALL mp3tag.0049D2C8
005F3BE7  8B0D 647B6000    MOV ECX,DWORD PTR DS:[607B64]            mp3tag.0061666C
005F3BED  A1 A47F6000      MOV EAX,DWORD PTR DS:[607FA4]
005F3BF2  8B00             MOV EAX,DWORD PTR DS:[EAX]
005F3BF4  8B15 94B15E00    MOV EDX,DWORD PTR DS:[5EB194]            mp3tag.005EB1E0
005F3BFA  E8 C996EAFF      CALL mp3tag.0049D2C8
005F3BFF  8B0D D87A6000    MOV ECX,DWORD PTR DS:[607AD8]            mp3tag.0060C7F8
005F3C05  A1 A47F6000      MOV EAX,DWORD PTR DS:[607FA4]
005F3C0A  8B00             MOV EAX,DWORD PTR DS:[EAX]
005F3C0C  8B15 D0455A00    MOV EDX,DWORD PTR DS:[5A45D0]            mp3tag.005A461C
005F3C12  E8 B196EAFF      CALL mp3tag.0049D2C8
005F3C17  8B0D 68826000    MOV ECX,DWORD PTR DS:[608268]            mp3tag.0060C804
005F3C1D  A1 A47F6000      MOV EAX,DWORD PTR DS:[607FA4]
```

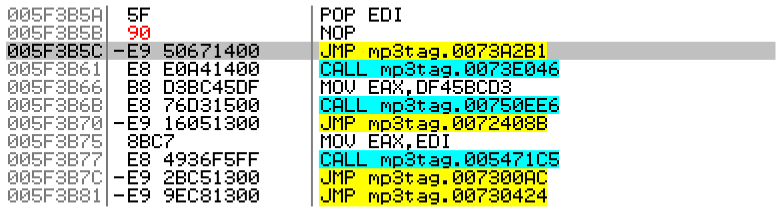
What we got here is **Stolen OEP Code**. Instead OEP, we see some weird code. By mine experience, ExeCryptor replaces first opcode at OEP with long jump (E9 xxxxxxxx) that leads to obfuscated code. **This jump at OEP will never be executed!** That jump is there just for a saftey purpose in case that something tries to use that code. Real OEP code is obfuscated at address where that jump points.

How we can find where is stolen OEP code? First we need to find long jump. Code in CPU looks bad because Olly didn't analyse it and we cannot see that jump right away. We can see some jumps and calls that leads to ExeCryptor sections. Above them code looks really messy. From there I search binary for first E9 byte:



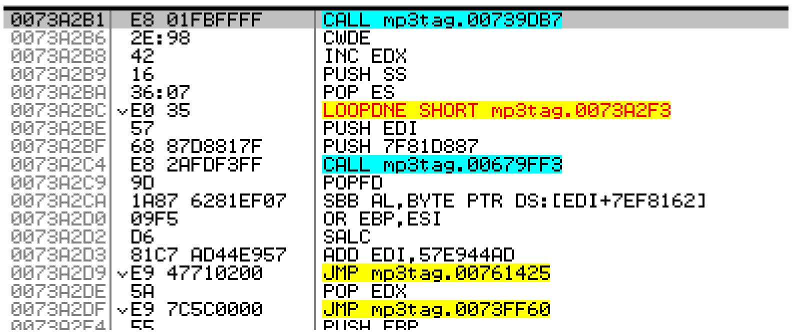
```
005F3B5A   5F              POP EDI
005F3B5B   00E9            ADD CL,CH
005F3B5D   50              PUSH EAX
005F3B5E   67:14 00        ADC AL,0
005F3B61   E8 E0A41400     CALL mp3tag.0073E046
005F3B66   B8 D3BC45DF     MOV EAX,DF45BCD3
005F3B6B   E8 76D31500     CALL mp3tag.00750EE6
005F3B70  -E9 16051300     JMP mp3tag.0072408B
005F3B75   8BC7            MOV EAX,EDI
005F3B77   E8 4936F5FF     CALL mp3tag.005471C5
005F3B7C  -E9 2BC51300     JMP mp3tag.007300AC
005F3B81  -E9 9EC81300     JMP mp3tag.00730424
```

Byte E9 is there. I patch 00 byte before E9 to see better:



```
005F3B5A   5F              POP EDI
005F3B5B   90              NOP
005F3B5C  -E9 50671400     JMP mp3tag.0073A2B1
005F3B61   E8 E0A41400     CALL mp3tag.0073E046
005F3B66   B8 D3BC45DF     MOV EAX,DF45BCD3
005F3B6B   E8 76D31500     CALL mp3tag.00750EE6
005F3B70  -E9 16051300     JMP mp3tag.0072408B
005F3B75   8BC7            MOV EAX,EDI
005F3B77   E8 4936F5FF     CALL mp3tag.005471C5
005F3B7C  -E9 2BC51300     JMP mp3tag.007300AC
005F3B81  -E9 9EC81300     JMP mp3tag.00730424
```

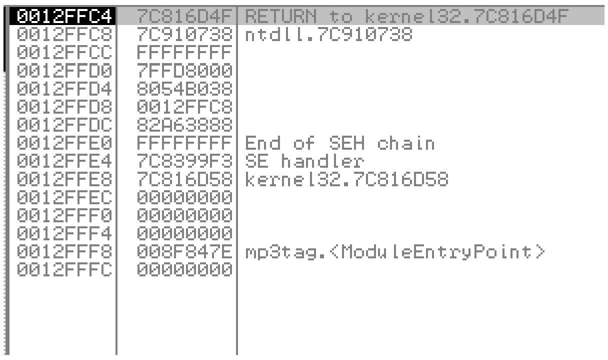
Now it's clear that this jump leads to section with ExeCryptor code. And by mine logic, that code should be stolen code:



```
0073A2B1   E8 01FBFFFF     CALL mp3tag.00739DB7
0073A2B6   2E:98           CWDE
0073A2B8   42              INC EDX
0073A2B9   16              PUSH SS
0073A2BA   36:07           POP ES
0073A2BC  vE0 35           LOOPDNE SHORT mp3tag.0073A2F3
0073A2BE   57              PUSH EDI
0073A2BF   68 87D8817F     PUSH 7F81D887
0073A2C4   E8 2AFDF3FF     CALL mp3tag.00679FF3
0073A2C9   9D              POPFD
0073A2CA   1A87 6281EF07   SBB AL,BYTE PTR DS:[EDI+7EF8162]
0073A2D0   09F5            OR EBP,ESI
0073A2D2   D6              SALC
0073A2D3   81C7 AD44E957   ADD EDI,57E944AD
0073A2D9  vE9 47710200     JMP mp3tag.00761425
0073A2DE   5A              POP EDX
0073A2DF  vE9 7C5C0000     JMP mp3tag.0073FF60
```

What now? Reastart application in olly, remove all breakpoints again, use script for killing traps again.

Now, go to that address where we think that stolen OEP should be. You should see nothing but zeros there now. Place **memory breakpoint on access** at that line. Keep pressing **Shift+F9** untill you break there. After couple stops in decrypt procedures, breakpoint check (yes it checks for beakpoint on that address), I stoped at that line. Fact that there is breakpoint check on that line is first confirmation of stolen code idea. If we check stack, we will get final confirmation:



```
0012FFC4  7C816D4F  RETURN to kernel32.7C816D4F
0012FFC8  7C910738  ntdll.7C910738
0012FFCC  FFFFFFFF
0012FFD0  7FFD8000
0012FFD4  8054B038
0012FFD8  0012FFC8
0012FFDC  82A63888
0012FFE0  FFFFFFFF  End of SEH chain
0012FFE4  7C8399F3  SE handler
0012FFE8  7C816D58  kernel32.7C816D58
0012FFEC  00000000
0012FFF0  00000000
0012FFF4  00000000
0012FFF8  008F847E  mp3tag.<ModuleEntryPoint>
0012FFFC  00000000
```

Stack looks like one in some target that is just loaded in Olly.

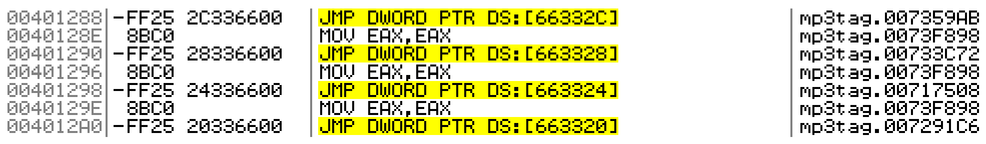
We have found stolen code. Btw, don't try find original opcodes. It is impossible.

6.8 [6] USING SCRIPT TO DECRYPT IMPORTS

We have found OEP, or stolen code, and now we have to find imports. For this we will use my second script:

ExeCryptor 2.x Delphi IAT.txt

This script finds imports in Delphi, Borland and MASM compiled application. Those applications use jumps to access imports (MSVC++ use calls). Let's see example in my target:



