火龙果•整理
uml.org.cn

单海波

.NET程序加解密技术

2008中国软件安全峰会
2008 China Software Security Summit

火龙果•整理
uml.org.cn

主要内容

- 一、.NET程序加解密技术基础
- 二、主流的.NET程序加解密技术
- 三、对.NET加解密技术的一些看法

火龙果•整理
uml.org.cn

一、.NET程序加解密技术基础

- 1、什么是.NET
  - .NET Framework 是支持生成和运行下一代应用程序和 XML Web Services 的内部 Windows 组件。
  - 符合ECMA-335（欧洲计算机制造商协会）规定的公共语言基础（CLI）的平台:MONO、SSCLI等。

火龙果•整理
uml.org.cn

.NET的目标：

- 一个一致的面向对象的编程环境。
- 一个将软件部署和版本控制冲突最小化的代码执行环境。
- 一个可提高代码执行安全性的代码执行环境。
- 一个可消除脚本环境或解释环境的性能问题的代码执行环境。
- 使开发人员的经验在面对类型大不相同的应用程序时保持一致。
- 确保基于 .NET Framework 的代码可与任何其他代码集成。

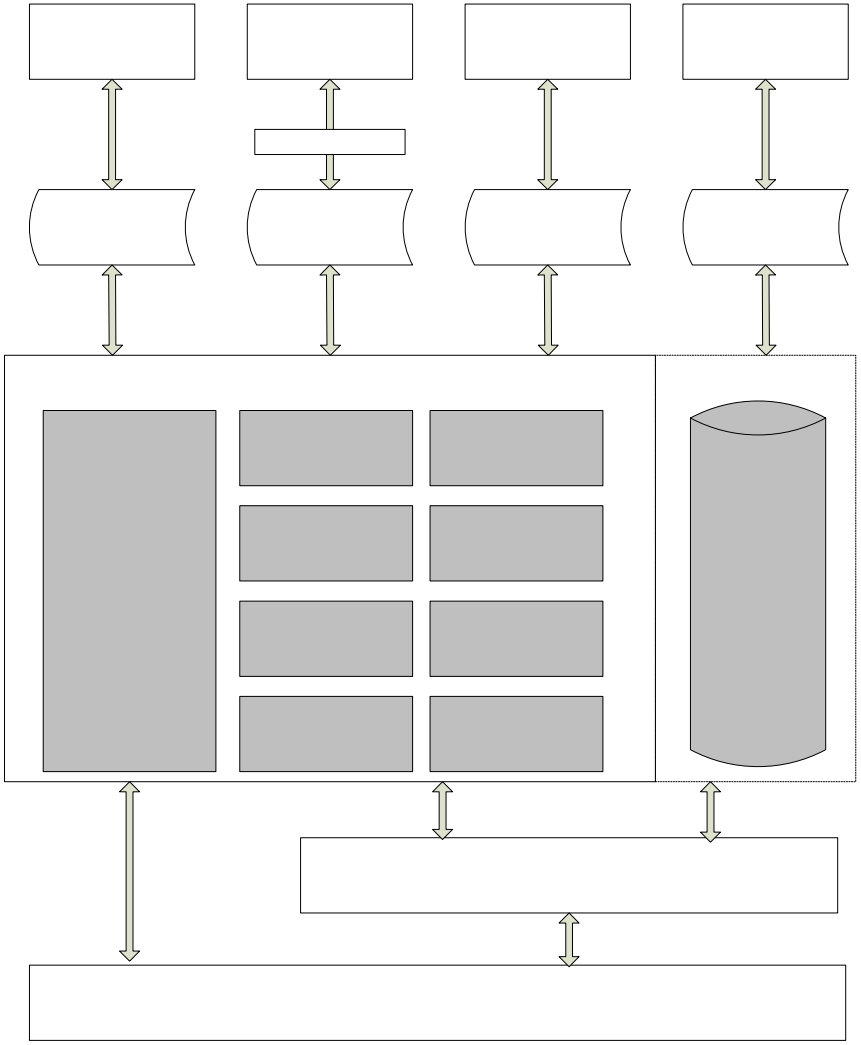
火龙果•整理
uml.org.cn

- 从加解密的角度看，.NET就是构建于操作系统（Windows）之上的虚拟机。

- 包含两个部分
  - CLR（通用语言运行时，也就是.NET程序的执行环境）
  - .NET类库

火龙果•整理
uml.org.cn

Windows系统上的.NET框架结构



C#

VB.net

msil/元数

程序集

程序集

火龙果•整理
uml.org.cn

2、加解密所关注的.NET带来的改变

❖(1)PE结构上的改变

- 普通PE文件存储x86/x64机器指令

```
004C3043   50                push eax
004C3044   64:8925 00000000  mov dword ptr fs:[0],esp
004C304B   83EC 58           sub esp,58
004C304E   53                push ebx
004C304F   56                push esi
004C3050   57                push edi
004C3051   8965 E8           mov dword ptr ss:[ebp-18],esp
004C3054   FF15 20235000     call dword ptr ds:[<&KERNEL32.GetVer kernel32.GetVersion
004C305A   33D2              xor edx,edx
004C305C   8AD4              mov dl,ah
004C305E   8915              mov dword ptr ds:[56C86C],edx
```

