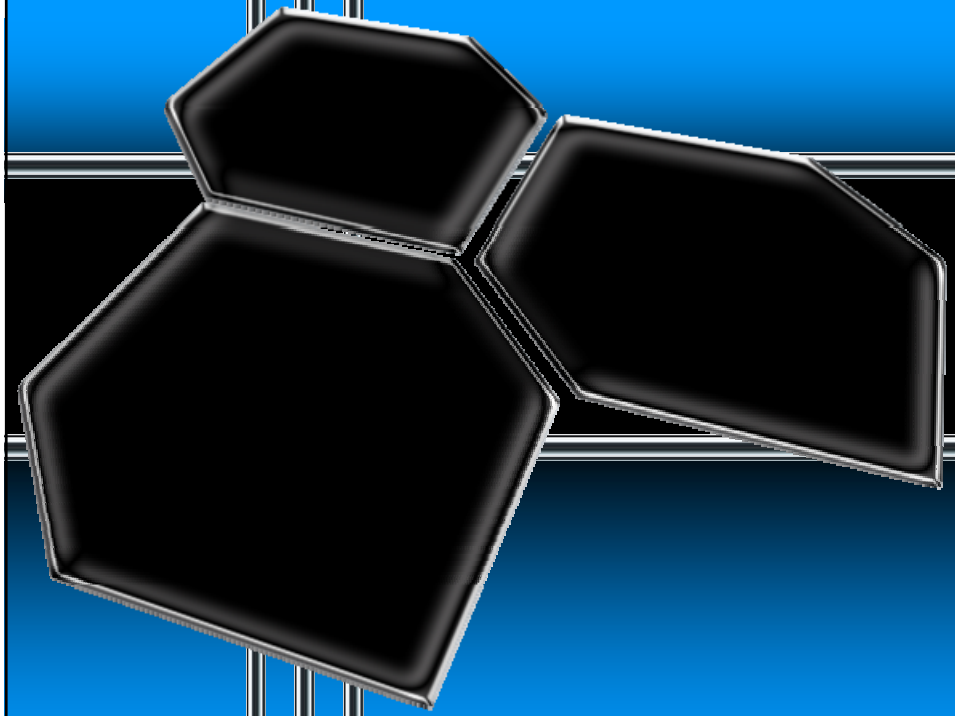


浅谈.NET软件保护的现状与发展趋势

单海波(tankaiha)

2008.看雪论坛.微软技术交流会
http://bbs.pediy.com

主要内容

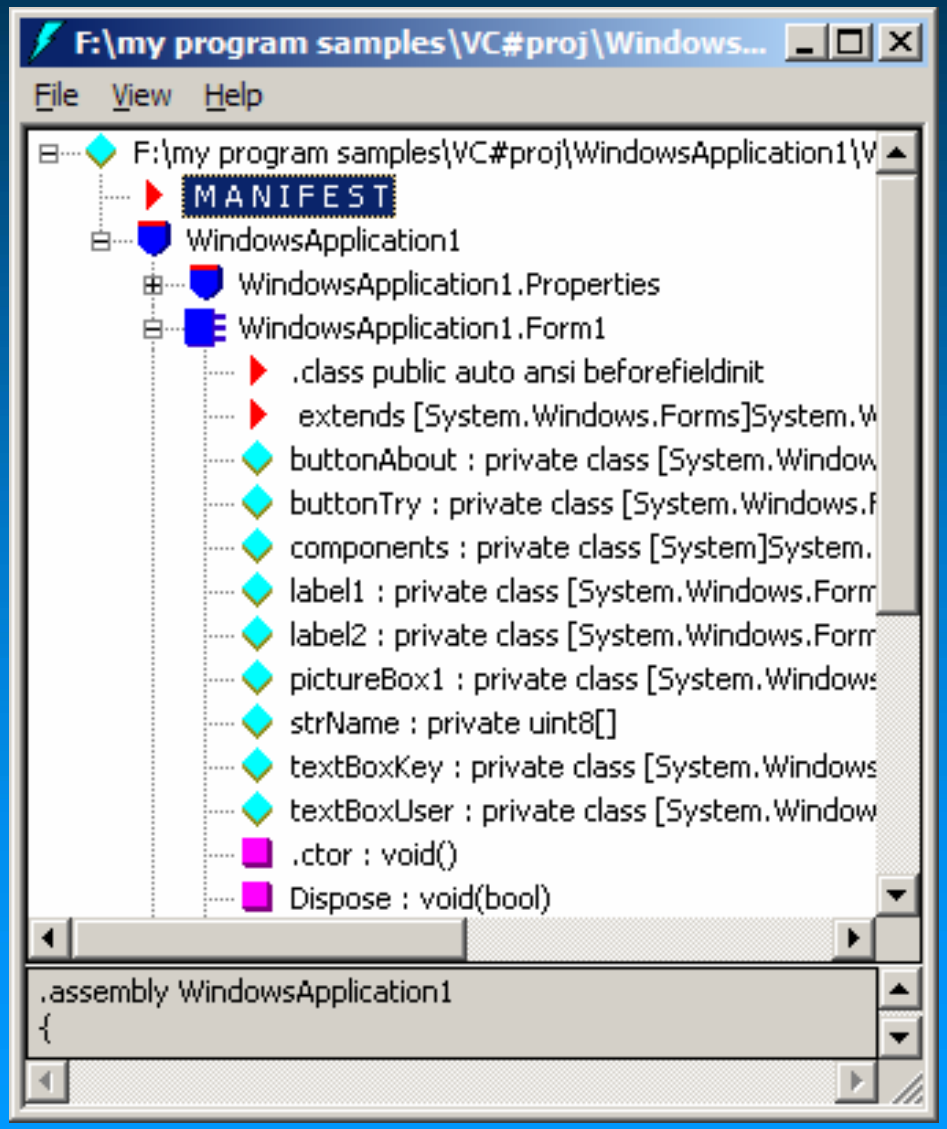
- 为什么要保护.NET程序
- .NET程序保护方式现状
- .NET程序保护的发展趋势

