

.NET Reverse Engineering Tutorial Episode 2

Version 1.0
November 2006



1. Forewords

We begin another chapter in the ongoing ARTeam series of Reverse Engineering tutorials. Today's topic will go over .NET Reverse Engineering. This topic has been hardly touched upon, thus giving me room to add some information to the reader

BEE Seeing YA.

MDMA



Editor: MaDMAn_H3rCuL3s



Disclaimers

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

All the commercial programs used within this document 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, and cracked versions were available since a lot of time. ARTeam or the authors of the paper cannot be considered responsible damages the companies holding rights on those programs. The scope of this tutorial 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.

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>

Table of Contents

Verification	2
1. .NET Reverse Engineering Tutorial Episode 2	3
1.1. Abstract.....	3
1.2. Targets.....	3
1.3. Reversing the Application.....	3
1.3.1 Preparation	3
1.3.2 Checking out the target.....	3
1.4. References.....	17
1.5. Conclusions	17
1.6. Greetings	17
Document History	17



1. .NET Reverse Engineering Tutorial Episode 2

1.1. Abstract

This tutorial is for informational purposes only. Once you complete the tutorial you are required to delete the application from your hard drive or other type of media. By reading on you consent to the requirements and the author is in no way responsible for your actions.

1.2. Targets

As software is constantly updated, you can find the exact version used in this tutorial at the flowing link:

WinXP Manager v4.98.4

<http://arteam.accessroot.com/tools/xpmanager.exe>

1.3. Reversing the Application

1.3.1 Preparation

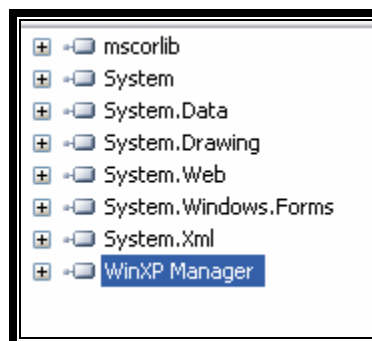
Obtain the following tools:

1. [Reflector](#) : Freeware
2. IDA : CommercialWare (google this)
3. FlexHEX : ShareWare (google this)

1.3.2 Checking out the target

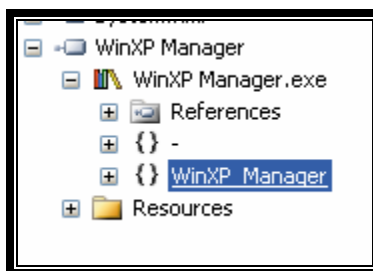
Today's target will be WinXP Manager. To start we get the victim program from link you are given earlier in this paper. We will load up the victim into Reflector and look for anything that might interest us. The good thing about .NET assemblies is that what you see is the source for the actual program. Like in C++ you have *{if,else}* statements, *for* loops, the same goes for .NET. If you closely inspect the code in Reflector we see this occur. Anyways, let us proceed with the paper on .NET assembly.

Upon first load in Reflector:

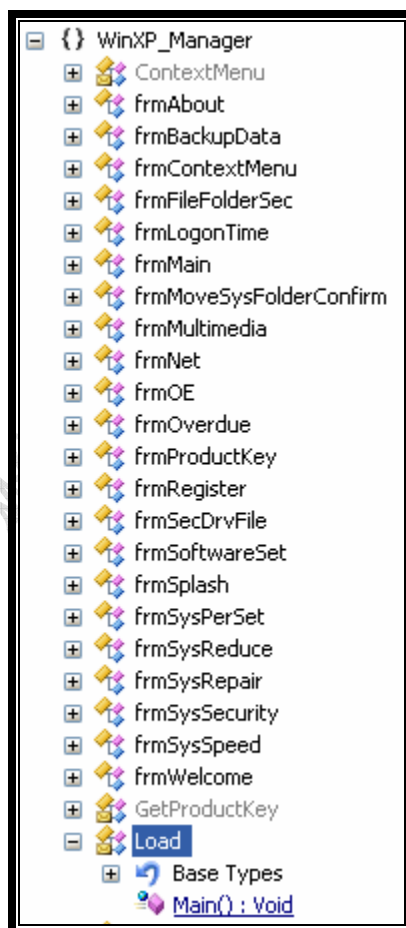


We see the Victim Executable.