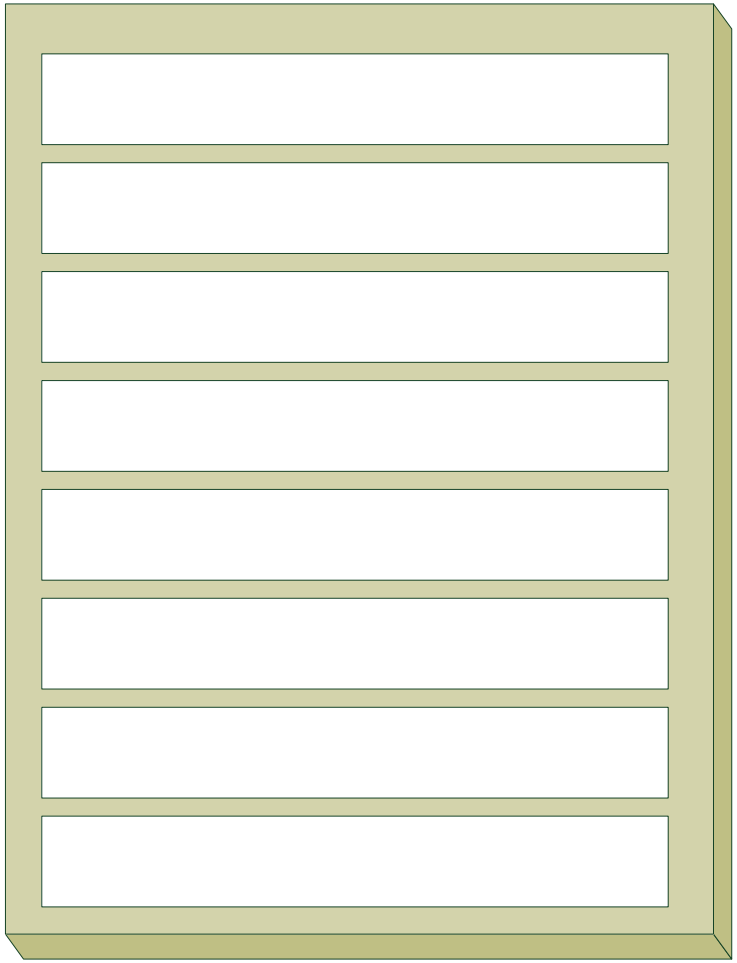
火龙果•整理
uml.org.cn

- .NET下PE文件（程序集）中保存的是元数据和MSIL，但不限于此。

```
.method public hidebysig static void  Main() cil managed
{
  .entrypoint
  // Code size       38 (0x26)
  .maxstack  8
  IL_0000:  br          IL_0007
  IL_0005:  pop
  IL_0006:  nop
  IL_0007:  br          IL_0017
  IL_000c:  ldstr       "hello .NET encryption and decryption"
  IL_0011:  br          IL_001c
  IL_0016:  ret
  IL_0017:  br          IL_000c
```

火龙果•整理
uml.org.cn

.NET下PE文件.text节的典型结构



输

Common Langu

msil代码、异常

火龙果•整理
uml.org.cn

- 元数据：描述数据的数据。

- 在.NET中，元数据：
  - a) 本身就是一段数据。
  - b) 描述了一个.NET程序的全部信息。（.NET程序是自描述的）

火龙果•整理
uml.org.cn

程序集的这种自描述性质使得反编译时信息更加丰富，结合MSIL的特性，使得反编译结果近似原代码。

```
sub_511FD0 proc near

var_10= dword ptr -10h
var_C= dword ptr -0Ch
var_4= dword ptr -4
arg_0= dword ptr  4
arg_4= dword ptr  8

push    0FFFFFFFFh
push    offset SEH_511FD0
mov     eax, large fs:0
push    eax
mov     large fs:0, esp
push    ecx
mov     eax, [esp+10h+arg_0
mov     [esp+10h+var_10], e
xor     edx, edx
cmp     eax, edx
mov     [esp+10h+var_4], 0
```

```
_Keygenme1.Net
RE_Keygenme1.Net.exe
References
AoRE_Keygenme1.Net
Form1
  Base Types
  Derived Types
  .ctor()
  Button1_Click(Objec
  Button2_Click(Objec
  CodeCrypt(String) :
  Dispose(Boolean) : V
  InitializeComponent(
  Button1 : Button
  Button2 : Button
  Label1 : Label
  Label2 : Label
  PictureBox1 : Picture
  TextBox1 : TextBox
  _Button1 : Button
  _Button2 : Button
  _Label1 : Label
```

```
Disassembler
private void Button1_Click(object sender, EventArg
{
    try
    {
        FileStream stream = new FileStream("key.dat
        StreamReader reader = new StreamReader(stre
        string text = reader.ReadToEnd();
        byte[] bytes = Encoding.Unicode.GetBytes(this.
        SHA512 sha = new SHA512Managed();
        sha.ComputeHash(bytes);
        if (Operators.ConditionalCompareObjectEqual(th
        {
            Interaction.MsgBox("Good job, make a keyma
        }
        else
        {
            Interaction.MsgBox("Try again, it is very simp
        }
        reader.Close();
        stream.Close();
    }
```

火龙果•整理
uml.org.cn

元数据的基本结构

- 元数据是以数据流（stream）的形式保存在文件中，这些流被分别命名，并有各自不同的含义。在.NET中，保存元数据的流被分为两大类：堆（heap）和表（table）。因此，流是一个统称，而堆和表则是流的两种存储形式。

火龙果•整理
uml.org.cn

.NET中的六个元数据堆

名称	含义
#Strings	UTF8格式的字符串堆，包含各种元数据的名称（比如类名、方法名、成员名、参数名等等）。堆的首部总有一个0作为空字符串，各字符串以0表示结尾。CLR中这些名称的最大长度是1024。
#Blob	二进制数据堆，存储程序中的非字符串信息，比如常量值、方法的签名、强名称等。每个数据的长度由该数据的前1至3位决定：0表示长度1字节，10表示长度2字节，110表示长度4字节。
#GUID	存储所有的全局唯一标识（Global Unique Identifier）。
#US	以Unicode格式存放的IL代码中使用的用户字符串（User String），比如ldstr调用的字符串。
#~	元数据中最重要的堆，几乎所有的元数据信息都以表的形式保存于此。每个.NET程序都必须包含。
#-	#~的未压缩（或称为未优化）存储，不常见。

火龙果•整理
uml.org.cn

元数据表

00 – Module	01 – TypeRef	02 - TypeDef
03 – FieldPtr	04 – Field	05 – MethodPtr
06 – MethodDef	07 – ParamPtr	08 - Param
09 – InterfaceImpl	10 – MemberRef	11 - Constant
12 – CustomAttribute	13 – FieldMarshal	14 - DeclSecurity
15 – ClassLayout	16 – FieldLayout	17 - StandAloneSig
18 – EventMap	19 – EventPtr	20 – Event
21 – PropertyMap	22 – PropertyPtr	23 – Property
24 – MethodSemantics	25 – MethodImpl	26 – ModuleRef
27 – TypeSpec	28 – ImplMap	29 – FieldRVA
30 – ENCLog	31 – ENCMap	32 – Assembly
33 – AssemblyProcessor	34 – AssemblyOS	35 – AssemblyRef
36 – AssemblyRefProcessor	37 – AssemblyRefOS	38 - File
39 – ExportedType	40 – ManifestResource	41 – NestedClass
42 – *GenericParam*	43 – *MethodSpec*	44 - *GenericParamConstraint*