1. 为什么要保护.NET程序

- 普通PE文件：保存x86/x64汇编编码
- 托管PE文件：元数据和MSIL

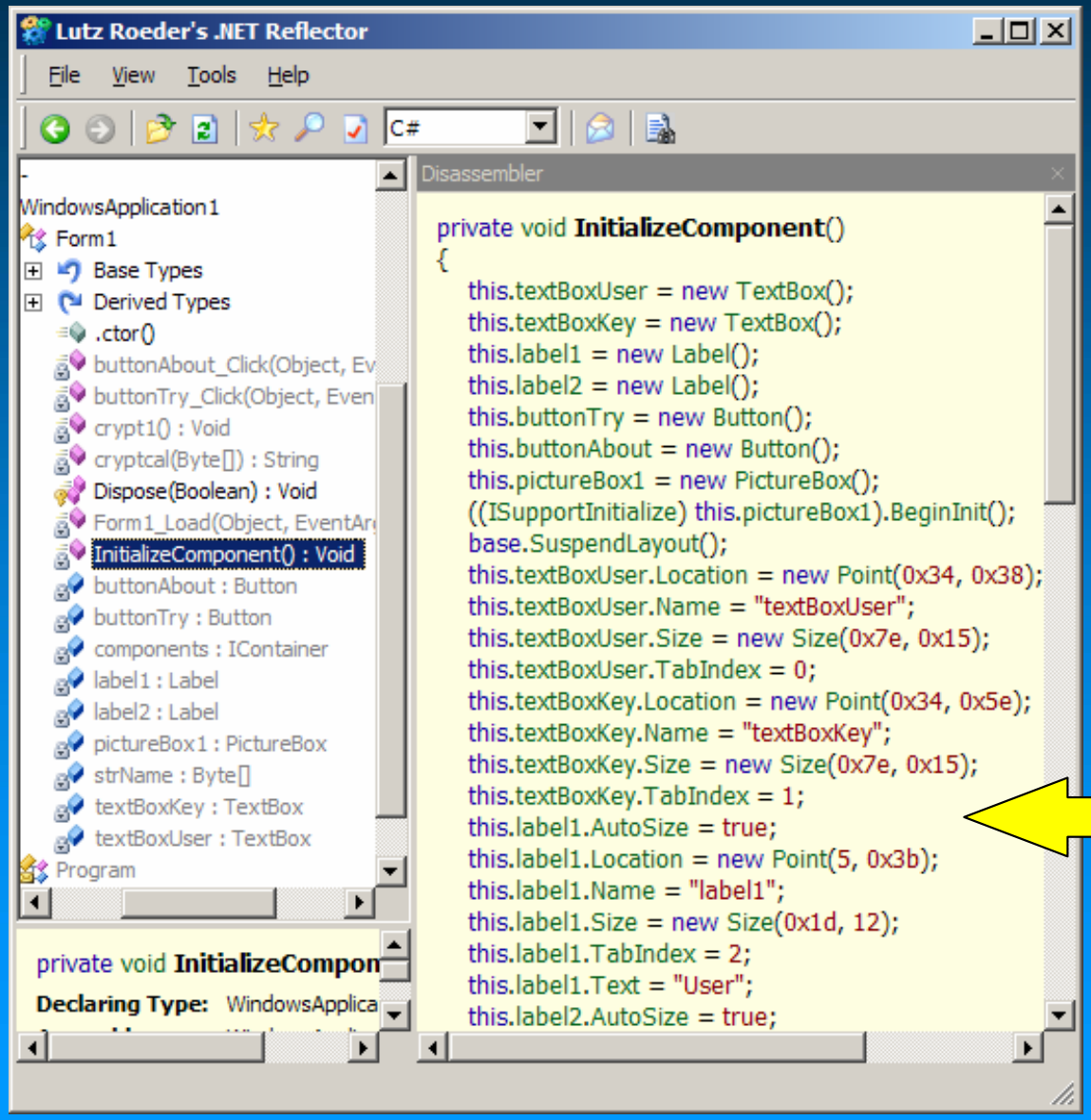
Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Ascii
00000860	38	00	00	00	02	00	00	11	2B	25	2B	2A	12	01	2B	29	8...▌..▌+%+*▌▌+)
00000870	0A	06	02	7B	02	00	00	04	59	20	80	96	98	00	6A	31	.▌▌{▌..▌Y.▌▌▌.j1
00000880	0D	02	06	7D	02	00	00	04	02	28	02	00	00	06	2A	28	.▌▌}▌..▌▌(▌..▌*(
00000890	2E	00	00	0A	2B	D4	0B	2B	D3	28	2F	00	00	0A	2B	D0	....+Ô▌+Ó(/...+Ð
000008A0	13	30	03	00	3D	00	00	00	03	00	00	11	02	28	2E	00	▌0▌.=...▌..▌▌(..
000008B0	00	0A	0A	12	00	28	2F	00	00	0A	7D	02	00	00	04	02	...▌.(/...}▌..▌▌

利用元数据，可以做些什么？



Decompile to MSIL

利用元数据，可以做些什么？

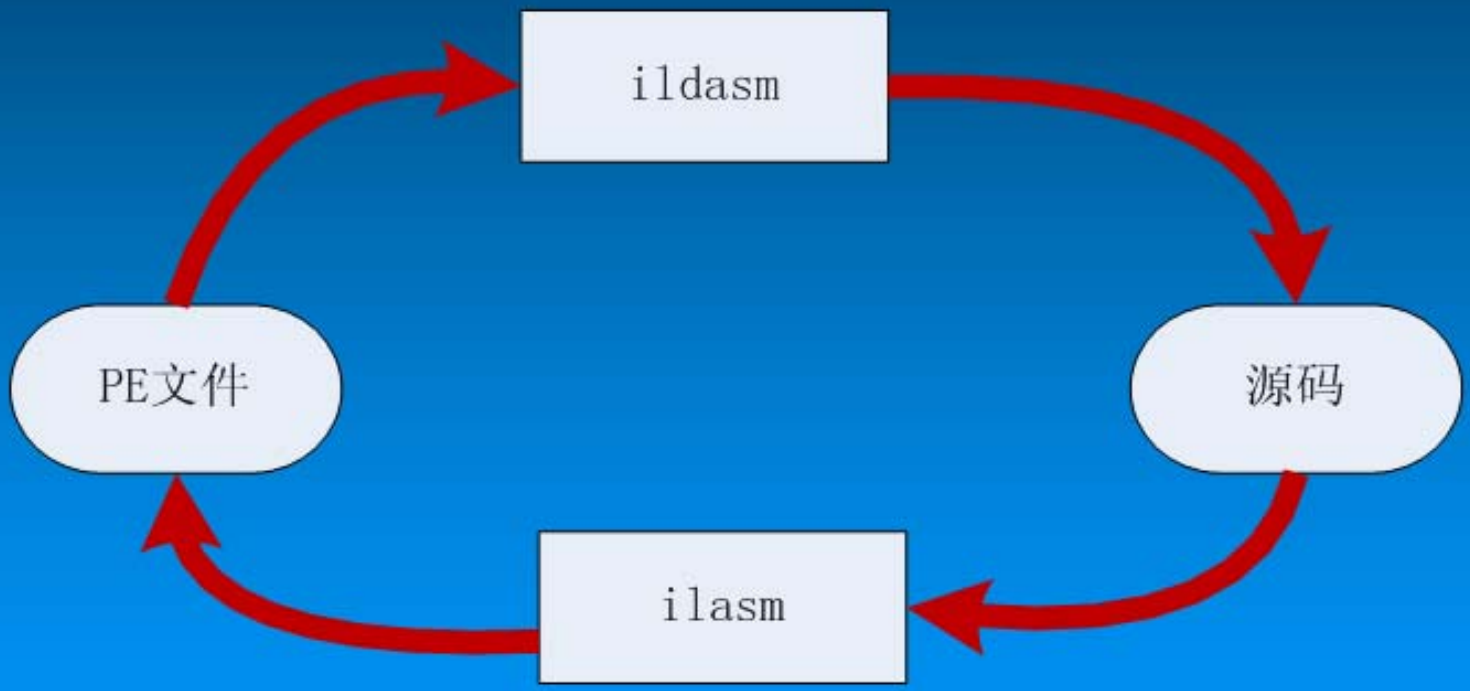


Decompile to
High Level Language

WOW!
Source Code!