```
00401288 -FF25 2C336600   JMP DWORD PTR DS:[66332C]     mp3tag.007359AB
0040128E  8BC0            MOV EAX,EAX                   mp3tag.0073F898
00401290 -FF25 28336600   JMP DWORD PTR DS:[663328]     mp3tag.00733C72
00401296  8BC0            MOV EAX,EAX                   mp3tag.0073F898
00401298 -FF25 24336600   JMP DWORD PTR DS:[663324]     mp3tag.00717508
0040129E  8BC0            MOV EAX,EAX                   mp3tag.0073F898
004012A0 -FF25 20336600   JMP DWORD PTR DS:[663320]     mp3tag.007291C6
```

My script for fixing imports needs to be changed a little for evenry new target. If you open script in Notepad, you will see this line:

find addr,#ff25????9999#   //THIS LINE NEEDS TO BE CHANGED!!!

That line is search pattern that finds all possible import jumps in code section. You can notice that import jumps in my target have these bytes (check image above):

FF25 xxxx6600

So that is byte mask for my example and I will set script to find those paterns:

find addr,#ff25????6600#   //THIS LINE NEEDS TO BE CHANGED!!!

Now script is ready to find imports. Just use it. Wait untill it finishes (that can last).

6.9 [7] DUMPING TO HARD DISC

- If script for fixing imports has finished without error, then you can use ImpREC to retrieve imports.
- Dump target to hard disk with LordPE or any similar dumping engine.
- Rebuild imports with ImpREC.

And that is it! Dumped target already works (at least in my case). But there could be some problems:

- First, you should change TLS settings in PE header. I will not explain here how and what to do it. I have already explained that in one of my previous tutorial. Check references.
- Second, if target has stolen code, it will probably work only on machine where it was dumped. I have already explained that problem in paper about official ExeCryptor crackme. Check references.

6.10 [9] REFERENCE MATERIAL

Here you can find scripts for unpacking ExeCryptor and some extra info about protection. Anti debug tricks, how scripts work, etc... all can be found in my previous tutorials. Don't be lazy. Read them.

At BIW reversing (http://www.reversing.be/) you can find four tutorials about ExeCryptor:

- "ExeCryptor official crackme" - Deplhi target, full protection, ExeCryptor 2.1.17, very complete tutorial.
- "Unpacking ExeCryptor 2.2.4" - Borland 1999 C++ target, no stolen code, completly removing ExeCryptor layer.
- "ExeCryptor 2.2.50 - unpacking MSVC+ target" - Unpacking MSVC++ target.
- "ExeCryptor 2.3.9" - just some thoughts on this version.

6.11 SCRIPTS

- Script "**ExeCryptor 2.x OEP.txt**". Just paste it in some text file. Script is also full of comments, so you can learn how it works:
- Script "**ExeCryptor 2.x Delphi IAT.txt**":
- Script "**ExeCryptor 2.x MSVC++ IAT.txt**". This script works little different. It is explained in one of previous tutorials:

7 OCR TOOLS WALKTHROUGH OF KEY CHECK ROUTINE BY ANHS!RK

7.1 INTRODUCTION

The following text is taken directly from Application (www.ocrtools.com).

OCRTools presents state-of-the-art Optical Character Recognition products developed entirely within the Microsoft .Net platform. Incorporating Neural Networks, Artificial Intelligence, and trained with over 4 million font variations; our products incorporate the latest optical character recognition technology to solve your OCR problems. And we offer OCR and a Barcode API/SDK, as well as Desktop Solutions.

All source code has been written in Microsoft VB.Net & C#.Net and compiled as Safe and Managed Code in the Microsoft .Net Framework.

7.2 TOOL REQUIRED

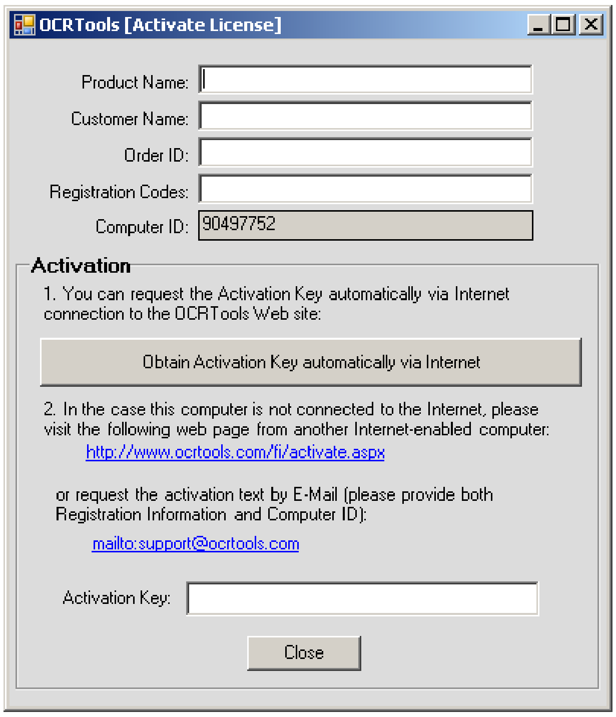
The main tool you ever required is BRAIN ☺ for doing some Logic and Reasoning. You will also need Reflector (the version I used was 5.0.50.0).

7.3 WALKTHROUGH

We need to find the following fields

- Product Name
- Registration Codes
- Activation Key

Customer Name & Order ID can be any thing


OCRTools [Activate License]

Product Name:
Customer Name:
Order ID:
Registration Codes:
Computer ID: 90497752

Activation

1. You can request the Activation Key automatically via Internet connection to the OCRTools Web site:

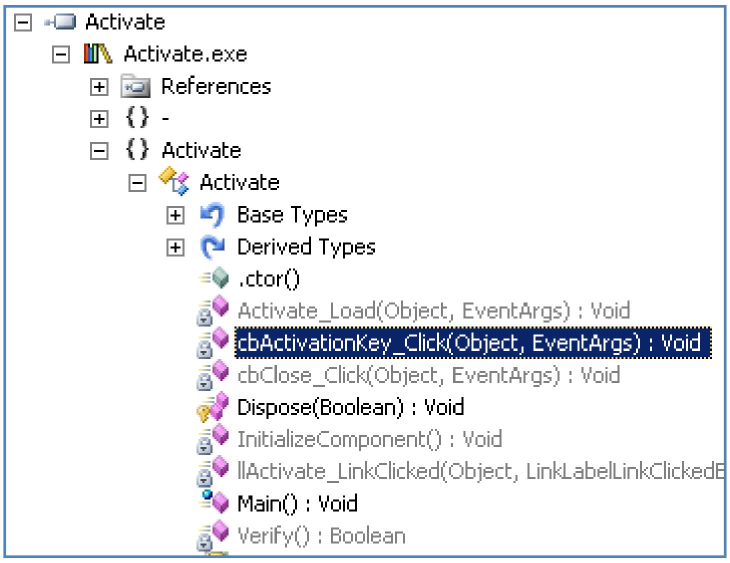
2. In the case this computer is not connected to the Internet, please visit the following web page from another Internet-enabled computer:
http://www.ocrtools.com/fi/activate.aspx

or request the activation text by E-Mail (please provide both Registration Information and Computer ID):

mailto:support@ocrtools.com

Activation Key:

Okay let's find out what is Product Name.


```
Activate
  Activate.exe
    References
    {} -
    {} Activate
      Activate
        Base Types
        Derived Types
        .ctor()
        Activate_Load(Object, EventArgs) : Void
        cbActivationKey_Click(Object, EventArgs) : Void
        cbClose_Click(Object, EventArgs) : Void
        Dispose(Boolean) : Void
        InitializeComponent() : Void
        llActivate_LinkClicked(Object, LinkLabelLinkClickedE
        Main() : Void
        Verify() : Boolean
```

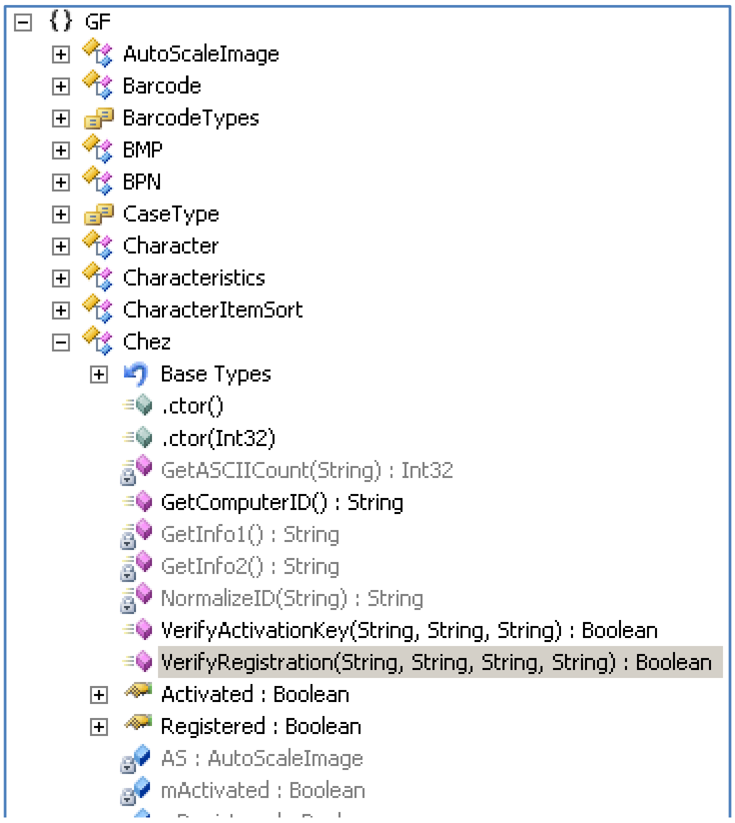
Open the app the in Reflector and open the following node as shown in figure
Note: Assembly **Activate** must be loaded into the Reflector

The code that looks more promising to us is the Method that returns a Boolean value

Code:

```
Private Sub cbActivationKey_Click(ByVal sender As Object, ByVal e As EventArgs)

    Try

        Dim prompt As String = ""

        Dim service As New Service

        Me.cbActivationKey.Text = "Retrieving Activation Key, Please wait..."

        Me.txtActivationKey.Text = "Retrieving Key, Please wait..."

        Me.cbActivationKey.Enabled = False

        Application.DoEvents()

        Me.txtComputerID.Text = Me.CZ.GetComputerID
        ' The VerifyRegistration Method looks promising

        Dim flag As Boolean = Me.CZ.VerifyRegistration(

                                    Me.txtProductName.Text.Trim,

                                    Me.txtCustomerName.Text.Trim,

                                    Me.txtOrderID.Text.Trim,

                                    Me.txtRegistrationCodes.Text.Trim)
        ' If the Property Registered is set to True we branch to Else

        If Not Me.CZ.Registered Then

            Me.txtActivationKey.Text = ""

            Interaction.MsgBox("Invalid Registration Information",
MsgBoxStyle.OkOnly, Nothing)

        Else
            ' This is a custom webservice which retrieves the Activation Key from the WebSite
            ' when the Registration Codes are valid

            prompt = service.GetActivationKey(Me.txtProductName.Text.Trim,

                                              Me.txtCustomerName.Text.Trim,

                                              Me.txtOrderID.Text.Trim,

                                              Me.txtRegistrationCodes.Text.Trim,

                                              Me.txtComputerID.Text.Trim)

            If ((prompt.Length > 0) AndAlso (prompt.Substring(0, 1) = "*")) Then

                Me.txtActivationKey.Text = ""

                Interaction.MsgBox(prompt, MsgBoxStyle.OkOnly, Nothing)

            ElseIf (prompt.Trim.Length = 0) Then

                Me.txtActivationKey.Text = ""
                Interaction.MsgBox("Invalid Activation Key", MsgBoxStyle.OkOnly,
Nothing)

            Else

                Me.txtActivationKey.Text = prompt
                ' The Verify is the Method which checks the Activation key is valid or not

                If Not Me.Verify Then

                    Me.txtActivationKey.Text = ""

                    Interaction.MsgBox("Invalid Activation Key",
MsgBoxStyle.OkOnly, Nothing)

                Else

                    Interaction.MsgBox("Successfully obtained Activation Key",
MsgBoxStyle.OkOnly, Nothing)

                End If

            End If

        End If

    Catch exception1 As Exception

        ProjectData.SetProjectError(exception1)

        Dim exception As Exception = exception1

        Me.txtActivationKey.Text = ""

        Interaction.MsgBox("Error creating Activation Key", MsgBoxStyle.OkOnly,
Nothing)

        ProjectData.ClearProjectError()

    Finally

        Me.cbActivationKey.Enabled = True

        Me.cbActivationKey.Text = "Obtain Activation Key automatically via
Internet"

    End Try
End Sub
```

Ok I followed VerifyRegistration Method and landed in the Assembly **GF**



```
{} GF
    AutoScaleImage
    Barcode
    BarcodeTypes
    BMP
    BPN
    CaseType
    Character
    Characteristics
    CharacterItemSort
    Chez
        Base Types
        .ctor()
        .ctor(Int32)
        GetASCIICount(String) : Int32
        GetComputerID() : String
        GetInfo1() : String
        GetInfo2() : String
        NormalizeID(String) : String
        VerifyActivationKey(String, String, String) : Boolean
        VerifyRegistration(String, String, String, String) : Boolean
        Activated : Boolean
        Registered : Boolean
        AS : AutoScaleImage
        mActivated : Boolean
```

The VerifyRegistration and VerifyActivationKey are almost identical