火龙果•整理
uml.org.cn

2、加解密所关注的.NET带来的改变

❖(2)统一了编程语言

- C#
- VB.Net
- C++/CLI
- 其它

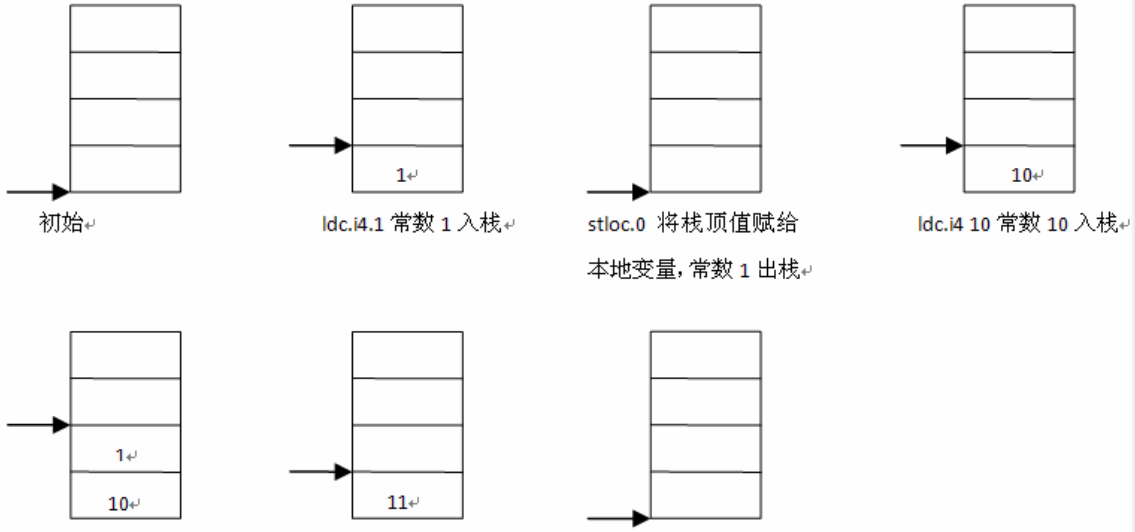
```
.method private instance void  InitializeComponent() cil manag
{
  .custom instance void [mscorlib]System.Diagnostics.DebuggerS
  // Code size       944 (0x3b0)
  .maxstack  4
  .locals init (class [System]System.ComponentModel.Componentk
           valuetype [System.Drawing]System.Drawing.Point V_1,
           valuetype [System.Drawing]System.Drawing.Size V_2)
  IL_0000:  ldtoken    AoRE_Keygenme1.Net.Form1
  IL_0005:  call       class [mscorlib]System.Type [mscorlib]S
  IL_000a:  newobj     instance void [System]System.ComponentMod
  IL_000f:  stloc.0
  IL_0010:  ldarg.0
  IL_0011:  newobj     instance void [System.Windows.Forms]Syste
  IL_0016:  callvirt   instance void AoRE_Keygenme1.Net.Form1::
  IL_001b:  ldarg.0
  IL_001c:  newobj     instance void [System.Windows.Forms]Syst
  IL_0021:  callvirt   instance void AoRE_Keygenme1.Net.Form1::
  IL_0026:  ldarg.0
  IL_0027:  newobj     instance void [System.Windows.Forms]Syst
  IL_002c:  callvirt   instance void AoRE_Keygenme1.Net.Form1::
  IL_0031:  ldarg.0
  IL_0032:  newobj     instance void [System.Windows.Forms]Syste
  IL_0037:  callvirt   instance void AoRE_Keygenme1.Net.Form1::s
  IL_003c:  ldarg.0
  IL_003d:  newobj     instance void [System.Windows.Forms]Syste
  IL_0042:  callvirt   instance void AoRE_Keygenme1.Net.Form1
```

火龙果•整理
uml.org.cn

MSIL的特点

- 一种面向对象的高级语言。
- 基于堆栈的运行机制。

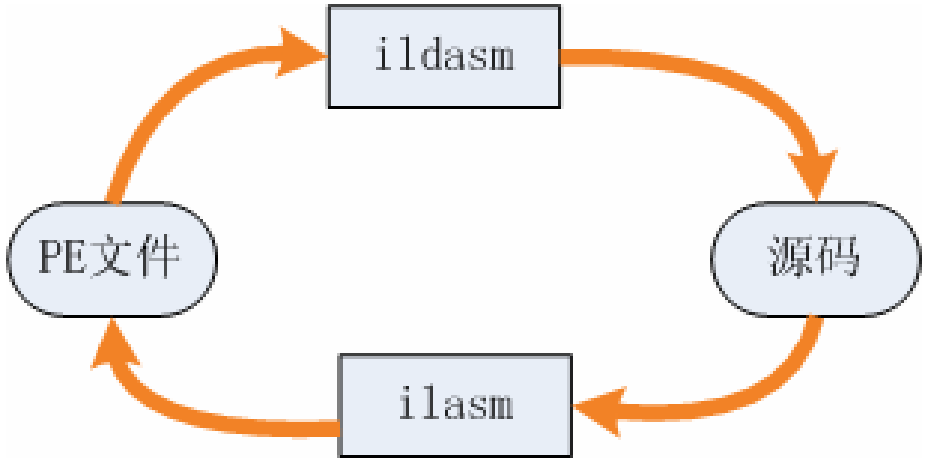
```
.locals init(int32 val)
ldc.i4.1
stloc.0
ldc.i4 10
ldloc.0
add
stsfld int32 class1::sum
```



火龙果•整理
uml.org.cn

- 编程语言统一到MSIL的优势：各语言间可以无缝地结合。

Round-Tripping



火龙果•整理
uml.org.cn

2、加解密所关注的.NET带来的改变

❖(3)程序运行方式的改变

- Windows不再直接负责程序的运行，而由.NET框架进行管理，框架中的JIT引擎负责在运行时将IL代码即时编译为本地汇编代码执行。

- .NET程序分为：托管、混合编译；
- 传统Windows程序：非托管/本地程序。

火龙果•整理
uml.org.cn

一、.NET程序加解密技术基础

2、几个重要的.NET基本概念

- 程序集（Assembly）

  .NET可执行程序的基本单元，通常见到的exe和dll文件都是程序集，由一个或多个模块及必要的清单信息（manifest）构成。

- MSIL的定义中，程序集没有扩展名，而模块是有文件扩展名的。

```
.assembly hello { }
.module hello.exe
```

火龙果•整理
uml.org.cn

- 类型（Type）

  .NET框架是面向对象的，类型则是这个概念的基本要素，最常见的类型是类（class），其次还有结构（struct）和枚举（enum）。

- 方法（Method）

  类型可以有很多成员（member），最重要的成员便是方法，因为方法中包含了关键代码。

火龙果•整理
uml.org.cn

标识（Token）

- 程序集中的每一元数据项都有一个唯一的身份证号，就是元数据的标识。标识在MSIL中的表示为0xAABBBBBB。

- 比如：Main方法为0x06000001
- 某字符串为0x70000001

火龙果•整理
uml.org.cn

签名（Signature）

- 签名就是存储在#Blob中的一段二进制数据，它的作用是描述特定元数据的性质。

- 共有6个表包含签名项：Field、Method、Property、MemberRef、StandAloneSig和TypeSpec。

火龙果•整理
uml.org.cn

- 既时编译（Just In-Time compile，JIT）
  - 这是.NET运行可执行程序的基本方式，也就是在需要运行的时候，将对应的IL代码编译为本机指令。传入JIT中的是IL代码，导出的是本机代码，所以部分加密软件通过挂钩JIT来进行IL加密，同时又保证程序正常运行。

