Reverse engineering an obfuscated .Net application

Travis Altman

RVAsec

June 2012

Huge thanks

Curtis Mechling

http://twitter.com/#!/curtismechling

Outline

- This talk is for all audiences, no experience and seasoned .Net developers
- I'll be covering basic technology behind a .Net application
- I'll discuss simple and more complex ways to reverse an obfuscated .Net application
- Demo reversing an obfuscated .Net app

Steps to reverse .Net app

1. Run the application to understand functionality
2. Decompile the application
3. Review source code and hone in on the functionality you're trying to understand
4. If obfuscated look for key constructs to understand functionality
5. Optional: Modify app to achieve your desired functionality

Topic not new

- Many others before me have discussed the insecurities of .Net applications
- Mark Pearl
  - http://1dl.us/va3
- Jon McCoy
  - http://digitalbodyguard.com/
- Cory Foy
  - http://1dl.us/va4

Reversing and obfuscation

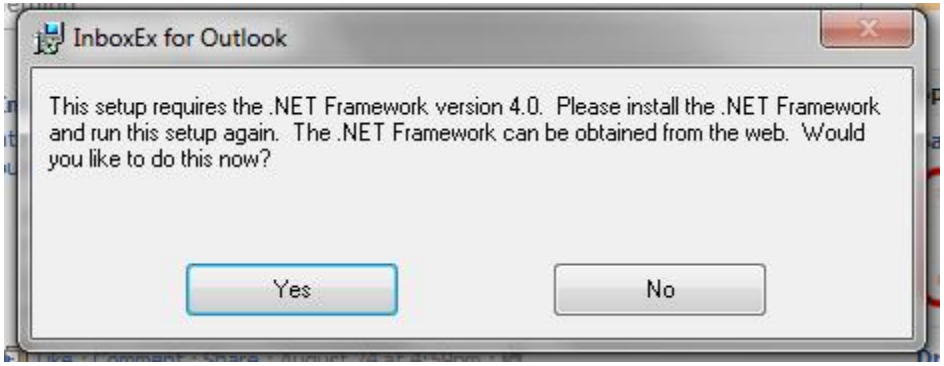
- What exactly is reversing and obfuscation?
- Reversing example
  - You're given an EXE and tasked with determining how it performs its functionality, aka secret sauce
  - Could also mean you're trying to subvert functionality such as licensing
- Obfuscation
  - Equivalent to hiding

.Net basics

- .Net, it’s a framework, nuff said
- Blanket term for microsoft family of technologies
- Most people think .Net web applications but that’s not the focus of this discussion
- The focus of this talk are stand alone executables written in C sharp although the programming language doesn’t really matter

.Net basics

- A stand alone executable built for .Net will run inside the application virtual machine, which sometimes may be referred to as the CLR (common language runtime)
- The .Net application virtual machine is a requirement for all .Net executables



Executables

- There are two main kinds of EXE's
  - Compiled and interpreted
- Compiled applications usually require an install, e.g Next > Next
- Interpreted applications require an application virtual machine
- So first you'll have to install the application virtual machine before running the interpreted application

Executables

- Compiled executables are often built with a higher programming language, such as C++, which then gets translated to a lower level language known as assembly machine language
- Due to the nature of compiled executables they are “harder” to reverse back to original source because of different compilers, architectures, and lack of information during compilation

Executables

- Interpreted executables, such as java and .Net, are much easier to reverse
- The interpreted compilation (JIT) process is more structured and retains more information about the executable
- Because of this it’s trivial to get original source code from executable
- I repeat, getting source code is trivial

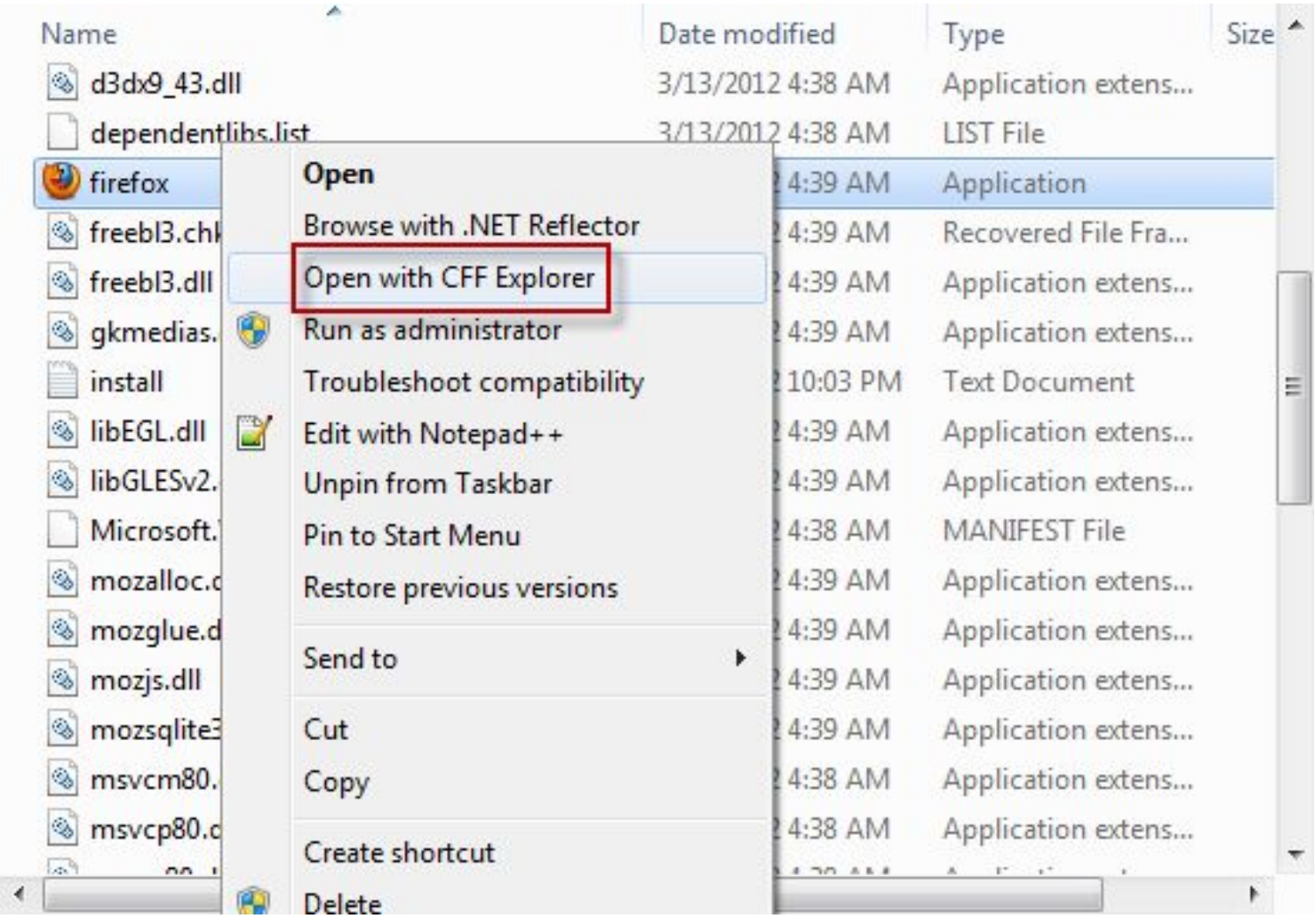
Obfuscation

- Because it’s trivial to get source code from a .Net application developers will use obfuscation to hide a majority of their source code
- There are a number of tools that will obfuscate your .Net application
- Most obfuscators will hide things such as variable and class names

