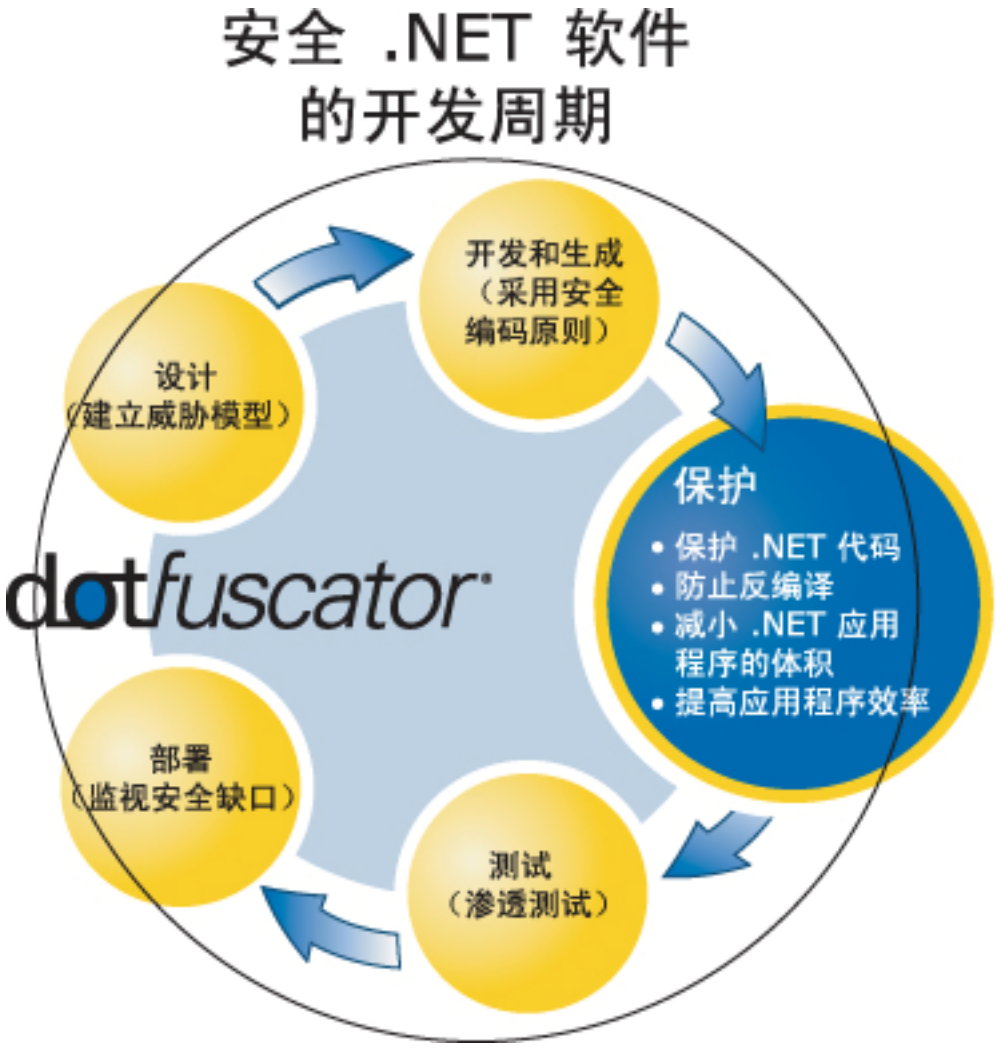
Dotfuscator3.0 快速指南

Wang Ke

Dotfuscator的定位



Dotfuscator的定位

- .NET 编写的程序很容易进行反向工程处理。
- 任何人只要有一个可免费得到的 .NET 反编译器就可以轻易对代码进行反向工程处理。
- 模糊处理这种技术能够对程序集中的符号进行无迹可寻的重命名，并提供很多其他手段，从而阻止进行反编译。

模糊处理

- 模糊处理 (obfuscation) 意在引起混淆。
- 原始的模糊处理程序实质上是将在代码中找到的标识符重命名为不可读的内容。它们可能使用哈希技术，或者对字符集进行算术偏移，将字符变为不可读字符或不可输出字符。这些技术虽然表面上很有效，可是很显然，它们是可逆的技术，因此很难起到保护作用。
- PreEmptive 的模糊处理工具远远超过了这种原始的重命名方法，它使用很多其他颇具独创性的“引起混淆”的方法，使得几乎不可能对他人的知识产权进行反向工程处理（而且代价太大，得不偿失）。