火龙果•整理
uml.org.cn

二、主流的.NET程序加解密技术

- 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、授权管理。

火龙果•整理
uml.org.cn

- 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...

火龙果•整理
uml.org.cn

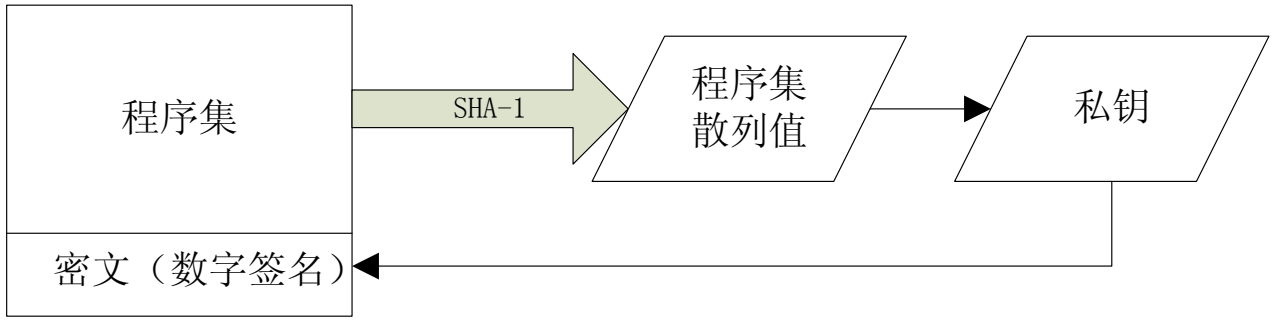
1 强名称（Strong Name）

- .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。

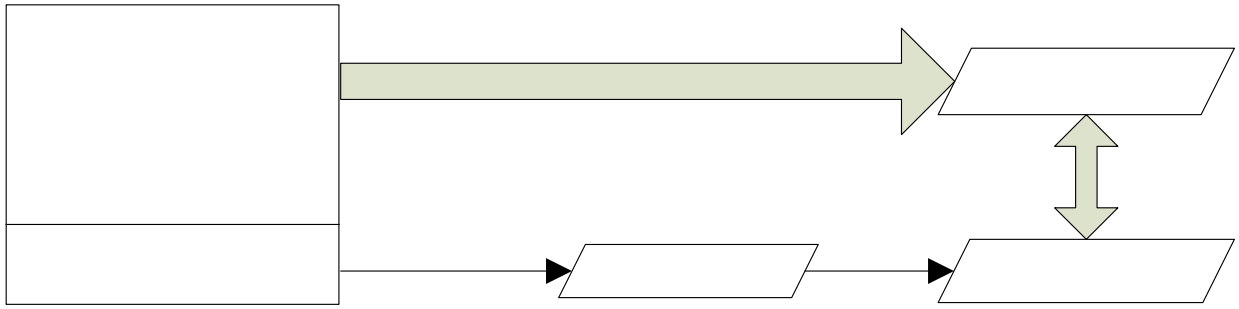
火龙果•整理
uml.org.cn

强名称的签署与验证

Sign



Validate



火龙果•整理
uml.org.cn

强名称保护的应用

保护强度取决于：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

火龙果•整理
uml.org.cn

强名称保护的一般分析方法

- 替换或去除原程序集中的强名称
- 在调用GetPublicKey或GetPublicKeyToken的地方将正确的数据填入程序中。
- Patch系统（mscorwks.dll）

火龙果•整理
uml.org.cn

2 名称混淆（Name Obfuscation）

- 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	[illegible]																[illegible]

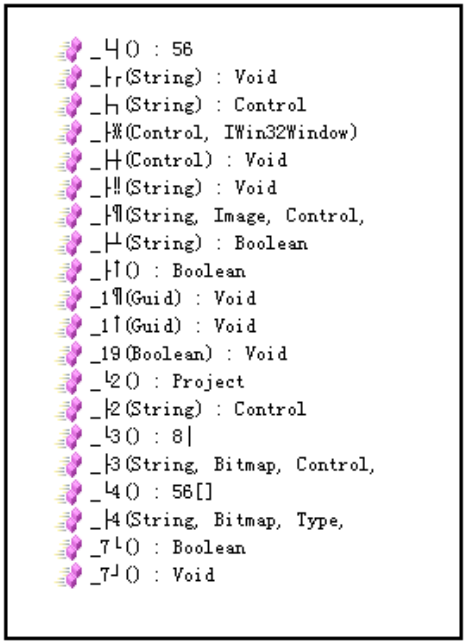
火龙果•整理
uml.org.cn

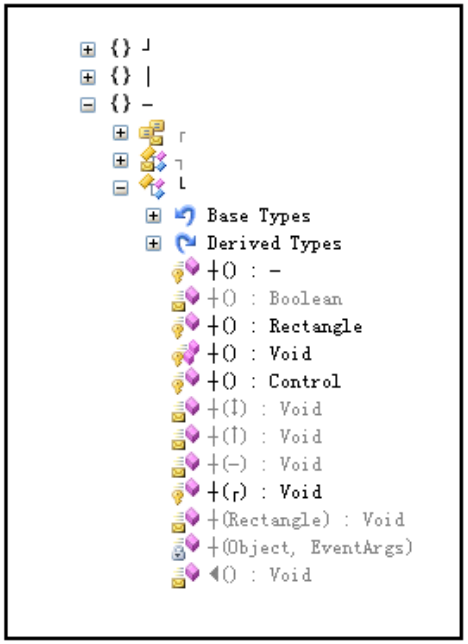
- 名称混淆的目的
  - 隐藏原作者编程意图，增加阅读难度；
  - 扰乱反编译软件的代码显示功能；
  - 增大通过反编译手段修改原程序的难度；
  - 阻止代码轻易被复用。

