.NET 软件代码分析与保护①

吕君可

(浙江师范大学 行知学院 金华 321004)

摘 要：从实用角度出发，结合实例，介绍了软件代码常见的几种分析技术；进而利用强名称、名称混淆、IL 代码混淆、加壳等保护技术，逐步给代码加上防护层，以增强软件代码被反编译破解的难度，从而实现软件代码的保护。

关键词：软件保护；代码分析；代码混淆；加壳

.NET Software Code Analysis and Protection

LV Jun-Ke

(Xingzhi College, Zhejiang Normal University, Jinhua 321004, China)

Abstract: From the practical perspective, the paper introduces several common analyzing techniques about software codes with some cases, and utilizes the protecting techniques: strong name, name obfuscation, IL code obfuscation and packing etc. The code is protected layer by layer to enhance the difficulty cracked by decompile, and to achieve the purpose of protecting the software codes.

Key words: software protection; code analysis; code obfuscation; packing

1 引言

随着互连网的发展，软件传播愈发开放与便捷，使得软件的盗版现象日益严重。据商业软件联盟(BSA)委托市场调查公司(IDC)的一项调查报告显示，2005 年全球商用软件中 35%为盗版，价值 342 亿美元，而中国的软件盗版更为猖獗，高达 86%[1]。随着. Net 时代的到来，越来越多的软件使用.Net 技术，这种以平台无关的中间代码来发布的软件代码与源码相似，比起传统的二进制可执行代码更易遭到静态分析甚至恶意修改。为了防止软件被非法复制、修改，除了用法律对软件知识产权进行保护之外，还需要在技术上对软件进行保护。

代码保护是目前最有效的方法之一。利用软件代码保护技术，给软件代码加上防护层，只要能使破解者对软件代码的分析破解付出较高的代价，就可认为软件在其生命周期内实现了知识产权与代码的保护。本文使用常见的密码验证程序为例，分析其中关键代码，使用多种技术手段，逐步给其加上防护层，从而实现软件代码的保护。文中使用的示例程序在 winxp 系统、VS2008 的 VB.net 中调试通过。

2 代码分析技术

代码分析技术是对程序控制流程，数据流程等抽象信息进行分析、提取及获得的分析技术。软件代码分析主要包括静态分析和动态分析两种。

2.1 静态分析

静态分析是对没有运行的静态程序进行的，它的主要任务是定位程序的关键代码，分析程序流程。即利用反汇编工具将程序的指令字节反编译成 IL 指令或高级语言，通过阅读反编译代码掌握程序的流程与功能。如图 2 所示为双击密码验证函数 CHECKMM 时反编译出的 VB 原码，从中可以得出用户的注册密码为“hello”。由此可见，程序中应避免使用未经加密的字符串。

① 收稿时间:2010-10-28;收到修改稿时间:2010-12-04

图 1　密码验证程序

图 2　对密码验证函数 CHECKMM 进行反编译

2.2 动态调试

定位出关键代码，程序也就破解了一半，但如果算法比较复杂，尤其是经过流程混淆的程序，利用静态分析程序的流程就比较困难，此时就需要用动态调试。动态调试主要是跟踪数据值的变化，跟踪控制流程等。利用调试工具对关键代码下断点，如果注册密码的明码完整地出现在内存中，则可以直接查看。图 3 是利用 PebrowseDbg 查看本例中寄存器 EDX 时所得到的信息。

```
+0x015178A4  68 00 65 00  h.e. +20
+0x015178A8  6C 00 6C 00  l.l. +24
 0x015178AC  6F 00 00 00  o... +28
```

图 3　用 PebrowseDbg 查看注册密码

2.3 代码修改

代码修改是另一种常见的软件破解技术。以“确定”按钮为例，打开反编译后的 IL 文件，让程序跳过密码验证程序，一种破解方式是修改粗体部份 IL_0011 不让 brfalse.s 跳转。根据堆栈平衡原则，修改为 POP 或 NOP，再编译回 exe 文件，则可成功破解登录。此时不管用户输入什么内容，程序都会进入到第二个窗体中。

