Reverse Engineering “Microsoft F#”.

Author:Aodrulez.

F# is another programming language added to the already crowded .NET Framework.F# is promising though! It is said to encompass functional programming as well as imperative object-oriented programming disciplines.

So far so good.... but the question that I have is... Why mix it with .NET? Yeah...maybe .NET apps are easy to code ..maybe they are GUI-wise amazing.... Portable, since its again similar to Java...all you need is .NET framework to run them... & so on n on... but dear M$... portability comes with a price!

There can be tonnes that I'd like to mention..but lemme concentrate on only one such price that one has to pay if they are tryin to achieve portability the Java way! And what am talking about is "Reverse Engineering"..or rather.. I should put it as "Ease in Reverse Engineering".First, lets analyze the "Portability Technique" thats being used here.

Portability Technique:

The basic idea here is to have three major components..

1] Programming Language
2] Intermediate Form of Code.
3] Framework.

Now how this works is simple.The Programming Language is designed in such a way that when you compile it.. Machine Code is not generated unlike rest of the programming Languages out there... instead... its converted to an Intermediate Form (ByteCode in case of Java & IL in case of .NET).Now a computer can understand what to do only if its in Machine Language... so its understood that this Intermediate Form is completely "Crap" to the computer.This is where the FrameWork comes into play... Each time you run such an application, something called as J-I-T... Just in Time Compiler..is called in.This is a part of the Framework & the sole purpose of this app is to compile the Intermediate Form into Machine specific code & then execute it.Benefit of this technique?.. u guessed it!... Portability.

Okies, that was a pathetic way to describe the whole thing but the sum-n-substance of it is correct.How is Protability achieved? See... each time you compile an app in this Programming Language... you always endup with Intermediate Form.This is common for all platforms... Now its the job of the Framework to make it work on a particular platform.So the only thing that has to be done is to code a Platform-Specific Framework...n thats it! All applications you code using this Programming Language can run on all Platforms....for which the Framework has been developed.

Downside?

Each application that you code & compile is in the Intermediate-Form.And what you distribute as an app is actually this Intermediate Form of your code.The problem with this is...

Decompilation!

Usually the applications coded in C/C++,Delphi etc contain Machine Language Code.So we run a Disassembler on these & ultimately end-up with the code of this applicaiton in Assembly Language Form. Since goin through lots n lots of assembler code is really a Head-Ache.. ripping out parts of your code & then converting them into C/C++ equivalent code is a really tedious thing to be doing.

Now the problem with our Portable Programming Language is that...the Intermediate Language has got its own OpCode & since this is not Machine specific..u cannot Disassemble this code. What I mean by this is.. if you put it through a Disassembler, most of the Assembly Listing that you'll get will be bogus... But thats a Blessing in Disguise! since this Intermediate Form has got its own OpCodes, if we have detailed info about the structure of this Intermediate Form, we can code Decompilers for it!

Decompiler?

A decompiler is a Tool that can go through a Programming Languages' Intermediate form & produce the actual "Source-Code". Yeah...u read it right...Source-Code! Most of the times, even Variable & Function names are preserved!Its just like a Disassembler..but also different in a lot of ways.For starters... its "Programming Language" Specific.That means you can't use a Java Decompiler for an app coded in .NET .

So,if a Programming Language uses this "Portability Technique", technically, a Decompiler can be written no matter how cryptic the Intermediate Form might be.That sounds grave does'nt it? So whats all this got to do with F#?? Everything!.....

As i said in the Beginning itself, F# has been made .NET Compliant.. that means...once compiled.. it'll be in the IL form. And there are tools already out there that can Decompile .NET Apps. One of my Favourites is ".NET Reflector".Its free, powerful, & has got plugins too!

Lets Reverse an F# App....

Since this is just a PoC Paper... lets code an App in F# & try to break (crack!) it.Quick search over internet shows that Visual Studio (.NET one) is needed to code apps in F# easily. A little more tinkering around showed me that all you really need to code an F# app is its compiler.you really don't need to install the overbloated Visual Studio to make our small PoC application. Just Download the compiler, install it & you are ready to have some fun.

Here is the Code for our PoC App:
(Save as Aodrulez.fs)

```
#light

open System
open System.Windows.Forms

let form = new Form()
form.Width  <- 170
form.Height <- 130
form.Visible <- true
form.Text <- "Aodrulez"

// Menu bar, menus
let mMain = form.Menu <- new MainMenu()
let mFile = form.Menu.MenuItems.Add("&File")

let mabout = form.Menu.MenuItems.Add("&About")
let miQuit  = new MenuItem("&Quit")
mFile.MenuItems.Add(miQuit)

let btn1 = new Button()
do btn1.Text <- "Register"
do btn1.Location <- new System.Drawing.Point(42,40)
do form.Controls.Add(btn1)

// TextBox
let textB = new TextBox()
//textB.Dock <- DockStyle.Fill
textB.Text <- "  Enter Code Here."
do textB.Location <- new System.Drawing.Point(30,10)
form.Controls.Add(textB)

// callbacks
mabout.Click.Add(fun _ ->
System.Windows.Forms.MessageBox.Show("Aodrulez's F# Crackme V1.0\nHappy
Cracking!","Aodrulez");())
miQuit.Click.Add(fun _ -> form.Close())
btn1.Click.Add(fun _ -> (if textB.Text="Awesome" then
System.Windows.Forms.MessageBox.Show("Correct!\n :)","Aodrulez");()
else System.Windows.Forms.MessageBox.Show("Wrong :( . Try
again!","Aodrulez");()))

#if COMPILED
// Run the main code. The attribute marks the startup application
thread as "Single
// Thread Apartment" mode, which is necessary for GUI applications.
[<STAThread>]
do Application.Run(form)
#endif
```

To compile it... save this as Batch file & run it:

```
-------------------------------------------------------------------------
@setlocal
@REM 1. Configure the sample, i.e. where to find the F# compiler and
TLBIMP tool.

@if "%FSHARP_HOME%"=="" ( set FSHARP_HOME=..\..\..)
@set FSC=%FSHARP_HOME%\bin\fsc.exe

@REM 2. Build the sample


%FSC% --target-winexe -g Aodrulez.fs
@if ERRORLEVEL 1 goto Exit


:Exit

@endlocal
@exit /b %ERRORLEVEL%
-------------------------------------------------------------------------
```

Reversing F#:

Okies..now that we have the Test Application ready.. lets see how it works!

