Reverse .NET Software IX

Unpack .Net Reactor 3.9.8.0

www.reaonline.net

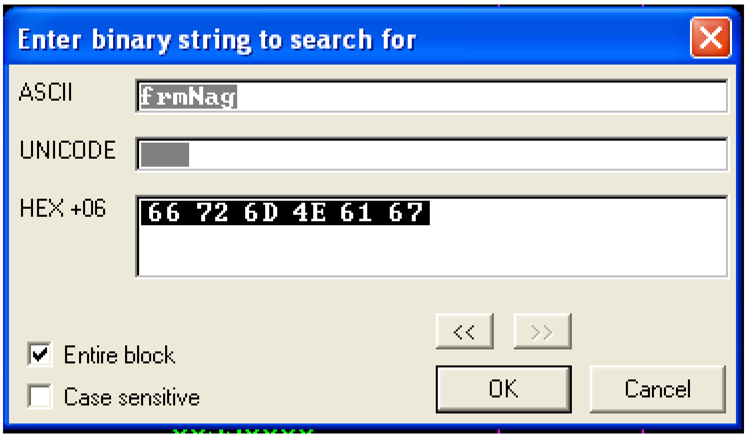
1 Introduction

- It has been a long time since I wrote my 8th article in "Reverse .Net Software" serie. This serie was written in Vietnamese because at the beginning I did not intend to publish it to internet community. It is just for members in REA group. In this serie I would like to dicuss about the protection ways of .net applications and their weaknesses but the .net protectors and obfuscators changed their methods day by day, the present is not like the past anymore. It gets harder to reverse a .net application than before so I decide to continue my series with this article with hoping that it will helps the others to understand more about .Net Framework and .Net Protectors.
- First, I would like to send a great thank to TQN. He helped me a lot by giving me much important information that he got during his reversing work. Without his help I can not finish this article, an article about "Unpacking .Net Reactor 3.9.8.0".
- " *.NET Reactor is a powerful .NET code protection & licensing system which assists developers in protecting their .NET software. Developers are able to protect their software in a safe and simple way now. This way developers can focus more on development than on worrying how to protect their intellectual property"*.
- Many reversers around the world had tried to unpack this packer and most of them had done their jobs successfully. The fact is that .Net Reactor is anyway not the powerful packer for .Net. He wrapped the original assembly and unpacked it again in memory. This method will lead, of course, to a security hole that a reverser can easily dump assembly from memory and get it back.
- The developers of .Net Reactor know about this hole but they can not prevent a reverser from dumping so they tried to modify the memory so that after dumping the memory to file the reverser can not easily start their reverse process because the format of file is now destroyed. A visible result of this anti-dump technique is that the dumped file can not be viewed with .Net Reflector. Therefore after dumping, the reverser must always fix their dump so that the file is exactly constructed again. This terrible job can be executed manually (which causes 100% a nightmare with calculation) or automatically through a tool (for example I wrote a tool .Net Reactor Unpacker to do something like that).
- I also used this method for 2 years to unpack many packers (for example Themida .Net, Cli Secure…) but I really do not like it. It is just so common, it does not tell me at least how the packer works. I just dump the assembly from memory and try to fix the header information to get the original back. It is the work of a PE fixer. However thank to this job now I have a good knowledge about .Net Pe File and write myself a library to parse a .net assembly and use this library in my tools (for example .Net Id). So I would like to introduce in this article a method to unpack .Net Reactor without fixing anything after dumping. That means I will dig deeper to find out how .Net Reactor work and dump the original assembly back which does not need any fixing after that. The version of .Net Reactor which I used is 3.9.8.0 which was

released on 12-Nov-2008. This new method is only academic. It will help us to understand more about how .Net Reactor works, but it can not be applied to unpack an application packed by .Net Reactor because it is time-consuming.

2 Unpacking with old method

- Before introducing new method I would like to use the current one which is used around the world (as I know) to unpack .Net Reactor. The target is my typical SampleCrackme which is packed with a demo version of .Net Reactor. You can find it as attachments of this article. I use OllyIce as my debugger. And let's start.
- Open OllyIce, load the target until it runs. Press Alt – M to open Memory Window, right click at the top of window, choose Search and enter the pattern to find Assembly in memory. The pattern can be the Window's name, caption of lable, caption of button or "Assembly Version" (as suggested by CodeRipper) or something like that.



Enter binary string to search for

ASCII: frmNag

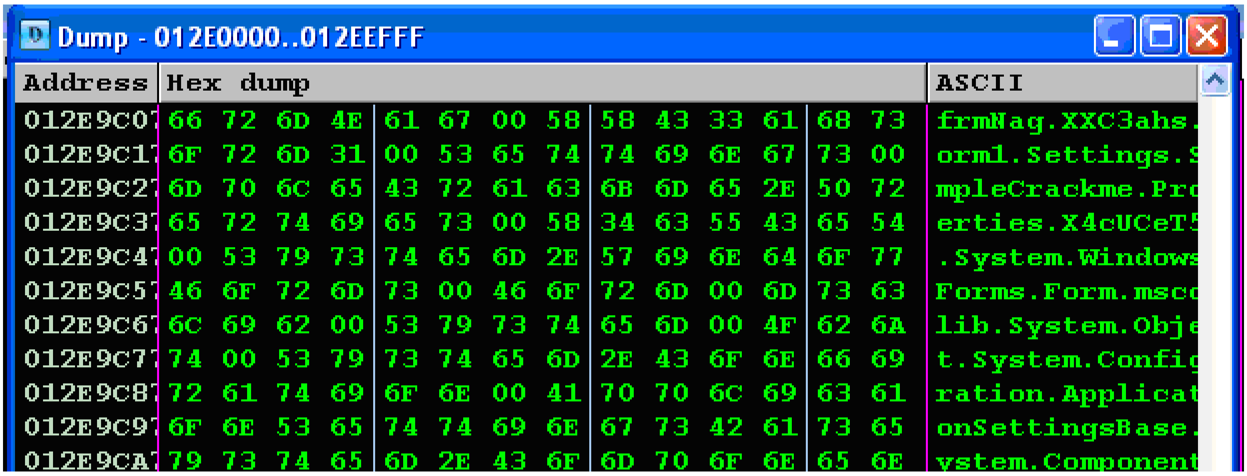
UNICODE:

HEX +06: 66 72 6D 4E 61 67

☑ Entire block

☐ Case sensitive

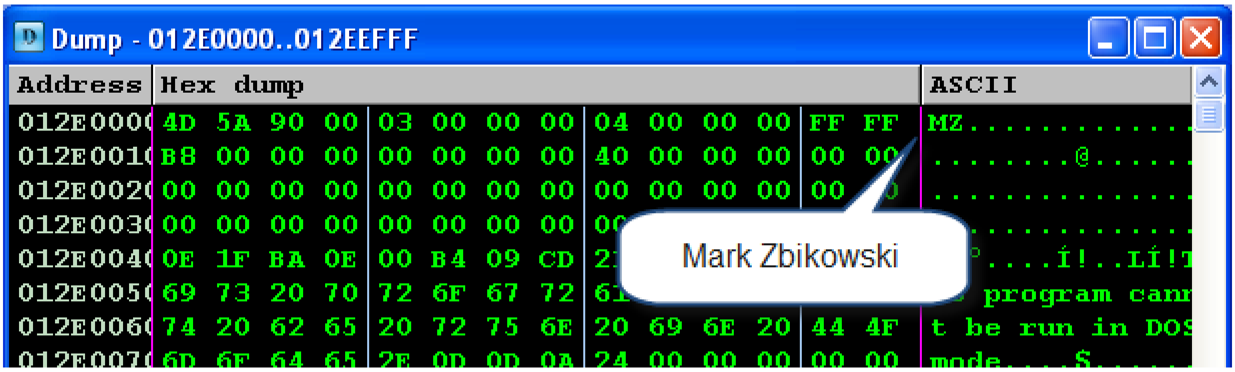
- The search engine will pause at here



Dump - 012E0000..012EEFFF

```
Address  Hex dump                                         ASCII
012E9C0  66 72 6D 4E 61 67 00 58 58 43 33 61 68 73        frmNag.XXC3ahs.
012E9C1  6F 72 6D 31 00 53 65 74 74 69 6E 67 73 00        orm1.Settings.S
012E9C2  6D 70 6C 65 43 72 61 63 6B 6D 65 2E 50 72        mpleCrackme.Pro
012E9C3  65 72 74 69 65 73 00 58 34 63 55 43 65 54        erties.X4cUCeT5
012E9C4  00 53 79 73 74 65 6D 2E 57 69 6E 64 6F 77        .System.Windows
012E9C5  46 6F 72 6D 73 00 46 6F 72 6D 00 6D 73 63        Forms.Form.msco
012E9C6  6C 69 62 00 53 79 73 74 65 6D 00 4F 62 6A        lib.System.Obje
012E9C7  74 00 53 79 73 74 65 6D 2E 43 6F 6E 66 69        t.System.Config
012E9C8  72 61 74 69 6F 6E 00 41 70 70 6C 69 63 61        ration.Applicat
012E9C9  6F 6E 53 65 74 74 69 6E 67 73 42 61 73 65        onSettingsBase.
012E9CA  79 73 74 65 6D 2E 43 6F 6D 70 6F 6E 65 6E        ystem.Component
```