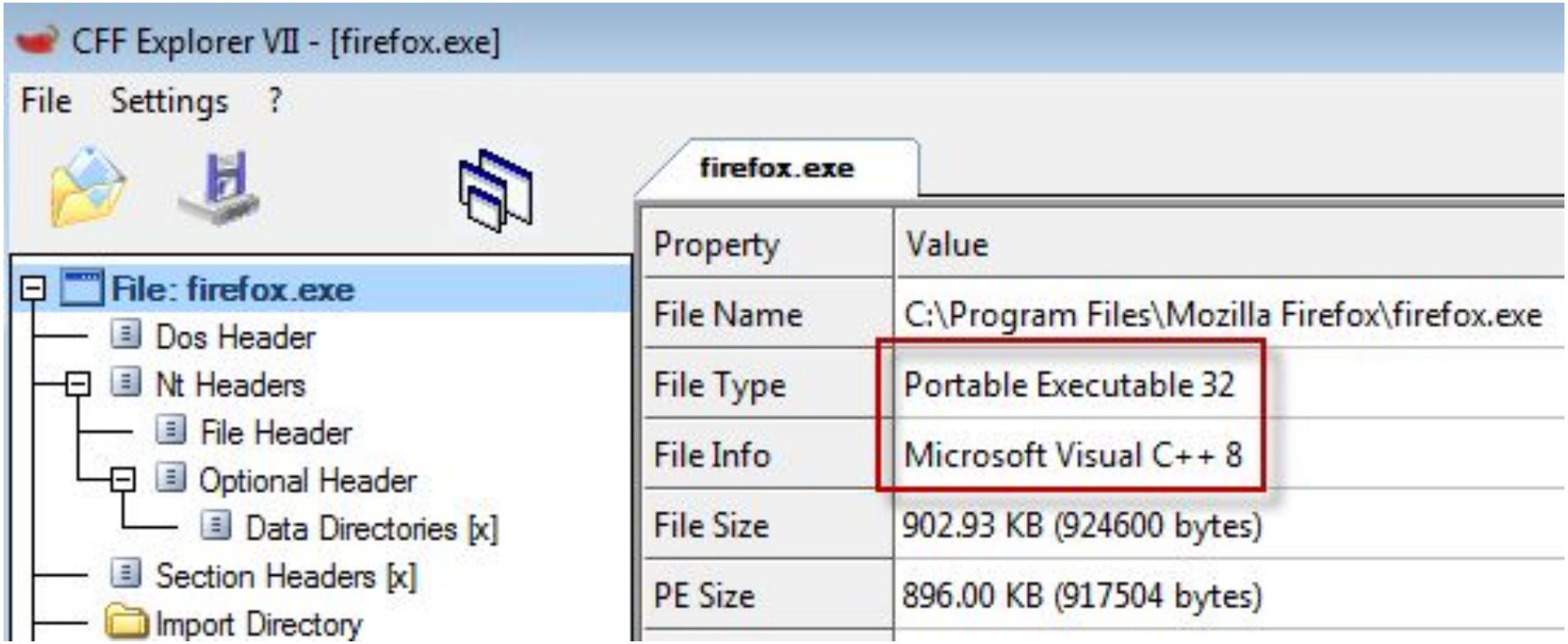
Determine type executable

- There are a number of ways to determine the type of executable you're dealing with
- The tool I like the best is CFF explorer
  - http://www.ntcore.com/exsuite.php
- Once installed you can simply right click on the executable to view the information via CFF explorer

Firefox with CFF explorer



Firefox with CFF explorer



GuessPassword.exe in CFF explorer

- GuessPassword.exe is a simple .Net application I wrote to check a password
- We'll continue to use GuessPassword.exe

GuessPassword.exe

Property	Value
File Name	C:\Users\travis\Documents\Visual Studio 2010\Projects\GuessPasswo..
File Type	Portable Executable 32 .NET Assembly
File Info	No match found.
File Size	8.00 KB (8192 bytes)

Next step > decompile

- Now you've identified the executable was built using .Net
- Next step is to decompile
- There are two tools that I like to use to decompile
- Reflector
  - www.reflector.net (paid)
- ILSpy
  - http://wiki.sharpdevelop.net/ILSpy.ashx (free)

ILSpy

ILSpy

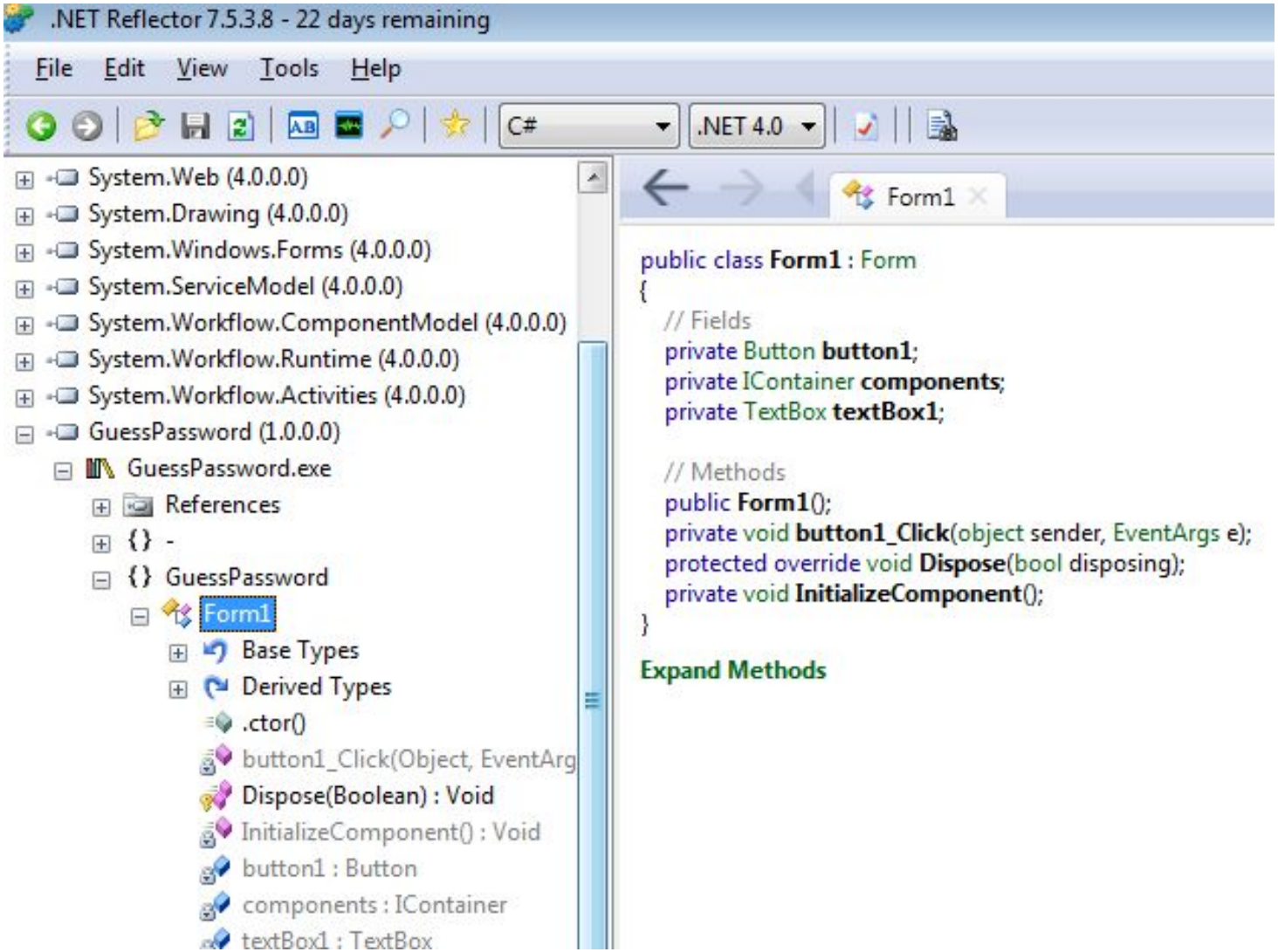
File View Debugger Help

C#

- GuessPassword
  - References
  - Resources
  - { } -
  - { } GuessPassword
    - Form1
      - Base Types
      - Derived Types
      - button1 : Button
      - components : IContainer
      - textBox1 : TextBox
      - .ctor() : void
      - button1_Click(object, EventArgs) : void
      - Dispose(bool) : void
      - InitializeComponent() : void
    - Program
      - Base Types
        - object
      - Main() : void
  - { } GuessPassword.Properties
- System.Windows.Forms
- mscorlib
- System
- System.Drawing

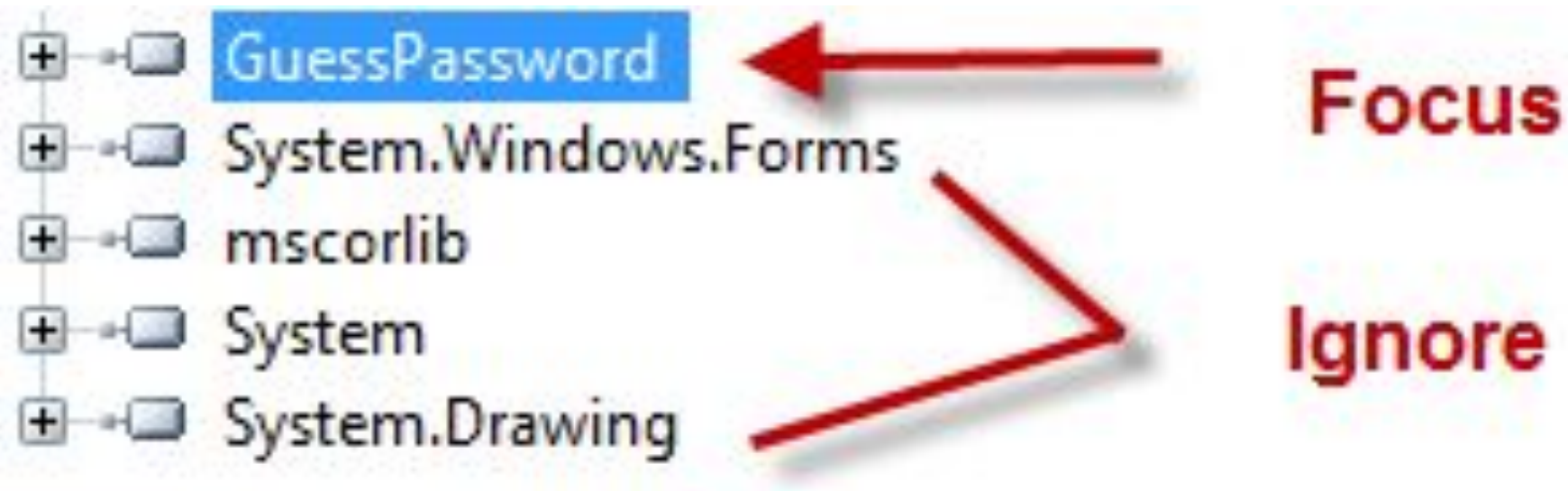
```
// C:\Users\travis\Documents\Visual Studi
// GuessPassword, Version=1.0.0.0, Cultur

// Entry point: GuessPassword.Program.Mai
// Architecture: x86
// Runtime: .NET 4.0

using ...
[assembly: AssemblyVersion("1.0.0.0")]
[assembly: Debuggable(DebuggableAttribute
[assembly: AssemblyCompany("")]
[assembly: AssemblyConfiguration("")]
[assembly: AssemblyCopyright("Copyright ©
[assembly: AssemblyDescription("")]
[assembly: AssemblyFileVersion("1.0.0.0")
[assembly: AssemblyProduct("GuessPassword
[assembly: AssemblyTitle("GuessPassword")
[assembly: AssemblyTrademark("")]
[assembly: CompilationRelaxations(8)]
[assembly: RuntimeCompatibility(WrapNonEx
[assembly: ComVisible(false)]
[assembly: Guid("1281978c-f709-497e-a293-
[assembly: TargetFramework(".NETFramework
```