```
Public Function VerifyRegistration(ByVal pProductName As String,
                ByVal pCustomerName As String,
                ByVal pOrderID As String,
                ByVal pRegistrationCodes As String) As Boolean
    Try
        Me.AS = New AutoScaleImage((DateTime.Now.Day * DateTime.Now.Month))
        Me.AS.ProductName = pProductName
        Me.AS.CustomerName = pCustomerName
        Me.AS.OrderID = pOrderID
        Me.AS.RegistrationCodes = pRegistrationCodes
        Me.mRegistered = Me.AS.Verify
        Return Me.mRegistered
     Catch exception1 As Exception
        Me.mRegistered = False
        Return False
    End Try
End Function


Public Function VerifyActivationKey(ByVal pProductName As String,
                ByVal pRegistrationCodes As String,
                ByVal pActivationKey As String) As Boolean
    Try
        Dim computerID As String = Me.GetComputerID
        If (computerID.Trim.Length = 0) Then
            Return False
        End If
        Me.AS = New AutoScaleImage((DateTime.Now.Day * DateTime.Now.Month))
        Me.AS.ProductName = pProductName
        Me.AS.CustomerName = computerID
        Me.AS.OrderID = pRegistrationCodes
        Me.AS.RegistrationCodes = pActivationKey
        Me.mRegistered = Me.AS.Verify
        Return Me.mRegistered
    Catch exception1 As Exception
        Me.mRegistered = False
        Return False
    End Try
End Function
```

From the above code it is obvious to us that the Method Verify is Important to us let break it down. Before venturing into the Method let me show what the statement `Me.AS = New AutoScaleImage((DateTime.Now.Day * DateTime.Now.Month))` is doing.

```
Public Sub New(ByVal pZ As Integer)
    Me.mSalt0 = "Gregory"                  '
    Me.mSalt1 = "Joseph"                   ' Keep these in Mind
    Me.mSalt2 = "Matthew"                  ' We might need them
    Me.mSalt3 = "Kalispal"                 '
    Me.mTargetPhrase = "ActionJackson"     '
    Me.mRandomGenerator = New RNGCryptoServiceProvider
```

‘ Hmmm SHA1 has been selected as HashAlogrithim

```
    Me.mHash = New SHA1CryptoServiceProvider

    Me.mProductID = &H3E7

    Me.mDemoName = "Demo"

    Me.mProductName = "Demo"

    Me.mCustomerName = "Demo"

    Me.mOrderID = "Demo"

    Me.mRegistrationCodes = "Demo"

    Me.mActivationKey = ""

    Me.INVALID = True

    Try

        If   (pZ   <>   (DateTime.Now.Day   *   DateTime.Now.Month))   Then   Throw   New
GF_RegistrationException("Invalid registration keys. ")

        Me.INVALID = False

    Catch exception1 As Exception

        Throw New GF_RegistrationException("OCR is not registered. ")

    End Try

End Sub
```

Now I’ll break down the Method Verify

```
Public Function Verify() As Boolean

    Try

        If Me.INVALID Then Throw New Exception("This product is not registered. ")
```

‘ Finally we find out that the ProductName must be one of the following 4 strings and only 3 strings are valid

```
            Dim str As String = "StandardOCR"

            Dim str2 As String = "StandardBar"

            Dim str3 As String = "CombinationOCR"

            Dim str4 As String = "Demo"

            Dim lList As String = ""

            Dim aSCIICount As Integer = 0
```

‘ Here the Properties which are assigned earlier turned into UPPER CASE and assigned to local variables

```
            Dim str7 As String = Me.mProductName.ToUpper.Trim

            Dim str8 As String = Me.mCustomerName.ToUpper.Trim
            Dim str9 As String = Me.mOrderID.ToUpper.Trim

            Dim mRegistrationCodes As String = Me.mRegistrationCodes

            Me.mRegistered = False

            Me.mProductID = &H3E7
```

‘ The str7 contains our entered ProductName and this is compared with the above predefined ProductName ‘string constants. We get a unique number based on the ProductName we choosed earlier

```
            If (str7 = str.ToUpper.Trim) Then Me.mProductID = 0

            ElseIf (str7 = str2.ToUpper.Trim) Then Me.mProductID = 1

            ElseIf (str7 = str3.ToUpper.Trim) Then Me.mProductID = 2

            Else

                If (str7 <> str4.ToUpper.Trim) Then Throw New
GF_RegistrationException("Invalid product name. ")

                Me.mProductID = 3

            End If
```

‘ The code starting from here is most important because most of the checking is done here

‘ Look here do the type name salt **ring any bell**

‘ I recommend you to look at these Method. Click GetSalt to launch the Reflector

```
            Dim salt As String = Me.GetSalt(Me.mProductID)
```

‘ Here HashString

```
            lList = Me.HashString((salt & str8 & str9 & Me.mTargetPhrase))
```

‘ Remember lList is the Refernce HASH which is cross checked with the HASH’s generated by entered

‘ RegistrationCode and ActivationKey, and here too GetASCIICount

```
            aSCIICount = Me.GetASCIICount(lList)
```

‘ The below condition require that the Registration code must be of length 0x13 or 19 characters in length
‘ CHECK 1

```
            If (mRegistrationCodes.Length = &H13) Then
```

‘ The below condition checks for the hyphens in some predefined positions
‘ XXXX-XXXX-XXXX-XXXX is what our code must look like
‘ CHECK 2

```
                If (((mRegistrationCodes.Substring(4, 1) <> "-") OrElse
(mRegistrationCodes.Substring(9, 1) <> "-")) OrElse

                     (mRegistrationCodes.Substring(14, 1) <> "-")) Then

                    Return False

                End If
```

‘ The following code removes the hyphens and once again checks its length and it must be equal to 0x10 or ‘16 characters
‘ CHECK 3

```
                Dim str11 As String = mRegistrationCodes.Replace("-", "")

                If (str11.Trim.Length <> &H10) Then

                    Return False

                End If
```

‘ The below condition checks that the first 4 characters in code must be number and divisible by 13
‘ This makes our serial look like NNNN-XXXX-XXXX-XXXX i.e., 0013-XXXX-XXXX-XXXX or
‘9997-XXXX-XXXX-XXXX
‘ CHECK 4
‘ QUICK NOTE: The serial cannot contain Hexadecimal even tough we can find numbers upto 0Xffff
‘ because the Conversion Method doesn’t specify the base to convert from this limits us to use decimal
‘numbers only

```
                If ((Convert.ToInt32(str11.Substring(0, 4)) Mod 13) <> 0) Then

                    Return False

                End If
```

‘ If we made up to this mark the str11 will be assigned the 12 characters form the registration code

```
                str11 = str11.Substring(4, 12)
```

‘ Look HashString is called once again but this time with Registration Code that we entered

```
                Dim str12 As String = Me.HashString((salt & str8 & str9 & str11))
```

‘ GetNumberCount?

‘Ans: Click Mee......

```
                Dim num3 As Integer = (Me.GetASCIICount(str12) +
Me.GetNumberCount(str11))
```

‘ Finally we are only ONE condition away from TRUE
‘ The First character in Reference Hash and our Registration Code/Activation Key needs to be equal
‘ plus num3 must contain a number that is to be same as aSCIICount
‘ How are we going satisfy these two

```
                If ((num3 = aSCIICount) AndAlso (str12.Substring(0, 1) =
lList.Substring(0, 1))) Then

                    Me.mRegistered = True

                    Return True

                End If

            End If

            Return False

        Catch exception1 As Exception

            Return False

        End Try
    End Function
```

There are enough hints for us to proceed. If you examined the code where I provided links now I am going show the code snippet which generates the required Registration Code and Serial

```
int counter = 0;

for (ulong i = 100000000000L; i < 999999999999L; ++i)

{

     string CryptHash = HashString(salt + text8 + text9 + i.ToString());

     if (CryptHash[0] == lList[0])

     {

         int check = GetASCIICount(CryptHash) + GetNumberCount(i.ToString());

         if (check == aSCIICount)

             ++counter;

         if (check == aSCIICount)

             MessageBox.Show(CryptHash + " <---> " + (i));

     }

 }
 MessageBox.Show(counter.ToString());
```

I choose the numbers `100000000000L` and `999999999999L` because these are 12 digits in length and the Method GetNumberCount only work on decimal numbers

8 THE STRANGE CASE OF DBG_PRINTEXCEPTION_C & DBG_RIPEXCEPTION BY MOID

8.1 INTRODUCTION

Hidden deep in the jungle we call the Internet, I found a wonderful anti-debugging trick. In a comment on rootkit.com, dsei shows a trick that is capable of detecting all ring3 debuggers. As far as I know, this trick is not used in any protection system nor defeated by any debugger or plugin. In this small essay I will show this trick, it's cause and how to defeat it.

8.2 TRICK DESCRIPTION

If we raise the DBG_PRINTEXCEPTION_C exception (0x40010006) without a ring3 debugger active, the exception is treated normally. However, if we do the same with a debugger attached, the debugger gets a OUTPUT_DEBUG_STRING_EVENT and after calling ContinueDebugEvent, the exception has disappeared.

Obviously this difference can be used to detect ring3 debuggers. As an example, here is the core of my xADT plugin to check for this:

```
;First, set up a SEH frame

push seh

push dword [fs:0]

mov [fs:0], esp

mov dword [return_code], POSITIVE        ;If it's swallowed, a debugger is detected

push 0

push 0

push 0

push DBG_PRINTEXCEPTION_C

call RaiseException

pop dword [fs:0]

pop eax

mov eax, [return_code]

ret

seh:                                     ;If the exception is not swallowed, there
is no debugger

mov dword [return_code], NEGATIVE

xor eax, eax

ret
```

8.3 THE CAUSE

This behaviour is by design. OutputDebugStringA uses it to send messages to the application debugger. The change from DBG_PRINTEXCEPTION_C to OUTPUT_DEBUG_STRING_EVENT happens in ntdll!DbgUiConvertStateChangeStructure.

Just take a look at this rebuild source snippet (by Alex Ionescu):

```
/* Any sort of exception */

case DbgExceptionStateChange:

case DbgBreakpointStateChange:

case DbgSingleStepStateChange:

   /* Check if this was a debug print */

   if(WaitStateChange->StateInfo.Exception.ExceptionRecord.ExceptionCode ==
DBG_PRINTEXCEPTION_C)

   {

       /* Set the Win32 code */

       DebugEvent->dwDebugEventCode = OUTPUT_DEBUG_STRING_EVENT;

       /* Copy debug string information */

       DebugEvent->u.DebugString.lpDebugStringData =

            (PVOID)WaitStateChange-
>StateInfo.Exception.ExceptionRecord.ExceptionInformation[1];

       DebugEvent->u.DebugString.nDebugStringLength =

           WaitStateChange-
>StateInfo.Exception.ExceptionRecord.ExceptionInformation[0];

       DebugEvent->u.DebugString.fUnicode = FALSE;

   }

   else

   if (WaitStateChange->StateInfo.Exception.ExceptionRecord.ExceptionCode ==
DBG_RIPEXCEPTION)