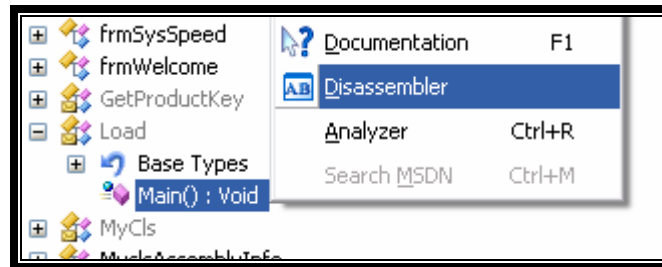


We expand the "+" signs

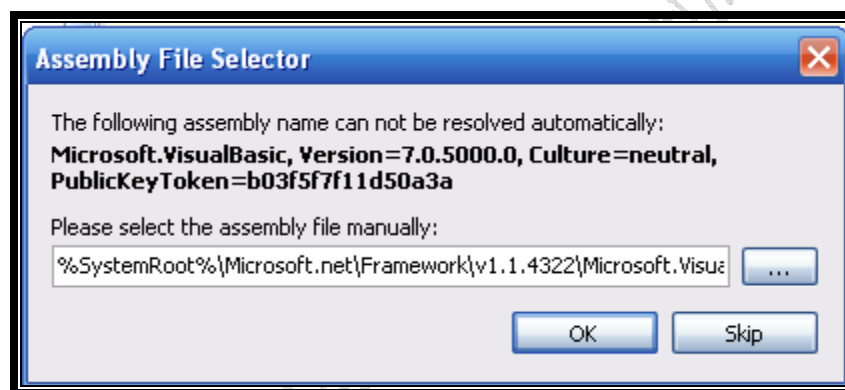


We expand it to see a possible method of attack.





Then we hit it with the disasm engine to see what it holds.



You should get this error, (and another one) just hit "Skip".



Disassembler

```
[STAThread]
public static void Main()
{
    string[] textArray1 = Environment.GetCommandLineArgs();
    foreach (string text1 in textArray1)
    {
        if (StringType.StrCmp(text1.Split(new char[] { '=' })[0].ToLower(), "/u", false) == 0)
        {
            string text2 = text1.Split(new char[] { '=' })[1];
            string text3 = Environment.GetFolderPath(Environment.SpecialFolder.System);
            ProcessStartInfo info1 = new ProcessStartInfo(text3 + @"%msiexec.exe", "/x " + text2);
            Process.Start(info1);
            Application.Exit();
            ProjectData.EndApp();
        }
    }
    Startup.StartupCondition();
    publicVarFun.OF = publicVarFun.GetImage("OPENFOLD.ICO");
    publicVarFun.MyXP = publicVarFun.GetImage("MyTradeLogo(ENG).png");
    publicVarFun.Line = publicVarFun.GetImage("Line.png");
    publicVarFun.MySideBarColor = StringType.FromObject(MyClsOther.MyClsRegConfig.ReadRegConfig("ColorScheme"));
    if (StringType.StrCmp(publicVarFun.MySideBarColor, "", false) == 0)
    {
        publicVarFun.MySideBarColor = "SystemColors";
    }
    publicVarFun.RemainDays = MyClsDetermineRegister.DetermineRegistered();
    if (StringType.StrCmp(publicVarFun.RemainDays, "Registered", false) == 0)
    {
        publicVarFun.IsRegistered = true;
        new frmSplash().Show();
    }
    else
    {
        new frmOverdue().Show();
    }
    Application.Run();
}
```

And our layout is this.

We look real hard at the disassembly to see what it tells us. At first glance we plainly see what we want to. A certain variable named "IsRegistered". Now I might not be the Albert Einstein of the reversing community, but even I know that this looks promising. So to disassemble it more:

IsRegistered

```
publicVarFun.RemainDays = MyClsDetermineRegister.DetermineRegistered();
if (StringType.StrCmp(publicVarFun.RemainDays, "Registered", false) == 0)
{
    publicVarFun.IsRegistered = true;
    new frmSplash().Show();
}
else
{
    new frmOverdue().Show();
}
```

So we are getting the infamous "are calls "Application.Run()". So now lets go to got from it. Open up IDA and load up the

we registered" check, then you can see it our favorite disassembler and see what we app.