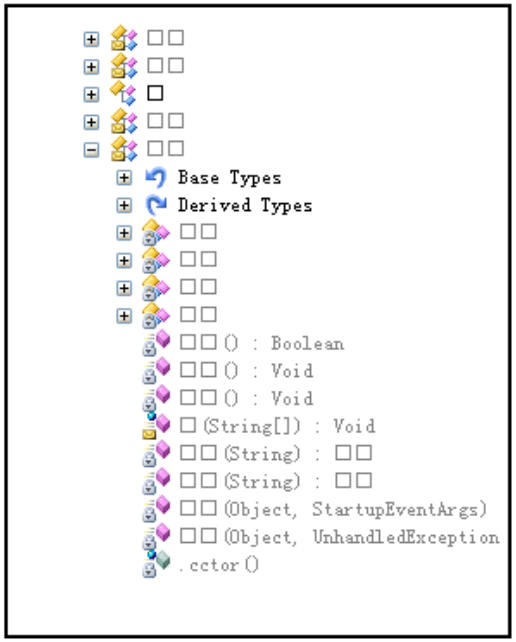
火龙果•整理
uml.org.cn

常见名称混淆形式

乱码



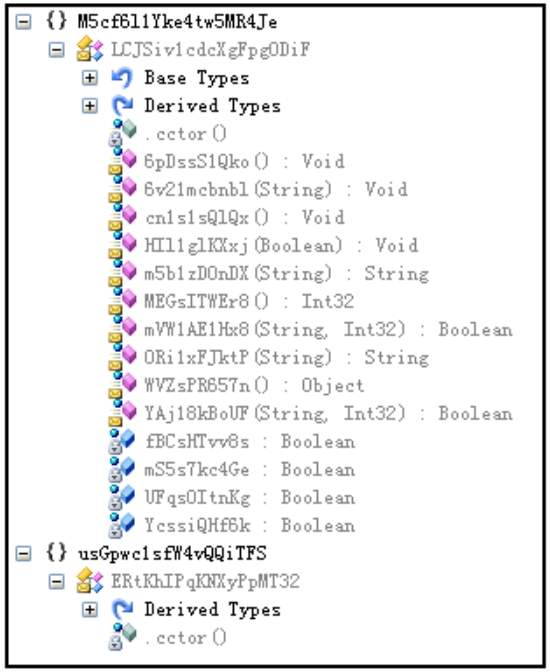


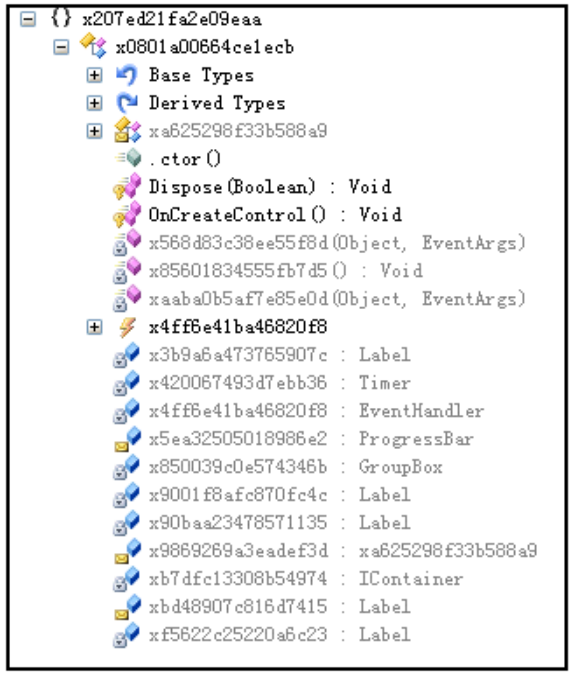


火龙果•整理
uml.org.cn

常见名称混淆形式

无明显规律字符串





火龙果•整理
uml.org.cn

常见名称混淆形式

引发歧义的字符串

```
MainForm
  Base Types
  Derived Types
  ()(Object, EventArgs) : Void
  ()()(Object, EventArgs) : Void
  ().(Object, EventArgs) : Void
  ()[](Object, EventArgs) : Void
  .(Object, EventArgs) : Void
  .ctor()
  []{}(Object, EventArgs) : Void
  {}(Object, EventArgs) : Void
  Dispose(Boolean) : Void
  Macrobject() : Void
  Macrobject(Object, EventArgs) : Void
  Macrobject(String, Dictionary<String, Boolean>) : Void
  () : ToolStripButton
  () : ToolStripSeparator
  (). : ToolStripButton
  . : ToolStripButton
  . : ToolStripSeparator
  . : ToolStripStatusLabel
  [] : ToolStripButton
  {} : ToolStripButton
  Macrobject : IContainer
  Macrobject : ListBox
```

```
Disassembler

public class MainForm : Form
{
    // Fields
    private ToolStripButton ();
    private ToolStripSeparator ();
    private ToolStripButton ().;
    private ToolStripButton .;
    private ToolStripSeparator .;
    private ToolStripStatusLabel .;
    private ToolStripButton [];
    private ToolStripButton {};
    private IContainer Macrobject;
    private ListBox Macrobject;
    private SplitContainer Macrobject;
    private StatusStrip Macrobject;
    private TextBox Macrobject;
    private ToolStrip Macrobject;
    private ToolStripButton Macrobject;
    private ToolStripContainer Macrobject;
    private ToolStripLabel Macrobject;
    private ToolStripProgressBar Macrobject;
    private ToolStripSeparator Macrobject;
    private ToolStripStatusLabel Macrobject;
    private ToolStripTextBox Macrobject;
```