Reflector


.NET Reflector 7.5.3.8 - 22 days remaining
File Edit View Tools Help
C#
.NET 4.0
System.Web (4.0.0.0)
System.Drawing (4.0.0.0)
System.Windows.Forms (4.0.0.0)
System.ServiceModel (4.0.0.0)
System.Workflow.ComponentModel (4.0.0.0)
System.Workflow.Runtime (4.0.0.0)
System.Workflow.Activities (4.0.0.0)
GuessPassword (1.0.0.0)
GuessPassword.exe
References
{} -
{} GuessPassword
Form1
Base Types
Derived Types
.ctor()
button1_Click(Object, EventArg
Dispose(Boolean) : Void
InitializeComponent() : Void
button1 : Button
components : IContainer
textBox1 : TextBox
Form1
public class Form1 : Form
{
// Fields
private Button button1;
private IContainer components;
private TextBox textBox1;
// Methods
public Form1();
private void button1_Click(object sender, EventArgs e);
protected override void Dispose(bool disposing);
private void InitializeComponent();
}
Expand Methods

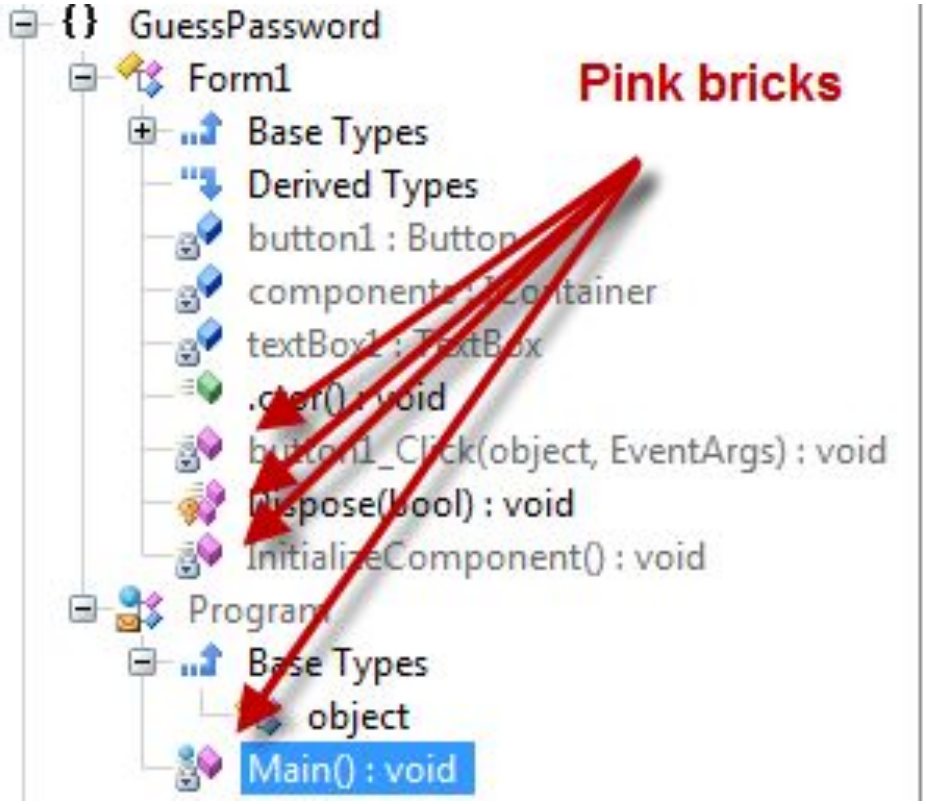
Decompilation tips

- Only analyze your exe tree and ignore dependencies that get pulled in



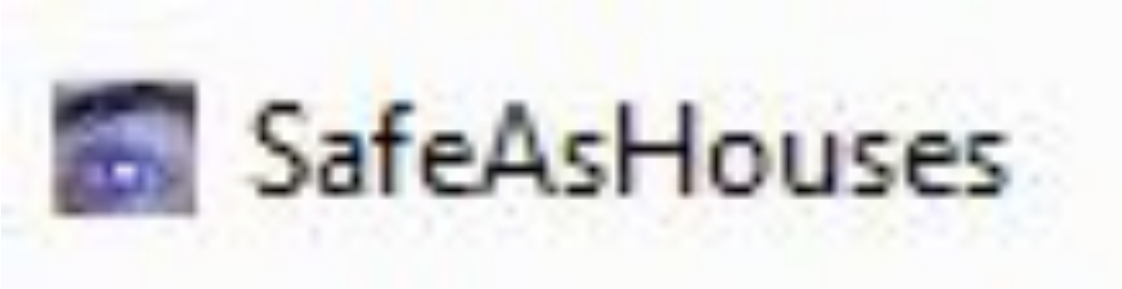
Decompilation tips

- Only focus on the “real” code
- Real code located in pink bricks

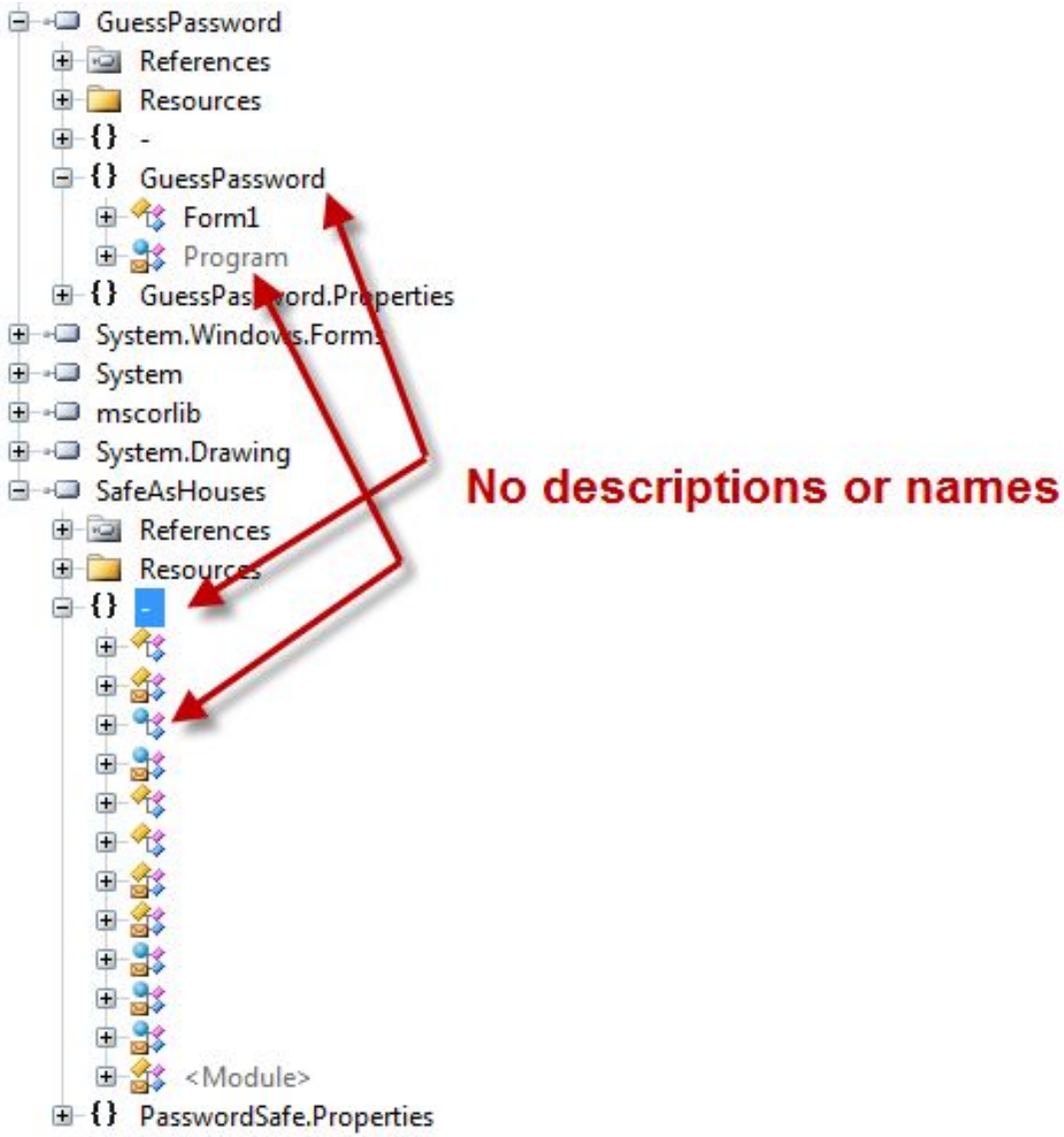


SafeAsHouses.exe

- Let’s take a look at a real life example
- SafeAsHouses.exe can be downloaded from from either download.com or softpedia.com
- SafeAsHouses is a password keeper, it’s designed to keep all your passwords safe in one location