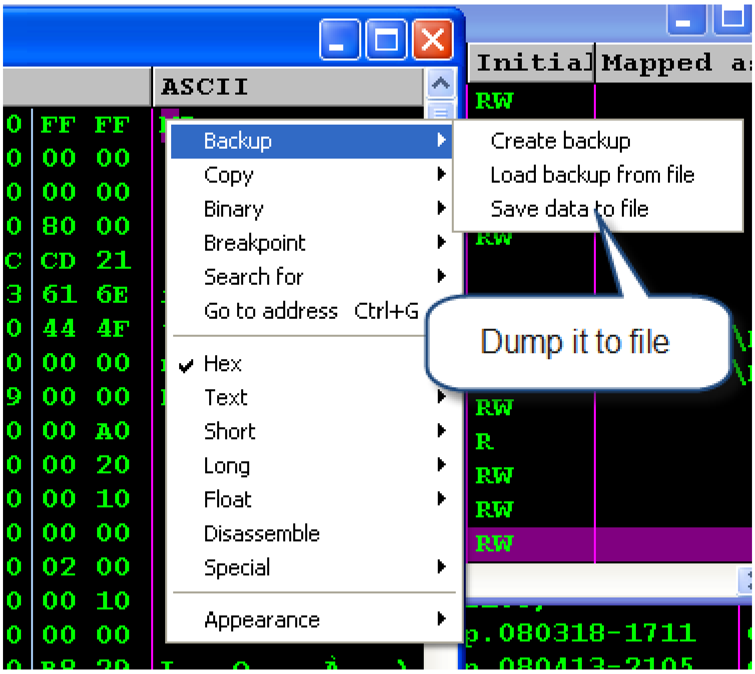
- Scroll up to the beginning of memory section we'll see the MZ-Word. MZ is for MaZic Word? Oh no, it is the initial of Mark Zbikowski, one of the developers of MS-DOS. It indicates that this memory section contains a PE (Portable Executable) file.



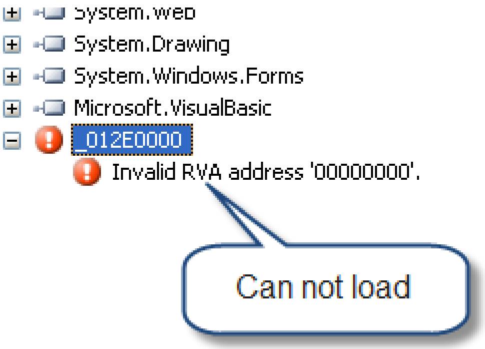
Dump - 012E0000..012EEFFF

```
Address  Hex dump                                     ASCII
012E000( 4D 5A 90 00|03 00 00 00|04 00 00 00|FF FF    MZ............
012E001( B8 00 00 00|00 00 00 00|40 00 00 00|00 00    ........@......
012E002( 00 00 00 00|00 00 00 00|00 00 00 00|00       ...............
012E003( 00 00 00 00|00 00 00 00|00                   ............
012E004( 0E 1F BA 0E|00 B4 09 CD|21                   º....Í!..LÍ!T
012E005( 69 73 20 70|72 6F 67 72|61                   program cann
012E006( 74 20 62 65|20 72 75 6E|20 69 6E 20|44 4F    t be run in DOS
012E007( 6D 6F 64 65|2E 0D 0D 0A|24 00 00 00|00 00    mode....$......
```

- If there is a MZ Word at memory section then scroll down slowly through the memory section to see if this section may be the assembly which we want. If not then search next. How do we know that this memory section is what we need? Then use our feeling, the wanted section will contain strings which are related to the assembly, for example name of assembly, caption of windows, name of company which writes the application…. The size of section can say something too. With this target, the section which we found above is the right one. Let’s dump it to file. I save it under the name _012E0000.exe.



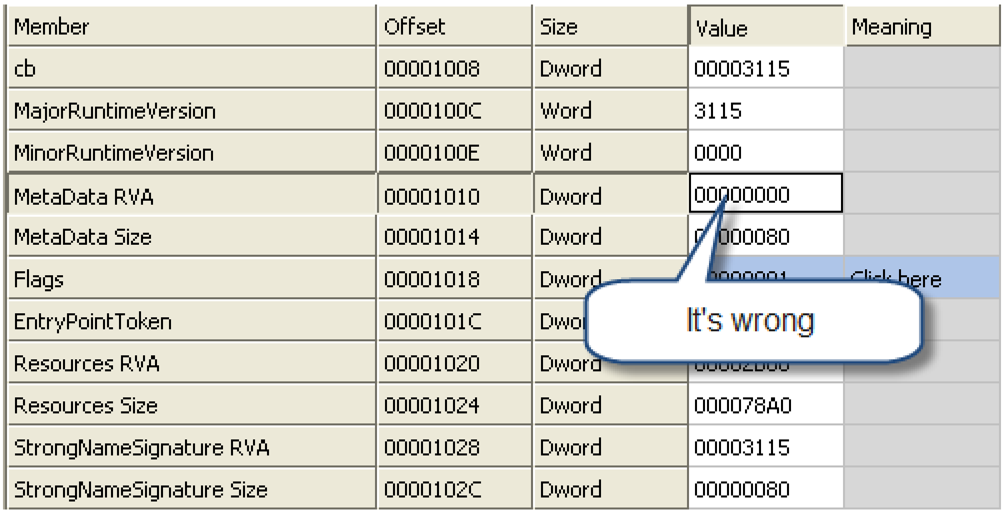
- OllyIce finishes his job, turn him off. Open Reflector and load our dump file.



System.Web
System.Drawing
System.Windows.Forms
Microsoft.VisualBasic
_012E0000
Invalid RVA address '00000000'.

Can not load

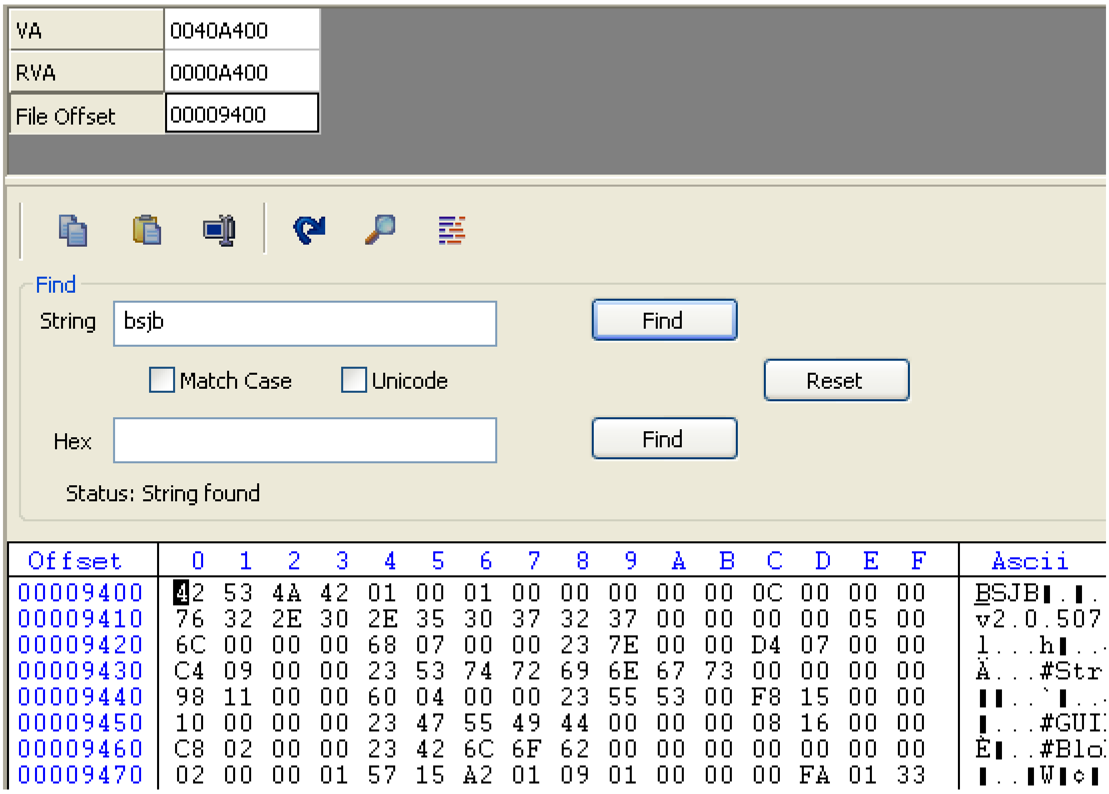
- We know that .Net Reactor destroyed the header of assembly. Such errors are what we are looking for. This is evidence telling us that .Net Reflector has destroyed the metadata so that we can not decompile file anymore after dumping. The .Net Reflector tells us that there is an error at RVA of a metadata element. Use CFF Explorer to open the dump file, go through and we found something wrong with MetaData RVA. It can not be 0x00000000.



Member	Offset	Size	Value	Meaning
cb	00001008	Dword	00003115	
MajorRuntimeVersion	0000100C	Word	3115	
MinorRuntimeVersion	0000100E	Word	0000	
MetaData RVA	00001010	Dword	00000000	
MetaData Size	00001014	Dword	00000080	
Flags	00001018	Dword	00000001	Click here
EntryPointToken	0000101C	Dword	[illegible]	
Resources RVA	00001020	Dword	00002D00	
Resources Size	00001024	Dword	000078A0	
StrongNameSignature RVA	00001028	Dword	00003115	
StrongNameSignature Size	0000102C	Dword	00000080	

It's wrong

- Let's fix it. In CFF Explorer, go to Address Converter, search string "BSJB", we found it at offset 0x9400, enter this value in textbox offset we'll get its RVA is 0XA400. Copy this value and paste it to MetaData RVA. This magic number "BSJB" refers to some of the original developers of the .NET Framework, Brain Harry, Susan Radke-Sproull, Jason Zander and Bill Evans. It seems that Microsoft like to honor their developers by adding their names to the file format. This magic string points to the first entry in the metadata table.
- Two figure below show result of searching and modify the MetaData RVA.