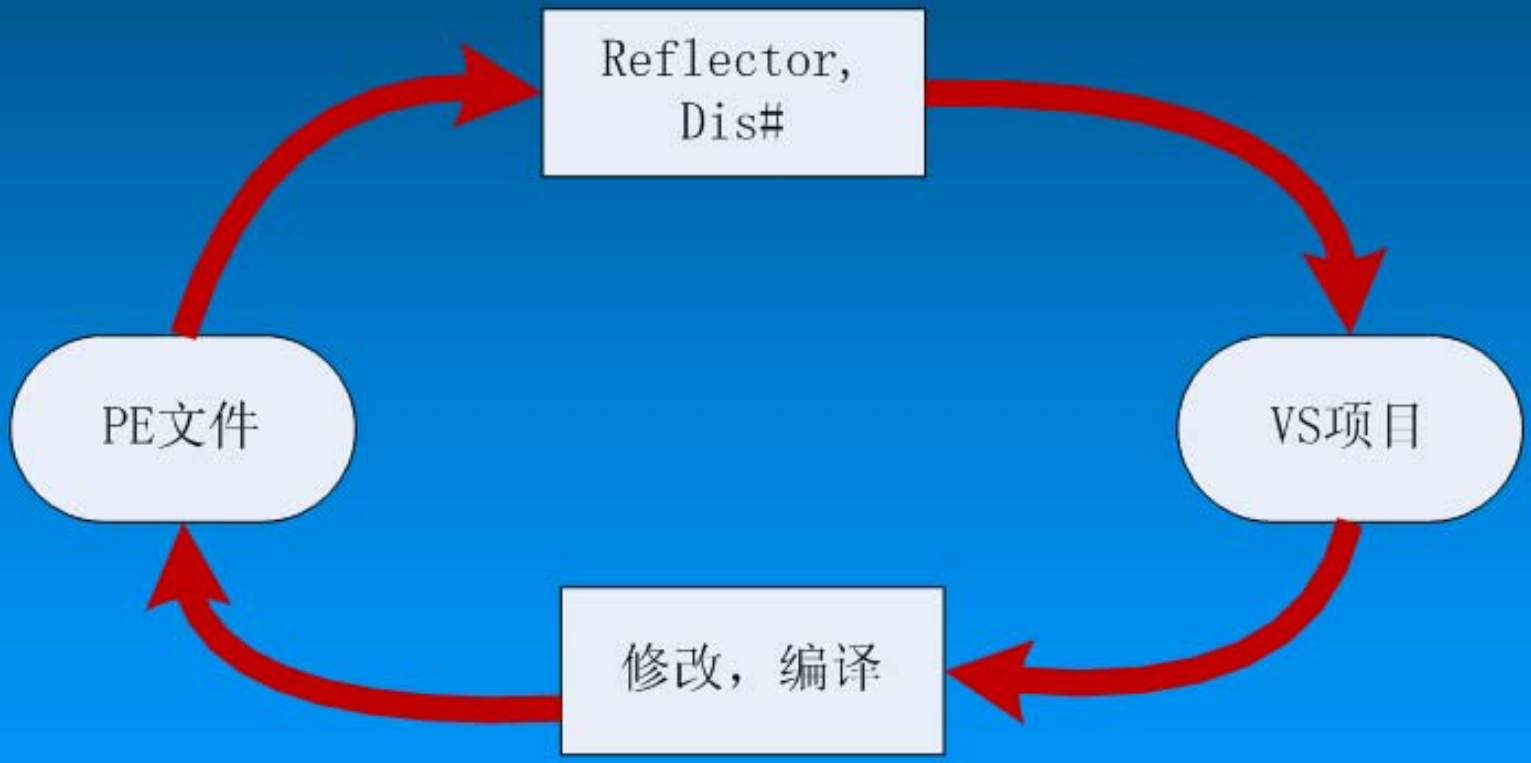
利用元数据，可以做些什么？

Round-Tripping



利用元数据，可以做些什么？

High Leveil Round-Tripping



1. 为什么要保护.NET程序

- 保护授权、注册机制
- 保护核心代码
- 保证程序的完整性和正确性
- 其它...

2. .NET程序保护方式现状

- 早期：　保护方式单一，保护强度较低
- 现在：　多种保护方式相结合；
  保护强度大；
  兼容性、执行效率可选择；
  整合了网络验证、授权。

主要保护手段

- 强名称
- 名称混淆
- 流程混淆
- 加密壳
- 授权系统
- 算法保护
- 虚拟机

非主流：打包、反调试、编译为本地代码...

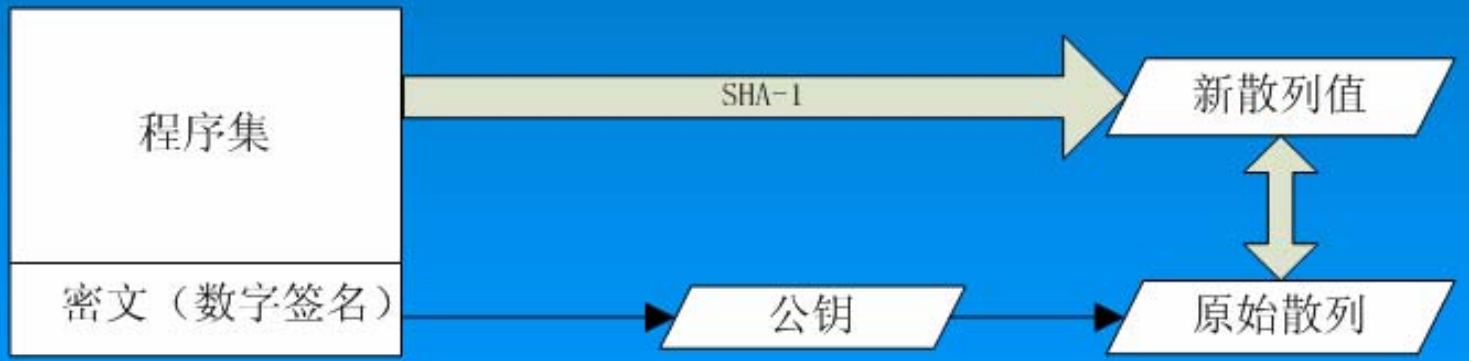
2.1 强名称（Strong Name）

- .NET中区分（或者说标识）程序集仍采用名称，但名称则分为强弱两类。弱名称包括三个要素：程序集名称、版本号和地域信息。强名称则是在弱名称的基础上再添加一个要素：数字签名。