   {

       /* Set the Win32 code */

       DebugEvent->dwDebugEventCode = RIP_EVENT;

       /* Set exception information */

       DebugEvent->u.RipInfo.dwType =

           WaitStateChange-
>StateInfo.Exception.ExceptionRecord.ExceptionInformation[1];
```

```
        DebugEvent->u.RipInfo.dwError =

            WaitStateChange-
>StateInfo.Exception.ExceptionRecord.ExceptionInformation[0];

    }

    else

    {

        /* Otherwise, this is a debug event, copy info over */

        DebugEvent->dwDebugEventCode = EXCEPTION_DEBUG_EVENT;

        DebugEvent->u.Exception.ExceptionRecord =

            WaitStateChange->StateInfo.Exception.ExceptionRecord;

        DebugEvent->u.Exception.dwFirstChance =

            WaitStateChange->StateInfo.Exception.FirstChance;

    }

    break;
```

As you see also DBG_RIPEXCEPTION gets a special treatment. By raising this exception you can send a RIP_EVENT to the debugger. This will make Olly break. Again this exception magically disappears when a debugger is present.

8.4 THE SOLUTION

With what you know now, it is very easy to find the solution. You might want to try finding it yourself before reading further.

Seeing how the problem is the special handling of those two exceptions, the trick is to remove that special handling. We have to make sure that the change to OUTPUT_DEBUG_STRING_EVENT or RIP_EVENT never happens.

The disassembly of the relevant code (in ntdll!DbgUiConvertStateChangeStructure) is this:

```
                lea     esi, [eax+0Ch]
                mov     ecx, [esi]
                cmp     ecx, 40010006h            ;<-- is it DBG_PRINTEXCEPTION_C?
                jnz     short loc_7C950969
                mov     [ebx], edi
                mov     ecx, [eax+24h]
                mov     [ebx+0Ch], ecx
                mov     ax, [eax+20h]
                and     word ptr [ebx+10h], 0
                mov     [ebx+12h], ax
                jmp     loc_7C950A28
loc_7C950969:
                cmp     ecx, 40010007h       ;<-- is it DBG_RIPEXCEPTION?
                jnz     short loc_7C950985
                mov     dword ptr [ebx], 9
                mov     ecx, [eax+24h]
                mov     [ebx+10h], ecx
                mov     eax, [eax+20h]
                jmp     loc_7C950A25
                lea     esi, [eax+0Ch]
                mov     ecx, [esi]
                cmp     ecx, 40010006h            ;<-- is it DBG_PRINTEXCEPTION_C?
                jnz     short loc_7C950969
                mov     [ebx], edi
                mov     ecx, [eax+24h]
                mov     [ebx+0Ch], ecx
                mov     ax, [eax+20h]
                and     word ptr [ebx+10h], 0
                mov     [ebx+12h], ax
                jmp     loc_7C950A28
loc_7C950969:
                cmp     ecx, 40010007h       ;<-- is it DBG_RIPEXCEPTION?
```

```
jnz     short loc_7C950985
mov     dword ptr [ebx], 9
mov     ecx, [eax+24h]
mov     [ebx+10h], ecx
mov     eax, [eax+20h]
jmp     loc_7C950A25
```

By patching the red jnz's to jmp, the change will never happen and they will be treated like normal exceptions. Make sure that the change is in the memory of the debugger, not the debuggee!

8.5 REFERENCES

Post by dsei - http://www.rootkit.com/board.php?thread=3360&did=edge284&disp=3360

Windows Native Debugging Internals - http://www.openrce.org/articles/full_view/25

9 CRACKING FOR FUN BY ARJUNS

I am going to discuss upon how cracking can be a fun. The topic I've chosen is how one can create multiple instances of Yahoo Messenger and Windows Live Messenger as we know one can't create multiple instances of them by default.

9.1 SOME THEORIES

Let's discuss upon how program knows if there is already a running instance of it. There may be various tricks to check it but the following Win32 AIPs are very helpful to achieve the same goal.

9.1.1 FINDING WINDOW

1. FindWindow (Exported by user32.dll)

The FindWindow function retrieves the handle to the top-level window whose class name and window name match the specified strings. This function does not search child windows.

```
HWND FindWindow (

    LPCTSTR lpClassName,             // pointer to class name

    LPCTSTR lpWindowName   // pointer to window name

   );
```

9.1.2 MUTEXES:

2. CreateMutex / OpenMutex (Exported by kernel32.dll):

A mutex (from mutually exclusive) is an object that can only be acquired by one thread at any given moment. Any threads that attempt to acquire a mutex while it is already owned by another thread will enter a wait state until the original thread releases the mutex or until it terminates. If more than one thread is waiting, they will each receive ownership of the mutex in the original order in which they requested it.

9.1.3 SEMAPHORES

3. CreateSemaphore/ OpenSemaphore (Exported by kernel32.dll)

A semaphore is like a mutex with a user-defined counter that defines how many simultaneous owners are allowed on it. Once that maximum number is exceeded, a thread that requests ownership of the semaphore will enter a wait state until one of the threads release the semaphore.

9.1.4 EVENTS

4. CreateEvent / OpenEvent (Exported by kernel32.dll)

The event object is a kernel object that stays nonsignaled until a condition is met. The programmer has the control over setting the event object to a signaled or a nonsignaled state, unlike a mutex or semaphore where the operating system governs the signaled and nonsignaled state of the object.

Notes:

All of the synchronization objects described above are managed by the kernel's object manager and implemented in kernel mode, which means that the system must switch into the kernel for any operation that needs to be performed on them.

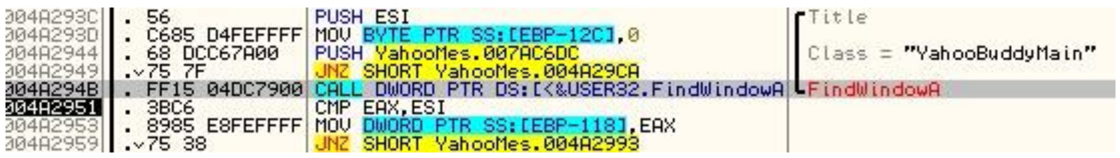
9.2 DIVING INTO THE SCENE

9.2.1 YAHOO MESSENGER

Let's begin our real job, first we go for Yahoo Messenger, version I'm using is 8.1.0.249

Open the target in Olly.

You can try putting breaking point on every APIs mentioned above. The only important Breakpoint is of at 4a294B.

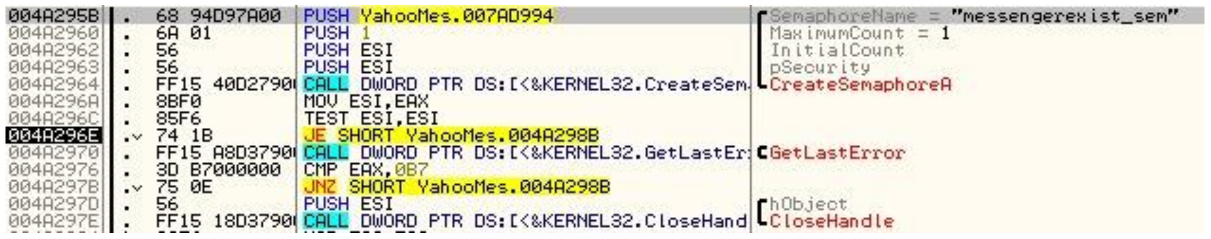


```
004A293C  . 56             PUSH ESI                                      ┌Title
004A293D  . C685 D4FEFFFF  MOV BYTE PTR SS:[EBP-12C],0                   │
004A2944  . 68 DCC67A00    PUSH YahooMes.007AC6DC                        │Class = "YahooBuddyMain"
004A2949  .v75 7F          JNZ SHORT YahooMes.004A29CA                   │
004A294B  . FF15 04DC7900  CALL DWORD PTR DS:[<&USER32.FindWindowA       └FindWindowA
004A2951  . 3BC6           CMP EAX,ESI
004A2953  . 8985 E8FEFFFF  MOV DWORD PTR SS:[EBP-118],EAX
004A2959  .v75 38          JNZ SHORT YahooMes.004A2993
```

As you can clearly see that there is a comparison being made at 4A2951, where ESI holds 0 and EAX being window handle. This is just to check that if there is already a window running of the same class "YahooBuddyMain". And if true it closes the newer instance by closing window handle at 4A297E.

So we just need to patch JNZ 4A2993 at 4A2959 to NOP so that it never jumps off there.

Now,



```
004A295B  .  68 94D97A00   PUSH YahooMes.007AD994                     ┌SemaphoreName = "messengerexist_sem"
004A2960  .  6A 01         PUSH 1                                     │MaximumCount = 1
004A2962  .  56            PUSH ESI                                   │InitialCount
004A2963  .  56            PUSH ESI                                   │pSecurity
004A2964  .  FF15 40D2790( CALL DWORD PTR DS:[<&KERNEL32.CreateSem.   └CreateSemaphoreA
004A296A  .  8BF0          MOV ESI,EAX
004A296C  .  85F6          TEST ESI,ESI
004A296E  .v 74 1B         JE SHORT YahooMes.004A298B
004A2970  .  FF15 A8D3790( CALL DWORD PTR DS:[<&KERNEL32.GetLastEr:   [GetLastError
004A2976  .  3D B7000000   CMP EAX,0B7
004A297B  .v 75 0E         JNZ SHORT YahooMes.004A298B
004A297D  .  56            PUSH ESI                                   ┌hObject
004A297E  .  FF15 18D3790( CALL DWORD PTR DS:[<&KERNEL32.CloseHand    └CloseHandle
```

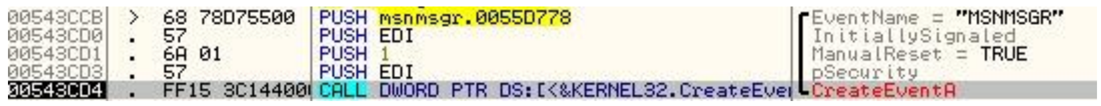
At 4A2964 we have a call to CreateSemaphoreA, and after execution it returns **not null,** if there is already Yahoo Messenger running and null if no instance of it running in EAX , in case of not null we assume that there is already Semaphore object exists of the name “messengerexists_sem”. Further we have a call to GetLastError at 4A2970 and a comparison is being made with the last error code to B7 (ERROR_ALREADY_EXISTS), if comparison is true newer window never gets executed.

So patch 4a297b: JNZ 4a298b too jmp 4a298b

Copy all modification and save, you are done. Enjoy as many instances as you want of yahooMessenger

9.2.2 WINDOWS LIVE MESSENGER 8.1

Easy, Windows Live Messenger doesn’t use FindWindow method to check if there is already one instance of it running nor it checks for Semaphore object but it uses an event object to know if there is one running already.

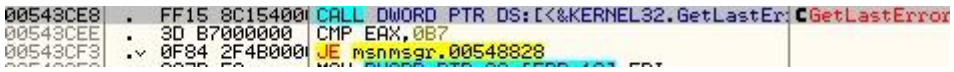