以下为 Button1_Click 事件的 IL 代码：

```
.method private instance void Button1_Click(object sender,
class [mscorlib]System.EventArgs e) cil managed
{
// 代码大小        48 (0x30)
.maxstack  8        ' 可容纳数据项的最大个数
IL_0000:  ldarg.0   ' 在堆栈中载入参数
IL_0001:  ldarg.0
IL_0002:  callvirt  instance class [System.Windows.Forms]System.Windows.Forms.TextBox test.Form1::get_TextBox1()
IL_0007:  callvirt  instance string [System.Windows.Forms]System.Windows.Forms.TextBox::get_Text()        ' 获取 TextBox1 中的字符串
IL_000c:  callvirt  instance bool test.Form1::CHECKMM(string)  '调用函数 CHECKMM
IL_0011:  brfalse.s IL_0024  ' 密码错误跳转
IL_0013:  call      class test.My.MyProject/MyForms test.My.MyProject::get_Forms()
IL_0018:  callvirt  instance class test.Form2 test.My.MyProject/MyForms::get_Form2()
IL_001d:  callvirt  instance void [System.Windows.Forms]System.Windows.Forms.Control::Show()' 密码正确调用 Show 方法显示 Form2
IL_0022:  br.s      IL_002f  ' 跳转到标号 IL_002f
IL_0024:  ldstr     "sorry"  ' 密码错误提示信息
IL_0029:  call      valuetype [System.Windows.Forms]System.Windows.Forms.DialogResult[System.Windows.Forms]System.Windows.Forms.MessageBox::Show(string) ' 调用 Show 方法显示密码错误对话框
IL_002e:  pop
IL_002f:  ret       ' 返回
} // end of method Form1::Button1_Click
```

3　代码保护技术

代码保护技术主要用于增强软件被反编译破解的难度，给程序签署强名称、代码混淆、加壳等是几种重要的代码保护方法。尤其是经过混淆的代码在反编译后，程序间的交叉引用会异常复杂；变量名、类名等数据会更加难以辨认。

3.1 强名称

强名称是.net 提供的一种验证机制，主要用于标识版本和标识原作者。利用强名称可以帮助用户检验自己的程序是否为原作者所写，而没有被人恶意修改过代码。在 SDK 中利用 Sn 命令可生成相应的密匙文件，给程序签署强名称后会在 manifest 中出现两条关键语句：.Publickey 和.Hash algorithm。此时修改 IL 代码，便会提示“Strong name validation failed”强名称验证失败。

3.2 名称混淆

名称混淆是最简单的混淆，即将变量名、类名、方法名、字段名等换成特殊符号或其它符号，以增加破解者对反编译代码的阅读难度。改名的方法很多，包括 Hashing 改名、名字交换、重载归纳等，图 4 是名称混淆前后的效果图。而名称混淆返混淆，由于名称换掉后没有备份表，所以基本上无法还原。

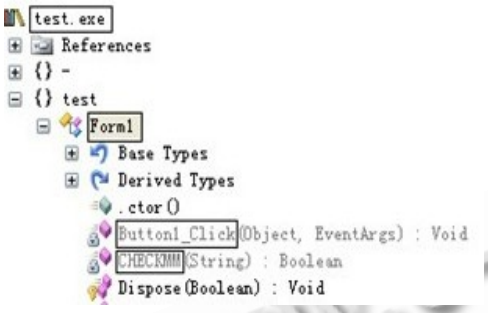

图 4(a) 名称混淆前

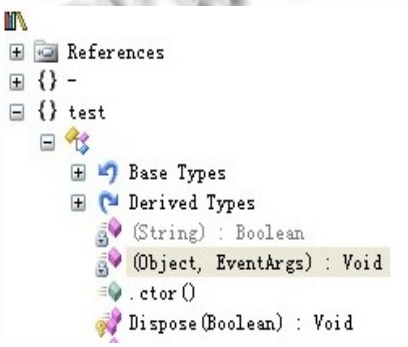

图 4(b) 名称混淆后

图 4(a)中的加边框部份经过名称混淆后，在图 4(b)中没有任何显示。在实际应用中，可以将其改成其它字符，以增加代码阅读的难度。但需要注意的是并不是所有的名称都可以进行混淆，比如上图中的.ctor，一旦这些名称被修改，运行程序将报错。一般来说，这种情况发生在 DLL 身上的比较多，因为 DLL 的某些 Public 方法是对外的接口，在程序开发和调试的时候一般使用源名称，因此一般不混淆对外提供的 Public 方法和混淆程序内部调用的程序。

3.3 流程混淆

流程混淆是指打乱程序流程，隐藏原作者的真实意图，增加代码阅读的难度，防止反编译工具直接将代码反编译为高级语言[2]。混淆的目标不是达到一种绝对的安全，而是尽可能的提高逆向分析的代价。对于大部分应用程序来说，只要能够让逆向分析的代价超出收益，其保护的目标也就达到了。根据混淆原理，对“确定”按钮的原 IL 代码进行混淆，经过混淆后，IL 代码为：

```
IL_0000:  ldarg.0
IL_0001:  ldarg.0
IL_0002:  br.s       IL_0028
IL_0003:  callvirt   instance bool test.Form1::CHECKMM(string)
IL_0008:  brfalse.s  IL_001b
IL_000a:  call  class test.My.MyProject/MyForms test.My.MyProject::get_Forms()
IL_000f:  callvirt   instance class test.Form2 test.My.MyProject/MyForms::get_Form2()
IL_0014:  callvirt   instance void [System.Windows.Forms]System.Windows.Forms.Control::Show()
IL_0019:  br.s       IL_0026
IL_001b:  ldstr      "sorry"
IL_0020:  call valuetype [System. Windows. Forms]System.Windows.Forms.DialogResult[System.Windows.Forms]System.Windows.Forms.MessageBox::Show(string)
IL_0025:  pop
IL_0026:  ret
IL_0028:  callvirt   instance class [System. Windows.Forms]System.Windows.Forms.TextBoxtest.Form1::get_TextBox1()
IL_002d:  callvirt   instance string [System. Windows.Forms]System.Windows.Forms.TextBox::get_Text()
IL_0032:  br.s       IL_0003
```