强名称的签署与验证

Sign

Validate



强名称保护的应用

关键：How to use it

- 程序集自身的签署
- 引用被签署的程序集
- 代码与强名称的结合

2.2 名称混淆（Name Obfuscat

- 混淆的对象：#Strings流中以UTF8格式保存的各类元数据的名称

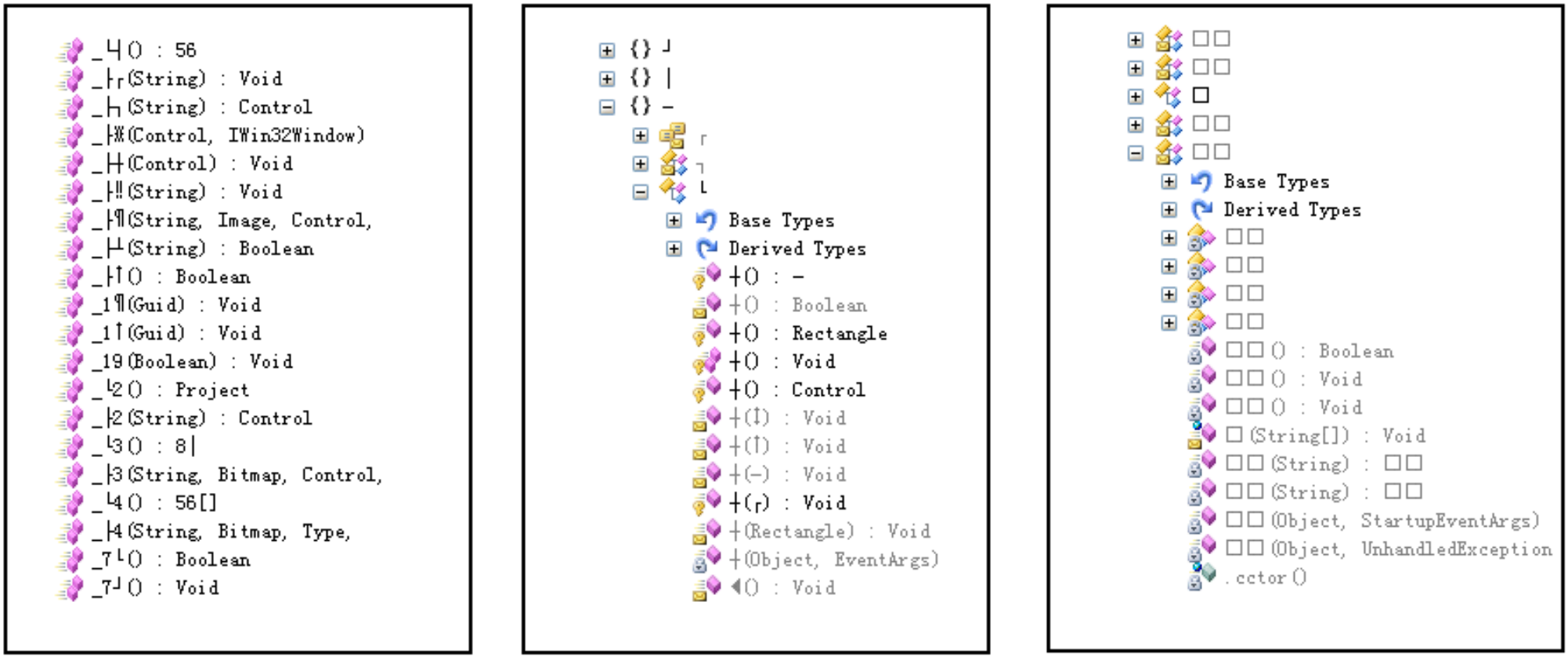
Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Ascii
00000400	74	68	45	76	65	6E	74	73	56	61	6C	75	65	00	54	65	thEventsValue.Te
00000410	78	74	42	6F	78	00	5F	54	65	78	74	42	6F	78	31	00	xtBox._TextBox1.
00000420	67	65	74	5F	54	65	78	74	42	6F	78	31	00	73	65	74	get_TextBox1.set
00000430	5F	54	65	78	74	42	6F	78	31	00	4C	61	62	65	6C	00	_TextBox1.Label.
00000440	5F	4C	61	62	65	6C	31	00	67	65	74	5F	4C	61	62	65	_Label1.get_Labe
00000450	6C	31	00	73	65	74	5F	4C	61	62	65	6C	31	00	42	75	l1.set_Label1.Bu
00000460	74	74	6F	6E	00	5F	42	75	74	74	6F	6E	31	00	67	65	tton._Button1.ge
00000470	74	5F	42	75	74	74	6F	6E	31	00	73	65	74	5F	42	75	t_Button1.set_Bu
00000480	74	74	6F	6E	31	00	5F	42	75	74	74	6F	6E	32	00	67	tton1._Button2.g
00000490	65	74	5F	42	75	74	74	6F	6E	32	00	73	65	74	5F	42	et_Button2.set_B
000004A0	75	74	74	6F	6E	32	00	5F	4C	61	62	65	6C	32	00	67	utton2._Label2.g
000004B0	65	74	5F	4C	61	62	65	6C	32	00	73	65	74	5F	4C	61	et_Label2.set_La
000004C0	62	65	6C	32	00	4B	65	79	00	43	6F	64	65	43	72	79	bel2.Key.CodeCry
000004D0	70	74	00	74	65	78	74	00	45	76	65	6E	74	41	72	67	pt.text.EventArg
000004E0	73	[illegible]					6F	6E	31	[illegible]							