```
00543CCB  >  68 78D75500   PUSH msnmsgr.0055D778                      ┌EventName = "MSNMSGR"
00543CD0  .  57            PUSH EDI                                   │InitiallySignaled
00543CD1  .  6A 01         PUSH 1                                     │ManualReset = TRUE
00543CD3  .  57            PUSH EDI                                   │pSecurity
00543CD4  .  FF15 3C14400( CALL DWORD PTR DS:[<&KERNEL32.CreateEve(   └CreateEventA
```

We can clearly see that one Event object named “MSNMSGR” is being created at 543cd4

When there is already one, we’ll have not null value in EAX which is of course an event handle to an earlier event object.

Further we have



```
00543CE8  .   FF15 8C15400( CALL DWORD PTR DS:[<&KERNEL32.GetLastEr: GetLastError
00543CEE  .   3D B7000000   CMP EAX,0B7
00543CF3  .v  0F84 2F4B000( JE msnmsgr.00548828
```

Call at 543ce8 gets last error and if that error code is B7 (ERROR_ALREADY_EXISTS)

Newer instance gets destroyed. We just need to patch JE 548828 to nop so it never jumps off to destroy the newer instance...

Save all modification, now you have a working multi messenger. ☺

10 WRITING A SELF-KEYGENERATOR LOADER WITH ABEL BY M1SCH13F

10.1 INTRODUCTION

Self keygenning is a method of patching an app, or in our case using a loader, to give the user a valid serial. This is a simplified way to keygen something. It can be used on very difficult to follow registration schemes or for anyone who just finds keygenning to be difficult. While this method might not be as 1337 as keygenning an app it has its place. Some assembly is required ;-). As always the more ASM you know, the better. The Program we will be examining today is, MP3 AVI MPEG WMV RM to Audio CD Burner, by Ether… Could they have made the name a little longer? Although I would not consider this a difficult program to serial fish, or to self keygen, it is not intended for n00bs either. This tutorial is also not intended to supplement lena151's tutorials, which I consider to be the definitive guide to cracking; hopefully this will complement and reinforce what she had in her tuts. If you haven't seen them I highly recommend doing so. They can be found at http://tuts4you.com. On a final note, this is my first tutorial. I hope you enjoy. Enough BS let's get down to business… Happy Cracking.

Tools Needed:

- PEiD, http://peid.has.it/
- OllyDbg, http://www.ollydbg.de/
- ABEL, http://www.tuts4you.com/download.php?view.385

Not Required, but recommended tools:

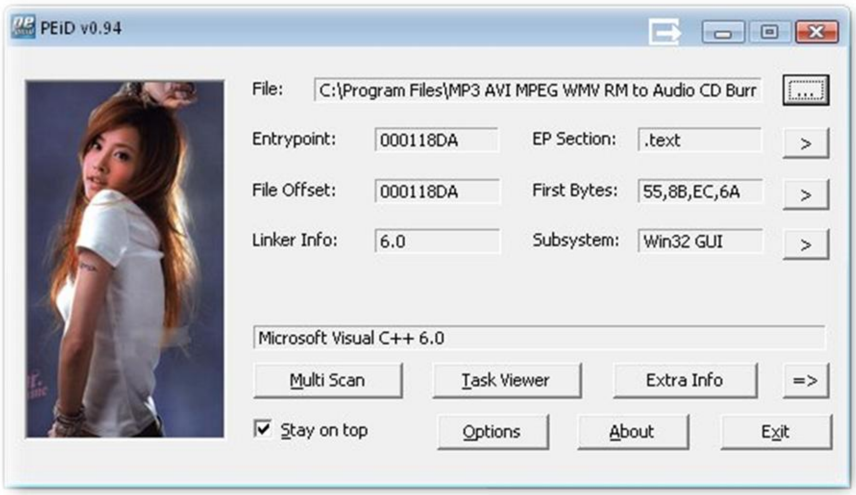
- MASM, http://www.masm32.com/masmdl.htm
- WinASM, http://www.winasm.net/index.php?ind=downloads

Further Reading:

- Lena151 Tutorials:
  - ✓ 17. Insights and practice in basic (self)keygenning
  - ✓ 24. Patching at runtime using loaders
- Goppit's Win32 Assembler Coding for Crackers

10.2 EXAMINING THE APPLICATION

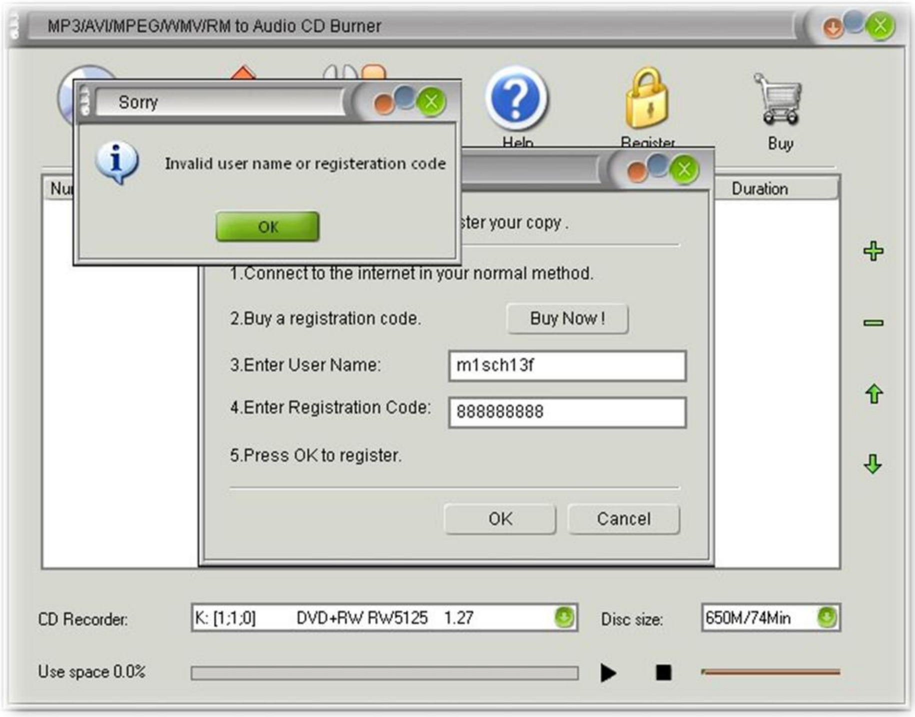
As with any other potential target we will start by looking at our exe in PEiD. BTW Thanks to SnD For the nice visual on the version I am using :)



File: C:\Program Files\MP3 AVI MPEG WMV RM to Audio CD Burr
Entrypoint: 000118DA
EP Section: .text
File Offset: 000118DA
First Bytes: 55,8B,EC,6A
Linker Info: 6.0
Subsystem: Win32 GUI
Microsoft Visual C++ 6.0
☑ Stay on top

Alright… Visual C++ 6. Usually easy to follow. Further examination, using kanal, reveals no known crypto algorithms used.

After loading the program in Olly we will land at EP press F9 to start and open the registration window. Type in your name as well as a bogus serial number. Here is what you will see :(. Don't worry about the Invalid user name though… we'll fix that ;)



Sorry

Invalid user name or registeration code

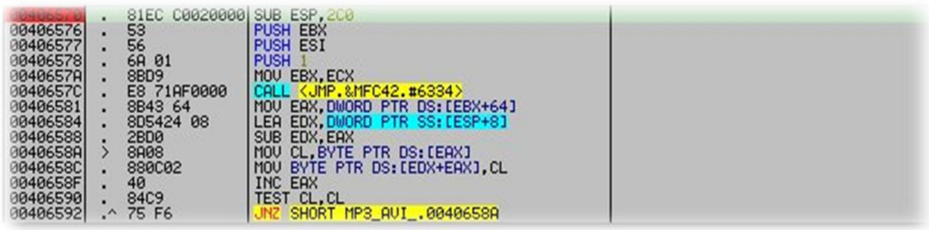
…ster your copy .
1.Connect to the internet in your normal method.
2.Buy a registration code.
3.Enter User Name: m1sch13f
4.Enter Registration Code: 888888888
5.Press OK to register.

CD Recorder: K: [1;1;0] DVD+RW RW5125 1.27
Disc size: 650M/74Min
Use space 0.0%

10.2.1 SERIAL FISHING THE TARGET

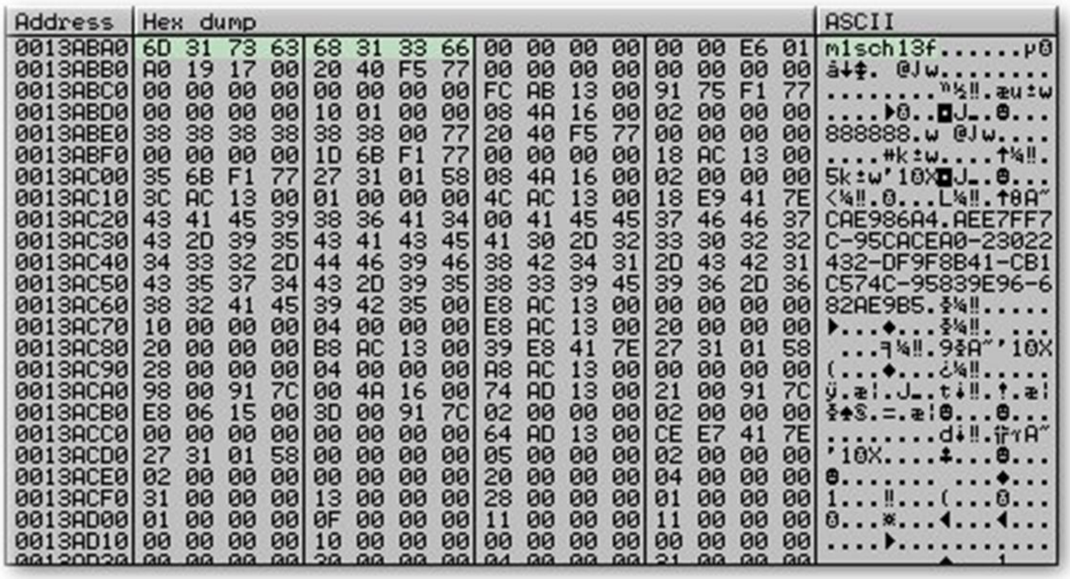
Alright, so right click the main window and go to “Search for” and then “All referenced Text Strings” As I’m sure you already know by now, unless you are a total n00b. Hit the ‘home’ key on your keyboard to start at the top of the list, Right click and search for “invalid user” or something like that. Make sure it’s not case sensitive either in case you DeCidEd tO tYpe LikE ThIs.

We find the text string at address 004066DC. Look around and examine the code a bit and study what is going on and how we will land there. You’ll see that there is a JNZ that jumps past a return into the function to call the bad boy. Let’s not get ahead of ourselves, BP the beginning of the registration scheme 00406570.