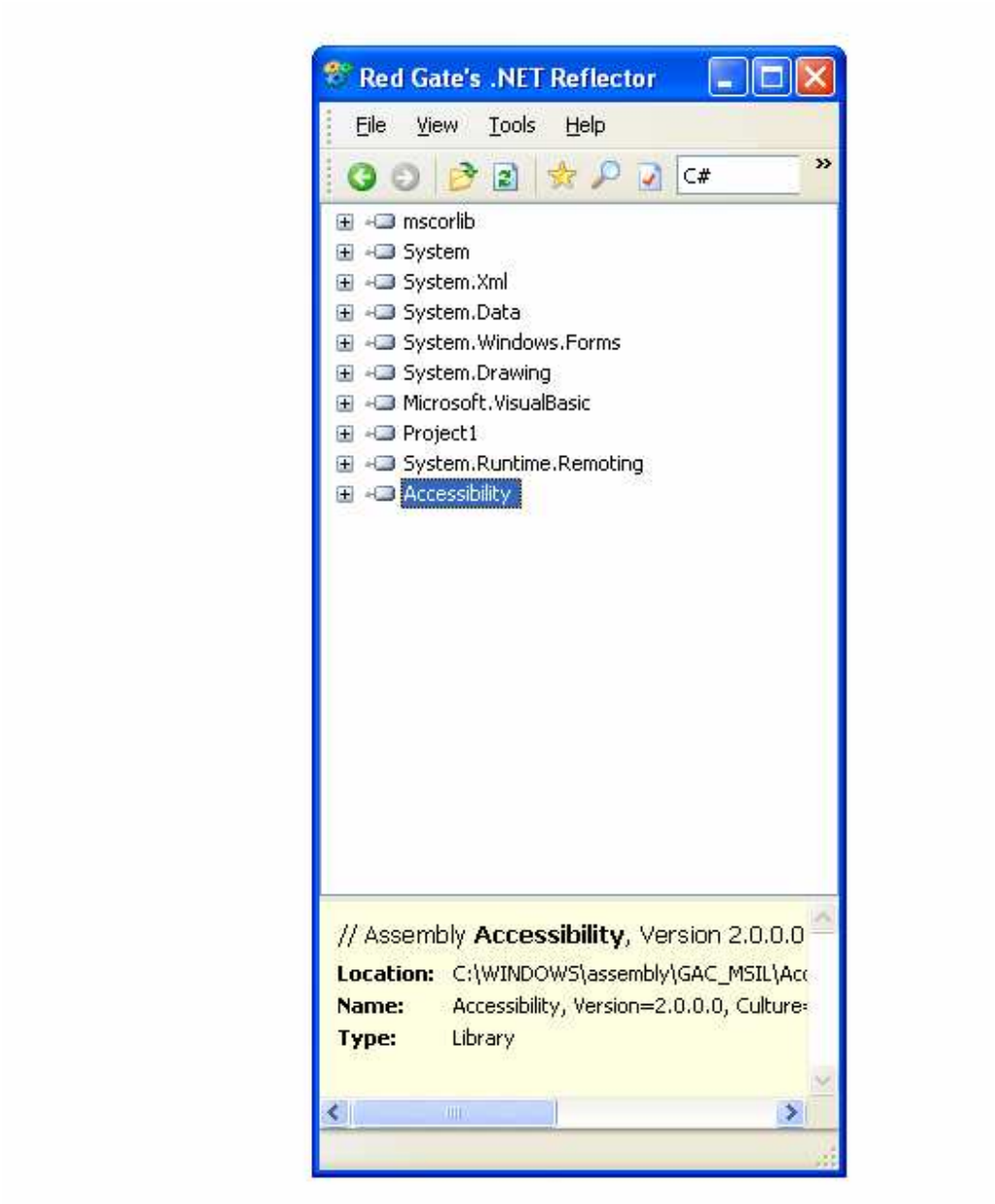
As you can see above...we've designed a GUI based application that needs some Code to be entered.It sure is'nt the one currently entered :) .If you have a look at the Applications' F# source-code above... you'll see that the actual code that the App is looking for is "Awesome".So lets try that one....

Yeah!..that was the Code our small little F# app was looking for. Now this was no big deal! Anyone can reverse an App if you have its Source-Code.So lets "Reverse Engineer" it the actual way....