名称混淆的目的

- 隐藏原作者编程意图，增加阅读难度；
- 扰乱反编译软件的代码显示功能；
- 增大通过反编译手段修改原程序的难度；
- 阻止代码轻易被复用。

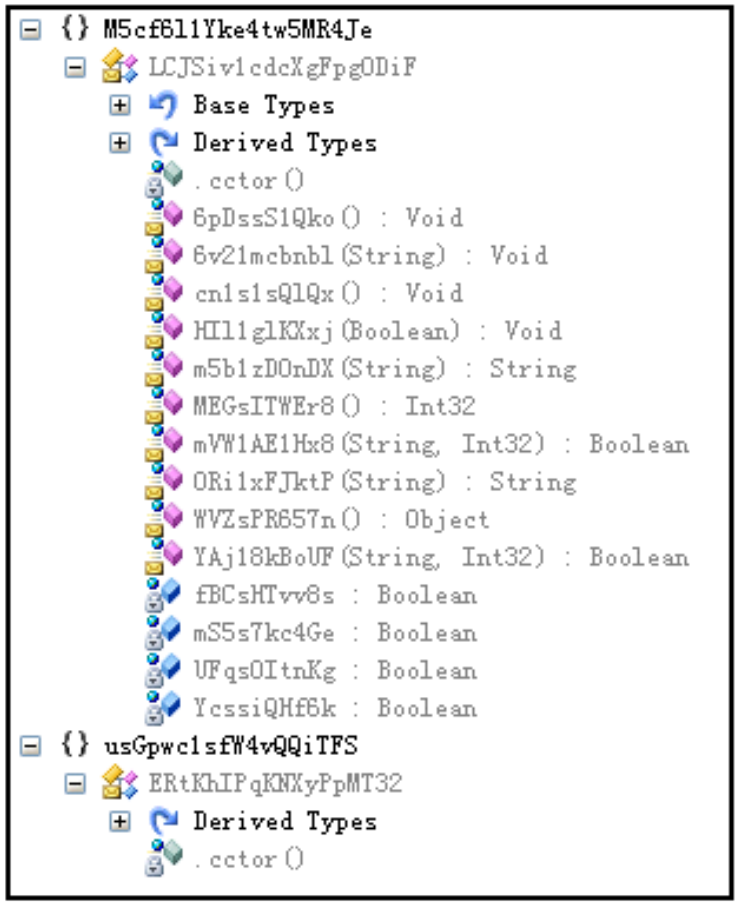
常见名称混淆形式

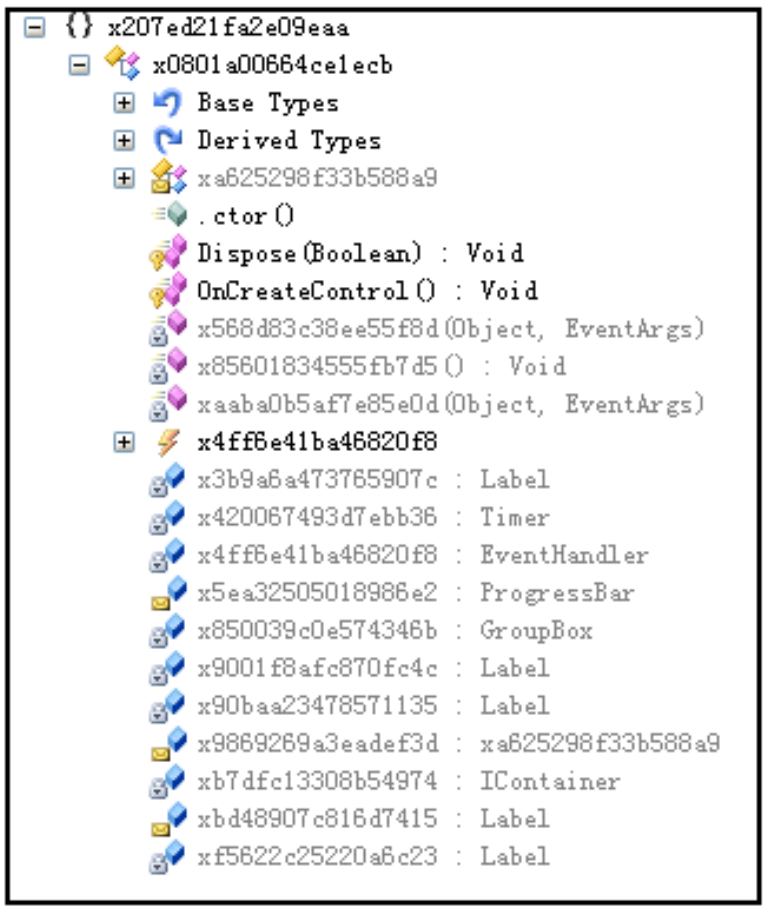
乱码



常见名称混淆形式

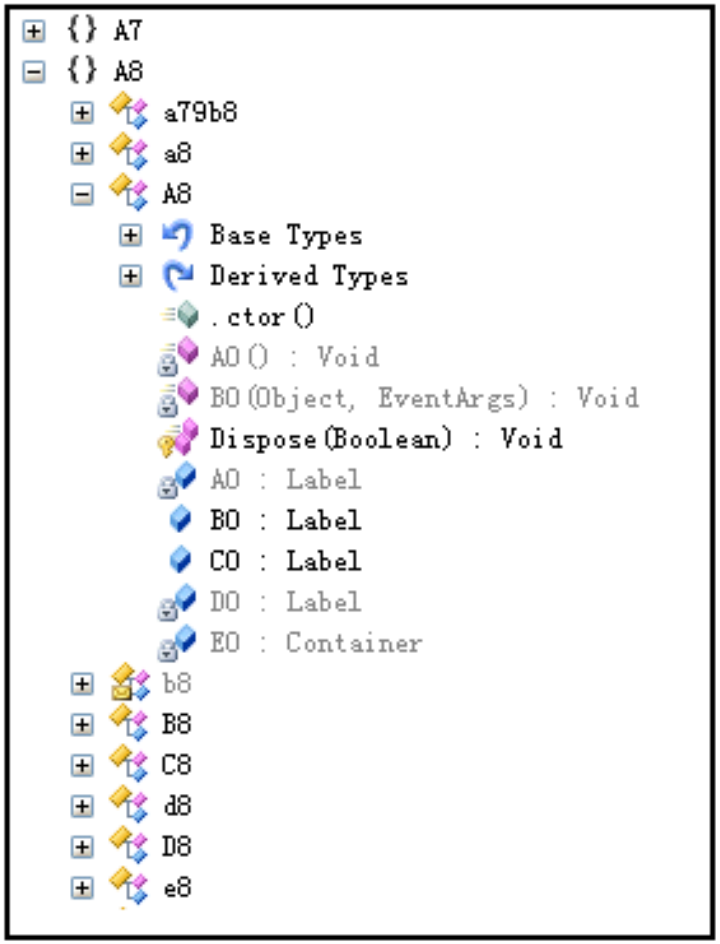
无明显规律字符串

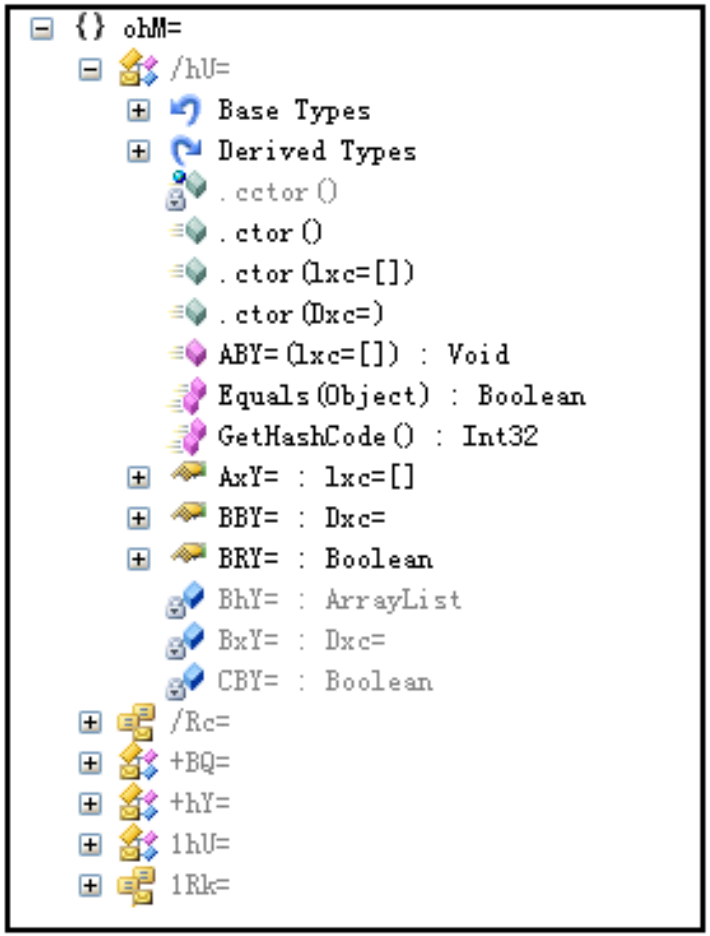




常见名称混淆形式

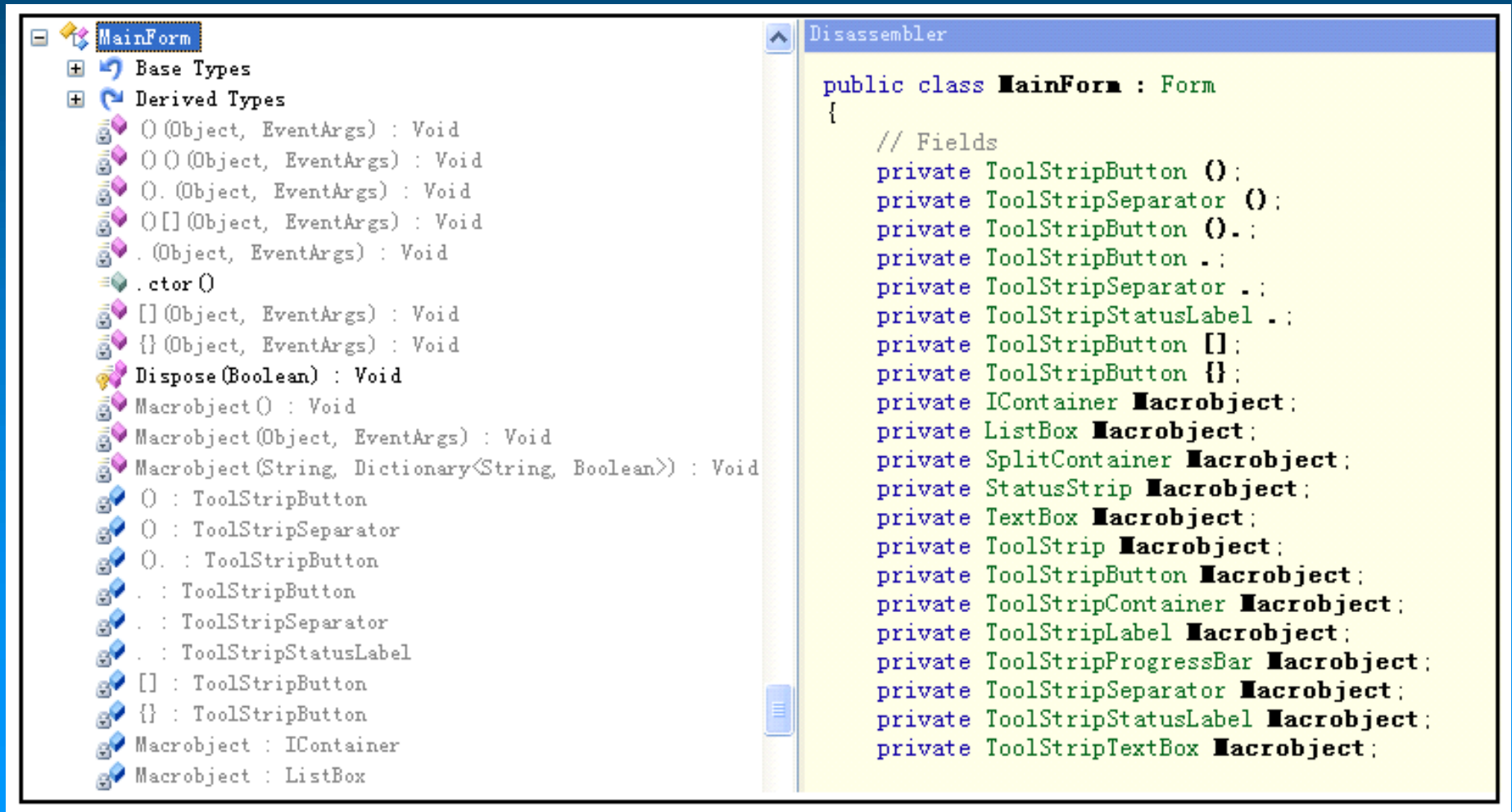
有明显规律字符串





常见名称混淆形式

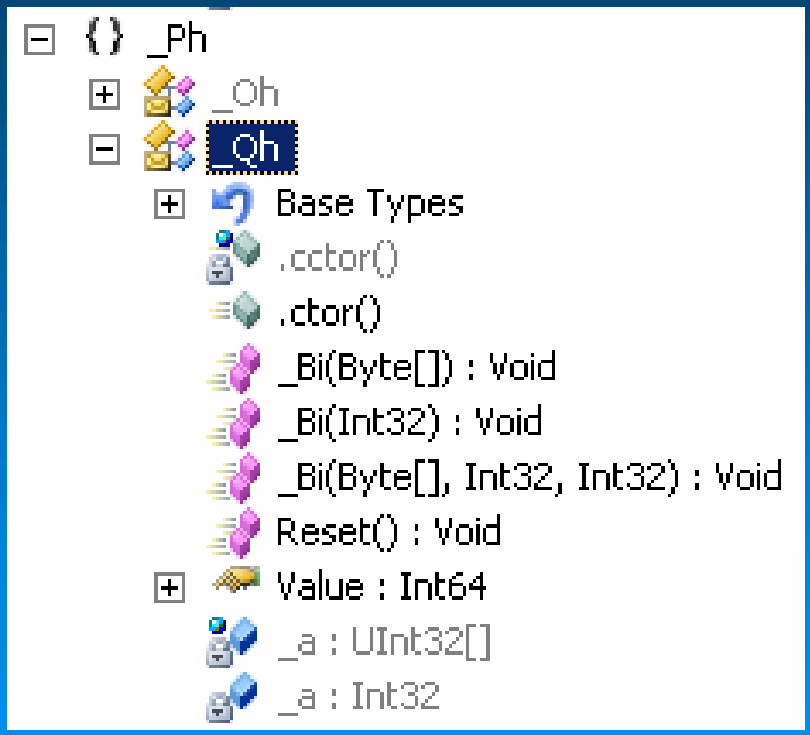
引发歧义的字符串

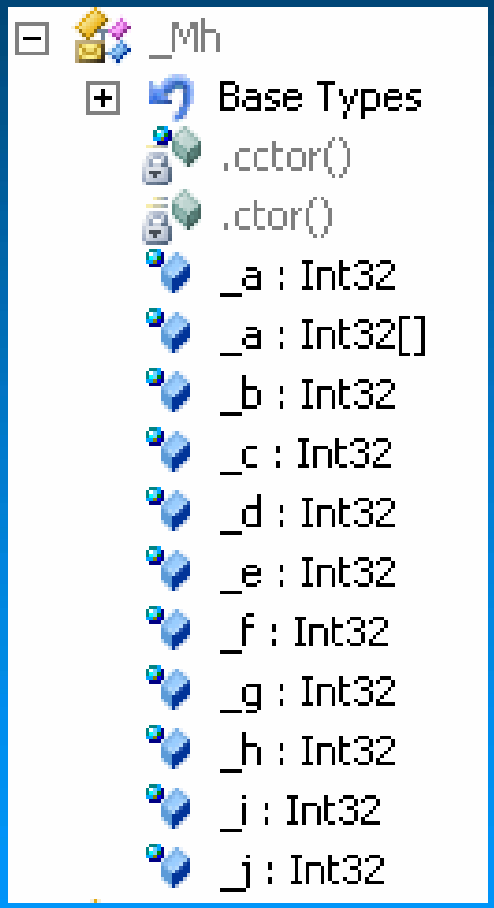


考察名称混淆的强度

- 字符串的可阅读性；
- 字符串的随机性；
- 名称的重载性。

考察名称混淆的强度





2.3 流程混淆（Control-Flow Obfuscation）

- What is it?

通过程序流程（如跳转、条件跳转、循环结构）的综合处理，得到可正确执行的代码，同时扰乱反编译器的语义分析，使得破解者难以进行分析。

流程混淆的保护效果

Disassembler

```
public static string ┐ (int)
{
    // This item is obfuscated and can not be translated.
}
```

reflector

dis#

```
[System.STAThread]
public static void   ()
{
    // decompiler error
}
```

常见的流程混淆保护方式

- 基于跳转指令
  - 代码块易位
  - 连续跳转
  - 跳转表
  - 逻辑跳转
  - 基于switch指令的跳转