SafeAsHouses in ILSpy

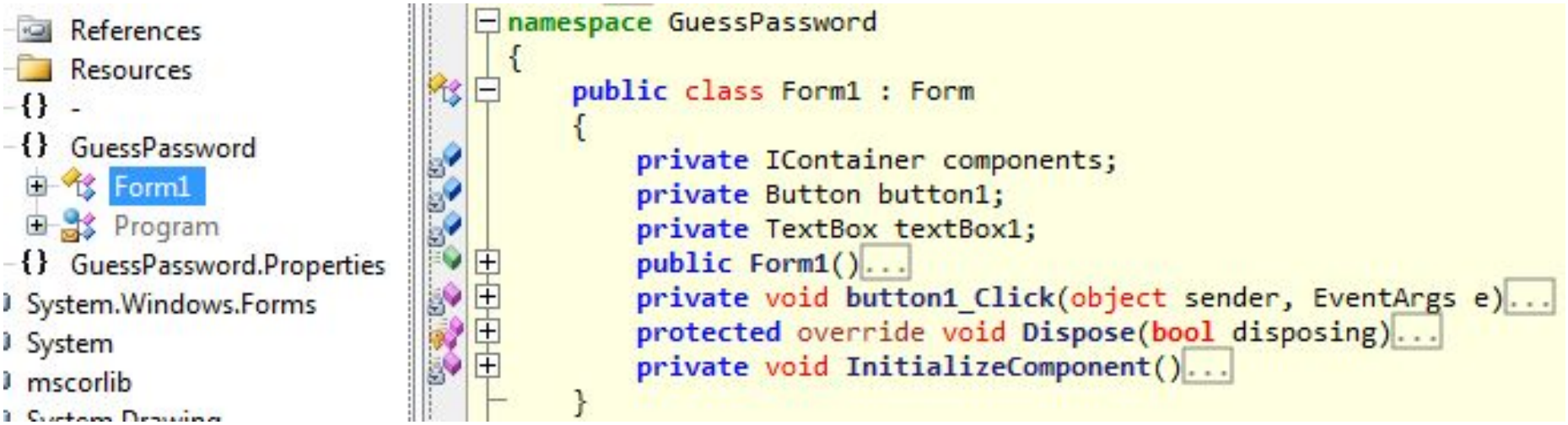


SafeAsHouses in ILSpy

- So soon as we open SafeAsHouses.exe in ILSpy we see signs of obfuscation
- In the GuessPassword example we see the class names are clearly visible
- SafeAsHouses masks this information to make it harder to understand the underlying functionality of the application

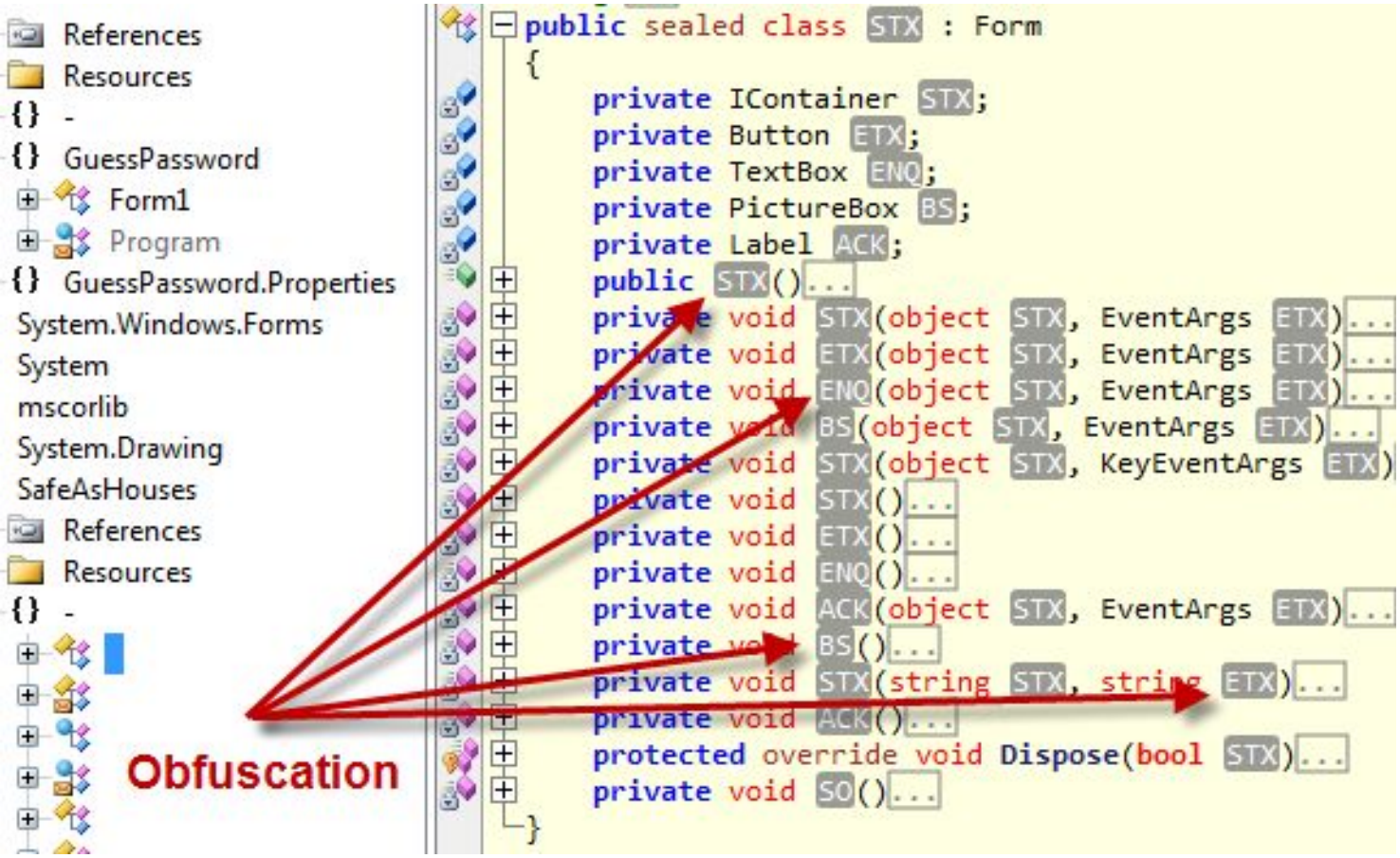
GuessPassword vs SafeAsHouses

- Next let's compare GuessPassword and SafeAsHouses in ILSpy
- First GuessPassword



GuessPassword vs SafeAsHouses

- SafeAsHouses

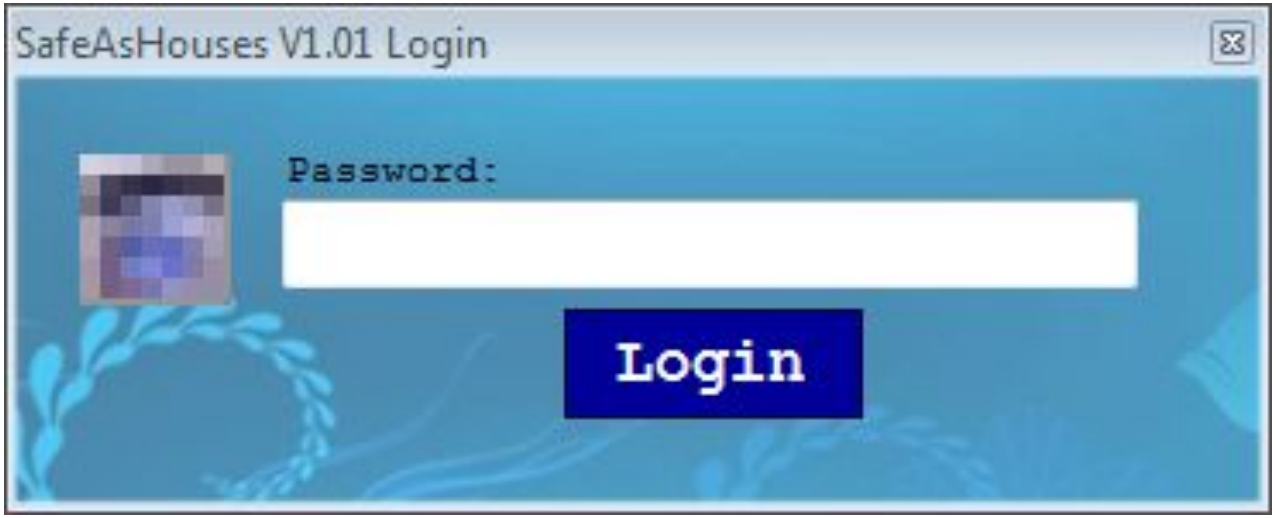


More obfuscation

- Here we see more signs of obfuscation
- In GuessPassword we see how things should look, we clearly get to see class names in the application
- In SafeAsHouses however things are hidden
- Instead of class names all we see are these grey boxes such as “STX”, “ETX”, etc
- STX, ETX, etc