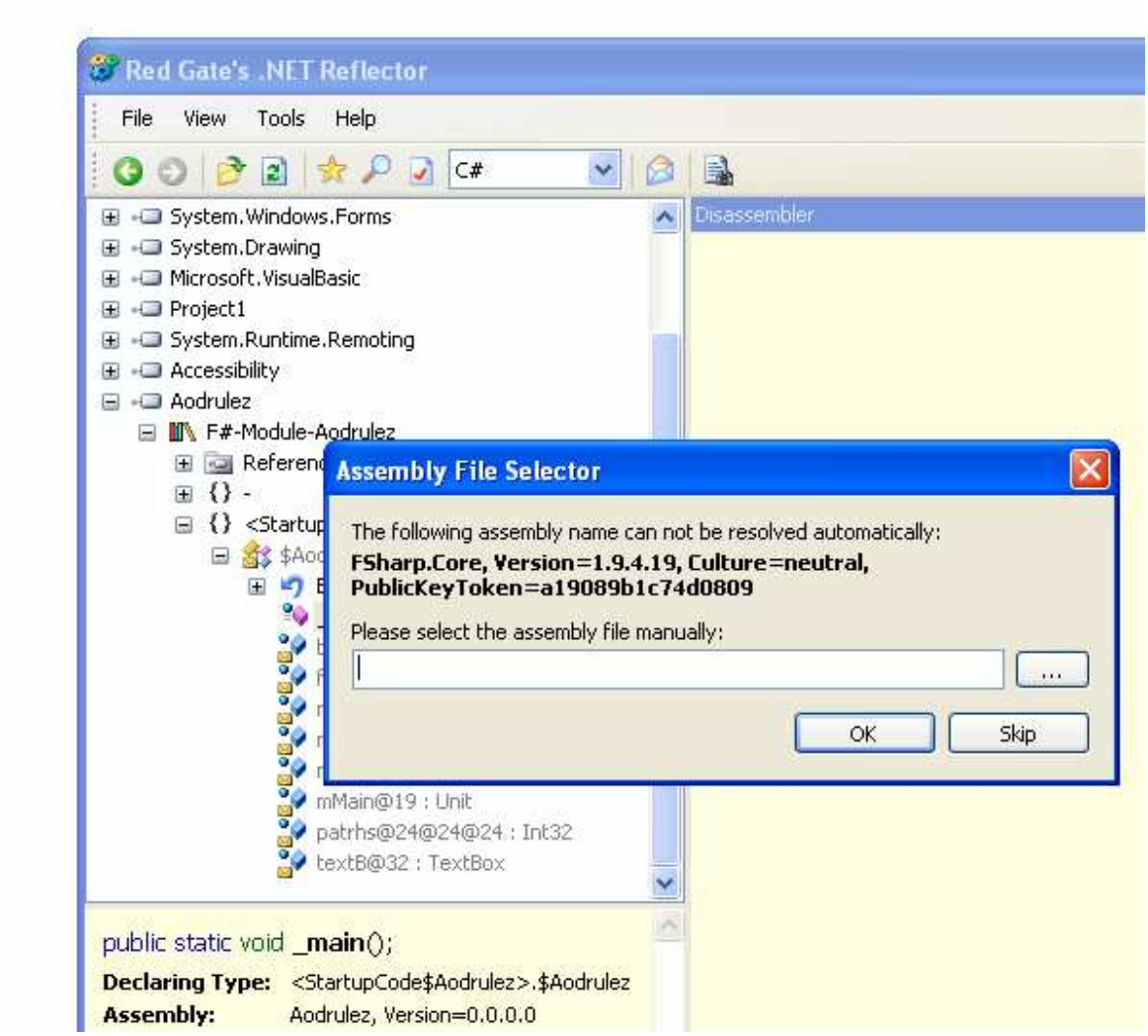
Time For Some Reverse Engineering……

.NET Reflector:

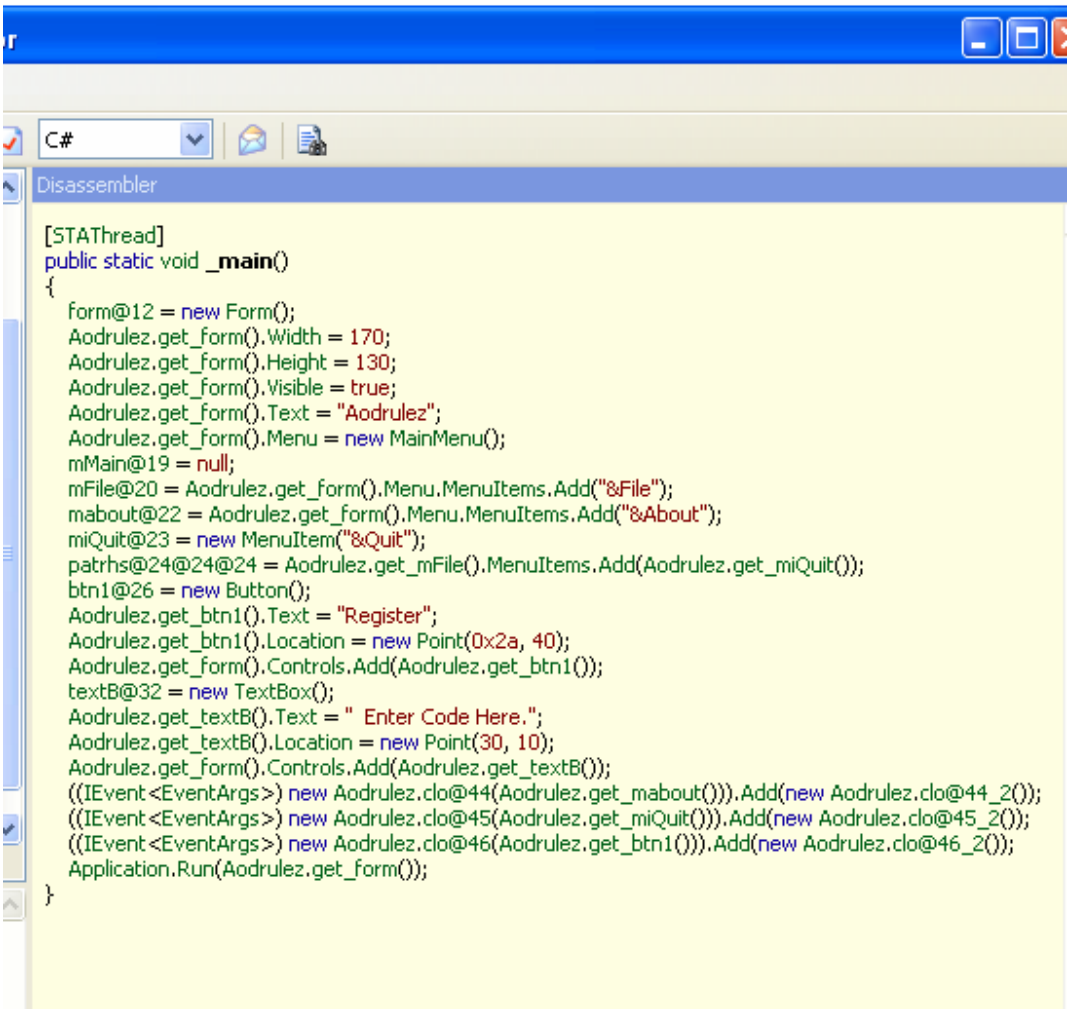
Am using .NET Reflector since I know that F# is already .NET Compatible.So heres how .NET Reflector looks like:



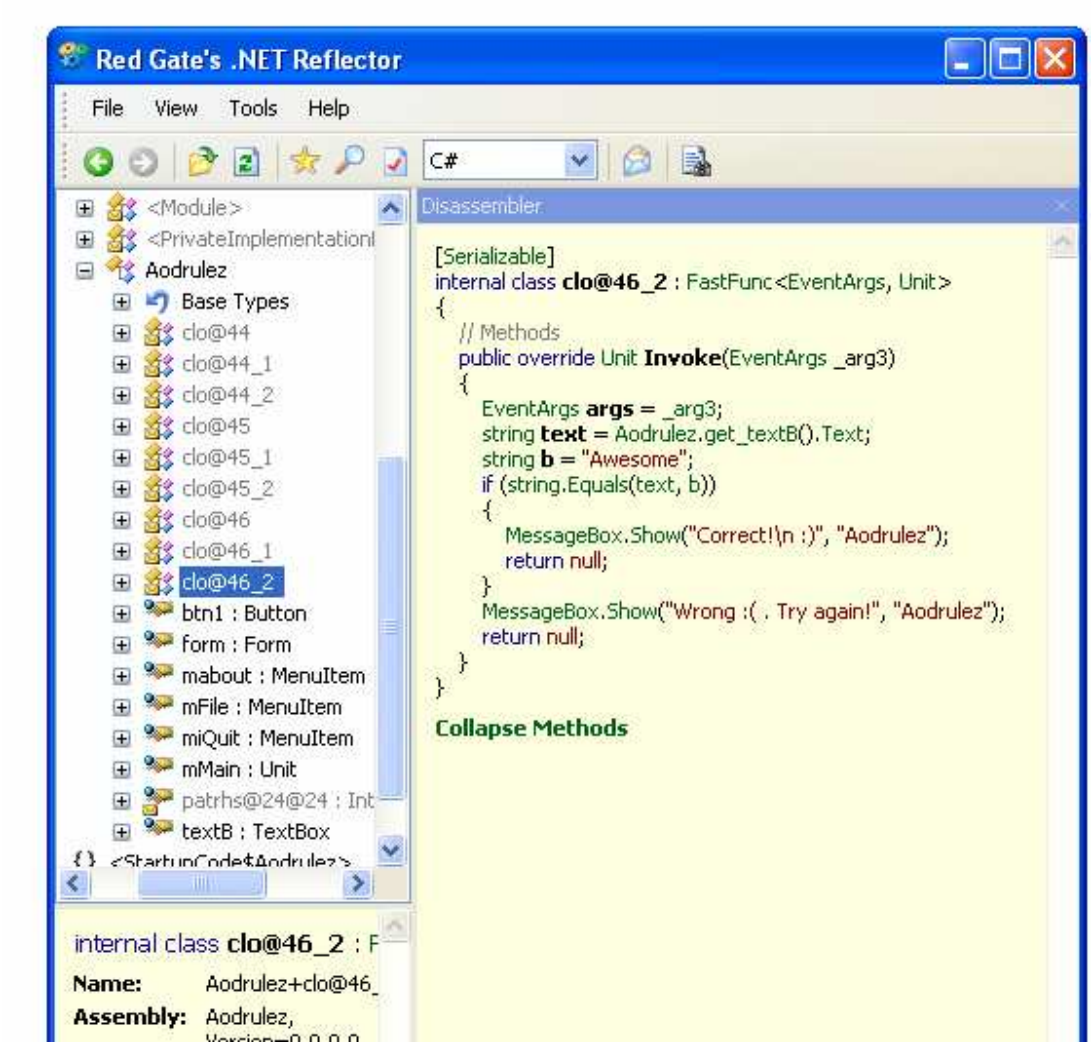
Now am opening my "Aodrulez.exe" which is our Compiled F# App in Reflector.Heres how it looks like then:



Oops! thats an Error saying it can't find some file thats required by F#. So just manually Browse & Select "FSharp.Core.dll" & Reflector is all happy! So now we are all set to reverse F# using .NET Reflector.