- 基于语法的混淆
  - 让堆栈溢出
  - 利用高级语言不支持的语法
  - 利用反汇编引擎的缺陷
  - 插入无效的编码
- etc.

2.4 壳保护

- 壳在Win32下发展的很成熟，在.NET中相对属于新兴事物
- 大体上分两类
  - 程序集整体保护
  - 基于每个方法的保护
  - 另类：基于类的方法的保护

程序集整体保护的壳

What is Whole Assembly Protection?

- 程序集在某一时刻被完整地在内存中解密，因此可轻易地使用自动脱壳工具转存，或是使用十六进制工具从内存中抓取数据，得到解密后的IL和元数据，并使用反编译软件进行分析。

程序集整体保护的壳

- 传统Win32的壳
- 纯.NET实现的压缩壳
  - bsp
  - AdeptCompressor(原Sixxpack)
  - .NETZ(.NET Executables Compressor)
- 其它一些壳的早期版本

Native Loader

基于每个方法的保护的壳

基于每个类的方法的保护

- 当某个类的某个方法运行时，解密该类中的所有方法

基于每个方法的保护的壳

What is Per-Method Protection?

- 只在需要执行某个方法时才将其编译为本地代码。如果壳保护可以以单个方法为保护单位，在需要JIT时才将该方法的代码解密，这样任何时刻都不会在内存中出现完成的程序集源代码。

壳保护的特点

- 托管壳、Native Loader
  - 兼容性好
  - 难以实现大强度
- 挂钩.NET内核的壳
  - 可实现随机算法
  - 保护强度大
  - 最大的缺陷：兼容性

2.5 许可证保护

按实现的原理分

- 基于.NET的许可证机制
- 完全重新编写

❖按保护的层次分

- 整体级
- 组件级

典型的.NET许可证保护验证流程

2.6 辅助保护手段

- 用户字符串编码
- 给程序集添加错误元数据
- 打包
- 特殊的.NET属性
- 利用系统特性
- 反调试（托管，非托管）

用户字符串编码(User String Enco

Why?