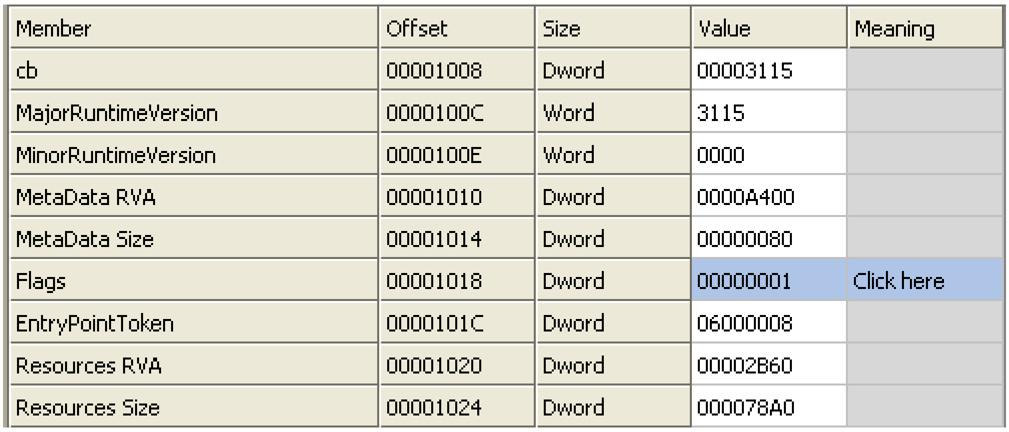
VA	0040A400
RVA	0000A400
File Offset	00009400

String: bsjb

□ Match Case □ Unicode

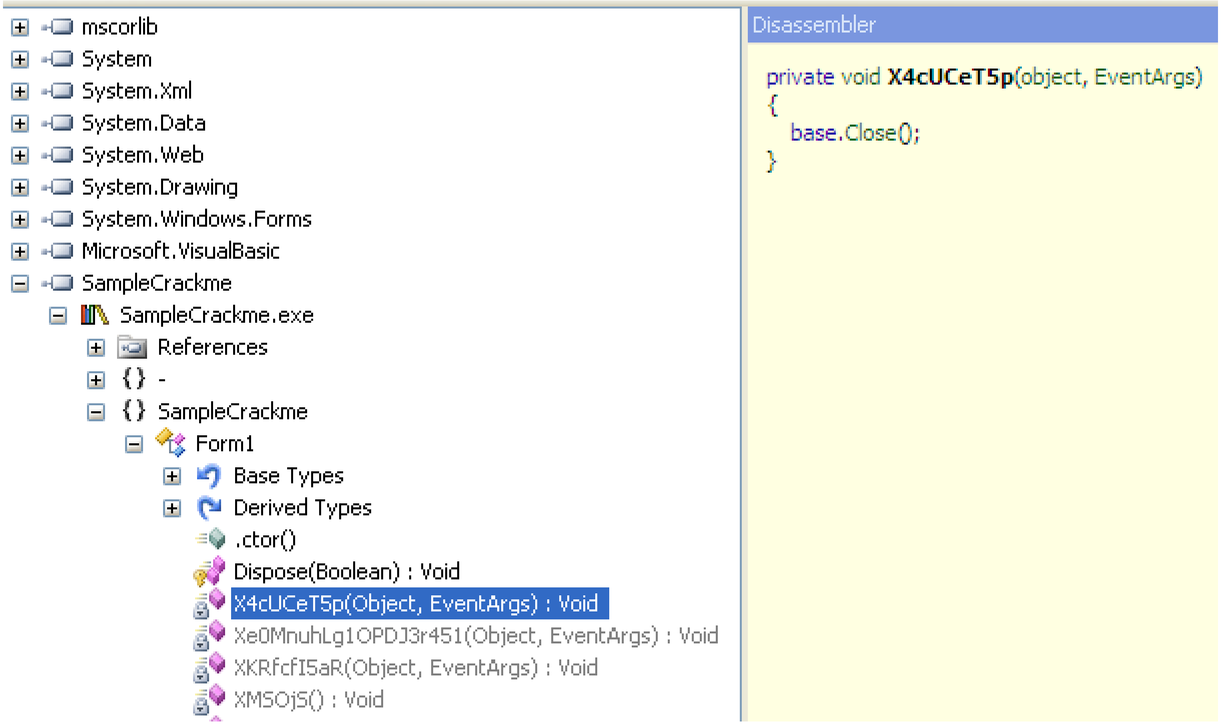
Status: String found

```
Offset     0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   Ascii
00009400  42 53 4A 42 01 00 01 00 00 00 00 00 0C 00 00 00  BSJB▌.▌.
00009410  76 32 2E 30 2E 35 30 37 32 37 00 00 00 00 05 00  v2.0.507
00009420  6C 00 00 00 68 07 00 00 23 7E 00 00 D4 07 00 00  l...h▌..
00009430  C4 09 00 00 23 53 74 72 69 6E 67 73 00 00 00 00  Ä...#Str
00009440  98 11 00 00 60 04 00 00 23 55 53 00 F8 15 00 00  ▌▌..`▌..
00009450  10 00 00 00 23 47 55 49 44 00 00 00 08 16 00 00  ▌...#GUI
00009460  C8 02 00 00 23 42 6C 6F 62 00 00 00 00 00 00 00  È▌..#Blo
00009470  02 00 00 01 57 15 A2 01 09 01 00 00 00 FA 01 33  ▌..▌W▌¢▌
```



Member	Offset	Size	Value	Meaning
cb	00001008	Dword	00003115	
MajorRuntimeVersion	0000100C	Word	3115	
MinorRuntimeVersion	0000100E	Word	0000	
MetaData RVA	00001010	Dword	0000A400	
MetaData Size	00001014	Dword	00000080	
Flags	00001018	Dword	00000001	Click here
EntryPointToken	0000101C	Dword	06000008	
Resources RVA	00001020	Dword	00002B60	
Resources Size	00001024	Dword	000078A0	

- Save our modifications and overwrite the original file. Use .Net Reflector to open it and now we got .Net Reflector to work. Reflector can now decompile the assembly. How easy it is! However how does .Net Reactor really work? When did he unpack the assembly to memory? When did he destroy the Metadata header? We'll find out in next section.

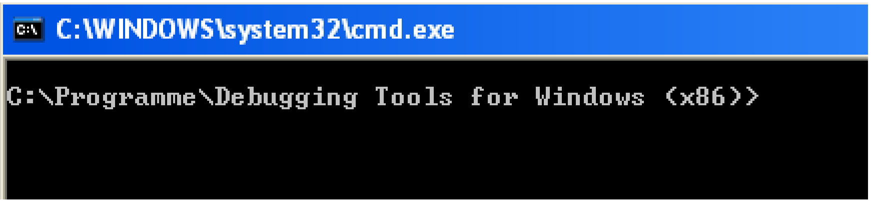


```
private void X4cUCeT5p(object, EventArgs)
{
   base.Close();
}
```

- The assembly can be view with Reflector. No error with Metadata anymore.