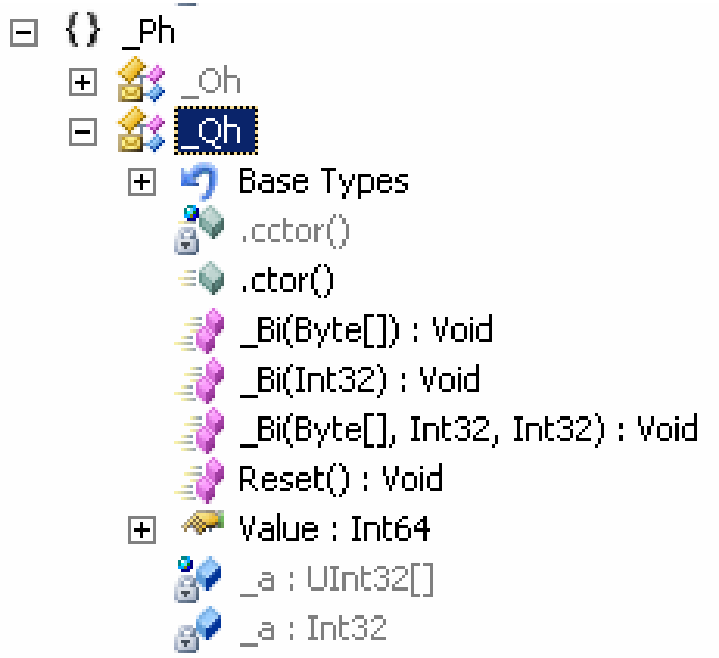
火龙果•整理
uml.org.cn

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。

火龙果•整理
uml.org.cn

考察名称混淆的强度



假设算法的关键代码在_Bi中，在反编译软件中搜索_Bi项时，会得到多个结果，增大定位关键方法和代码阅读的难度。

火龙果•整理
uml.org.cn

名称混淆的一般分析方法

- 名称混淆是一个单向过程，大多数情况下是不可逆的。
- 暂不进行反混淆，直接分析被保护程序；若混淆影响了操作和代码理解，则对名称进行一定修改后再分析。
- 不同的.NET编译器在生成代码时，往往会有一些约定俗成的名称，可以辅助反混淆。

火龙果•整理
uml.org.cn

3 流程混淆（Control-Flow Obfuscation）

- 定义
  - 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。

火龙果•整理
uml.org.cn

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
}
```

火龙果•整理
uml.org.cn

常见的流程混淆保护方式

- 基于代码的混淆
  - IL级别的混淆
  - .NET的本质决定了IL的混淆无法做到x86混淆的保护强度。
- 基于结构的混淆
  - 比如类型的拆分、改变程序结构等。

火龙果•整理
uml.org.cn

流程混淆的一般分析方法

- 第一步仍是使用Reflector（或其它反编译软件）载入，看其能否被反编译为高级语言。即使反编译得到的代码很复杂，包括很多跳转，但只要分析能继续进行，就不需要对程序进行特殊处理。
- 使用针对特定保护软件的反混淆程序。
- 通用的反混淆程序（可借助编译器的优化功能）

火龙果•整理
uml.org.cn

编写自己的流程反混淆程序

- 利用MONO的Cecil库。
- 利用微软的Phoenix SDK编译优化框架。
- 其它

火龙果•整理
uml.org.cn

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
- 由于Win32保护技术无法直接应用于.NET程序，因此部分传统强壳在.NET下的保护效果很一般。
- 目前常见的壳是基于.NET重新编写的。

火龙果•整理
uml.org.cn

.NET壳的分类

- 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）

火龙果•整理
uml.org.cn

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。

火龙果•整理
uml.org.cn

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

火龙果•整理
uml.org.cn

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。
- 通过挂钩.NET内核（如JIT引擎）实现。

火龙果•整理
uml.org.cn

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

火龙果•整理
uml.org.cn

5 许可证保护

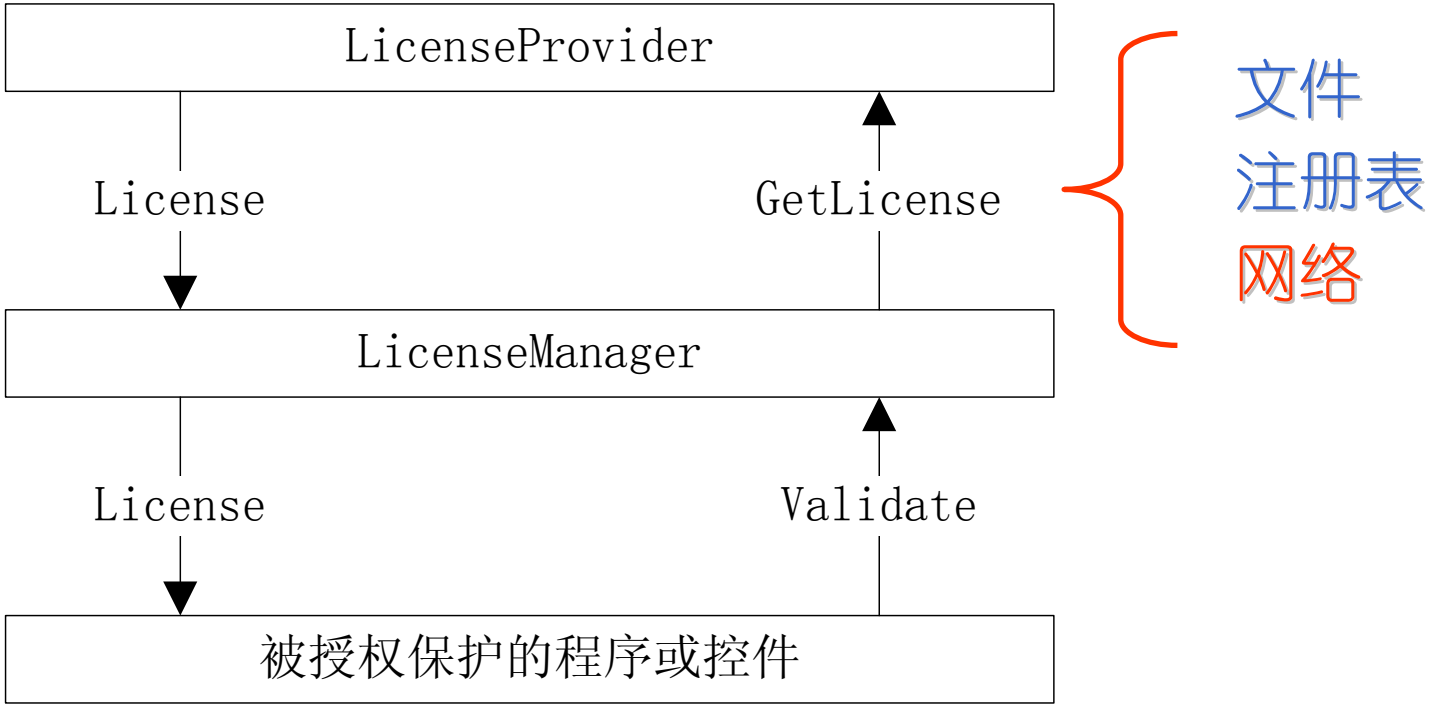
- 许可证（License）提供了更丰富的授权管理手段。

- 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

火龙果•整理
uml.org.cn

典型的.NET许可证保护验证流程



火龙果•整理
uml.org.cn

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）

火龙果•整理
uml.org.cn

用户字符串编码(User String Encoding)

Why?

- 用户字符串中包含太多敏感信息