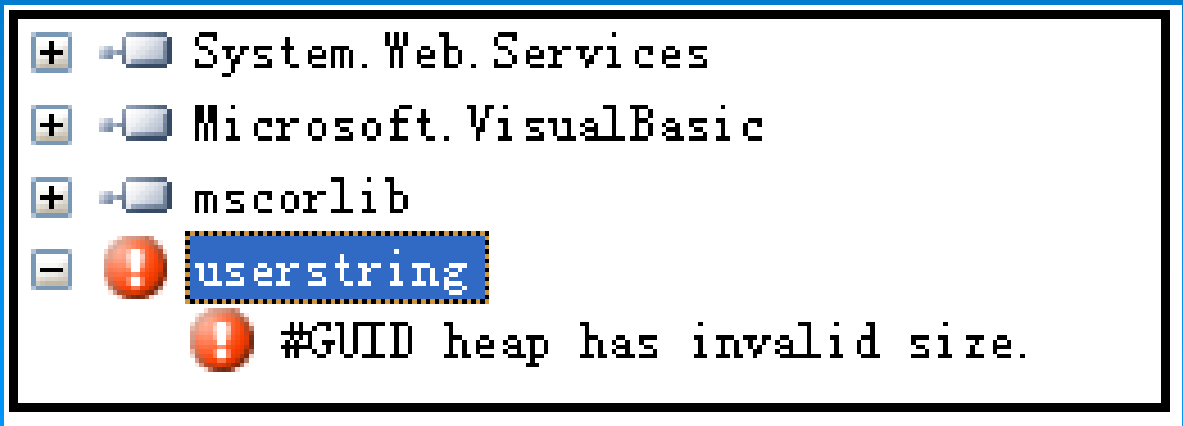
- 用户字符串中包含太多敏感信息

How?

- 直接编码
- 利用强名称编码

添加错误的元数据

常见的如：#GUID大小错误



添加错误的元数据

- 如何利用？
  - #GUID堆的定义：The Guid heap is an array of GUIDs, each 16 bytes wide. Its first element is numbered 1, its second 2, and so on。
  - 根据这个定义，#GUID堆的大小只可能是16的倍数，如果不符合这个原则，则元数据定义错误，

打包

- 当一个程序集含有多个模块时，可以通过打包的方式整合为一个单独的可执行文件。
- 打包可以达到许多效果，比如方便程序发布，对原程序进行加密，或通过压缩减小程序体积。

Whole Assembly Protection

特殊的.NET属性

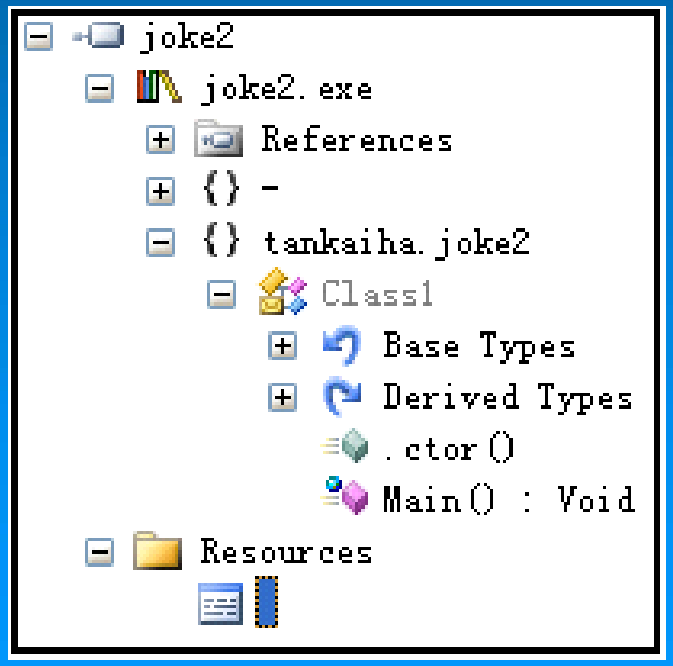
- 与调试相关的属性
  - [DebuggerHidden]
  - [DebuggerNonUserCode]
  - [DebuggerStepThroughAttribute]

特殊的.NET属性

- 与反编译相关的属性
  - SuppressIldasmAttribute

利用系统特性

- 托管资源的名称不合法，不能导出为文件



反调试

- 托管方式
  - System.Diagnostics.Debugger.IsAttached
- 非托管方式
  - 利用Win32下各种反调试技术
  - 调用机制：p/invoke，C++/CLI混合编译

2.7 其它保护方式

这里“其它”的定义：

- 应用范围不广
- 暂时没有成为主流，但很有希望
- 很重要

动态方法委托调用

- 原理：
  - 委托（Delegate）
  - 动态方法（Dynamic Method）

```
MD5CryptoServiceProvider provider = new MD5CryptoServiceProvider();
byte[] bytes = Encoding.ASCII.GetBytes(Field_06.Method_00(Method_04()));
byte[] inputBuffer = Encoding.ASCII.GetBytes(Field_06.Method_00(text1));
provider.Initialize();
```

基本原理

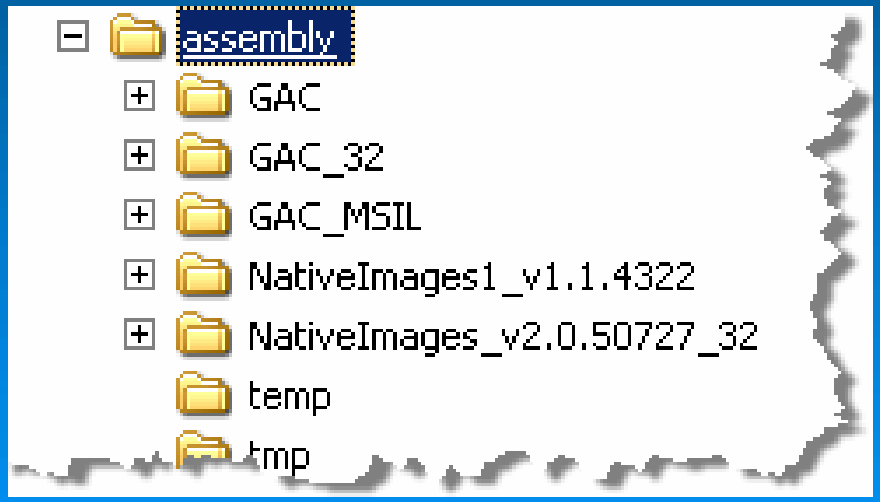

```
static r()
{
    r = new byte[] {
        0x4c, 0x43, 80, 0xa5, 0xd8, 0xa7, 8, 0x1b, 0x56, 0x25, 0x2d,
        0xc4, 0x36, 0xfc, 0xd8, 0xe4, 2, 0xb0, 0x4b, 0xf8, 0xb0, 130,
        0x79, 0x61, 0x3e, 0x16, 0xec, 0xc0, 15, 0x62, 0x8e, 0x41, 0
        0x87, 0x2a, 0xe5, 0xc0, 0xf8, 0x37, 0x9a, 0xf4, 0xd1, 0xfb, 0
```

```
ProtectorRuntime.Init();
r = (r) EncryptedMethodHelper.Deserialize(new MemoryStream(r)).CreateDelegate(typeof(r));
this.r();
```

```
MD5CryptoServiceProvider provider = new MD5CryptoServiceProvider();
byte[] bytes = Encoding.ASCII.GetBytes(r(r()));
byte[] inputBuffer = Encoding.ASCII.GetBytes(r(text1));
```

编译为本地代码

- 原理：利用框架的GAC机制



编译为本地代码

VTableFixups Size	000[illegible]	Dword	000[illegible]
ExportAddressTableJumps RVA	00001038	Dword	00000000
ExportAddressTableJumps Size	0000103C	Dword	00000000
ManagedNativeHeader RVA	00001040	Dword	00002050
ManagedNativeHeader Size	00001044	Dword	000000B0

Original MetaData RVA	000010A8	Dword	00076000
Original MetaData Size	000010AC	Dword	00012BF8
Original MSIL Code RVA	000010B0	Dword	0017E8E0
Original MSIL Code Size	000010B4	Dword	0000D3ED