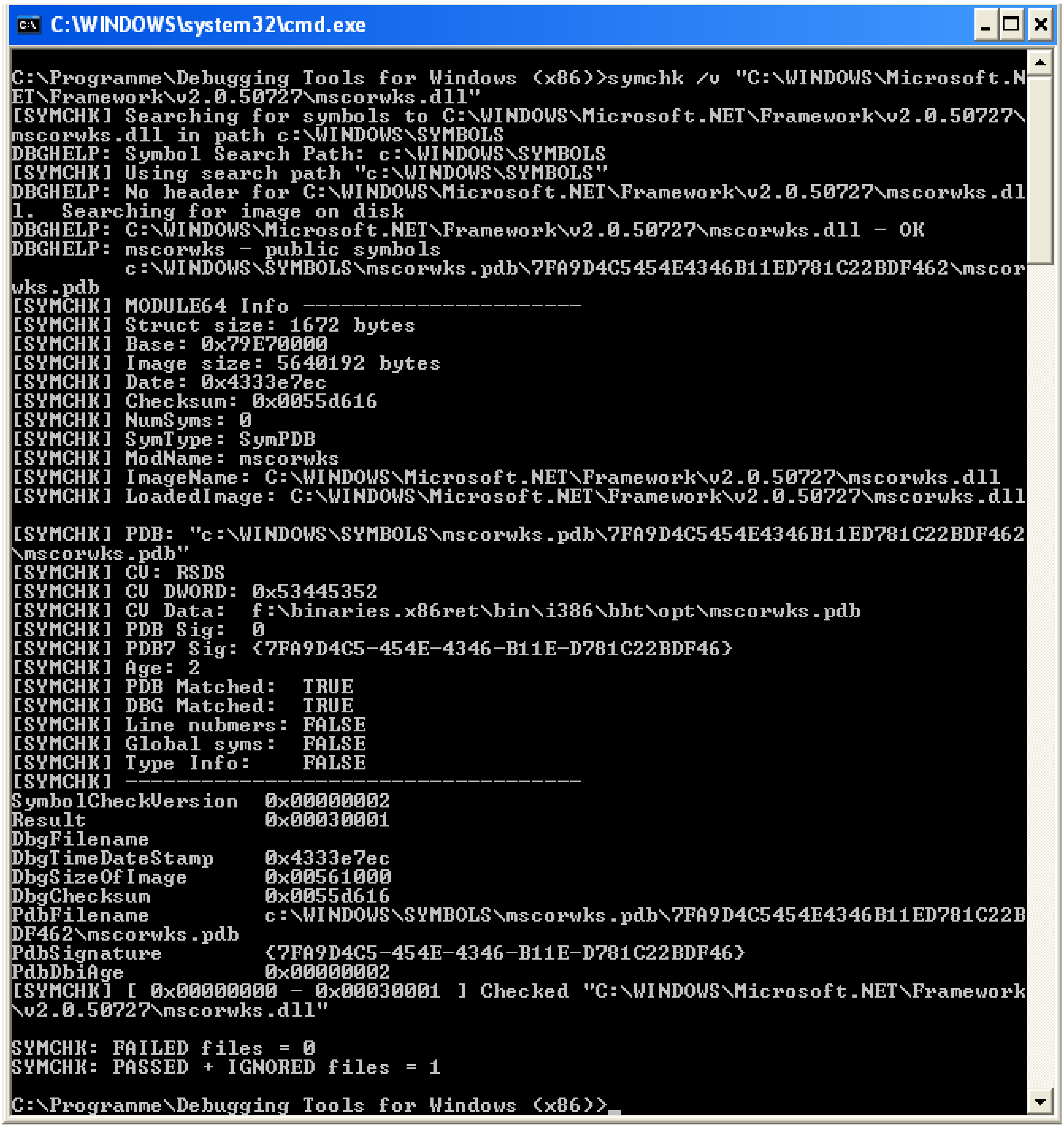
3 Unpack with new method

- To find out more about how .Net Reactor works, we need to debug the .Net Framework, set breakpoint at some important functions and see what happened. To do that we need to make our OllyIce to be able to load with symbol file of .Net Framework which provides much useful information about the functions of a file? The symbol file may be achieved in many ways but I know only one way through WinDbg. If you know more, then please share your way with me.
- So go to download WinDbg, install it. Open command console, browse to the folder where we installed WinDbg. For example I install it under the folder Programme\Debugging Tools for Windows (x86)



```
C:\Programme\Debugging Tools for Windows (x86)>
```

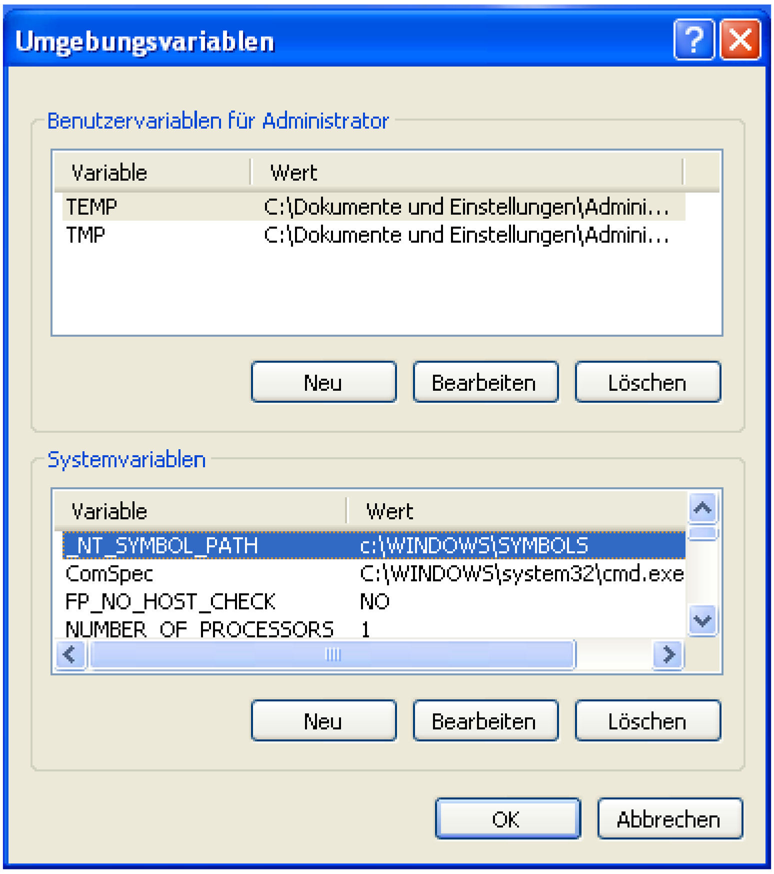
- Enter this command *symchk /v "C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\mscorwks.dll*". It will load the symbol file of mscorwks.dll from Microsoft server. The link to your mscorwks.dll at local computer may vary with mine. So please be sure that you provide the correct link to symchk. If not, symchk can not load the symbol file to our local computer. After execution of symchk, it will give the result back. In my result, symchk did his job successfully. No failed file and one passed/ignored file because I downloaded the symbol for mscorwks before. Symchk just check to find out if there are any updates for this file, he found no update so he just passed.
- Mscorwks.dll and Mscorjit.dll are two significant DLLs of .net framework. When a assembly is loaded, mscorwks.dll will validate its PE Header, IL format, verify strong name,… So we will load its symbol to provide more info to OllyIce so that we can make our debug better.



```
C:\Programme\Debugging Tools for Windows (x86)>symchk /v "C:\WINDOWS\Microsoft.N
ET\Framework\v2.0.50727\mscorwks.dll"
[SYMCHK] Searching for symbols to C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\
mscorwks.dll in path c:\WINDOWS\SYMBOLS
DBGHELP: Symbol Search Path: c:\WINDOWS\SYMBOLS
[SYMCHK] Using search path "c:\WINDOWS\SYMBOLS"
DBGHELP: No header for C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\mscorwks.dl
l.  Searching for image on disk
DBGHELP: C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\mscorwks.dll - OK
DBGHELP: mscorwks - public symbols
         c:\WINDOWS\SYMBOLS\mscorwks.pdb\7FA9D4C5454E4346B11ED781C22BDF462\mscor
wks.pdb
[SYMCHK] MODULE64 Info ----------------------
[SYMCHK] Struct size: 1672 bytes
[SYMCHK] Base: 0x79E70000
[SYMCHK] Image size: 5640192 bytes
[SYMCHK] Date: 0x4333e7ec
[SYMCHK] Checksum: 0x0055d616
[SYMCHK] NumSyms: 0
[SYMCHK] SymType: SymPDB
[SYMCHK] ModName: mscorwks
[SYMCHK] ImageName: C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\mscorwks.dll
[SYMCHK] LoadedImage: C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\mscorwks.dll

[SYMCHK] PDB: "c:\WINDOWS\SYMBOLS\mscorwks.pdb\7FA9D4C5454E4346B11ED781C22BDF462
\mscorwks.pdb"
[SYMCHK] CV: RSDS
[SYMCHK] CV DWORD: 0x53445352
[SYMCHK] CV Data:  f:\binaries.x86ret\bin\i386\bbt\opt\mscorwks.pdb
[SYMCHK] PDB Sig:  0
[SYMCHK] PDB7 Sig: {7FA9D4C5-454E-4346-B11E-D781C22BDF46}
[SYMCHK] Age: 2
[SYMCHK] PDB Matched:  TRUE
[SYMCHK] DBG Matched:  TRUE
[SYMCHK] Line nubmers: FALSE
[SYMCHK] Global syms:  FALSE
[SYMCHK] Type Info:    FALSE
[SYMCHK] ------------------------------------
SymbolCheckVersion  0x00000002
Result              0x00030001
DbgFilename
DbgTimeDateStamp    0x4333e7ec
DbgSizeOfImage      0x00561000
DbgChecksum         0x0055d616
PdbFilename         c:\WINDOWS\SYMBOLS\mscorwks.pdb\7FA9D4C5454E4346B11ED781C22B
DF462\mscorwks.pdb
PdbSignature        {7FA9D4C5-454E-4346-B11E-D781C22BDF46}
PdbDbiAge           0x00000002
[SYMCHK] [ 0x00000000 - 0x00030001 ] Checked "C:\WINDOWS\Microsoft.NET\Framework
\v2.0.50727\mscorwks.dll"

SYMCHK: FAILED files = 0
SYMCHK: PASSED + IGNORED files = 1

C:\Programme\Debugging Tools for Windows (x86)>_
```

- We have now the symbol file of mscowks; in next step we must configure our Olly so that he can work with this symbol file. A command plugin for OllyDbg of anonymouse can do this job perfectly.
- Before using this plugin we need to configure it. Let's add an environment variable _NT_SYMBOL_PATH with the value C:\Windows\Symbols. The value of this environment variable is the path to where symchk saved the symbol file at local computer. This value stands in the result of symchk command

after execution too. For example, we can find it in the figure above at some first rows.



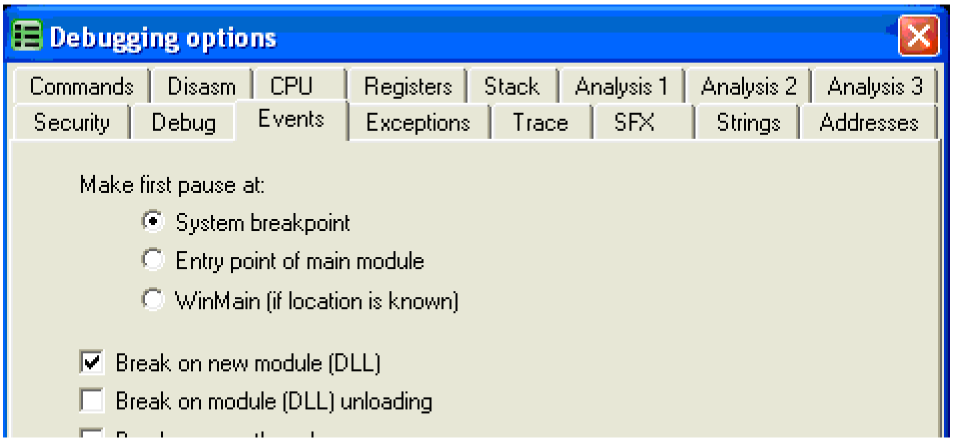
Benutzervariablen für Administrator

Variable	Wert
TEMP	C:\Dokumente und Einstellungen\Admini...
TMP	C:\Dokumente und Einstellungen\Admini...