D	ProductParameter	00049C80
D	WinXP_Manager.Load	00049CC0
F	Main	00049CD0 P
D	WinXP_Manager.MyCls	00049E30
D	MyClsBlnREG DWORD	00049E40

This is our Member name we found from Reflector.

We double click the line "WinXP_Manager.Load"

```

loc_49DDC:                                // CODE XREF: Main+100↑j
    nop
    call class System.String [PCL]PCL.MyClsDetermineRegister::DetermineRegistered()
    stsfld class System.String WinXP_Manager.publicVarFun::RemainDays
    ld sfld class System.String WinXP_Manager.publicVarFun::RemainDays
    ldstr "Registered"
    ldc.i4.0
    call int32 [Microsoft.VisualBasic]Microsoft.VisualBasic.CompilerServices.StringTy
    ldc.i4.0
    bne.un.s loc_49E11
    ldc.i4.1
    stsfld bool WinXP_Manager.publicVarFun::IsRegistered
    newobj void WinXP_Manager.frmSplash::.ctor()
    stloc.s 5
    ldloc.s 5
    callvirt void [System.Windows.Forms]System.Windows.Forms.Control::Show()

```

And we scroll down to our code area we found from reflector.

```

bne.un.s loc_49E11
ldc.i4.1
stsfld bool WinXP_Manager.publicVarFun::IsRegistered
newobj void WinXP_Manager.frmOverdue::.ctor()
stloc.s 6
ldloc.s 6
callvirt void [System.Windows.Forms]System.Windows.Forms.Control::Show()
nop
nop
loc_49E21:                                // CODE XREF: Main+13F↑j
    nop

```

The reason I have this certain area highlighted, with text bubble is found below.

In ASM we had JE, JNE. In .NET we have something similar to it called:

3.14 bne.un<length> – branch on not equal or unordered

Format	Assembly Format	Description
40 <int32>	bne.un target	Branch to target if unequal or unordered.
33 <int8>	bne.un.s target	Branch to target if unequal or unordered, short form.

So we have a JNZ here, or JNZ Short.



The opposite of JNZ(JNE) is JE. So let's refer again to our master manual for some document information on the opcode.

3.5 beq.<length> – branch on equal

Format	Assembly Format	Description
3B <int32>	beq target	Branch to target if equal.
2E <int8>	beq.s target	Branch to target if equal, short form.

So now we have the opcode for JE.

So in beginner terms, we are making a simple JNZ -> JE. By switching the (bne.un.s, or JNZ Short), to (beq.s, or JE Short). So we are attacking this application in the simplest of terms, by doing the ever famous Jump switch. In ASM I would not be caught dead doing this, but in .NET anything goes.

So if we refer back to IDA, we can grab our Binary code to do the "search" for the Hexadecimal characters in the executable. Now, assuming you have read my first tutorial on .NET, I will assume you already are aware of the members. We cannot just patch anywhere we like. Sometimes our changes will affect the program in other places. Lets analyze the function "IsRegistered". To do this lets get back to the place we started from in Reflector, then click on the "IsRegistered" member name.

```

{
    publicVarFun.IsRegistered = true;
    new frmSplash().Show();
}
else
{
    new frmOverdue().Show();
}

```

Click the "IsRegistered" member name.

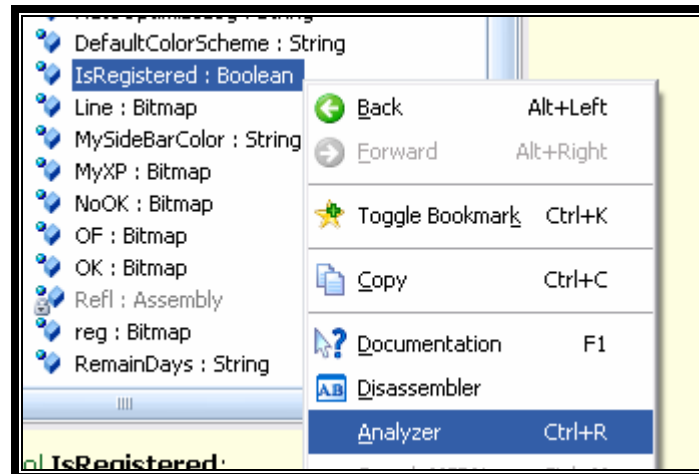
```

DefaultColorScheme : String
IsRegistered : Boolean
Line : Bitmap
MySideBarColor : String

```

We see the member name.