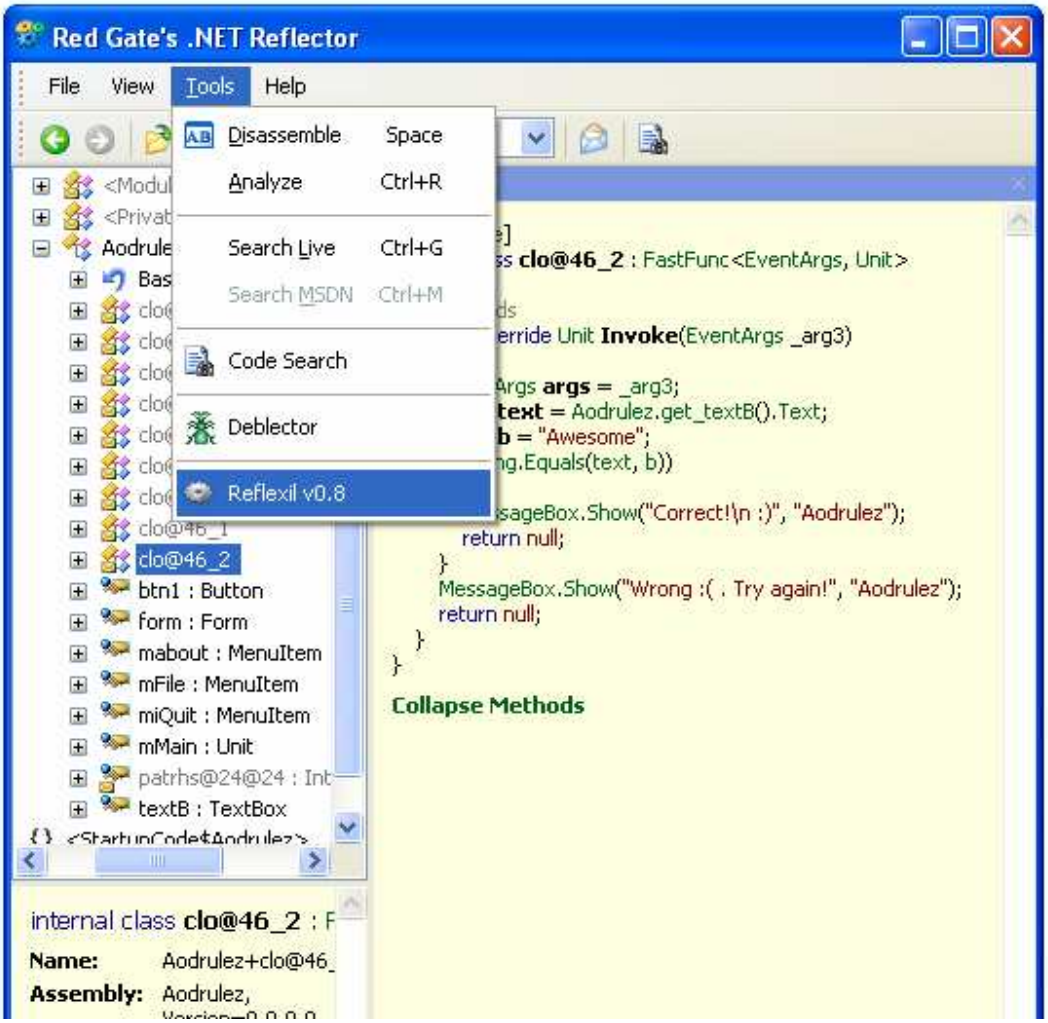
The above picture shows the Decompiled Listing of our App in C# code... ".NET Reflector" simply works fine with F# too :)
Lets look for some more interesting code in our app's Decompilation!



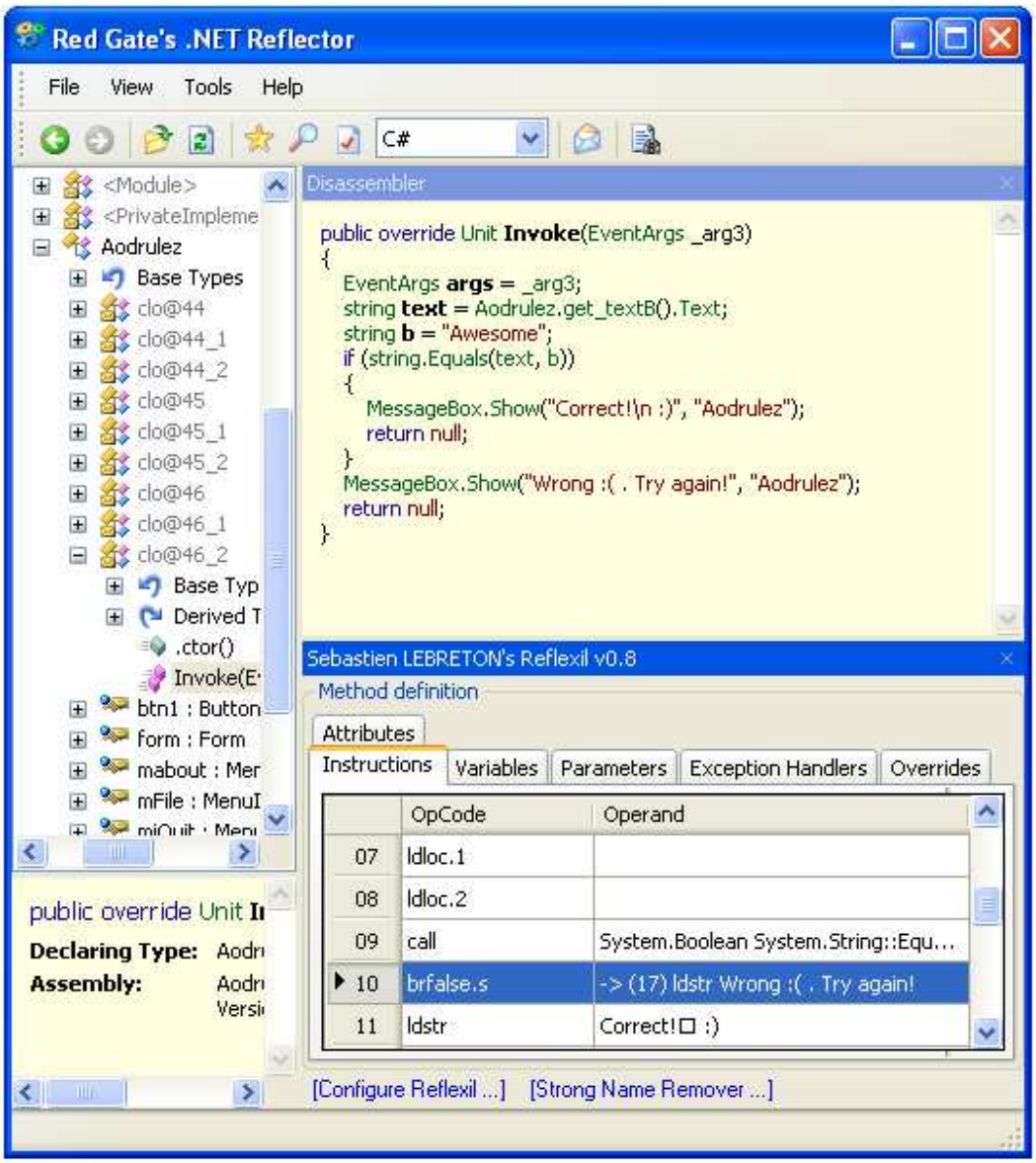
Are you seeing what am seeing too???? Thats my Code in all its Glory! Its intact.. & I believe, even a 2yr old can understand that!