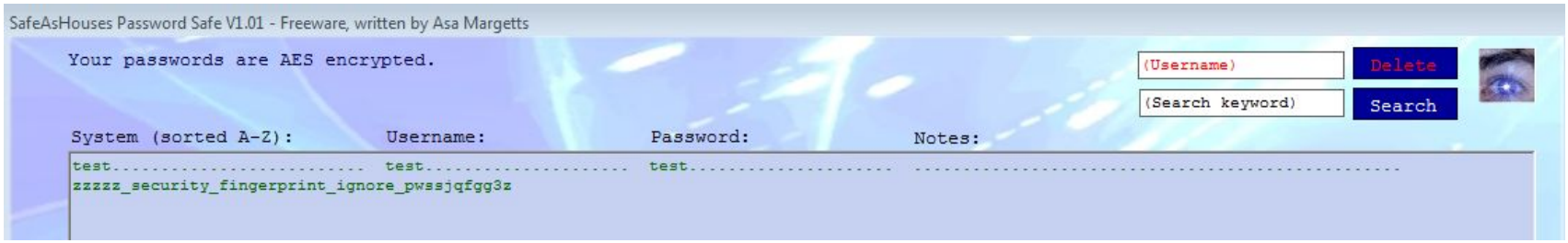
Understanding the app

- Typically you would run the application first to get an idea of the functionality but wanted to show the obfuscation first
- Run the application



Understanding the app

- If we authenticate with the correct password we are granted access to the application



- Incorrect password we get denied

Items of interest

1. The application asks for input
2. Incorrect password presents a pop up stating “Password incorrect”
3. The application exits after you click OK when your password is incorrect
4. Successful authentication brings up another window

Items of interest

- The main idea is to take certain areas of interest inside the application and find that functionality
- Even if the application is obfuscated hopefully identifying an item of interest will lead us to the code we want to reverse
- I’ve highlighted four items of interest but you could easily focus on other areas

Searching for items of interest

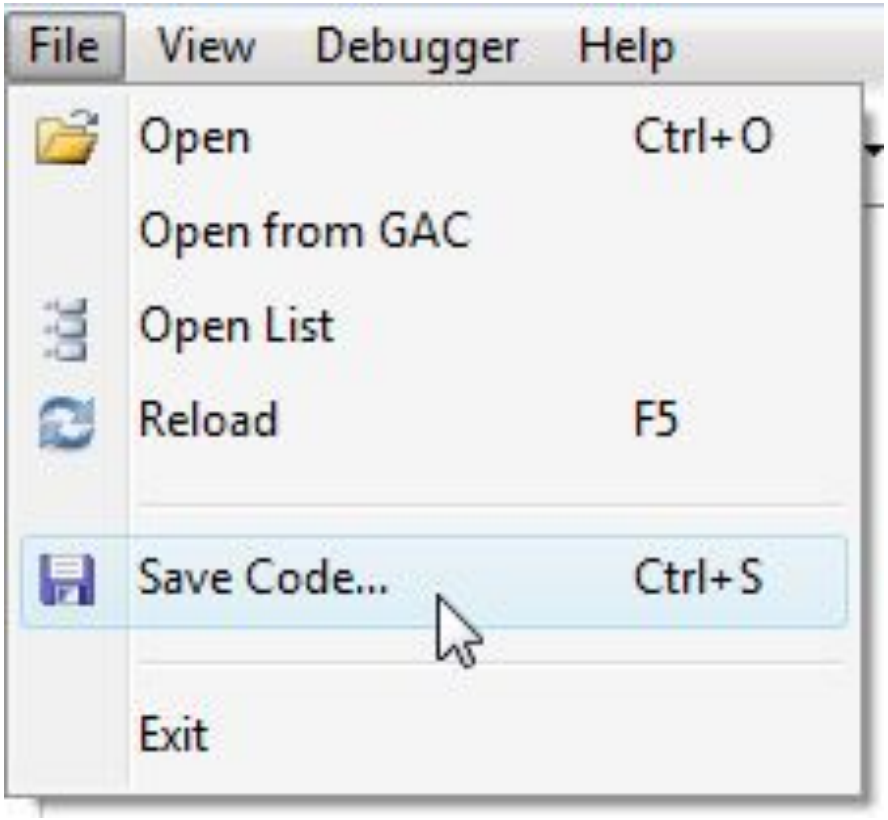
- Now that we've done some recon and understand the type of functionality we want to go after we need to search for that
- There are a number of ways to search for items of interest but I'll highlight two

1. Decompile all code into one text file
2. Use the Reflector plugin "Code search"

Decompile code into text file

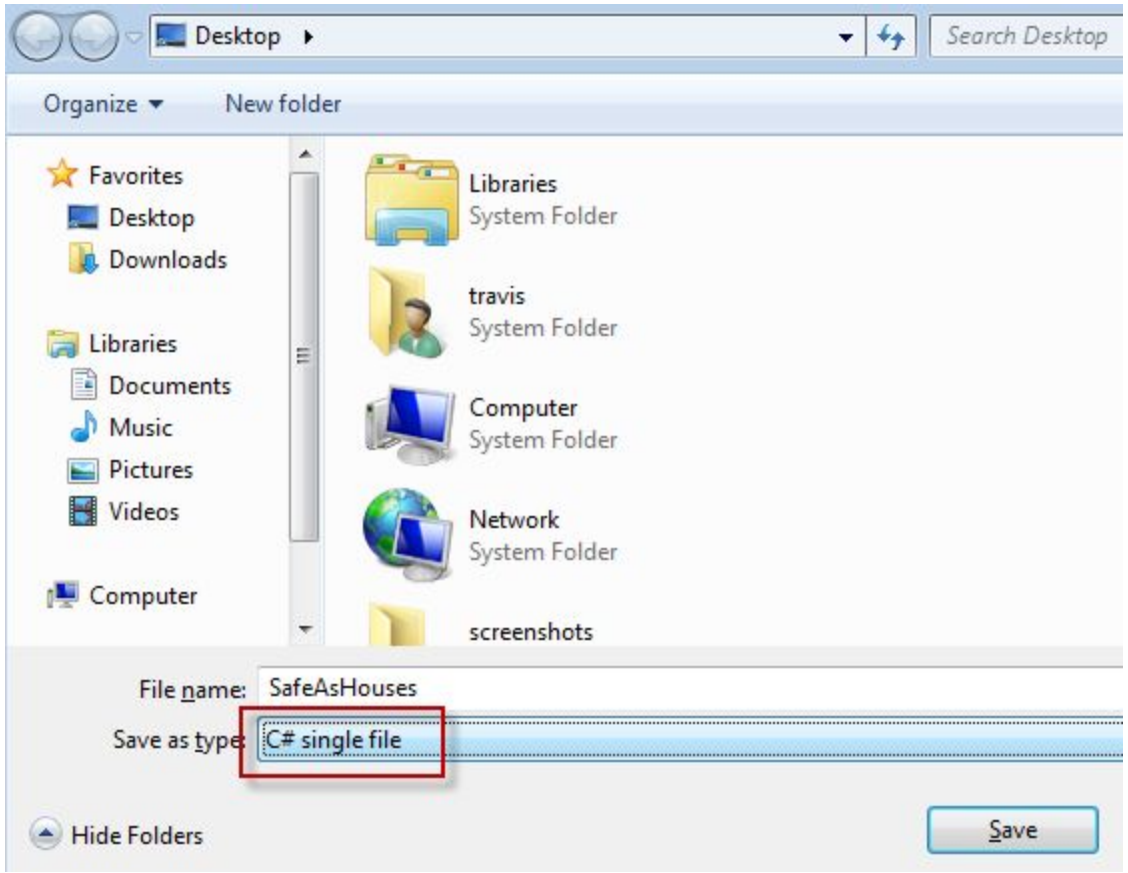
Decompile code into text file

- Next choose File > Save Code



Decompile code into text file

- Then save the file as a “C# single file”



Decompile code into text file

- At this point you can use your favorite text editor to search through the decompiled code
- Disadvantage is that searching through a flat file doesn’t present a lot of context
- On the other hand it’s always handy to have a raw dump and the ability to save for historical purposes

Reflector code search plugin

- Reflector's code search plugin is very convenient in that you don't have to leave the reflector tool
- With the code search plugin you also don't loose the context with where code functionality is located
- Using code search we'll search for all four items of interest
- http://reflectoraddins.codeplex.com/wikipage?title=CodeSearch&referringTitle=Home

Items of interest

1. The application asks for input
2. Incorrect password presents a pop up stating “Password incorrect”
3. The application exits after you click OK when your password is incorrect
4. Successful authentication brings up another window

Application asks for input

- A popular way of taking form input in a .Net application is through the text box class where the convention is “this.textBox1.Text”
- Here textBox1 is a variable name
- Most obfuscators will hide the variable name with something like “this.STX.Text”
- So better to search for “this.*.Text” inside the code search reflector plugin

Application asks for input

Code Search 5.0.0.28288

this.*.Text

Path	Hit Count
. () : Void	1
.() : Void	5
. () : Void	7
. () : Void	4
. () : Void	32
. (Object, EventArgs) : Void	1
. () : Void	3

- Searching for this.*.Text revealed numerous hits, probably best to keep searching for different terms, you might also want to search for just *.Text

Pop up message

- A pop up box is typically done with the “MessageBox.Show” method
- Two arguments can be given to this method, window title and window message
- MessageBox.Show(“title”, “message”)
- Use code search to see how many message boxes are in the application, the idea is to hopefully pinpoint this functionality