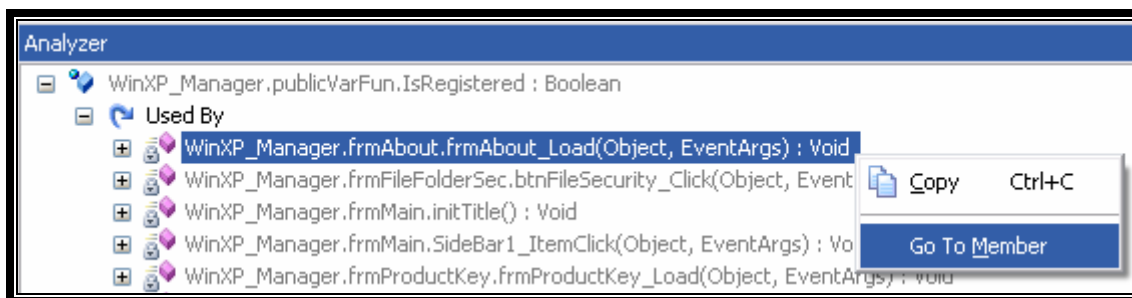
Then we right click the name, and analyze.



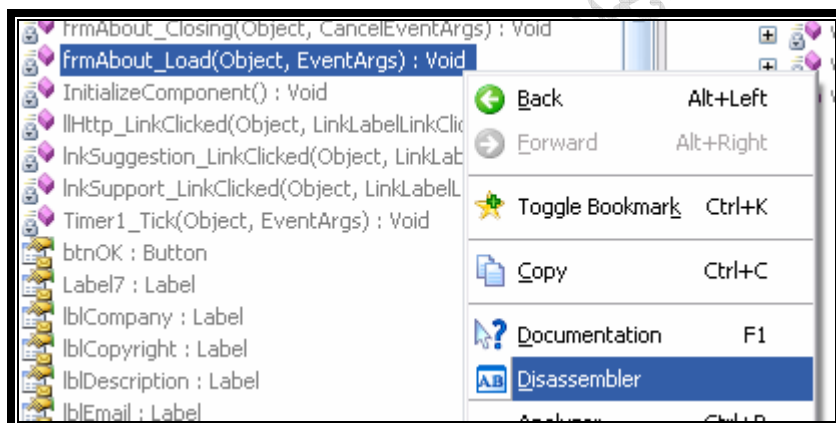
Now we see what other functions use the same member name.

So we can conclude that the "IsRegistered" name is used more often than once. Since our original place we discussed before will set the flag for "IsRegistered" we are good here. So what we must do is go to each function and see what they tell us.

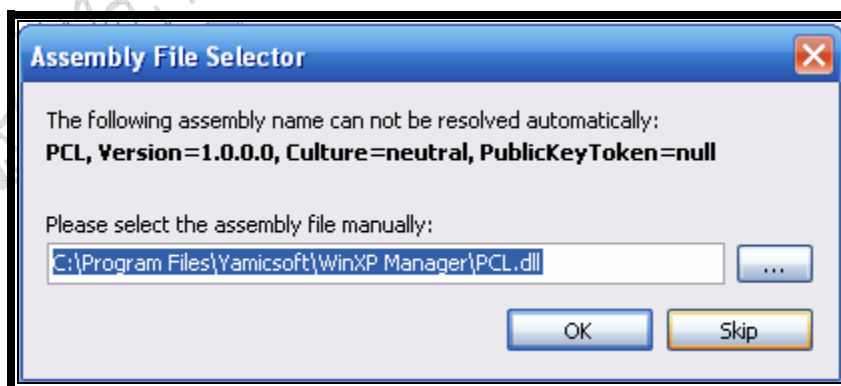




We click the first member name, and then we go to the member like in the picture.