Patching F# App???

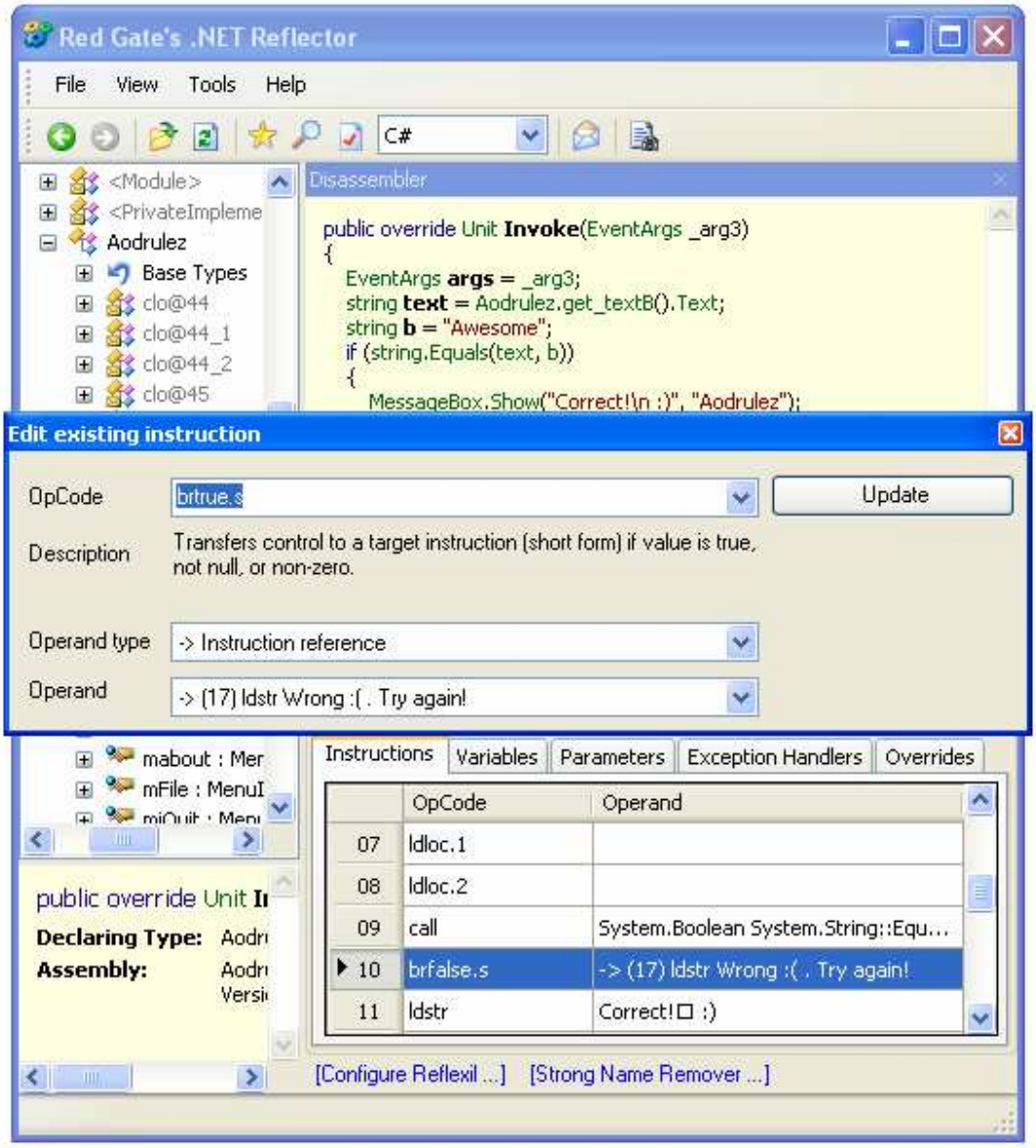
Yeah..u read it right too... Lets try to patch this small app of ours to accept any String as Valid code.. :). For that we'll use one of Reflector's Plugins called as "Reflexil".