Pop up message

Code Search 5.0.0.28288

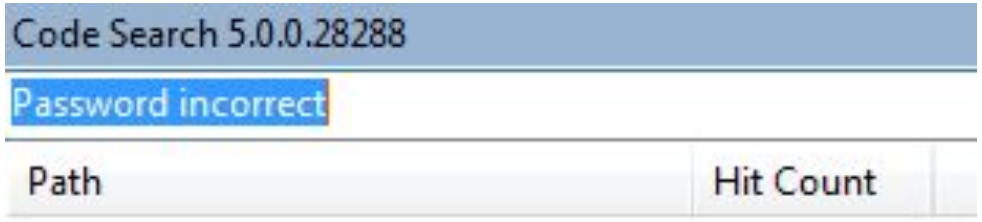
MessageBox.Show

Path	Hit Count
.└(Object, EventArgs) : Void	1
. 0 : Void	1
.└0 : Void	1
.|0 : Void	1
.◘0 : Void	1
.–0 : Void	1
◘.└0 : Void	4
◘.|0 : Void	1

- So quite a few hits on MessageBox.Show, let’s continue searching for other constructs

Pop up message

- The pop up message states “Password incorrect”
- We should search the code for strings like this



- No dice, the phrase “Password incorrect” must be obfuscated

Application closes

- If you enter an incorrect password you'll get a pop up, after clicking OK the application will close
- .Net can handle this in a couple of ways, with Application.Exit and Environment.Exit
- Let's search for these terms as well

Application closes

Code Search 5.0.0.28288

Environment.Exit

Path	Hit Count
.└(Object, EventArgs) : Void	1
. () : Void	1
.└() : Void	1
.|() : Void	1
.◘() : Void	1

- Less results which means which means less manual reviewing of code

Successful authentication opens another window

- Probably the most popular way to show one window then hide another is to use the window.Show() and window.Hide() methods
- They are used in tandom
- Even though they are commonly used in tandom it’s a good idea to search for both terms

Successful authentication opens another window

Code Search 5.0.0.28288

Hide()

Path	Hit Count
. () : Void	1
. (Object, EventArgs) : Void	1
. └(Object, EventArgs) : Void	1

- Only three hits, we're money

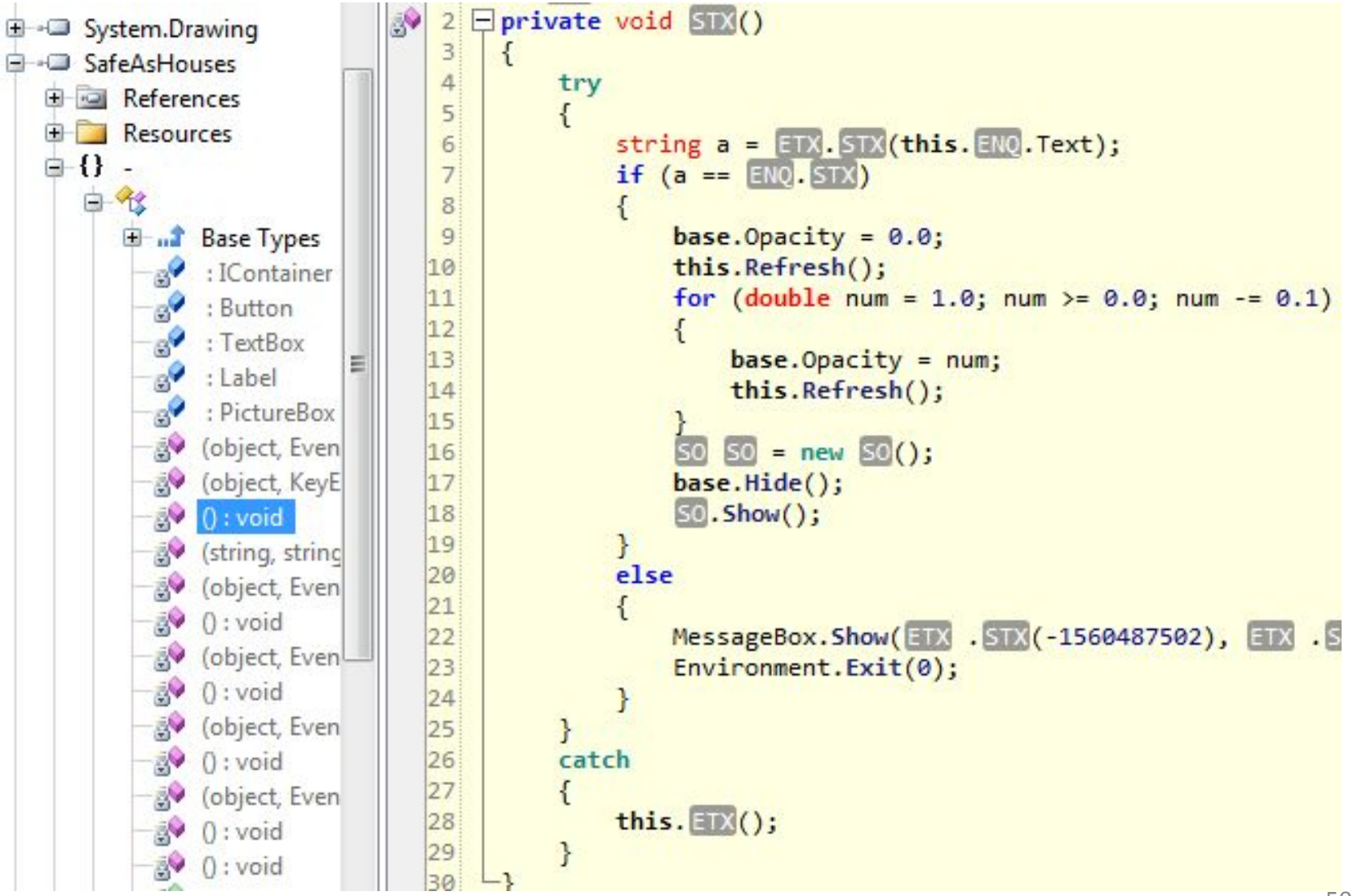
Code search plugin

- Using this plugin we were able to narrow down to only three locations where our authentication functionality is most likely hiding
- Click on each result to view the obfuscated code
- Look for the other constructs, Environment.Exit, this.*.Text, etc
- Code search is case sensitive

First hit, found the functionality

```
private void ┐()
{
    try
    {
        if (└.┐(this.|.Text) == |.┐)
        {
            base.Opacity = 0.0;
            this.Refresh();
            for (double i = 1.0; i >= 0.0; i -= 0.1)
            {
                base.Opacity = i;
                this.Refresh();
            }
            ♫ ♫ = new ♫();
            base.Hide();
            ♫.Show();
        }
        else
        {
            MessageBox.Show(└ .┐(-1560487502), └ .┐(-1560487461));
            Environment.Exit(0);
        }
    }
    catch
    {
```

Same code in ILSpy



Explanation of code

- Line 6: developer is assigning variable “a” to whatever you type into the text box
- Line 7: if statement comparing password values
- Line 17: if password is correct base.Hide will hide the login box
- Line 18: will show the main window
- Line 22: message box that tells you your password is incorrect
- Line 23: closes the application

Subverting authentication

- Now that we've found the code that performs the authentication we want to subvert that functionality to gain access without knowing the password
- There are two tools that will allow us modify the executables
- Graywolf
  - http://digitalbodyguard.com/GrayWolf.html
- Reflexil
  - http://reflexil.net/

Reflexil

```
private void ┐ ()
{
    try
    {
        if (└ .┐ (this.| .Text) == | .┐ )
        {
            base.Opacity = 0.0;
            this.Refresh();
            for (double i = 1.0; i >= 0.0; i -= 0.1)
            {
```

Sebastien LEBRETON's Reflexil v1.5

Method definition

Instructions | Variables | Parameters | Exception Handlers | Overrides | Attributes | C

	Offset	OpCode	Operand
00	0	ldarg.0	
01	1	ldfld	System.Windows.Forms.TextBox ::|
02	6	callvirt	System.String System.Windows.Forms.(
03	11	call	System.String └:: (System.String)
04	16	stloc.0	

Reflexil

- The bottom half of the previous screen shot shows the reflexil plugin to reflector
- With reflexil we can edit the executable
- We'll be editing content in the "Instructions" tab within reflexil
- These instructions are referred to as CIL (common intermediate language)

CIL

- Lower level language used by the .Net application virtual machine
- So your higher level programming, such as C#, gets converted to CIL (sometimes called IL)
- The CIL instructions will then be JIT compiled into native machine code at run time
- Opcodes are at the heart of CIL and tell the application what to do

CIL

- Not that important to understand all the technical details behind CIL
- On lines 01, 02, and 03 we can see these three lines are more than likely responsible for getting input from user via a text box

01	1	ldfld	System.Windows.Forms.TextBox ::|
02	6	callvirt	System.String System.Windows.Forms.
03	11	call	System.String ::(System.String)

Modifying CIL

- Looking through the code and CIL we see an interesting instruction on line 07
- The operand to the instruction is “op_Equality” that compares passwords
- The next opcode instruction on line 08 is “brfalse”
- Stands for branch if false, so it’s the if statement

Modifying CIL

- You'll also notice the operand for the opcode on line 08 is "->(36)"
- This means branch to line 36 if the password doesn't match
- This branches all the way down to the message box functionality
- To break this functionality we can change the brfalse opcode to the opposite which is brtrue