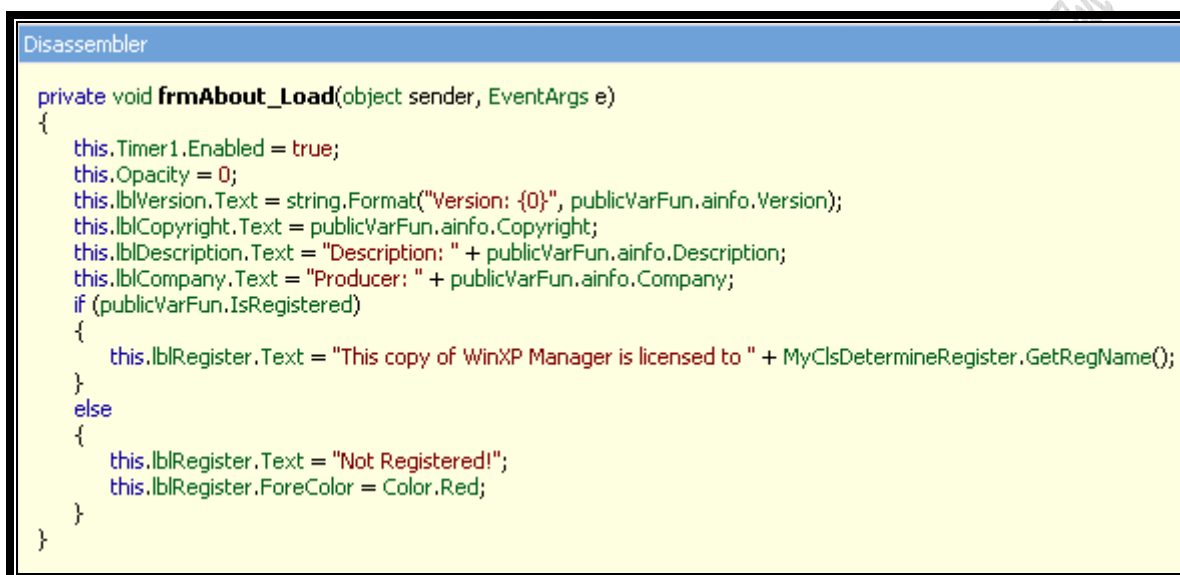


We will follow the "About_Load" into disassembly.



Again skip the error warnings.





```

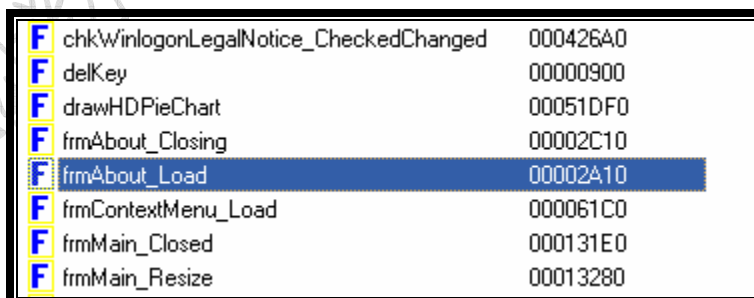
Disassembler

private void frmAbout_Load(object sender, EventArgs e)
{
    this.Timer1.Enabled = true;
    this.Opacity = 0;
    this.lblVersion.Text = string.Format("Version: {0}", publicVarFun.ainfo.Version);
    this.lblCopyright.Text = publicVarFun.ainfo.Copyright;
    this.lblDescription.Text = "Description: " + publicVarFun.ainfo.Description;
    this.lblCompany.Text = "Producer: " + publicVarFun.ainfo.Company;
    if (publicVarFun.IsRegistered)
    {
        this.lblRegister.Text = "This copy of WinXP Manager is licensed to " + MyClsDetermineRegister.GetRegName();
    }
    else
    {
        this.lblRegister.Text = "Not Registered!";
        this.lblRegister.ForeColor = Color.Red;
    }
}

```

We see the "registered" and "not registered" dialogs.

Let's switch to IDA view again and see what we got as far as code for this function.