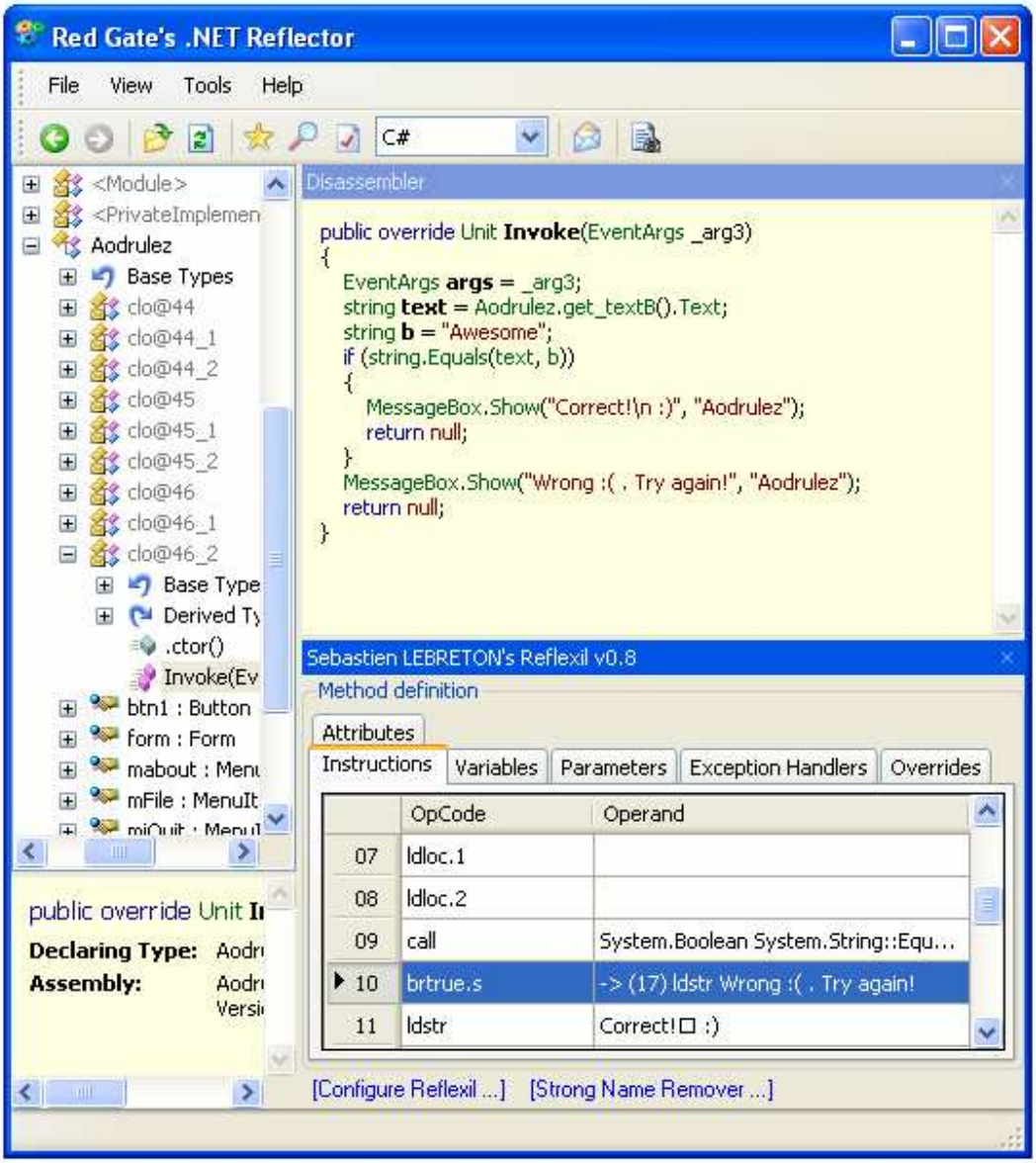


Reflexil Shows IL Disassembly as shown below which we can Modify as we wish.