Systemvariablen

Variable	Wert
_NT_SYMBOL_PATH	c:\WINDOWS\SYMBOLS
ComSpec	C:\WINDOWS\system32\cmd.exe
FP_NO_HOST_CHECK	NO
NUMBER_OF_PROCESSORS	1

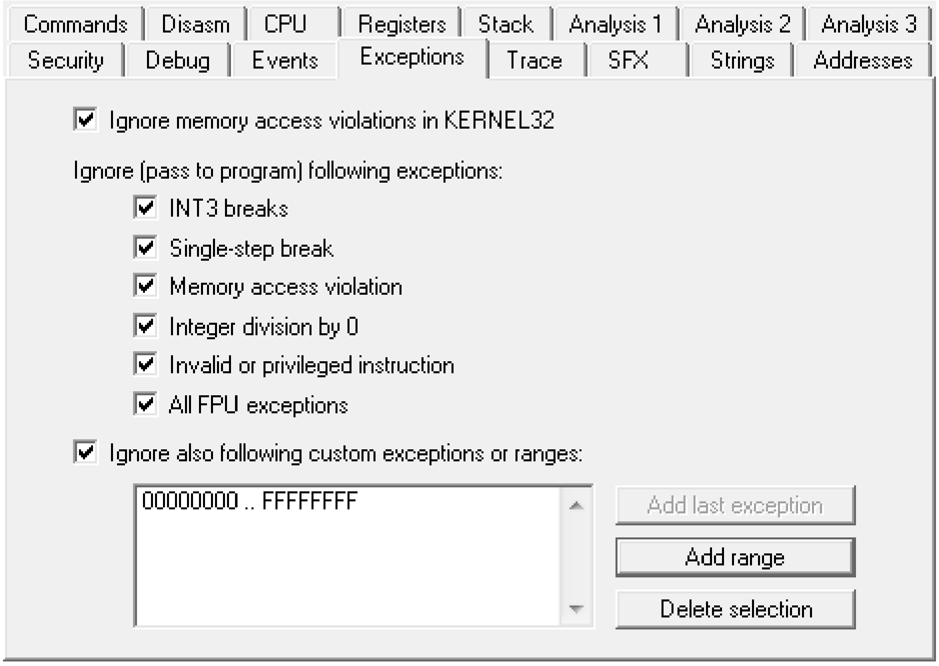
- With this help of this plugin, OllyIce can now work with the symbol file. Open OllyIce, go to Debugging Options, be sure that "Make first pause at: System breakpoint" and "Break on new module (DLL)" and all exceptions must be passed



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



☑ Ignore memory access violations in KERNEL32

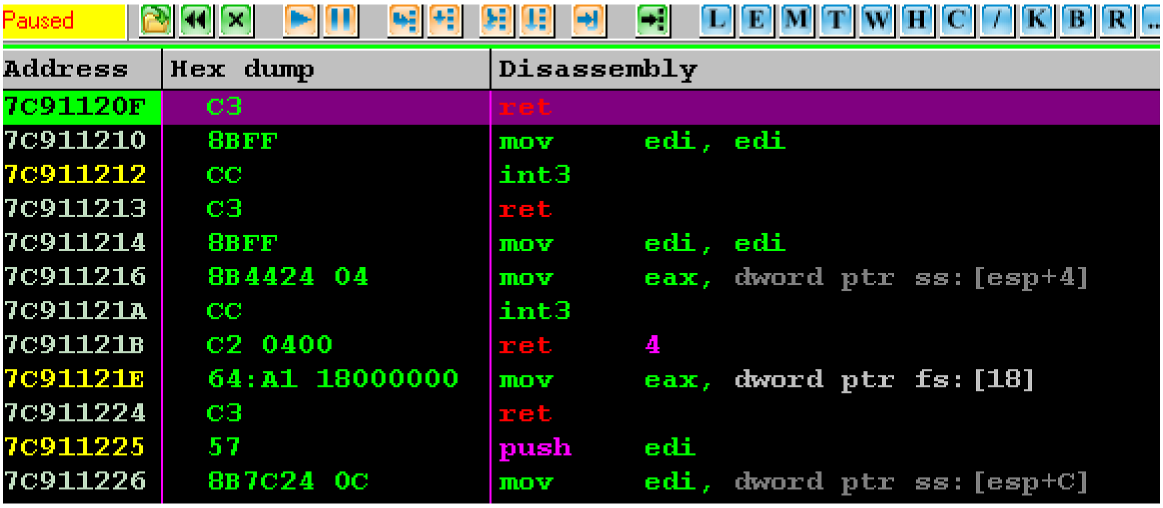
Ignore (pass to program) following exceptions:

- ☑ 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:

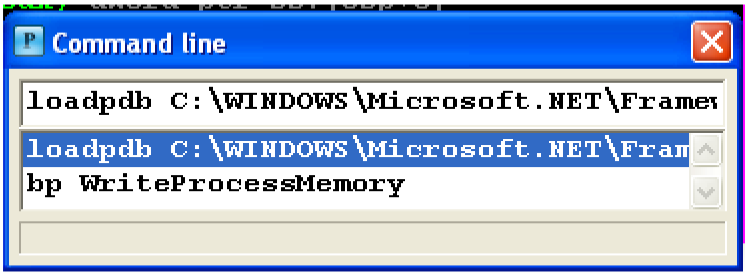
00000000 .. FFFFFFFF

- Load the SampleCrackme into OllyIce, it will land at this command



```
Address   Hex dump         Disassembly
7C91120F  C3               ret
7C911210  8BFF             mov     edi, edi
7C911212  CC               int3
7C911213  C3               ret
7C911214  8BFF             mov     edi, edi
7C911216  8B4424 04        mov     eax, dword ptr ss:[esp+4]
7C91121A  CC               int3
7C91121B  C2 0400          ret     4
7C91121E  64:A1 18000000   mov     eax, dword ptr fs:[18]
7C911224  C3               ret
7C911225  57               push    edi
7C911226  8B7C24 0C        mov     edi, dword ptr ss:[esp+C]
```

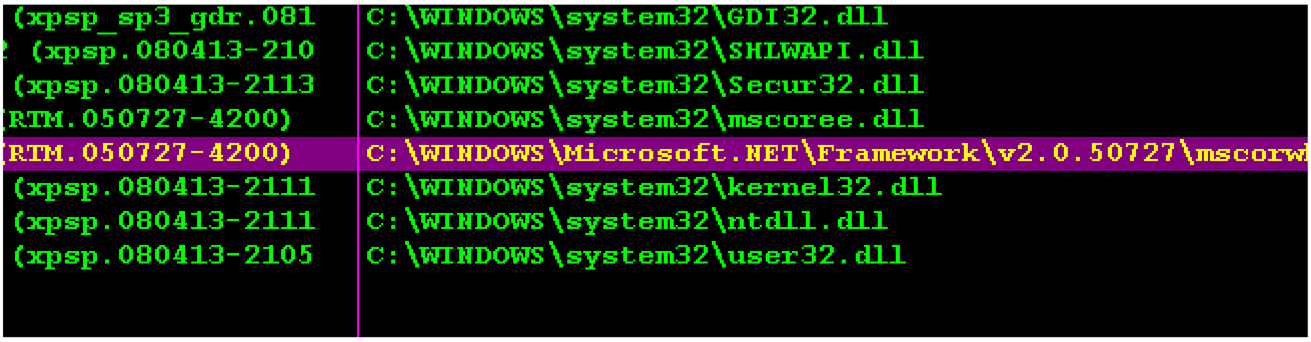
- Open Command line plugin, enter this command *loadpdb C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\mscorwks.dll*. This command will load the symbol file into OllyIce and we'll have more information when debugging mscorwks.dll.



```
loadpdb C:\WINDOWS\Microsoft.NET\Framew
```

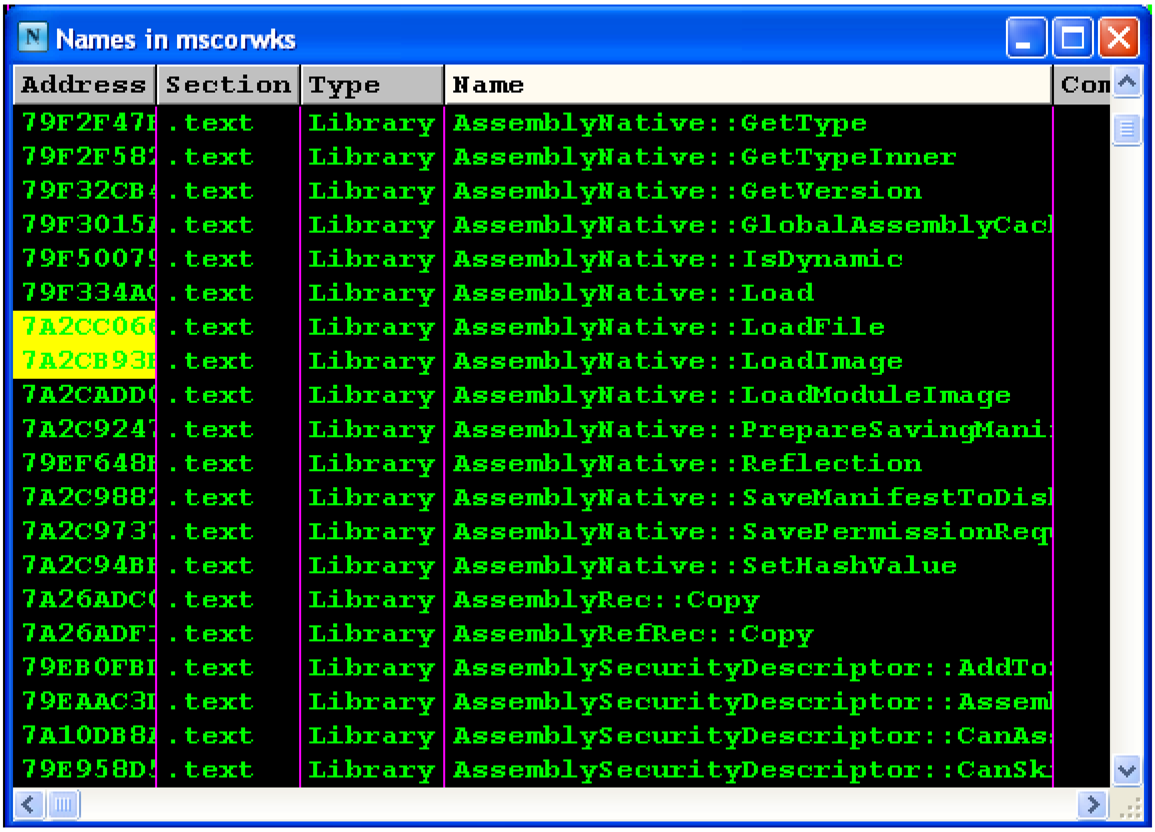
```
loadpdb C:\WINDOWS\Microsoft.NET\Fram
bp WriteProcessMemory
```