F	chkWinlogonLegalNotice_CheckedChanged	000426A0
F	delKey	00000900
F	drawHDPieChart	00051DF0
F	frmAbout_Closing	00002C10
F	frmAbout_Load	00002A10
F	frmContextMenu_Load	000061C0
F	frmMain_Closed	000131E0
F	frmMain_Resize	00013280

Here is the function name, double click and see the code for it.



```

*      ldsfld bool WinXP_Manager.publicVarFun::IsRegistered
*      brfalse.s loc_2AC8
*      ldarg.0
*      callvirt class [System.Windows.Forms]System.Windows.Forms.Label WinXP_Manager.frmAbout::get_lb1
*      ldstr "This copy of WinXP Manager is licensed to "
*      call class System.String [PCL]PCL.MyClsDetermineRegister::GetRegName()
*      call class System.String [mscorlib]System.String::Concat(class System.String, class System.String)
*      callvirt void [System.Windows.Forms]System.Windows.Forms.Control::set_Text(class System.String)
*      nop
*      br.s loc_2AEB
*
loc_2AC8:                                // CODE XREF: frmAbout_Load+99↑j
*      nop
*      ldarg.0
*      callvirt class [System.Windows.Forms]System.Windows.Forms.Label WinXP_Manager.frmAbout::get_lb1
*      ldstr "Not Registered!"
*      callvirt void [System.Windows.Forms]System.Windows.Forms.Control::set_Text(class System.String)
*      nop

```

NOW WE LIKE WE DID BEFORE.

3.17 brfalse.<length> – branch on false, null, or zero

Format	Assembly Format	Description
39 <int32>	brfalse target	Branch to target if value is zero (false).
2C <int8>	brfalse.s target	Branch to target if value is zero (false), short form.
39 <int32>	brnull target	Branch to target if value is null (alias for brfalse).
2C <int8>	brnull.s target	Branch to target if value is null (alias for brfalse.s), short form.
39 <int32>	brzero target	Branch to target if value is zero (alias for brfalse).
2C <int8>	brzero.s target	Branch to target if value is zero (alias for brfalse.s), short form.

So "brfalse.s" is Branch to target if zero (short)

3.18 brtrue.<length> – branch on non-false or non-null

Format	Assembly Format	Description
3A <int32>	brtrue target	Branch to target if value is non-zero (true).
2D <int8>	brtrue.s target	Branch to target if value is non-zero (true), short form.
3A <int32>	brinst target	Branch to target if value is a non-null object reference (alias for brtrue).
2D <int8>	brinst.s target	Branch to target if value is a non-null object reference, short form (alias for brtrue.s).

So the opposite is "brtrue.s"

If we were to click on the other member names, we will see the same idea as before. A "true/false" switch. So we had eight other member look closely at the initial disassembly, we see "IsRegistered" flag is set for us in the



names popup in our member search. If we that we do not need to reverse this. The beginning when we load.

```

    publicVarFun.RemainDays = MyClsDetermineRegister.DetermineRegistered();
    if (StringType.StrCmp(publicVarFun.RemainDays, "Registered", false) == 0)
    {
        publicVarFun.IsRegistered = true;
        new frmSplash().Show();
    }
    else
    {
        new frmOverdue().Show();
    }
    Application.Run();
}

```

See we set "IsRegistered" to True.

Disassembler

```

.method private instance void btnFileSecurity_Click(object sender, [mscorlib]System.EventArgs e) cil managed
{
    .maxstack 4
    L_0000: nop
    L_0001: ld sfld bool WinXP_Manager.publicVarFun::IsRegistered
    L_0006: brtrue.s L_001b
    L_0008: ldstr "This software has not been registered, please decrypt the files before the trial expires!"
    L_000d: ldstr "Warning"
    L_0012: ldc.i4.0
    L_0013: ldc.i4.s 48
    L_0015: call [System.Windows.Forms]System.Windows.Forms.DialogResult [System.Windows.Forms]System.Wir
    L_001a: pop
    L_001b: nop
    L_001c: nop
    L_001d: ldstr "FileSecurity.exe"
    L_0022: call [System]System.Diagnostics.Process [System]System.Diagnostics.Process::Start(string)
    L_0027: pop
    L_0028: leave.s L_0037
    L_002a: call void [Microsoft.VisualBasic]Microsoft.VisualBasic.CompilerServices.ProjectData::SetProjectError([mscorlib]System.Exception)
    L_002f: nop
    L_0030: call void [Microsoft.VisualBasic]Microsoft.VisualBasic.CompilerServices.ProjectData::ClearProjectError()
    L_0035: leave.s L_0037
    L_0037: nop
    L_0038: nop
    L_0039: ret
    .try L_001d to L_002a catch [mscorlib]System.Exception handler L_002a to L_0037
}