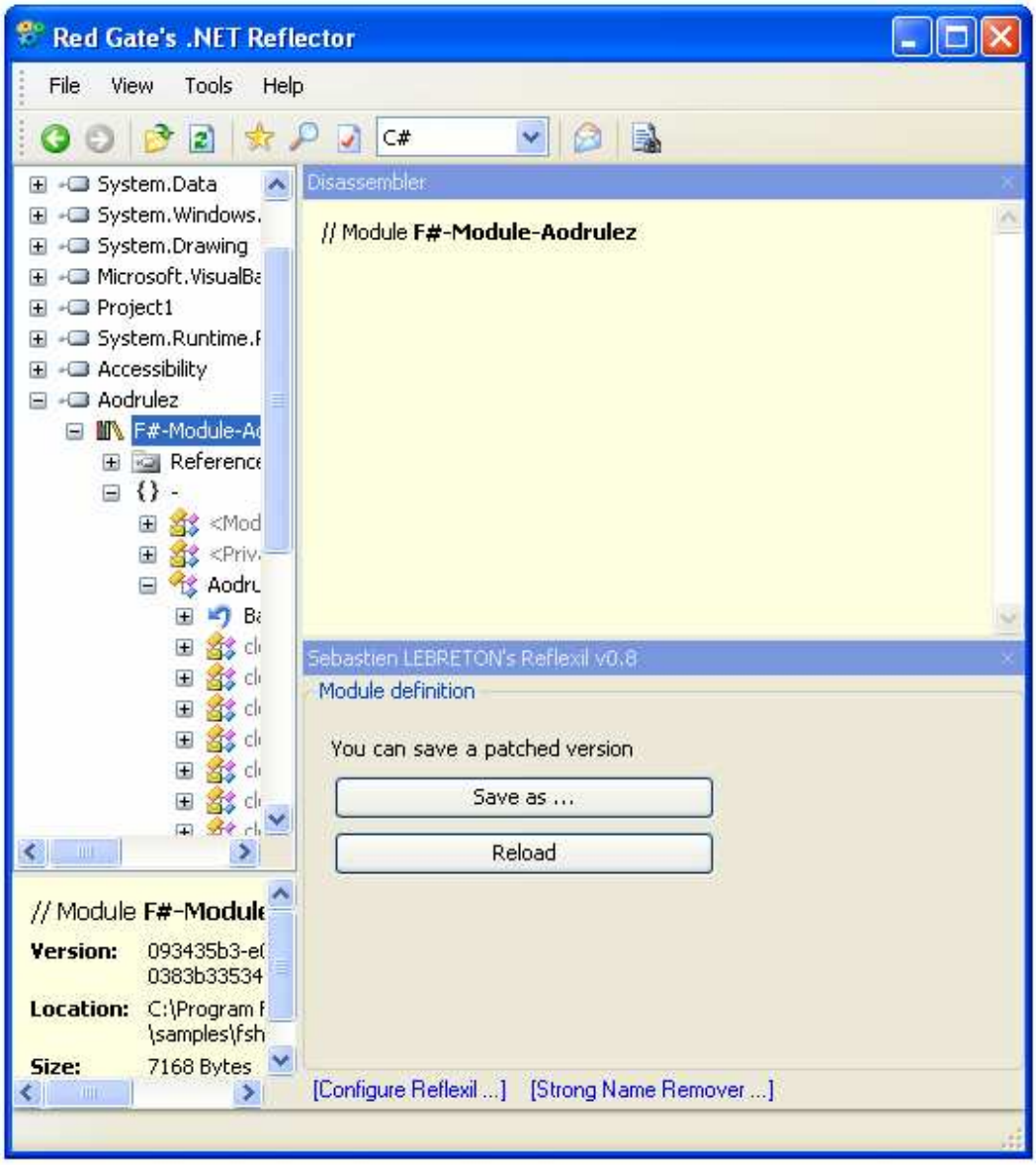


Lets modify that "brfalse.s" to "brtrue.s". What will that do?
As you can see in the above pic, thats the conditional jump. So what we are tryin to do is to make it jump to the "Correct! :)" MessageBox no matter what Text we enter.(Note that here, the actual code..ie Awesome will give me the "Wrong :(" MessageBox!)
So lets patch it!

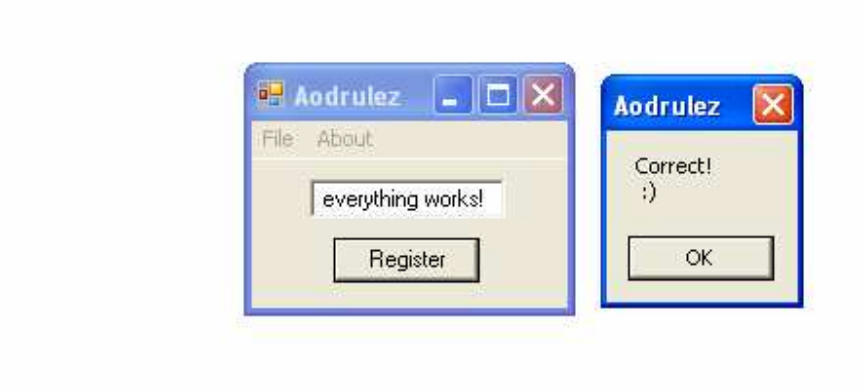

Red Gate's .NET Reflector
File View Tools Help
C#
<Module>
<PrivateImpleme
Aodrulez
Base Types
clo@44
clo@44_1
clo@44_2
clo@45
Disassembler
public override Unit Invoke(EventArgs _arg3)
{
EventArgs args = _arg3;
string text = Aodrulez.get_textB().Text;
string b = "Awesome";
if (string.Equals(text, b))
{
MessageBox.Show("Correct!\n :)", "Aodrulez");
Edit existing instruction
OpCode
brtrue.s
Update
Description
Transfers control to a target instruction (short form) if value is true, not null, or non-zero.
Operand type
-> Instruction reference
Operand
-> (17) ldstr Wrong :(. Try again!
mabout : Mer
mFile : MenuI
public override Unit I
Declaring Type: Aodr
Assembly: Aodr
Versi
Instructions
Variables
Parameters
Exception Handlers
Overrides
OpCode
Operand
07 ldloc.1
08 ldloc.2
09 call System.Boolean System.String::Equ...
10 brfalse.s -> (17) ldstr Wrong :(. Try again!
11 ldstr Correct!□ :)
[Configure Reflexil ...]
[Strong Name Remover ...]



And now lets save this Executable!



I've saved my patched exe as "Aodrulez_patched.exe". Now lets try if our patch works or not :)



It sure does! Thats how simple it is to Crack an F# Application :)

Moral of the Story:

As of now, F# is in its infancy I believe.But if its .NET Framework Compatibility is continued, I don't think it'll last for long.Why? Think of it this way....

Suppose you own some Software Company thats developing a new Algorithm... something that you just don't want to disclose. As long as you code your apps using this Algo in traditional programming languages...its very tedious to rip your algorithm & to reverse them.But lets say...you chose one of those .NET Languages.Lets say F#.... when you compile your Application & Sell/Distribute it as a Product, its as good as saying you are distributing Pamphlets of your Secret Algoritm's Source-code!

As a Software company thats the last thing you want to happen to you..is'nt it?

So...Microsoft...Please! I think F# has a long way to go.Its a really beautiful Idea in itself.Don't mix this Programming Language with your .NET Framework.

Disclaimer:

This paper was solely put together for Informational Purpose & to pointout the weaker aspects of .NET Framework & the recently introduced F# language.The author shall in no way be responsible for any damage caused by misuse of the information provided here.

Greetz Fly out to:

Amforked() :My Mentor.
www.OrchidSeven.com :For givin me this Opportunity.
The Blue Genius : :) .