- Press F9 so that OllyIce starts to run SampleCrackme. Everytime when a module loads, OllyIce will stop. Just press F9 until mscorwks is loaded.



```
(xpsp_sp3_gdr.081    C:\WINDOWS\system32\GDI32.dll
 (xpsp.080413-210    C:\WINDOWS\system32\SHLWAPI.dll
(xpsp.080413-2113    C:\WINDOWS\system32\Secur32.dll
RTM.050727-4200)     C:\WINDOWS\system32\mscoree.dll
RTM.050727-4200)     C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\mscorw
(xpsp.080413-2111    C:\WINDOWS\system32\kernel32.dll
(xpsp.080413-2111    C:\WINDOWS\system32\ntdll.dll
(xpsp.080413-2105    C:\WINDOWS\system32\user32.dll
```

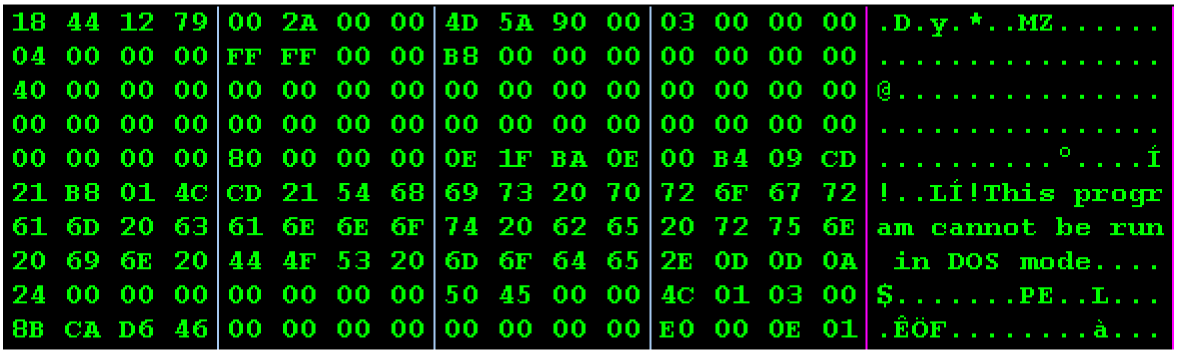
- Right click on the record of mscorwks, click View names.



Names in mscorwks

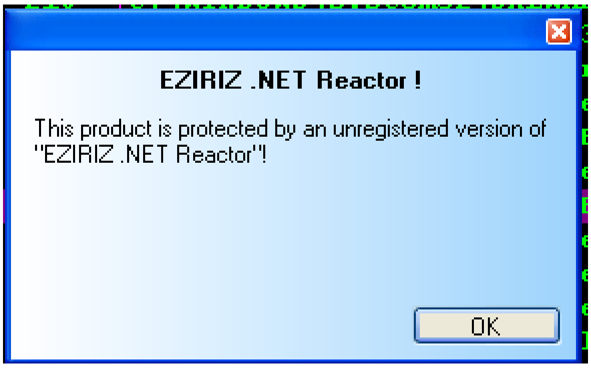
Address	Section	Type	Name	Con
79F2F47	.text	Library	AssemblyNative::GetType	
79F2F58	.text	Library	AssemblyNative::GetTypeInner	
79F32CB	.text	Library	AssemblyNative::GetVersion	
79F3015	.text	Library	AssemblyNative::GlobalAssemblyCac	
79F5007	.text	Library	AssemblyNative::IsDynamic	
79F334A	.text	Library	AssemblyNative::Load	
7A2CC06	.text	Library	AssemblyNative::LoadFile	
7A2CB93	.text	Library	AssemblyNative::LoadImage	
7A2CADD	.text	Library	AssemblyNative::LoadModuleImage	
7A2C924	.text	Library	AssemblyNative::PrepareSavingMani	
79EF648	.text	Library	AssemblyNative::Reflection	
7A2C988	.text	Library	AssemblyNative::SaveManifestToDis	
7A2C973	.text	Library	AssemblyNative::SavePermissionReq	
7A2C94B	.text	Library	AssemblyNative::SetHashValue	
7A26ADC	.text	Library	AssemblyRec::Copy	
7A26ADF	.text	Library	AssemblyRefRec::Copy	
79EB0FB	.text	Library	AssemblySecurityDescriptor::AddTo	
79EAAC3	.text	Library	AssemblySecurityDescriptor::Assem	
7A10DB8	.text	Library	AssemblySecurityDescriptor::CanAs	
79E958D	.text	Library	AssemblySecurityDescriptor::CanSk	

- It is very beautiful. We have the names of all functions. They are very meaningful. It'll surely help us a lot in reversing .Net application. After going through this list, let's set breakpoint on the function AssemblyNative:LoadImage. It looks so interesting and may bring us much useful information.
- Press F9 so that OllyIce continues his job until we break at AssemblyNative:LoadImage, right click on ECX register, follow in dump and we see



```
18 44 12 79|00 2A 00 00|4D 5A 90 00|03 00 00 00|.D.y.*..MZ......
04 00 00 00|FF FF 00 00|B8 00 00 00|00 00 00 00|................
40 00 00 00|00 00 00 00|00 00 00 00|00 00 00 00|@...............
00 00 00 00|00 00 00 00|00 00 00 00|00 00 00 00|................
00 00 00 00|80 00 00 00|0E 1F BA 0E|00 B4 09 CD|..........°....Í
21 B8 01 4C|CD 21 54 68|69 73 20 70|72 6F 67 72|!..LÍ!This progr
61 6D 20 63|61 6E 6E 6F|74 20 62 65|20 72 75 6E|am cannot be run
20 69 6E 20|44 4F 53 20|6D 6F 64 65|2E 0D 0D 0A| in DOS mode....
24 00 00 00|00 00 00 00|50 45 00 00|4C 01 03 00|$.......PE..L...
8B CA D6 46|00 00 00 00|00 00 00 00|E0 00 0E 01|.ÊÖF........à...
```

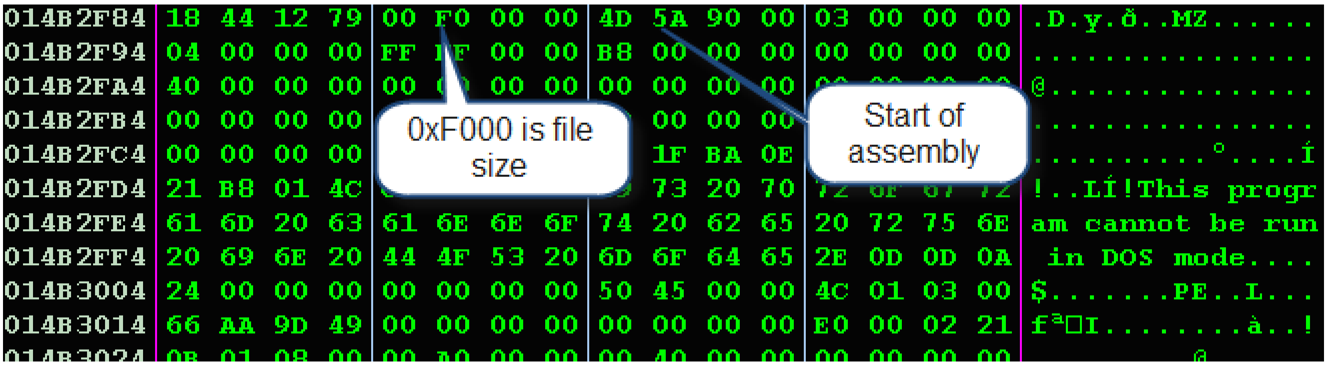
- A great jump, ECX pointed to an unknown struct. This struct contains an unknown pointer (DWORD), the size of assembly and the complete assembly. We can now easily dump assembly from memory with the exact size (no redundant bytes anymore) and the dumped file can be an original one with the probability up to 90%. However it is not easy to find out when we can get the assembly that we need. As you know, when an assembly is loaded, the .Net framework will load it references so when this breakpoint stops OllyIce, it does not mean that ECX points to the assembly which we want but it can be its references, the loader or something like that. So we must to be sure that ECX points to our wanted assembly.
- In this example, I packed my Sample Crackme with a demo version of .Net Reactor. Hence a nag reminding me to buy a full version comes always up before my Sample Crackme runs. So I take this nag as a signal telling me that the breakpoint which breaks after this nag will point to my assembly. Or like AS Protect, you can count the number of times that this breakpoint breaks and the last break before the assembly completely runs will lead us to the memory section of the assembly.