```
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.ToBase64S
{
    Interaction.MsgBox("Good job, make a keymaker", MsgBoxStyle.Information, "Done
}
else
{
    Interaction.MsgBox("Try again, it is very simple", MsgBoxStyle.Critical, "No ....");
```

火龙果•整理
uml.org.cn

How?

- 直接编码，或是让强名称参与编码

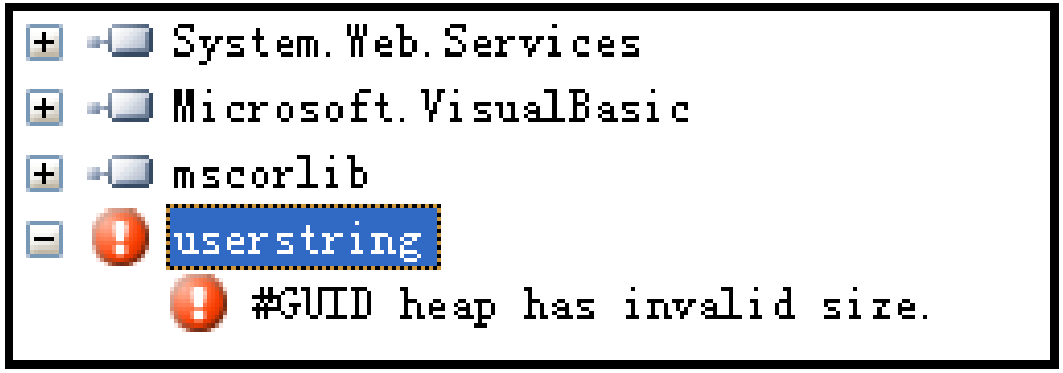
```
FileStream stream = new FileStream(┌.┌.┐ (0x13c), FileMode.Open, FileAccess.Read);
StreamReader reader = new StreamReader(stream);
string text = reader.ReadToEnd();
byte[] bytes = Encoding.Unicode.GetBytes(this.┐ ().Text);
SHA512 sha = new SHA512Managed();
sha.ComputeHash(bytes);
if (Operators.ConditionalCompareObjectEqual(this.┐ (text), Convert.ToBase64String(sha.Hash), true
{
    Interaction.MsgBox(┌.┌.┐ (0x149), MsgBoxStyle.Information, ┌.┌.┐ (0x16e));
}
else
{
    Interaction.MsgBox(┌.┌.┐ (0x177), MsgBoxStyle.Critical, ┌.┌.┐ (0x1a0));
```

火龙果•整理
uml.org.cn

添加错误的元数据

基本原理：添加与.NET标准相违背，但又不影响程序正常运行的元数据（或其它结构）。

常见的如：#GUID大小错误



火龙果•整理
uml.org.cn

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。

火龙果•整理
uml.org.cn

利用错误元数据流的：

复件 {smartassembly}
Module contains unkown meta-data stream '{smartassembly}'.

不一定停留在.NET层次，还利用错误PE结构：

dotNET_Reactor
Invalid number of data directories in NT header.

一般分析方法：
定位错误的位置（比如利用Peverify.exe）；
去除错误（直接修改或重新编译）。

火龙果•整理
uml.org.cn

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。

火龙果•整理
uml.org.cn

特殊的.NET属性

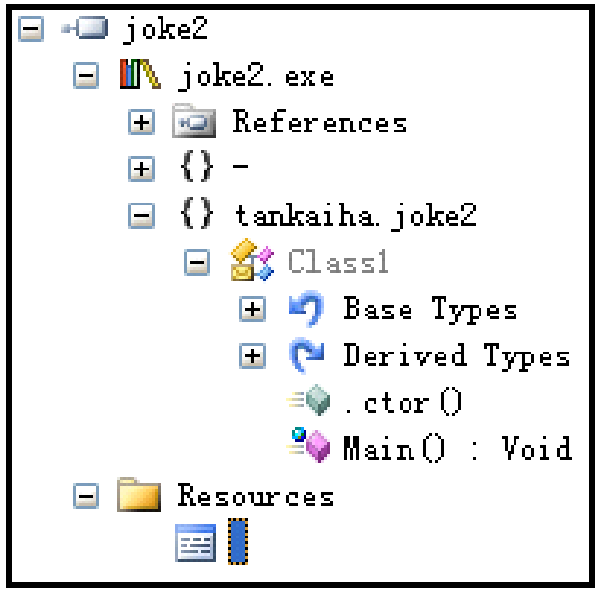
- 与调试相关的属性
  - [DebuggerHidden]
  - [DebuggerNonUserCode]
  - [DebuggerStepThroughAttribute]

- 与反编译相关的属性
  - SuppressIldasmAttribute

火龙果•整理
uml.org.cn

利用系统特性

- 托管资源的名称不合法，不能导出为文件



火龙果•整理
uml.org.cn

反调试

- 托管方式
  - System.Diagnostics.Debugger.IsAttached
  - EnumWindows
  - （底层是调用了Win32 API）
- 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

火龙果•整理
uml.org.cn

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

火龙果•整理
uml.org.cn

动态方法委托调用

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

  - 方法的IL代码加密保存，运行时解密，创建委托并填入解密后的方法体，并动态调用。

火龙果•整理
uml.org.cn

动态方法委托调用的应用

- 对单个或几个关键方法使用该种保护。
- 可以使用逆向算法得到原始IL。
- 或是使用WinDbg等调试软件得到IL。
- 缺点：速度慢

火龙果•整理
uml.org.cn

动态标识

- 基本原理：程序集在编译成功完成时，各个元数据项的token就固定了。将关键方法的调用保存为委托，利用方法的token作为定位委托的依据，在运行时动态利用委托进行调用。
- 保护效果：

```
try
{
    if (![illegible] .[illegible] ())
    {
        return;
    }
    Application.EnableVisualStyles();
    Application.DoEvents();
    [illegible] .[illegible] ([illegible] .[illegible] (0xd123));
    try
    {
        builder = new StringBuilder();
        builder.Append(Path.GetTempPath());
        if (builder[builder.Length - 1] != '\\')
```