Modifying CIL

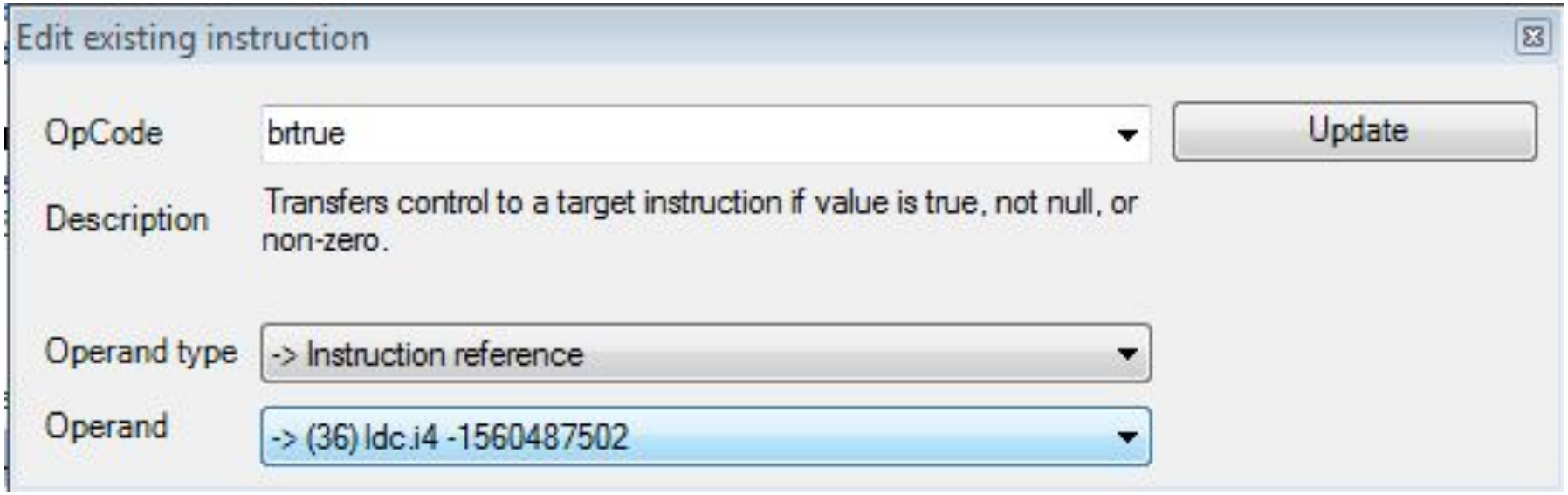
- Right click on opcode and choose edit

05	17	ldloc.0	
06	18	ldsfld	System.String |::
07	23	call	System.Bo
08	28	brfalse	-> (36) ldc
09	33	ldarg.0	

Create new...
Edit...
Replace all with code...

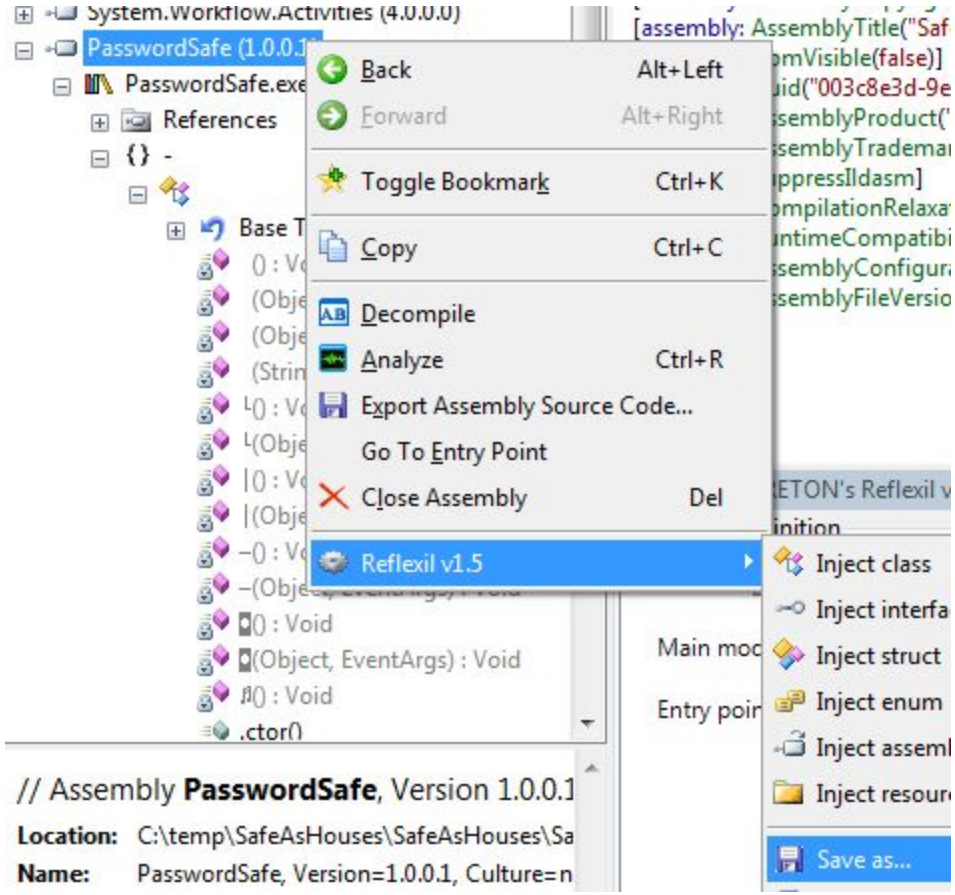
Modifying CIL

- Next change to brtrue then click update



Modifying CIL

- Next right click on root tree and save



Patched executable

- Now you've successfully patched a .Net executable
- If we run this we can provide the incorrect password and successfully authenticate but if we provide the correct password the application will close
- We don't have to stop there we can delete chunks of code to remove that functionality as well

Deleting code blocks

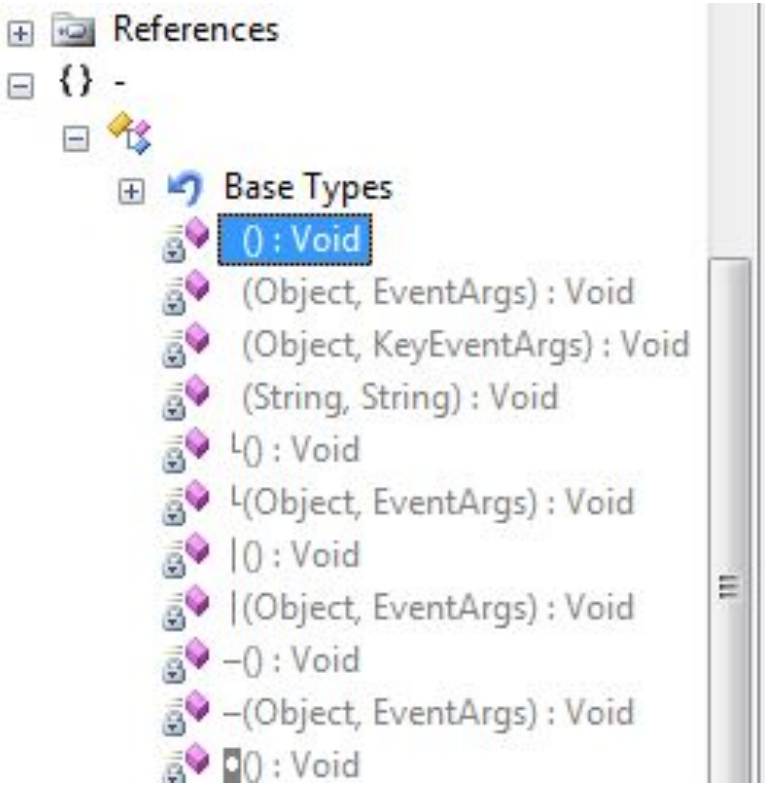
```
2   private void STX()
3   {
4       try
5       {
6           string a = ETX.STX(this.ENQ.Text);
7           if (a == ENQ.STX)
8           {
9               base.Opacity = 0.0;
10              this.Refresh();
11              for (double num = 1.0; num >= 0.0; num -= 0.1)
12              {
13                  base.Opacity = num;
14                  this.Refresh();
15              }
16              SO SO = new SO();
17              base.Hide();
18              SO.Show();
19          }
20          else
21          {
```

Delete this code

Delete code blocks

- Delete lines 1-28
- Save the patched application
- Run it again and now it won’t matter what password you type in because we’ve deleted that entire if statement that checks for the password
- Next open the patched application in reflector to see if the code block was deleted

Missing code block




```
private void ┐()
{
    try
    {
        ♫ ♫ = new ♫();
        base.Hide();
        ♫.Show();
        return;
        MessageBox.Show(└ ┐(-1560487502), └ ┐(-1560487461));
        Environment.Exit(0);
    }
    catch
    {
        this.└();
    }
}
```