EZIRIZ .NET Reactor !

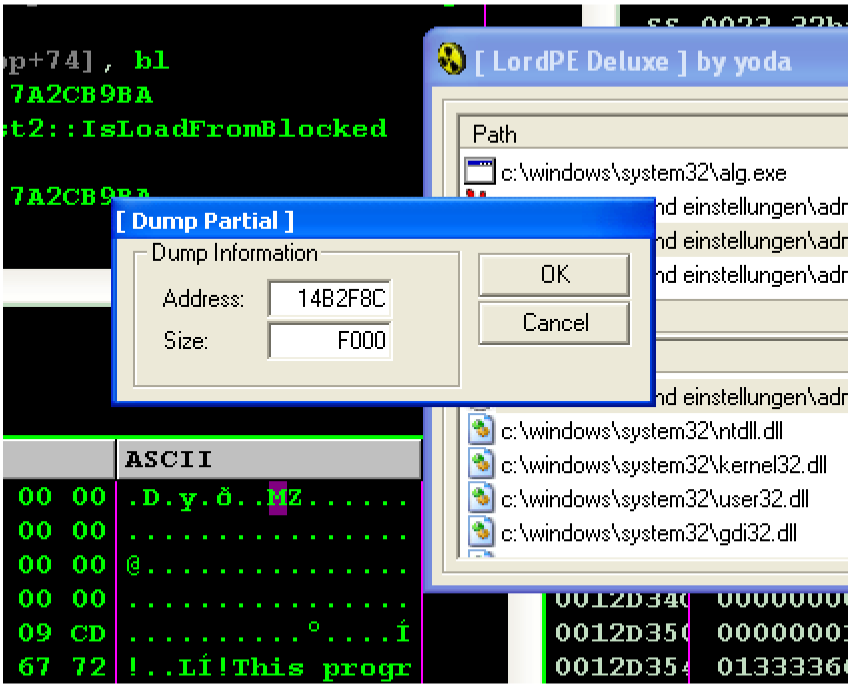
This product is protected by an unregistered version of "EZIRIZ .NET Reactor"!

- After this nag is loaded, OllyIce will break at the breakpoint AssemblyNative:LoadImage again. We have now ECX point to our assembly with size of 0xF000 as shown in figure below.



```
014B2F84 18 44 12 79 00 F0 00 00 4D 5A 90 00 03 00 00 00 .D.y.ð..MZ......
014B2F94 04 00 00 00 FF FF 00 00 B8 00 00 00 00 00 00 00 ................
014B2FA4 40 00 00 00 00    00 00 00 00 00 00             @...............
014B2FB4 00 00 00 00                00 00 00             ................
014B2FC4 00 00 00 00                1F BA 0E             ..........º....Í
014B2FD4 21 B8 01 4C                73 20 70             !..LÍ!This progr
014B2FE4 61 6D 20 63 61 6E 6E 6F 74 20 62 65 20 72 75 6E am cannot be run
014B2FF4 20 69 6E 20 44 4F 53 20 6D 6F 64 65 2E 0D 0D 0A  in DOS mode....
014B3004 24 00 00 00 00 00 00 00 50 45 00 00 4C 01 03 00 $.......PE..L...
014B3014 66 AA 9D 49 00 00 00 00 00 00 00 00 E0 00 02 21 fª□I........à..!
014B3024 0B 01 08 00 00 A0 00 00 00 40 00 00 00 00 00 00
```

- Let’s dump this memory to file; I use LordPE to do that. At my local computer the address is 0x14B2F8C and size is of course 0xF000.



[Dump Partial]

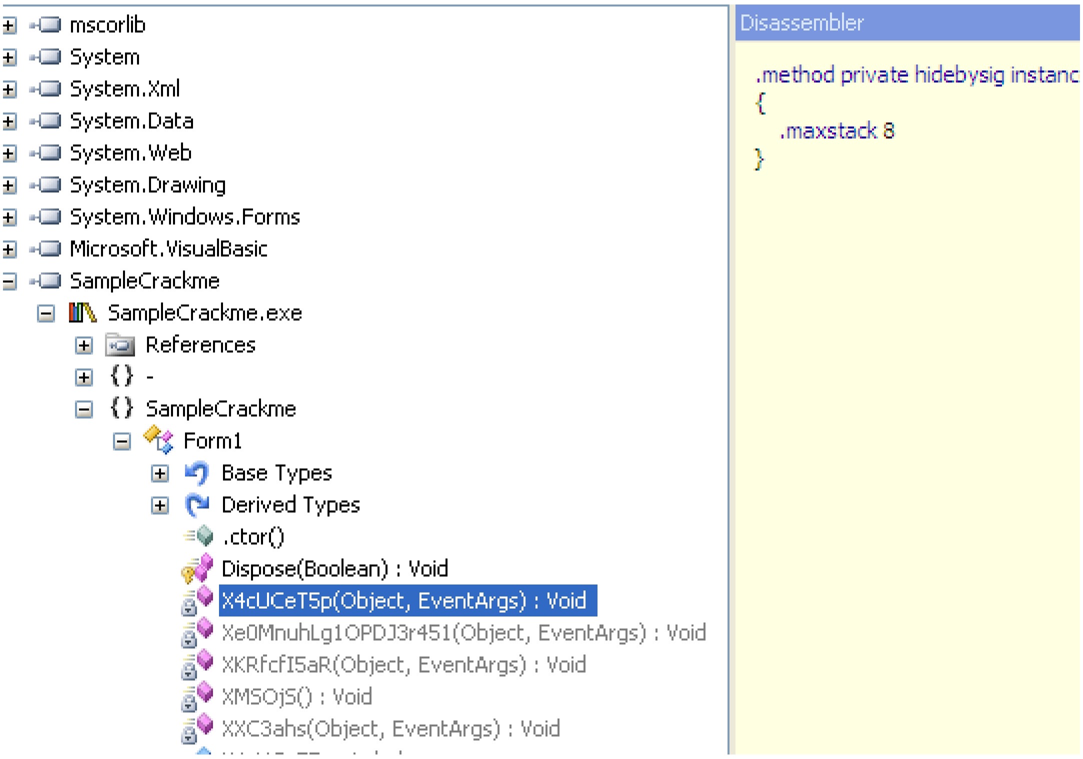
Dump Information

Address: 14B2F8C

Size: F000

[LordPE Deluxe] by yoda
Path
c:\windows\system32\alg.exe
c:\windows\system32\ntdll.dll
c:\windows\system32\kernel32.dll
c:\windows\system32\user32.dll
c:\windows\system32\gdi32.dll
OK
Cancel
ASCII

- Let’s view the dumped file in Reflector. Oh, we have a correct header. Everything looks beautiful, no crashes, no need to fix any value in header file but where is our IL code? Did .Net Reactor hide it anywhere? Is there any IL Code in file or are they all changed to native code?

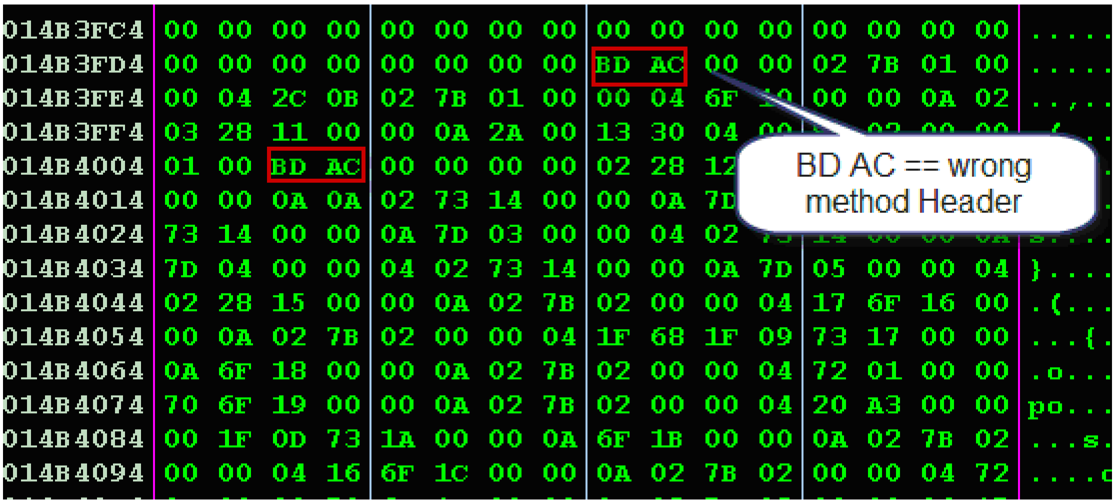


```
+ mscorlib
+ System
+ System.Xml
+ System.Data
+ System.Web
+ System.Drawing
+ System.Windows.Forms
+ Microsoft.VisualBasic
- SampleCrackme
    - SampleCrackme.exe
        + References
        + {} -
        - {} SampleCrackme
            - Form1
                + Base Types
                + Derived Types
                  .ctor()
                  Dispose(Boolean) : Void
                  X4cUCeT5p(Object, EventArgs) : Void
                  Xe0MnuhLg1OPDJ3r451(Object, EventArgs) : Void
                  XKRfcfI5aR(Object, EventArgs) : Void
                  XMSOjS() : Void
                  XXC3ahs(Object, EventArgs) : Void
```