```
try
{
    if (![illegible] .[illegible] ())
    {
        return;
    }
    [illegible] .[illegible] ();
    [illegible] .[illegible] ();
    [illegible] .[illegible] ([illegible] (0xdcf8));
    try
    {
        builder = new StringBuilder();
        builder.Append([illegible] .[illegible] ());
        if (builder[builder.Length - 1] != '\\')
```

火龙果•整理
uml.org.cn

动态标识

- 为什么能达到保护效果？

  因为不同的编译器在编译同一个程序时，生成元数据的顺序是不同的，因此再次编译后的token也就不一样。如果token错误，则无法定位到正确的IL代码。

效果：阻止了解密时的round-tripping。

火龙果•整理
uml.org.cn

编译为本地代码

- 原理：利用框架的GAC机制，将IL代码编译为本地汇编，并删除原有的元数据和IL，以达到保护的目的。

- 加解密的战场又回到了x86汇编。

火龙果•整理
uml.org.cn

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（较难实现）
  - 部分替换MSIL指令集
- 按实现的方式分
  - 托管
  - 非托管

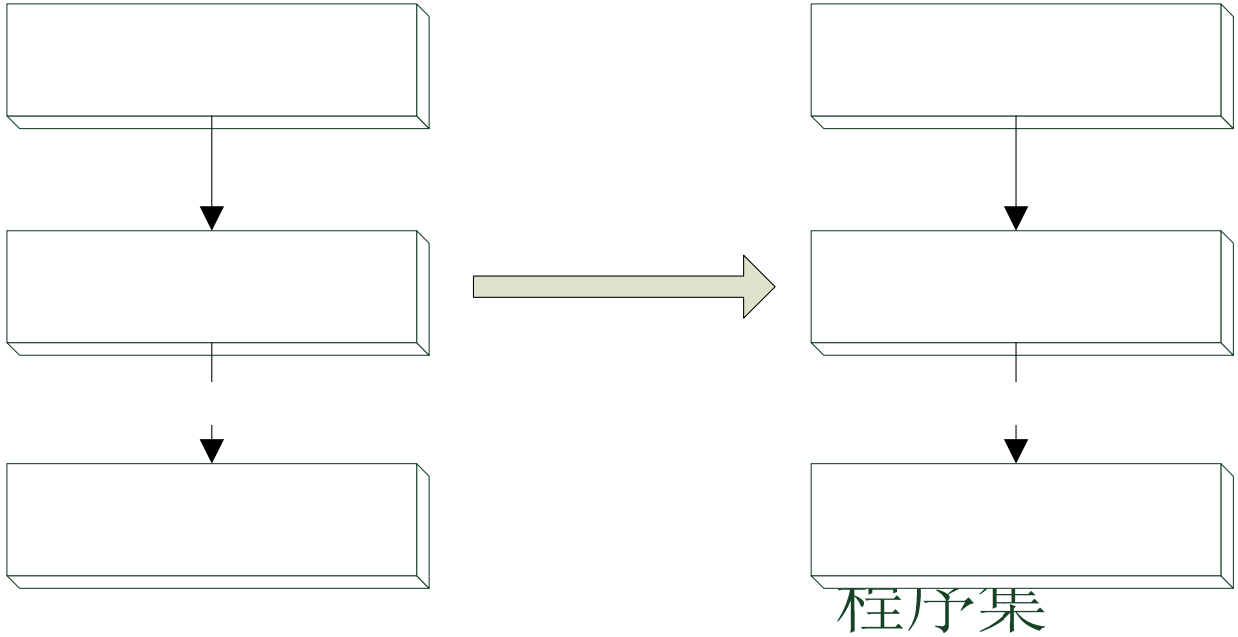
火龙果•整理
uml.org.cn

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。

火龙果•整理
uml.org.cn

非托管的虚拟机



- 优点：执行效率高，保护强度大
- 缺点：兼容性

VM

MSIL代码

火龙果•整理
uml.org.cn

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.OrdeCrypt(text), Convert.ToBase64String(sha.Hash), true
```

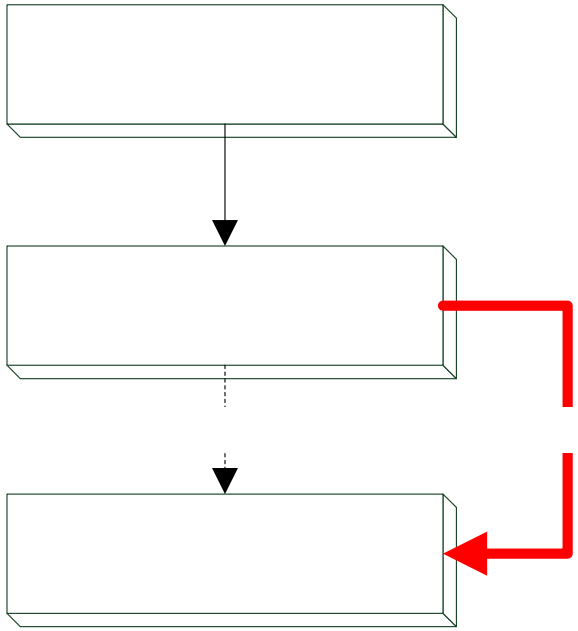
before protection

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

after protection

火龙果•整理
uml.org.cn

纯托管的虚拟机



最大缺点：
执行效率低

程序集

火龙果•整理
uml.org.cn

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

火龙果•整理
uml.org.cn

三、对.NET加解密技术的一些看法

❖.NET保护技术的发展

- 更强大的虚拟机保护配合完善的授权管理。
- 借助编译原理，实现更强大的混淆。
- 更强大的壳保护，同时提高兼容性。

火龙果•整理
uml.org.cn

学习.NET加解密技术的目的

- 了解.NET保护的基本原理，选择合适的保护产品。
- 掌握程序调试技术，在信息很少的情况下解决问题。
- 学习掌握底层框架，提高应用开发水平。
- 其它

火龙果•整理
uml.org.cn

学习.NET加解密与底层框架的方法

- 要注重创新。加密与解密本身就是不断创新挑战的过程，任何行业也都是这样。
- 学习.NET，但不要局限于.NET，要能触类旁通。
- 掌握技巧，更要转化为技术。掌握基础知识是很重要的。

火龙果•整理
uml.org.cn

- .NET加解密是一个很有意思的方向。 .NET加解密技术的发展使得保护方式多样化，强度远大于诞生初期，已经不能单纯地从托管的层次解决了。上至托管层，下至Win32（Ring3）乃至Ring0层，.NET加解密需要掌握综合性的知识。

火龙果•整理
uml.org.cn

Thank You !

2008中国软件安全峰会
2008 China Software Security Summit