```
00406570  .   81EC C0020000 SUB ESP,2C0
00406576  .   53            PUSH EBX
00406577  .   56            PUSH ESI
00406578  .   6A 01         PUSH 1
0040657A  .   8BD9          MOV EBX,ECX
0040657C  .   E8 71AF0000   CALL <JMP.&MFC42.#6334>
00406581  .   8B43 64       MOV EAX,DWORD PTR DS:[EBX+64]
00406584  .   8D5424 08     LEA EDX,DWORD PTR SS:[ESP+8]
00406588  .   2BD0          SUB EDX,EAX
0040658A  >   8A08          MOV CL,BYTE PTR DS:[EAX]
0040658C  .   880C02        MOV BYTE PTR DS:[EDX+EAX],CL
0040658F  .   40            INC EAX
00406590  .   84C9          TEST CL,CL
00406592  .^  75 F6         JNZ SHORT MP3_AVI_.0040658A
```

Step, F8, through the code. We don’t see anything too interesting beginning to happen until 004065CD. This is where the serial is calculated. So, if you are more interested in patching, or keygenning this may be of interest, but today we will continue on. Step, F8, once more and we see our valid serial on the stack, below our name. Follow it in the dump so we can see what we might be playing with later. Scroll around a little bit and you’ll see your name and your bogus serial not far away. Take notice that the correct serial is 1 Q-Word Long, or 2 D-Words long. No matter how long your name is on this app the serial is always the same length.

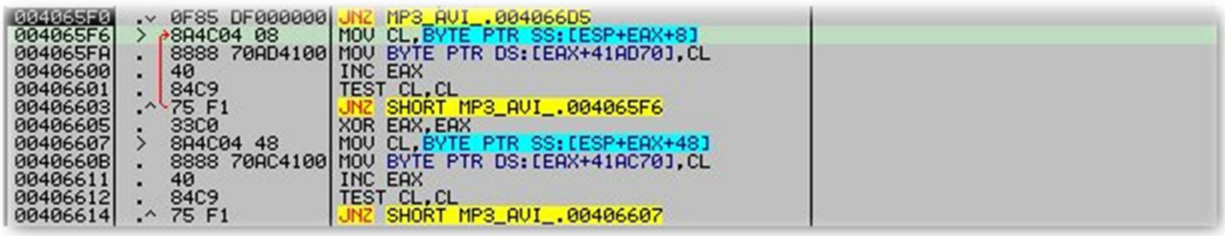


```
Address  Hex dump                                         ASCII
0013ABA0 6D 31 73 63|68 31 33 66|00 00 00 00|00 00 E6 01| m1sch13f......µ☺
0013ABB0 A0 19 17 00|20 40 F5 77|00 00 00 00|00 00 00 00| á↓↨. @Jw........
0013ABC0 00 00 00 00|00 00 00 00|FC AB 13 00|91 75 F1 77| ........ⁿ½‼.æu±w
0013ABD0 00 00 00 00|10 01 00 00|08 4A 16 00|02 00 00 00| ....►☺..◘J▬.☻...
0013ABE0 38 38 38 38|38 38 00 77|20 40 F5 77|00 00 00 00| 888888.w @Jw....
0013ABF0 00 00 00 00|1D 6B F1 77|00 00 00 00|18 AC 13 00| ....↔k±w....↑¼‼.
0013AC00 35 6B F1 77|27 31 01 58|08 4A 16 00|02 00 00 00| 5k±w'1☺X◘J▬.☻...
0013AC10 3C AC 13 00|01 00 00 00|4C AC 13 00|18 E9 41 7E| <¼‼.☺...L¼‼.↑θA~
0013AC20 43 41 45 39|38 36 41 34|00 41 45 45|37 46 46 37| CAE986A4.AEE7FF7
0013AC30 43 2D 39 35|43 41 43 45|41 30 2D 32|33 30 32 32| C-95CACEA0-23022
0013AC40 34 33 32 2D|44 46 39 46|38 42 34 31|2D 43 42 31| 432-DF9F8B41-CB1
0013AC50 43 35 37 34|43 2D 39 35|38 33 39 45|39 36 2D 36| C574C-95839E96-6
0013AC60 38 32 41 45|39 42 35 00|E8 AC 13 00|00 00 00 00| 82AE9B5.Φ¼‼.....
0013AC70 10 00 00 00|04 00 00 00|E8 AC 13 00|20 00 00 00| ►...♦...Φ¼‼. ...
0013AC80 20 00 00 00|B8 AC 13 00|39 E8 41 7E|27 31 01 58|  ...╕¼‼.9ΦA~'1☺X
0013AC90 28 00 00 00|04 00 00 00|A8 AC 13 00|00 00 00 00| (...♦...¿¼‼.....
0013ACA0 98 00 91 7C|00 4A 16 00|74 AD 13 00|21 00 91 7C| ÿ.æ|.J▬.t¡‼.!.æ|
0013ACB0 E8 06 15 00|3D 00 91 7C|02 00 00 00|02 00 00 00| Φ♠§.=.æ|☻...☻...
0013ACC0 00 00 00 00|00 00 00 00|64 AD 13 00|CE E7 41 7E| ........d¡‼.╬τA~
0013ACD0 27 31 01 58|00 00 00 00|05 00 00 00|02 00 00 00| '1☺X....♣...☻...
0013ACE0 02 00 00 00|00 00 00 00|20 00 00 00|04 00 00 00| ☻....... ...♦...
0013ACF0 31 00 00 00|13 00 00 00|28 00 00 00|01 00 00 00| 1...‼...(...☺...
0013AD00 01 00 00 00|0F 00 00 00|11 00 00 00|11 00 00 00| ☺...☼...◄...◄...
0013AD10 00 00 00 00|10 00 00 00|00 00 00 00|00 00 00 00| ....►...........
```

If you look a little bit further in the code, you’ll see that if you input a correct serial, it is written to a file, option.ini, so if we can replace our bogus serial with the correct one in the dump we should have a fully functioning program that wasn’t patched right?

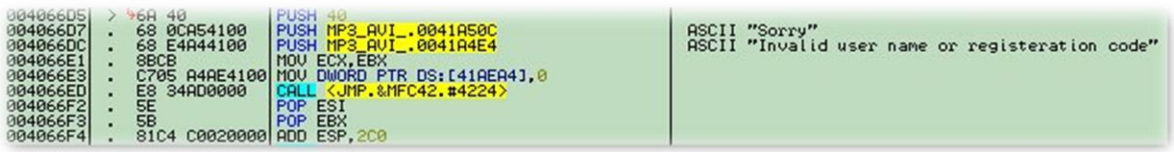
10.2.2 FINDING WHERE TO PATCH

Alright… now that we have our correct serial, let's see what happens when registering. Copy and paste the correct serial into the registration box. Step, F8, note the values of the registers particularly EAX when you get to the jump to bad boy. Continue past the jump to the bad boy and we will see that there are two loops following it. The first one reads your name from the dump, the second reads the serial. This is where the program figures out what data goes into the configuration file.



```
004065F0  .v 0F85 DF000000  JNZ MP3_AVI_.004066D5
004065F6  > >8A4C04 08      MOV CL,BYTE PTR SS:[ESP+EAX+8]
004065FA  .  8888 70AD4100  MOV BYTE PTR DS:[EAX+41AD70],CL
00406600  .  40             INC EAX
00406601  .  84C9           TEST CL,CL
00406603  .^ 75 F1          JNZ SHORT MP3_AVI_.004065F6
00406605  .  33C0           XOR EAX,EAX
00406607  >  8A4C04 48      MOV CL,BYTE PTR SS:[ESP+EAX+48]
0040660B  .  8888 70AC4100  MOV BYTE PTR DS:[EAX+41AC70],CL
00406611  .  40             INC EAX
00406612  .  84C9           TEST CL,CL
00406614  .^ 75 F1          JNZ SHORT MP3_AVI_.00406607
```

The Code after this is pretty much useless to us. Now let's figure out where to patch the program. The ideal spot would to follow the jump to the bad boy since that code is useless for us anyhow and we don't want to dig too deep :)



```
004066D5  > >6A 40          PUSH 40
004066D7  .  68 0CA54100    PUSH MP3_AVI_.0041A50C          ASCII "Sorry"
004066DC  .  68 E4A44100    PUSH MP3_AVI_.0041A4E4          ASCII "Invalid user name or registeration code"
004066E1  .  8BCB           MOV ECX,EBX
004066E3  .  C705 A4AE4100  MOV DWORD PTR DS:[41AEA4],0
004066ED  .  E8 34AD0000    CALL <JMP.&MFC42.#4224>
004066F2  .  5E             POP ESI
004066F3  .  5B             POP EBX
004066F4  .  81C4 C0020000  ADD ESP,2C0
```

As you can see we have 25(HEX) bytes to play with here, which should suffice for what we need to do.

You may be asking yourself why we don't patch it at the source, where the serial is originally written onto the dump. Well further examination reveals that it seems to be happening in a system DLL, msvcrt. And to be honest with you I don't feel like messing around in there. In other programs it may be a very viable option though, so don't limit yourself to the method we are using today.

10.2.3 PATCHING THE APP

Now comes the fun part, patching the app to do our bidding >:)

We have to think of what exactly we want to do.

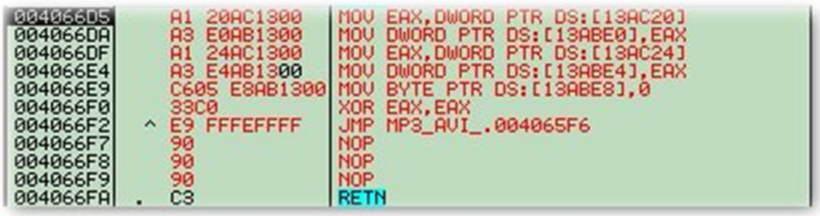
1. Move DWORD at memory address 0013AC20 to DWORD 0013ABE0
2. Move DWORD at memory address 0012AC24 to DWORD 0013ABE4
3. You'll notice the bogus serial has an empty byte after it. Make sure to patch the byte at 0013ABE8 to be empty otherwise you'll end up with funky data in the configuration file unless the user put an 8 character long serial.

4. Clear EAX to be Zero just like it was when passing the jump to the bad boy when registration was successful. This is important because the loops after the jump to the bad boy use EAX as a counter.

5. Jump back to where we would be right after the jump to the bad boy, 004065F6.

You cannot just assemble code that says `MOV DWORD PTR DS:[69696969], DWORD PTR DS:[00311311]`. You will have to move the d word's contents to a register. Since we already are going to XOR EAX I am going to move use that.

Here is what the patch will look like in olly



```
004066D5     A1 20AC1300     MOV EAX,DWORD PTR DS:[13AC20]
004066DA     A3 E0AB1300     MOV DWORD PTR DS:[13ABE0],EAX
004066DF     A1 24AC1300     MOV EAX,DWORD PTR DS:[13AC24]
004066E4     A3 E4AB1300     MOV DWORD PTR DS:[13ABE4],EAX
004066E9     C605 E8AB1300   MOV BYTE PTR DS:[13ABE8],0
004066F0     33C0            XOR EAX,EAX
004066F2   ^ E9 FFFEFFFF     JMP MP3_AVI_.004065F6
004066F7     90              NOP
004066F8     90              NOP
004066F9     90              NOP
004066FA   . C3              RETN
```

```
MOV EAX,DWORD PTR DS:[0013AC20]