Great success

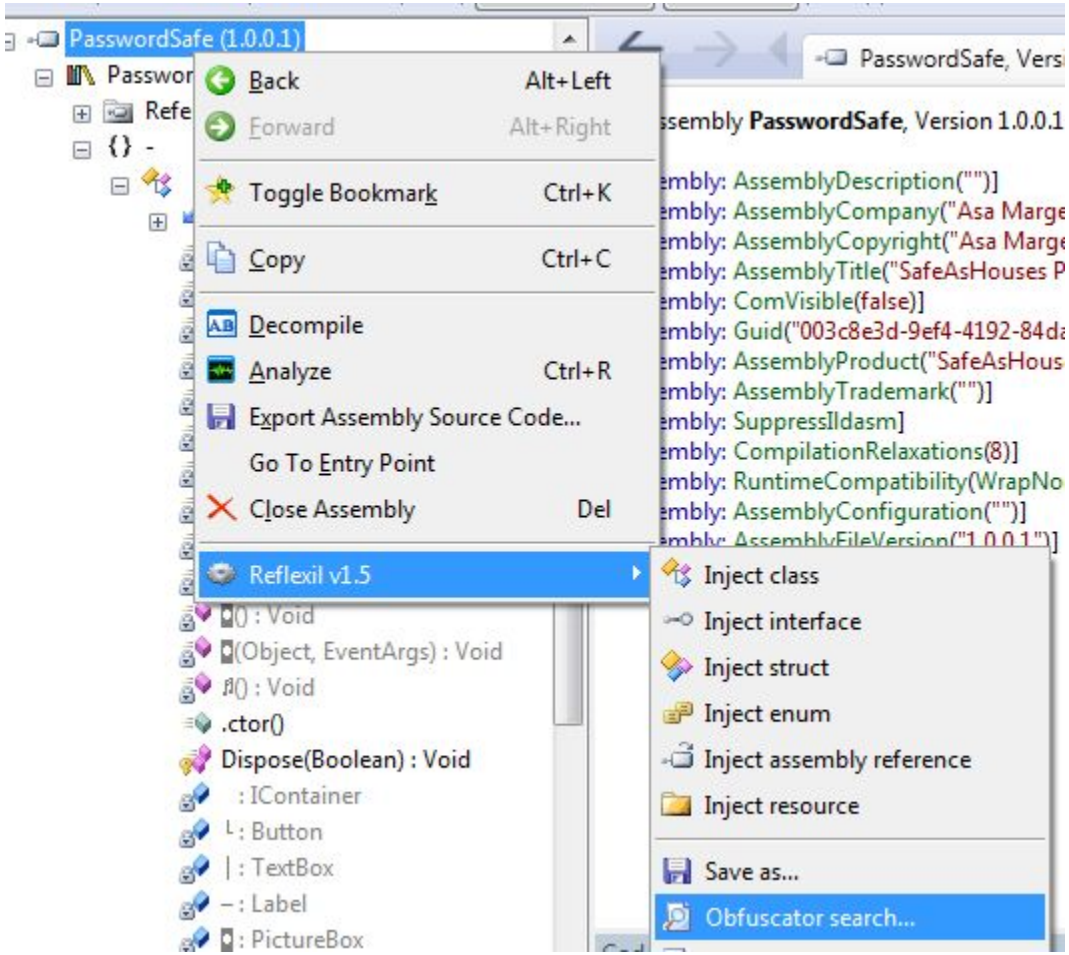
- We've successfully modified the application to subvert authentication
- If we had obtained this executable from another user then we would have all their passwords
- Hopefully you see how easy it is to control and modify .Net applications to your heart's content

Easier reversing via deobfuscation

- There is a slightly easier way to go about reversing an obfuscated .Net application
- If we can deobfuscate the obfuscated code then we'll have a much easier time understanding the functionality of the application
- Luckily reflexil can deobfuscate many obfuscation tools

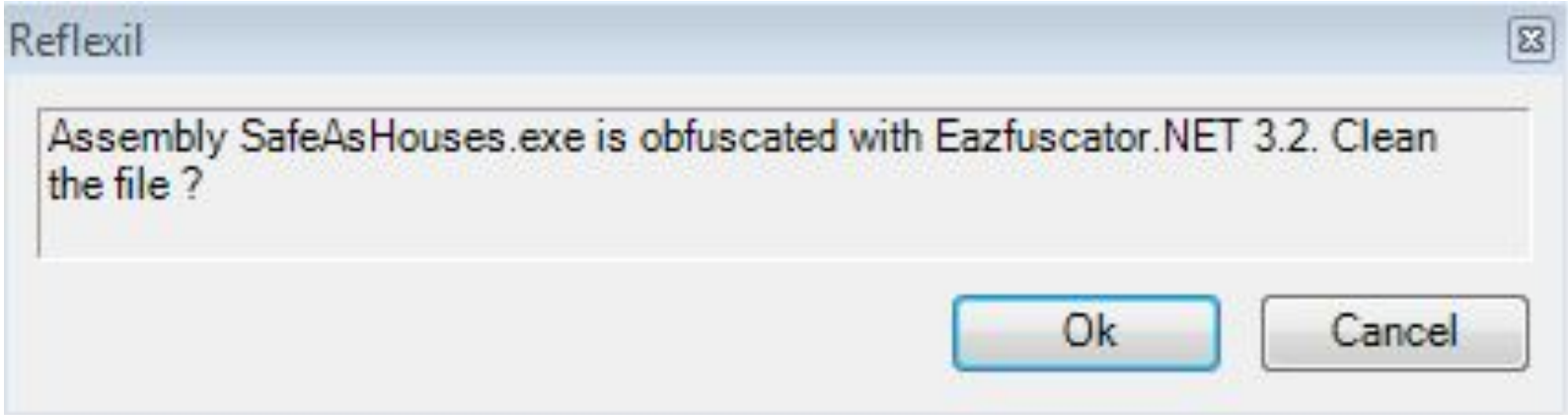
Reflexil deobfuscation

- Right click > Reflexil > Obfuscator search



Reflexil deobfuscation

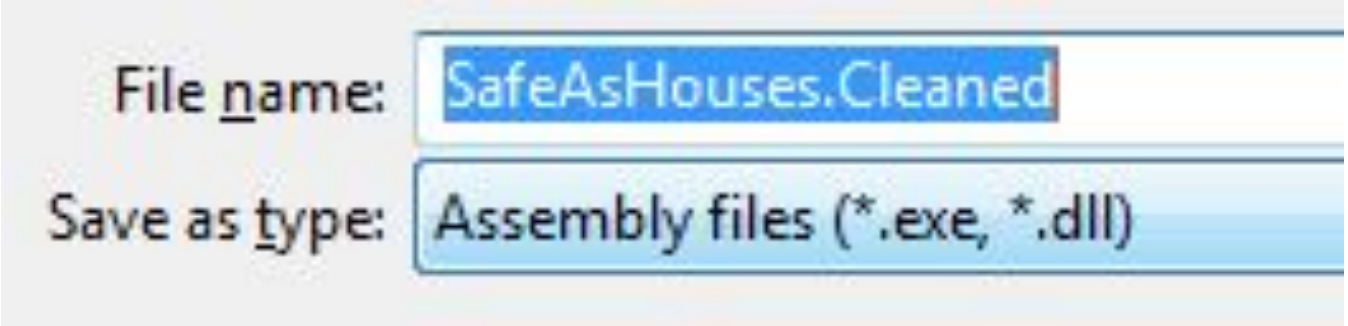
- Next reflexil will try to determine the type of obfuscation used



- Here it successfully determined it was obfuscated with Eazfuscator.NET 3.2

Reflexil deobfuscation

- Next you save the executable, you can stick with the default *.Cleaned extension so you don't accidentally write over the original exe



- Next open your saved SafeAsHouses.Cleaned in reflector to view deobfuscated code

Deobfuscated code

- GClass0
- LogonForm
  - Base Types
  - .ctor()
  - button_login_Click(Object, Ever
  - Dispose(Boolean) : Void
  - field_logonpassword_KeyDown
  - field_logonpassword_TextChan
  - InitializeComponent() : Void
  - LogonForm_Load(Object, Event
  - method_0() : Void
  - method_1() : Void
  - method_2() : Void
  - method_3() : Void
  - method_4(String, String) : Void
  - method_5() : Void
  - pictureBox1_Click(Object, Event
  - pictureBox1_MouseHover(Obje

```
private void method_0()
{
    try
    {
        if (Class0.smethod_0(this.field_logonpassword.Text) == GClass0.string_0)
        {
            base.Opacity = 0.0;
            this.Refresh();
            for (double i = 1.0; i >= 0.0; i -= 0.1)
            {
                base.Opacity = i;
                this.Refresh();
            }
            MainForm form = new MainForm();
            base.Hide();
            form.Show();
        }
        else
        {
            MessageBox.Show("Password incorrect", "ACCESS DENIED");
            Environment.Exit(0);
        }
```

Deobfuscated code

- So reflexil actually does a nice job on getting us better source code
- It may not retrieve original variables or class names but it will at least name them var1, var2, etc to give better meaning
- In this case it actually revealed the pop up message of “Password incorrect” to whereas before that was obfuscated

Steps to reverse .Net app

1. Run the application to understand functionality
2. Decompile the application
3. Review source code and hone in on the functionality you're trying to understand
4. If obfuscated look for key constructs to understand functionality
5. Optional: Modify app to achieve your desired functionality

Wrapping up

- A standalone .Net executable will more than likely be very easy to decompile to get original source code
- Obfuscation techniques only make it a little bit harder to figure out the original source code
- If you want to save your intellectual property then don’t write the software that utilizes the .Net framework

Questions

http://twitter.com/#!/curtismechling

http://travisaltman.com