```

We see the same switch. (which is set true from earlier)

Now let's concentrate on making our name appear in the about box. So we go to the "frm.about.load" member, and disassemble it to see.

```

FocusInternal() : Boolean { System.Windows.Forms.Form }
frmAbout_Closing(Object, CancelEventArgs) : Void
frmAbout_Load(Object, EventArgs) : Void
FromChildHandle(IntPtr) : Control { System.Windows.Forms

```

Then disasm it to see.



```

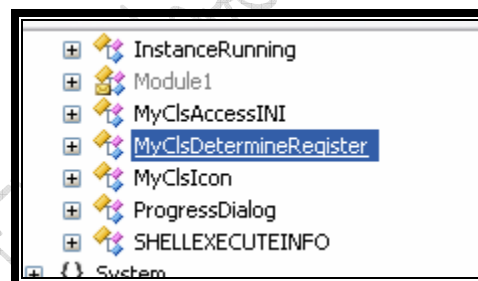
Disassembler

private void frmAbout_Load(object sender, EventArgs e)
{
    this.Timer1.Enabled = true;
    this.Opacity = 0;
    this.lblVersion.Text = string.Format("Version: {0}", publicVarFun.ainfo.Version);
    this.lblCopyright.Text = publicVarFun.ainfo.Copyright;
    this.lblDescription.Text = "Description: " + publicVarFun.ainfo.Description;
    this.lblCompany.Text = "Producer: " + publicVarFun.ainfo.Company;
    if (publicVarFun.IsRegistered)
    {
        this.lblRegister.Text = "This copy of WinXP Manager is licensed to " + MyClsDetermineRegister.GetRegName();
    }
    else
    {
        this.lblRegister.Text = "Not Registered!";
        this.lblRegister.ForeColor = Color.Red;
    }
}

```

And we see if it says registered to: then tries to read a different member.

The member it tries to read from is in the "PCL.dll" file. So let's click the "MyClsDetermineRegister" place.