MOV DWORD PTR DS:[0013ABE0],EAX

MOV EAX,DWORD PTR DS:[0013AC24]

MOV DWORD PTR DS:[0013ABE4],EAX

MOV BYTE PTR DS:[0013ABE8],00

XOR EAX,EAX

JMP 004065F6
```

Which correspond to this byte sequence:

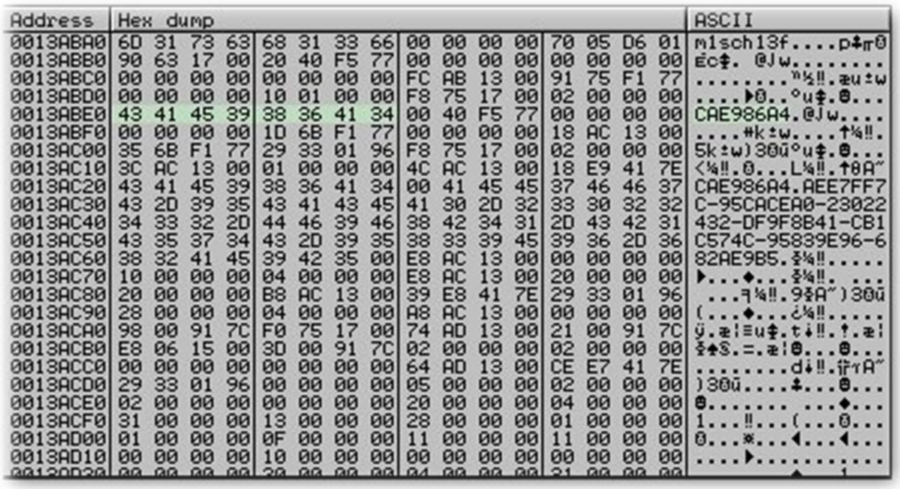
```
A1 20 AC 13 00 A3 E0 AB 13 00 A1 24 AC 13 00 A3 E4 AB 13 00 C6 05 E8 AB 13 00 00 33
C0 E9 FF FE FF FF 90 90 90
```

If you don't know how to find that right click the code and "follow in dump"

Hopefully you are following along so far.

10.2.4 TESTING

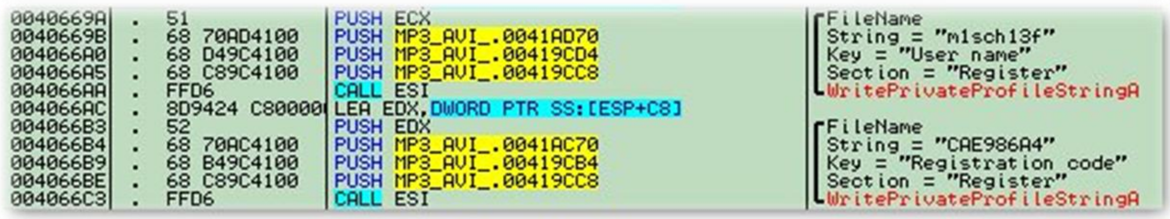
At this point it is, as always, advisable to test your patch. Right Click, Copy to executable, all modifications. Would also be a good idea to save your work thus far. Let's keep our breakpoints intact and step through the code to make sure it is doing exactly what we want it to. Unless you screwed it up you should see the contents of 0013AC20 move to EAX and then to 0012ABE0 and so on.



```
Address  Hex dump                                         ASCII
0013ABA0 6D 31 73 63 68 31 33 66 00 00 00 00 70 05 D6 01 m1sch13f....p╪╓☺
0013ABB0 90 63 17 00 20 40 F5 77 00 00 00 00 00 00 00 00 Éc↨. @Jw........
0013ABC0 00 00 00 00 00 00 00 00 FC AB 13 00 91 75 F1 77 ........ⁿ½‼.æu±w
0013ABD0 00 00 00 00 10 01 00 00 F8 75 17 00 02 00 00 00 ....►☺..°u↨.☻...
0013ABE0 43 41 45 39 38 36 41 34 00 40 F5 77 00 00 00 00 CAE986A4.@Jw....
0013ABF0 00 00 00 00 1D 6B F1 77 00 00 00 00 18 AC 13 00 ....↔k±w....↑¼‼.
0013AC00 35 6B F1 77 29 33 01 96 F8 75 17 00 02 00 00 00 5k±w)3☺û°u↨.☻...
0013AC10 3C AC 13 00 01 00 00 00 4C AC 13 00 18 E9 41 7E <¼‼.☺...L¼‼.↑ΘA~
0013AC20 43 41 45 39 38 36 41 34 00 41 45 45 37 46 46 37 CAE986A4.AEE7FF7
0013AC30 43 2D 39 35 43 41 43 45 41 30 2D 32 33 30 32 32 C-95CACEA0-23022
0013AC40 34 33 32 2D 44 46 39 46 38 42 34 31 2D 43 42 31 432-DF9F8B41-CB1
0013AC50 43 35 37 34 43 2D 39 35 38 33 39 45 39 36 2D 36 C574C-95839E96-6
0013AC60 38 32 41 45 39 42 35 00 E8 AC 13 00 00 00 00 00 82AE9B5.Φ¼‼.....
0013AC70 10 00 00 00 04 00 00 00 E8 AC 13 00 20 00 00 00 ►...♦...Φ¼‼. ...
0013AC80 20 00 00 00 B8 AC 13 00 39 E8 41 7E 29 33 01 96  ...╕¼‼.9ΦA~)3☺û
0013AC90 28 00 00 00 04 00 00 00 A8 AC 13 00 00 00 00 00 (...♦...¿¼‼.....
0013ACA0 98 00 91 7C F0 75 17 00 74 AD 13 00 21 00 91 7C ÿ.æ|≡u↨.t¡‼.!.æ|
0013ACB0 E8 06 15 00 3D 00 91 7C 02 00 00 00 02 00 00 00 Φ♠§.=.æ|☻...☻...
0013ACC0 00 00 00 00 00 00 00 00 64 AD 13 00 CE E7 41 7E ........d¡‼.╬τA~
0013ACD0 29 33 01 96 00 00 00 00 05 00 00 00 02 00 00 00 )3☺û....♣...☻...
0013ACE0 02 00 00 00 00 00 00 00 20 00 00 00 04 00 00 00 ☻....... ...♦...
0013ACF0 31 00 00 00 13 00 00 00 28 00 00 00 01 00 00 00 1...‼...(...☺...
0013AD00 01 00 00 00 0F 00 00 00 11 00 00 00 11 00 00 00 ☺...☼...◄...◄...
0013AD10 00 00 00 00 10 00 00 00 00 00 00 00 00 00 00 00 ....►...........
```

Here's what the dump should look like after our patch has run its course.

Also notice what happens on lines 004066B9 and 004066B4 as we pass through the loops after the jump to the bad boy.



```
0040669A  .  51             PUSH ECX                              ┌FileName
0040669B  .  68 70AD4100    PUSH MP3_AVI_.0041AD70                │String = "m1sch13f"
004066A0  .  68 D49C4100    PUSH MP3_AVI_.00419CD4                │Key = "User name"
004066A5  .  68 C89C4100    PUSH MP3_AVI_.00419CC8                │Section = "Register"
004066AA  .  FFD6           CALL ESI                              └WritePrivateProfileStringA
004066AC  .  8D9424 C80000( LEA EDX,DWORD PTR SS:[ESP+C8]
004066B3  .  52             PUSH EDX                              ┌FileName
004066B4  .  68 70AC4100    PUSH MP3_AVI_.0041AC70                │String = "CAE986A4"
004066B9  .  68 B49C4100    PUSH MP3_AVI_.00419CB4                │Key = "Registration code"
004066BE  .  68 C89C4100    PUSH MP3_AVI_.00419CC8                │Section = "Register"
004066C3  .  FFD6           CALL ESI                              └WritePrivateProfileStringA
```

Name and correct serial :)

Job well done. Now all that is left to do is make it distributable. We have several ways of doing this. We can make an offset patch, make a search and replace patch, which could possibly work on other versions, or we can make a loader that the user will only have to run once and they don't have 'cracked' software on their machine. I like the third method myself, it's 1337er which is always a plus… right? So let's move on.

10.3 CREATING A LOADER IN ABEL

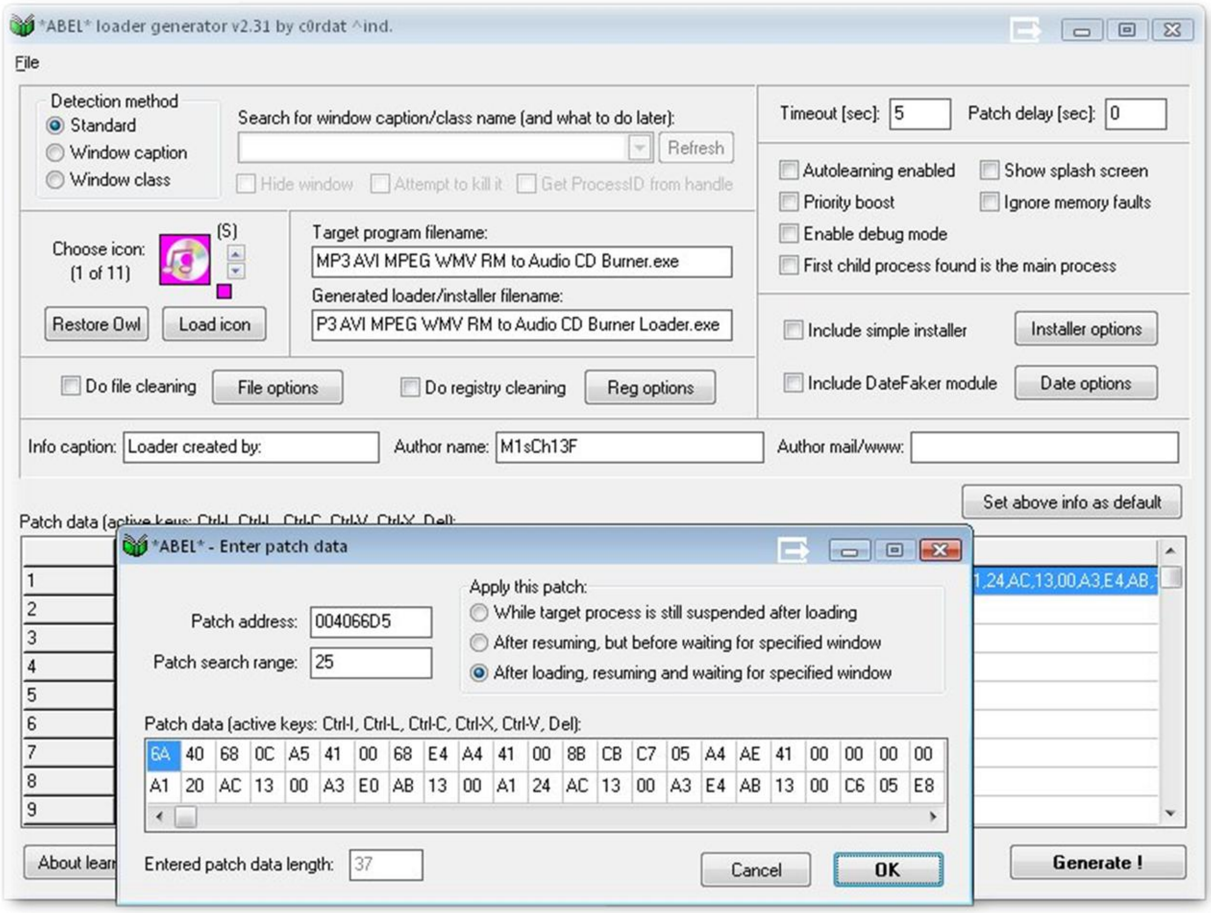
Alright this is the easy way at going about making a loader. In case you have never made a loader, don't worry I'll be showing you. Let's fire up ABEL. First thing let's change the timeout, which is the time the loader waits for the program to load before patching. Sometimes when you can't find a suitable memory address to follow to make a loader in dUP this is the best alternative.

I changed the timeout to 5 seconds because 15 is far longer than we would require. I also disable Auto learning. Auto learning searches for byte sequences, this works well in polymorphic code or a packed executable where the address of the patch might change.

Next I put the target program's file name. I just right clicked the target file and pressed rename and copied the name and added .exe at the end when I pasted it into ABEL. The Loader name is your call. Next comes the tricky part. The patch data in ABEL can be kind of annoying because you can't just copy and paste an entire byte sequence like you can in dUP. Despite its downfall it is pretty straight forward and easy.

First you put the address where you want to patch. You don't need to use raw offset, just type what you see in Olly, as you can see in my screenshot. For the "apply this patch" section you can pick any of the options they will all work for our example. For the final touch I added the custom icon by loading the icon used by the program we are patching. That is up to you though as well.

Alright well enough around lets generate this loader. Press OK out of the patch data window and then press generate!


ABEL loader generator v2.31 by c0rdat ^ind.

Detection method: [✓ Standard] Window caption / Window class

Search for window caption/class name (and what to do later):

Choose icon: (1 of 11)

Target program filename: MP3 AVI MPEG WMV RM to Audio CD Burner.exe

Generated loader/installer filename: P3 AVI MPEG WMV RM to Audio CD Burner Loader.exe

Timeout [sec]: 5

Patch delay [sec]: 0

Info caption: Loader created by:

Author name: M1sCh13F

Author mail/www:

ABEL - Enter patch data

Patch address: 004066D5

Patch search range: 25

Apply this patch: While target process is still suspended after loading / After resuming, but before waiting for specified window / [✓ After loading, resuming and waiting for specified window]

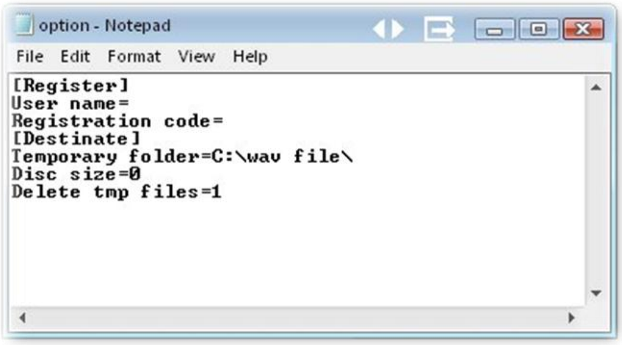
Patch data (active keys: Ctrl-I, Ctrl-L, Ctrl-C, Ctrl-X, Ctrl-V, Del):