Remove Original Metadata and MSIL

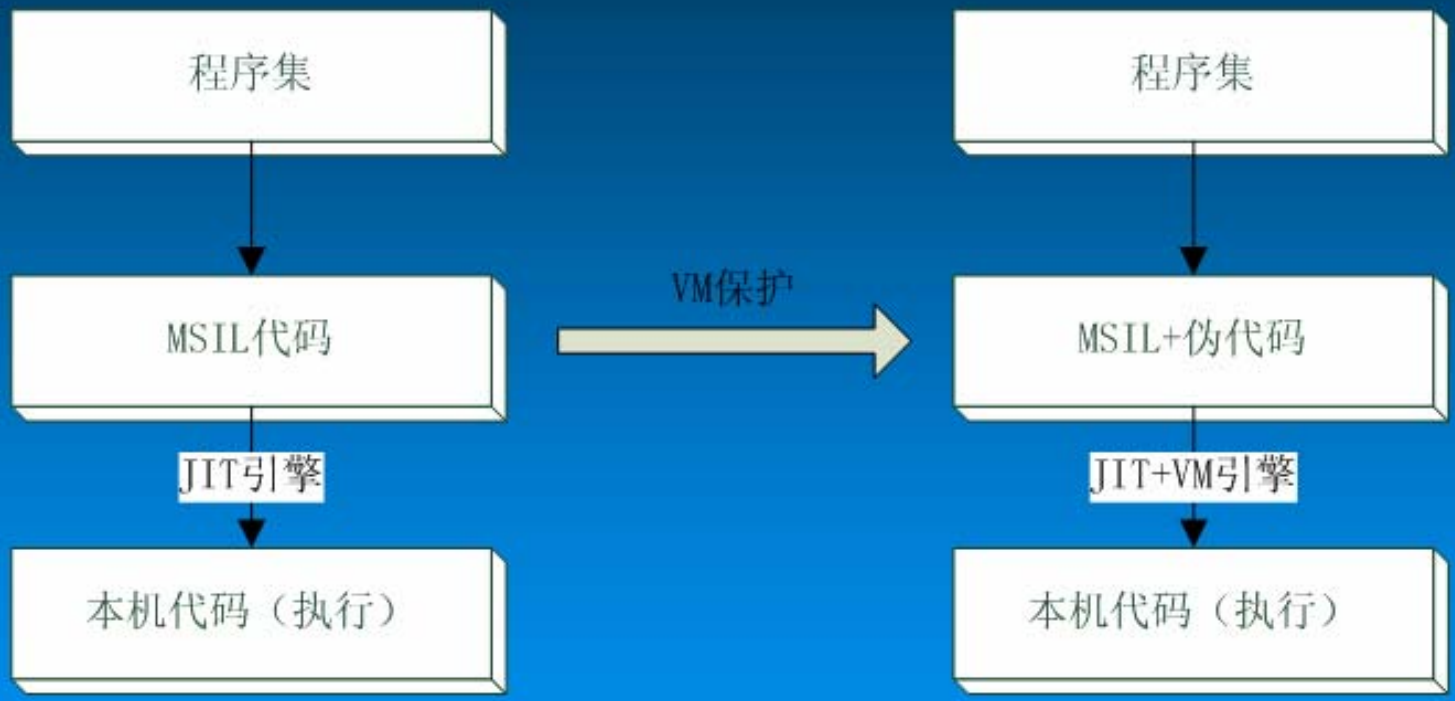
虚拟机保护

- 按保护的级别分
  - 整体级
  - 代码级
- 按虚拟的程度分
  - 完全替换MSIL指令集
  - 部分替换MSIL指令集（较难实现）
- 按实现的方式分
  - 托管
  - 非托管

整体级的虚拟机保护

- 实现原理：模拟文件系统与注册表
- 主要用途：程序的打包与发布
- 保护强度：类似于程序集整体保护，保护强度低。

非托管的虚拟机



- 优点：执行效率高，保护强度大
- 缺点：兼容性

纯托管的虚拟机

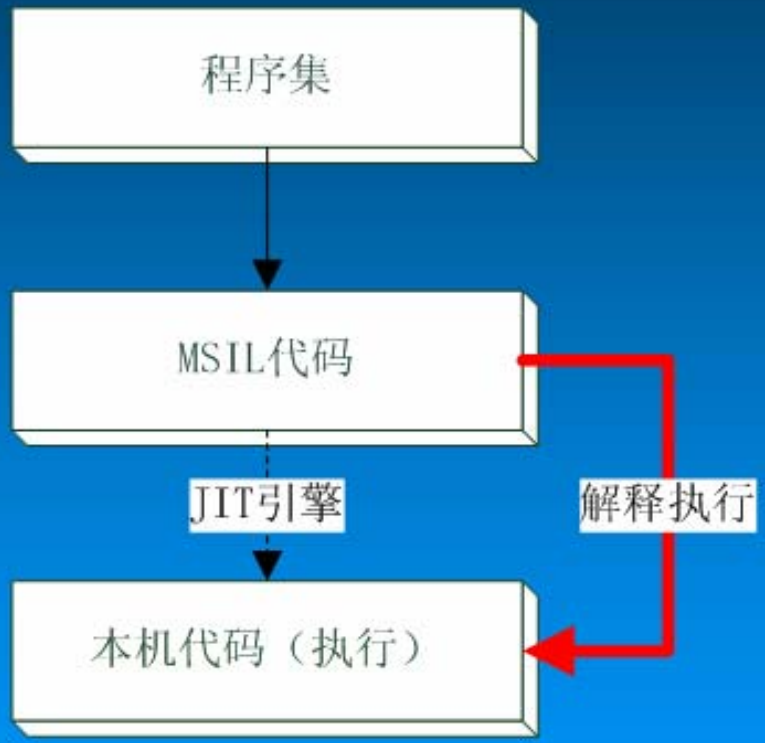

```
private void Button1_Click(object sender, EventArgs e)
{
    try
    {
        FileStream stream = new FileStream("key.dat", FileMode.Open, FileAccess.Read);
        StreamReader reader = new StreamReader(stream);
        string text = reader.ReadToEnd();
        byte[] bytes = Encoding.Unicode.GetBytes(this.TextBox1.Text);
        SHA512 sha = new SHA512Managed();
        sha.ComputeHash(bytes);
        if (Operators.ConditionalCompareObjectEqual(this.CodeCrypt(text), Convert.ToBase64String(sha.Hash), true
```

before protection

```
private void Button1_Click(object sender, EventArgs e)
{
    object[] args = new object[] { sender, e };
    SLMRuntime.SVMExecMethod(this, "60db2c9f00d34d26b3cbb2e0a6501adc", args);
}
```

after protection

纯托管的虚拟机



最大缺点：
执行效率低

增大虚拟机保护的强度

增大虚拟机保护强度的手段：

- 伪指令的随机化
- 加密算法的随机化
- 其它辅助反分析手段

3 .NET保护的发展趋势

- .NET软件保护技术现在是，以后一段时间仍将是.NET领域中发展最快的方向之一。

- 用户的需求决定发展方向
  - 个人用户：简单有效的注册机制
  - 企业用户：兼容性，管理性

3 .NET保护的发展趋势

代码保护技术的发展方向

- 名称与流程混淆
  - 在与反混淆技术的斗争中前进
- 加密壳
  - 不断完善兼容性
  - 不断增大强度（接管越来越多的JIT内核）
- 虚拟机
  - 更完善的伪指令集
  - 更高的执行速度

3 .NET保护的发展趋势

- 保护层次的飞跃
  - 现在：纯代码层次的保护
  - 将来：网络、服务、保护的结合。
    将程序保护作为一个整体，而不仅仅是保护代码。（MSLP）

That's all.

Thank you!