将 IL 代码重新编译成可执行文件，程序运行正常，但用反编译工具打开该事件后，提示：“未将对象引用设置到对象的实例”，从而阻止了代码被直接反编译为高级语言。上述事件使用 br.s 实现程序的直接跳转，在实际运用过程中，可在源程序中添加一些垃圾代码，比如加入模糊谓词、伪装的条件判断语句等，让程序不停地进行跳转，来增加 IL 代码的阅读难度。

3.4 加壳

所谓“壳”是指计算机软件里的一段专门负责保护软件不被非法修改或反编译的程序。它们一般先于

(下转第 220 页)

由表 1 和表 2 比较，经过优化后的内核训练速度大约是源程序的 5 倍。由表 2 得知，对于 2-3-1 网络，64 个工作项的工作组性能更优，这是由于相比更大的工作组，该工作组对于 2-3-1 网络训练各步骤所浪费的工作项更少。

表 3 显示了不同大小的工作组对 2-300-1 网络的使用 RPROP 算法的训练时间，由于 256 个工作项更满足 2-300-1 网络的需要，所以其性能要优于其他两个工作组。但对于各层神经元数量相差很大的网络，运算时浪费的工作项会很多，所以训练的速度大约只有源程序的 2 倍。

表 3　2-300-1 网络单内核实现与源程序的训练时间

实现类型和训练次数	OpenCL单内核实现单GPU运行训练1000次			源程序 训练1000次
工作组大小	64	128	256	/
耗时（秒）	1.0349206	0.9766208	0.9562168	2.3481872

4　结论

由表 1、表 2 和表 3 来看，在人工神经网络各层神经元数量相差不很大的情况下，其训练算法在支持 OpenCL 的 GPU 上运算比在 CPU 上的效率高数倍。但也存在一些限制，由于没有工作组之间的同步，为了减少数据传输消耗的时间，只能使用一个工作组来计算，而指定的 GPU 其执行内核的工作组中所能存放的工作项的最大数目是确定的，所以不能使用更多的工作项来进行计算；由于网络各层神经元数量不一定是 GPU 的 wave 大小的整数倍，可能导致同一 wave 中有的工作项不能被充分利用；由于给定 GPU 的工作组内存大小有限，对全局内存的访问不一定能够完全优化。

参考文献

1 Hagan MT, Demuth HB, Beale MH. 神经网络设计.北京:机械工业出版社,2005.197–244.

2 Wang B, Zhu L, Jia KB, Zheng J. Accelerated Cone Beam CT Reconstruction Based on OpenCL. Image Analysis and Signal Processing (IASP), 2010 International Conference on. 2010.291–295.

3 The OpenCL Specificatin Version:1.0. Khronos OpenCL Working Group. 2008.

4 The OpenCL Specificatin Version:1.1. Khronos OpenCL Working Group. 2010.

5 Programming Guide-ATI Stream Computing OpenCLTM. Advanced Micro Devices,Inc.

6 跨平台的多核与众核编程讲义—OpenCL 的方式.AMD 上海研发研发中心.

(上接第 235 页)

程序运行并拿到控制权，然后完成保护软件的任务[3]。壳分为两种：保护壳与压缩壳。压缩壳的主要功能在于如何优化算法，增加压缩比，进而减少程序的体积，保护能力相对较弱；保护壳则运用各种加密算法和先进的加密技术来保护程序，使得解密变得异常困难，甚至无法脱壳解密。常见的保护壳有 ASProtect、Armadillo、EncryptPE 等，但有加壳就有脱壳，软件开发者最好根据自己的需要，建立模型，设计自己的加壳保护方案。

4　结语

随着软件代码分析技术的不断提升，只要有足够的时间和精力，代码保护技术总是能够被破解的，它只能够增加软件被逆向分析的复杂度，在一定程度上延缓对程序的分析与破解。但是如果一种保护技术的强度强到足以让破解者在软件的生命周期内无法实现完全破解，这种保护技术就可以被认为是成功的。因此，将多种保护技术结合起来将是一种有效的选择。

参考文献

1 Business software alliance.URL:http://www.bsa.org.

2 段钢.加密与解密.第 3 版.北京:电子工业出版社, 2008.241.

3 武新华,张慧娟,李秋菊.加密解密全攻略.第 2 版.北京:中国铁道出版社, 2008.203.