Disassembler

```
.method private hidebysig instanc
{
    .maxstack 8
}
```

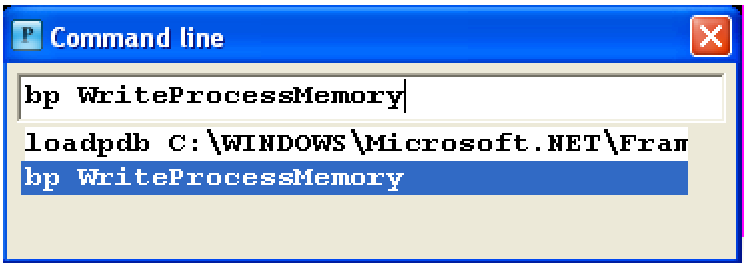
- Return to OllyIce windows, browse to the section of IL Code which starts normally at offset 0x1050. We see that the Method Header of each method is destroyed through replacing 4 bytes at the beginning with 0xBDAC0000. You can read more about Method Header (Fat Header/Tiny Header) in the documentation of Microsoft about .Net PE File Format.



```
014B3FC4 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
014B3FD4 00 00 00 00 00 00 00 00 BD AC 00 00 02 7B 01 00 .....
014B3FE4 00 04 2C 0B 02 7B 01 00 00 04 6F 10 00 00 0A 02 ..,..
014B3FF4 03 28 11 00 00 0A 2A 00 13 30 04 00
014B4004 01 00 BD AC 00 00 00 00 02 28 12
014B4014 00 00 0A 0A 02 73 14 00 00 0A 7D
014B4024 73 14 00 00 0A 7D 03 00 00 04 02
014B4034 7D 04 00 00 04 02 73 14 00 00 0A 7D 05 00 00 04 }....
014B4044 02 28 15 00 00 0A 02 7B 02 00 00 04 17 6F 16 00 .(...
014B4054 00 0A 02 7B 02 00 00 04 1F 68 1F 09 73 17 00 00 ...{.
014B4064 0A 6F 18 00 00 0A 02 7B 02 00 00 04 72 01 00 00 .o...
014B4074 70 6F 19 00 00 0A 02 7B 02 00 00 04 20 A3 00 00 po...
014B4084 00 1F 0D 73 1A 00 00 0A 6F 1B 00 00 0A 02 7B 02 ...s.
014B4094 00 00 04 16 6F 1C 00 00 0A 02 7B 02 00 00 04 72 ....
```

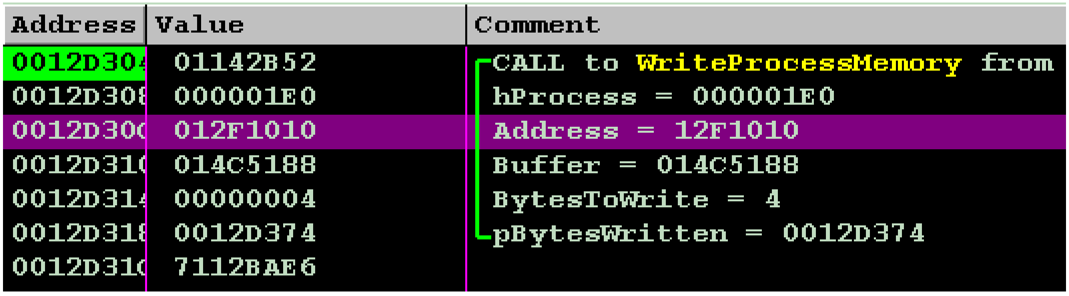
BD AC == wrong method Header

- .Net Reactor must anywhere restore these bytes back so that CLR can compile the code. A breakpoint at WriteProcessMemory should be a good candidate to find out where .Net Reactor tries to write the original value back.



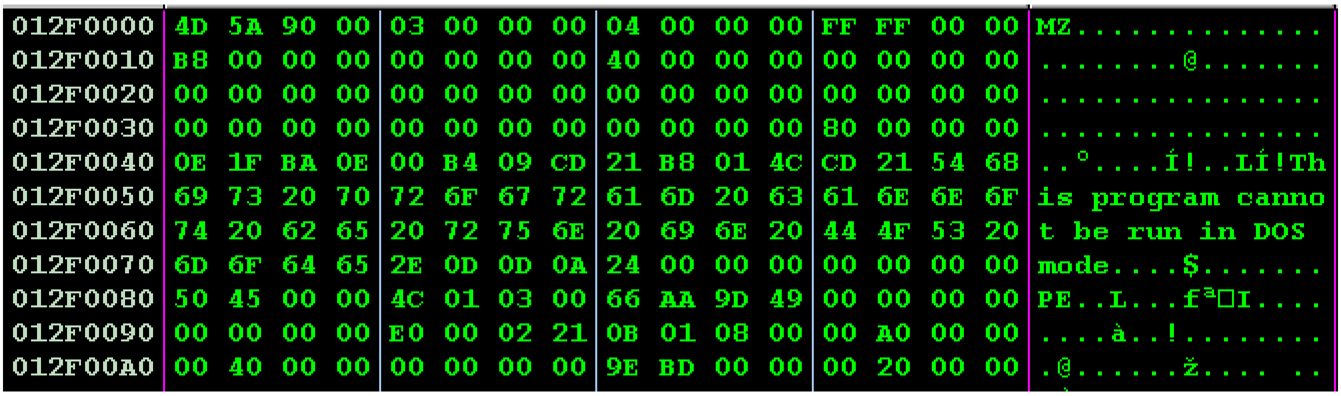
```
Command line
bp WriteProcessMemory
loadpdb C:\WINDOWS\Microsoft.NET\Fran
bp WriteProcessMemory
```

- Press F9 so that OllyIce runs forward and we will land here.



```
Address   Value      Comment
0012D304  01142B52   ┌CALL to WriteProcessMemory from
0012D308  000001E0   │hProcess = 000001E0
0012D30C  012F1010   │Address = 12F1010
0012D310  014C5188   │Buffer = 014C5188
0012D314  00000004   │BytesToWrite = 4
0012D318  0012D374   └pBytesWritten = 0012D374
0012D31C  7112BAE6
```

- Scroll up to 0x12F0000 we see a magic word MZ, scroll down to take a look at whole memory section we see that it may be contain the assembly which we want.



```
012F0000 4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00 MZ..............
012F0010 B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 ........@.......
012F0020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
012F0030 00 00 00 00 00 00 00 00 00 00 00 00 80 00 00 00 ................
012F0040 0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68 ..º....Í!..LÍ!Th
012F0050 69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F is program canno
012F0060 74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20 t be run in DOS 
012F0070 6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00 mode....$.......
012F0080 50 45 00 00 4C 01 03 00 66 AA 9D 49 00 00 00 00 PE..L...fª□I....
012F0090 00 00 00 00 E0 00 02 21 0B 01 08 00 00 A0 00 00 ....à..!........
012F00A0 00 40 00 00 00 00 00 00 9E BD 00 00 00 20 00 00 .@......ž.... ..
```

- The offset 0x1000 starts usually the MetaData Header of .Net Directory. With the WriteProcessMemory function, .Net Reactor try to write garbage value into this MetaData Header and restore the Method Header of each method back. To stop him to do what he wants, everytime when he tries to modify value from offset 0x1000 to 0x1050 (a usual range for MetaData Header of .net application) , we modify the BytesToWrite to 0 so that he can not write any value to destroy the header and with that way we only allow him to restore the method header back.
- .Net Reactor will modify the MetaData Header first and he’ll restore the method header back and at last he will modify the MetaData Header again. So to get our original assembly back, at the first round when he modifies our

MetaData Header, we modify the BytesToWrite to 0. In the second round when he starts to restore our method header back, we just press F9 to go through. In the last round, when he wants to modify our metadata header again, we have already let him fix all Method Header. We just dump the original assembly back and we'll get full working file which can be viewed in Reflector.

4 Conclusion

- We have tried to unpack .Net Reactor with old method and new method. You can realize that the new one is just for studying how .Net Reactor works actually. It can not be applied in reality because for example unpacking a file with more than thousands of method, we can not sit and press F9 until the third round (WriteProcessMemory tries to write at offset 0x1000 to 0x1050 again).
- However the new method provides us a deeper look about .Net Framework and the way .Net Reactor works. There are a lot of interesting functions of mscorwks.dll which we can set a breakpoint and see what happens. I am looking forward to see other articles showing the art of playing with mscorwks.dll and mscorjit.dll file.

5 Links in article

- REA http://reaonline.net/index.php
- .Net Reactor http://www.eziriz.com/
- .Net Reflector http://www.red-gate.com/products/reflector/
- .Net Reactor Unpacker http://rongchaua.net/tools-mainmenu-36/80-reacfixer
- .Net PE Library http://rongchaua.net/tools-mainmenu-36/117-net-pe-file-format-library
- .Net Id http://rongchaua.net/tools-mainmenu-36/131-net-id
- Command Line Plugin of anynomouse http://www.openrce.org/downloads/details/206/Modified_CmdLine_Plug-in
- .Net PE File Format http://download.microsoft.com/download/7/3/3/733AD403-90B2-4064-A81E-01035A7FE13C/MS%20Partition%20II.pdf

6 References

- http://portal.acm.org/citation.cfm?id=579355

7 The end

- I wrote this documentation just for storing my thinking flow during reverse process. It is just a notice and I do not intend to write it as an article. Therefore in some section I just discuss main idea and of course it is not completely explained. I hope you can emphatize with me.
- I do write something wrong please contact to correct me.
- This artice was written as a reference for member in REA so I would like to present REA members with this one.

- This artice is aimed for education. I am not responsible for the reader's activity when the reader use it for their aims.

RONGCHAUA

Rongchaua
My Email : rongchaua@rongchaua.net
My Website: www.rongchaua.net

If I make a mistake in this article, please correct me.
12.04.2009-13.04.2009