Then expand it.

```

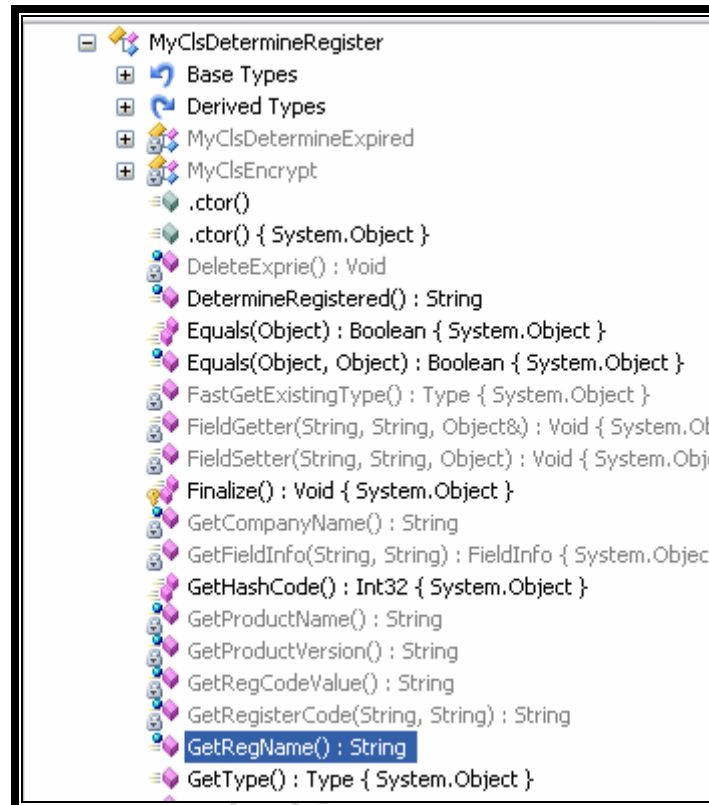
Disassembler

public class MyClsDetermineRegister
{
    public MyClsDetermineRegister();
    private static void DeleteExprie();
    public static string DetermineRegistered();
    private static string GetCompanyName();
    private static string GetProductName();
    private static string GetProductVersion();
    private static string GetRegCodeValue();
    private static string GetRegisterCode(string strCodeWord, string strValue);
    public static string GetRegName();
    private static bool Registered();
    public static bool WriteRegisterCode(string CodeValue, string UserName);
}

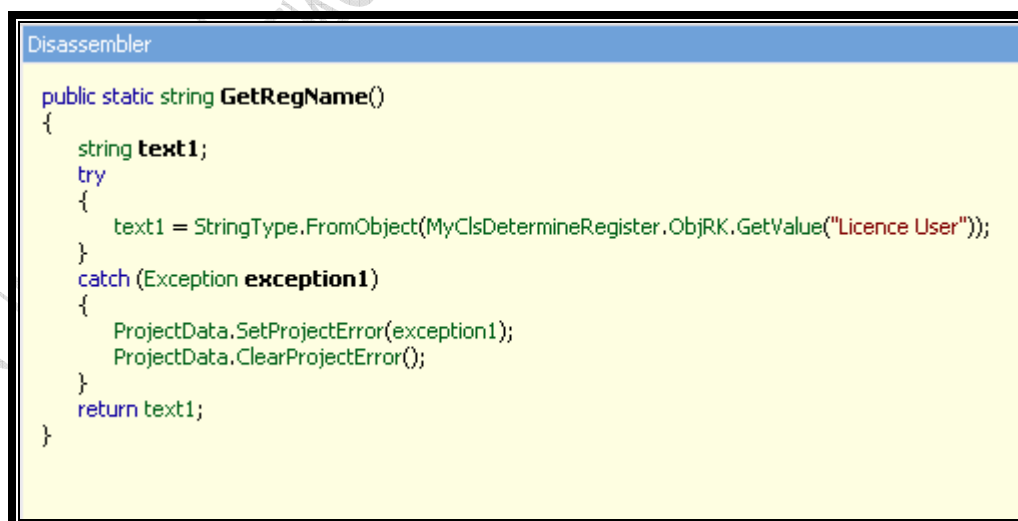
```

We see the "GetRegName()"





Expand the function and see the member name.



And we see the Value it tried to read from. "Licence User" (yes its spelled wrong)



```
to " + MyClsDetermineRegister.GetRegName());
string PCL.MyClsDetermineRegister.GetRegName();
```

So let's look in the registry and see what we got.



My Computer\HKEY_CURRENT_USER\Software\Yamicsoft\WinXP Manager

We look under the following directory.

Name	Type	Data
(Default)	REG_SZ	(value not set)
CreateSR	REG_SZ	True
Licence User	REG_SZ	ARTeam

Now run the app...





Wasn't that bad, was it? ☺

1.4. References

"Primer on .NET", Shub-Niggurath, Googleplex, zyzygy, <http://tutorials.accessroot.com>
".NET Reverse Engineering Tutorial Episode 1" MaDMAN_H3RCUL3s, <http://tutorials.accessroot.com>

1.5. Conclusions

We have reached the end, and my hope is that you, the reader, learned a new technique. Instead of the old, Disassemble, re-assemble technique, we simply patched the target and saved ourselves more headaches. This technique will only work on "non-obfuscated" executables. If this program was obfuscated, then the simple patching would not be possible. We would have to rely on the decrypt, disassemble, patch, and re-assemble method.

1.6. Greetings

ARTeam, fly, shoo00, heXer, unpack.cn, PEdiy forum, SECTiON-8, Like maybe one or two 0day groups, Anyone who has done any sort of chemical brain enhancement (☺), Anyone who makes their own chemical brain enhancers, especially the old HiVE dwellers (you know who you are), and of course.... YOU!

Document History

- Version 1.0 first public release





BEE HAPPY!

17-March-2014-10:45:00 AM