```
6A 40 68 0C A5 41 00 68 E4 A4 41 00 8B CB C7 05 A4 AE 41 00 00 00 00
A1 20 AC 13 00 A3 E0 AB 13 00 A1 24 AC 13 00 A3 E4 AB 13 00 C6 05 E8
```

Entered patch data length: 37

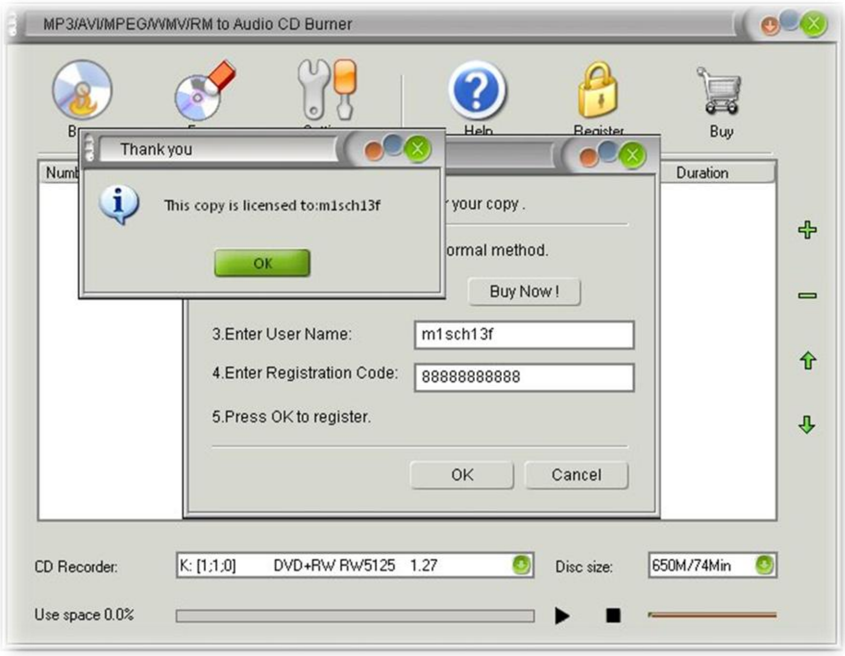
10.3.1 TESTING LOADER

Let's first delete the data from "option.ini" that contains our user name and registration code. So you should be looking at something like this…



```
[Register]
User name=
Registration code=
[Destinate]
Temporary folder=C:\wav file\
Disc size=0
Delete tmp files=1
```

Alright now it's time to test our loader! Click 'Register' and let's try it out.



This copy is licensed to:m1sch13f

3.Enter User Name: m1sch13f

4.Enter Registration Code: 88888888888

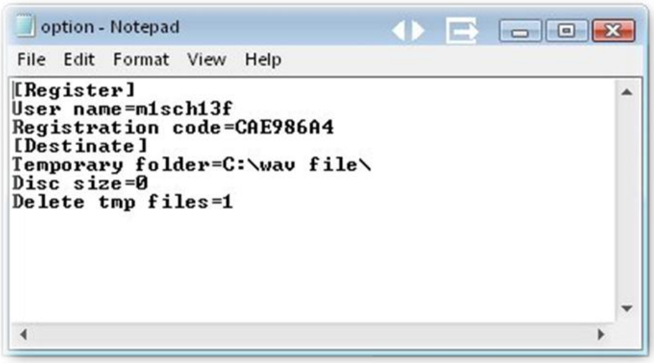
5.Press OK to register.

CD Recorder: K: [1;1;0] DVD+RW RW5125 1.27

Disc size: 650M/74Min

Use space 0.0%

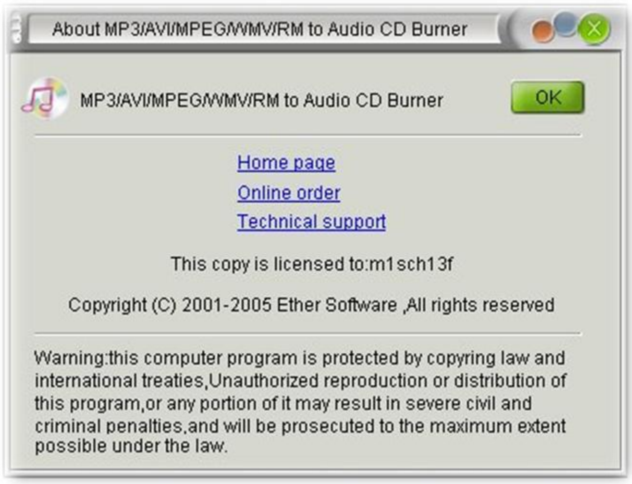
Alright, now let's take a look at option.ini again and make sure our patch did what it was supposed to.



```
[Register]
User name=m1sch13f
Registration code=CAE986A4
[Destinate]
Temporary folder=C:\wav file\
Disc size=0
Delete tmp files=1
```

As you can see, we have our name and our correct 8 character long serial.

If you want to double check, restart the program without the loader and look in the about box.

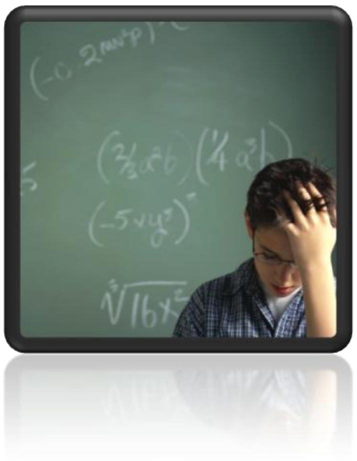

About MP3/AVI/MPEG/WMV/RM to Audio CD Burner
MP3/AVI/MPEG/WMV/RM to Audio CD Burner
OK
Home page
Online order
Technical support

This copy is licensed to:m1sch13f

Copyright (C) 2001-2005 Ether Software ,All rights reserved

Warning:this computer program is protected by copyring law and international treaties,Unauthorized reproduction or distribution of this program,or any portion of it may result in severe civil and criminal penalties,and will be prosecuted to the maximum extent possible under the law.

ARTEAM EZINE #4 CALL FOR PAPERS



ARTeam members are asking for your article submissions on subjects related to Reverse-Engineering.

We wanted to provide the community with somewhere to distribute interesting, sometimes random, reversing information. Not everyone likes to write tutorials, and not everyone feels that the information they have is enough to constitute a publication of any sort. I'm sure all of us have hit upon something interesting while coding/reversing and have wanted to share it but didn't know exactly how. Or if you have cracked some interesting protection but didn't feel like writing a whole step by step tutorial, you can share the basic steps and theory here. If you have an idea for an article, or just something fascinating you want to share, let us know.

Examples of articles are a new way to detect a debugger, or a new way to defeat debugger detection, or how to defeat an interesting crackme..

The eZine is more about sharing knowledge, as opposed to teaching. So the articles can be more generic in nature. You don't have to walk a user through step by step. Instead you can share information from simple theory all the way to "sources included"

What we are looking for in an article submission:

1. Clear thought out article. We are asking you to take pride in what you submit.
2. It doesn't have to be very long. A few paragraphs is fine, but it needs to make sense.
3. Any format is fine, but to save our time possibly send them in WinWord Office or text format.
4. If you include pictures please center them in the article. If possible please add a number and label below each image.
5. If you use references please add them as footnotes where used.
6. If you include code snippets inside a document other than .txt please use a monospace font to allow for better formatting and possibly use a syntax colorizer
7. Anonymous articles are fine. But you must have written it. No plagiarism!
8. Any other questions you may have feel free to ask

We are accepting articles from anyone wanting to contribute. That means you.

We want to make the eZine more of a community project than a team release. If your article is not used, it's not because we don't like it. It may just need some work. We will work with you to help develop your article if it needs it.

Questions or Comments please visit http://forum.accessroot.com