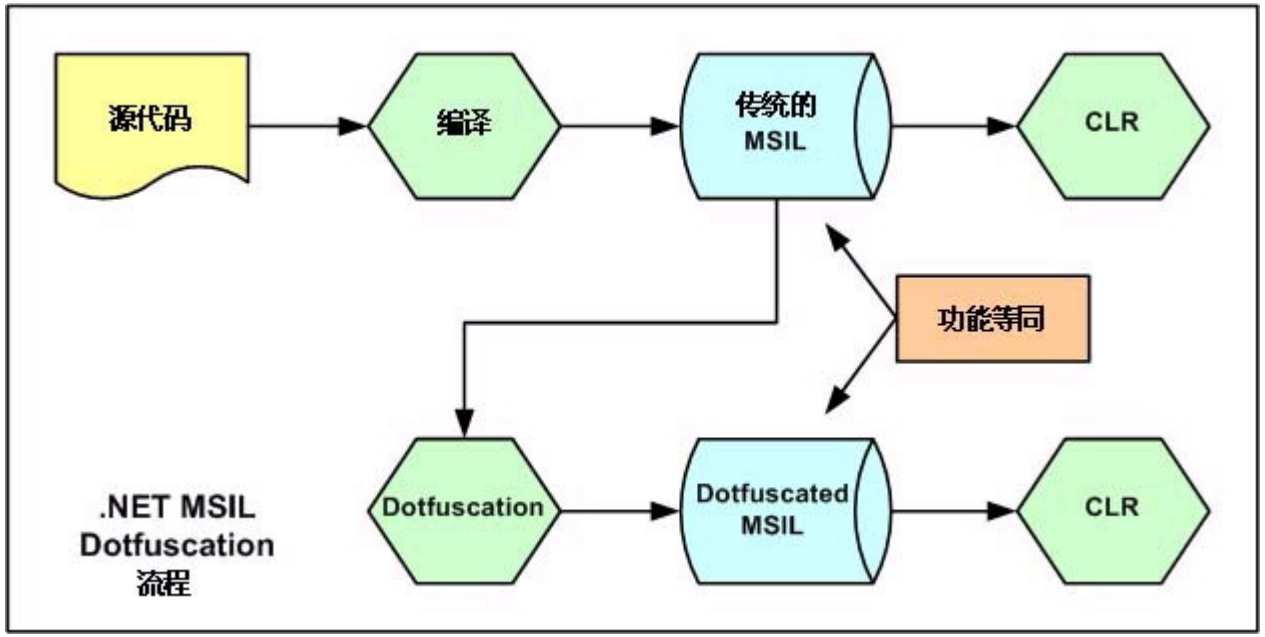
Dotfuscator 的益处

- Dotfuscator 是一种用于 .NET 应用程序的开发后重编译系统。它分析应用程序并使它们更小巧、更快捷且更难于进行反向工程。简而言之，Dotfuscator 使应用程序变得“更好”。
- **Dotfuscator 极大地增强了代码安全性**
- Dotfuscator 采用最新的技术以保护 .NET 应用程序 — 即保护应用程序内所包含的重要知识产权。
- **Dotfuscator 极大地减小了 .NET 程序的体积**
- Dotfuscator Professional Edition 分析应用程序并精确计算出您真正需要使用的应用程序部分。据此，它可以剥离出这些部分，使您得到尽可能小的可执行文件。
- **Dotfuscator 提高运行时性能**
- 通过移除不需要的程序元素并将标识符重命名为短小的名称，Dotfuscator 可切实提高程序运行速度。
- 此外，Dotfuscator 还提供了很多其他的益处 (Benefits)，如将很多程序集链接到一个程序集中，以及使用隐藏信息为应用程序添加水印。

模糊处理过程

- 模糊处理这一过程应用于已编译的 MSIL 代码而不是应用于源代码，理解这一点十分重要。开发环境和工具无需更改就可以适应重命名技术。源代码不会以任何方式被更改，甚至不会被读取。经过模糊处理的 MSIL 代码在功能上等效于传统的 MSIL 代码，它在公共语言运行库 (CLR) 上执行并产生与原始 MSIL 代码相同的结果。（但是，不能进行逆向操作。即使可能将经过高度模糊处理的 MSIL 进行反编译，和原始的源代码相比，它在语义上也将有重大不同。）下图说明使用 Dotfuscator 进行模糊处理的流程。

模糊处理过程


源代码
编译
传统的
MSIL
CLR
功能等同
Dotfuscation
Dotfuscated
MSIL
CLR
.NET MSIL
Dotfuscation
流程

Dotfuscator Community Edition的局限性

- Dotfuscator Community Edition 是提供基本模糊处理功能的免费版本。其主要用途是重命名标识符，为反向工程处理设置障碍。Dotfuscator Community Edition 合并了多种进行这种保护的高级技术。此外，还会实现一些减小程序大小的效果（这是重命名为普通标识符的结果）。
- Dotfuscator Community Edition 3.0：
- 不能脱离 Visual Studio 单独运行，这意味着它不能用于商用生成环境中。
- 不能重命名 .NET 2.0 泛型类型和方法。
- 不能使用 Dotfuscator 对针对 Microsoft Office 集成编写的托管代码进行模糊处理。
- 不能对旨在运行于 Microsoft SQL Server 2005 内部的托管代码进行模糊处理。
- 仅支持以全局选项的方式使用库模式（对所有的输入程序集启用或禁用库模式）。
- 仅支持以全局选项的方式应用声明性模糊处理设置（对所有的输入程序集启用或禁用声明性模糊处理设置）。
- 不支持托管 C++ 应用程序。

独立 **GUI** 快速入门步骤

- 步骤 **1 --** 启动 **Dotfuscator GUI**
- 步骤 **2 --** 创建一个默认项目
- 步骤 **3 --** 生成项目
- 步骤 **4 --** 配置项目
- 步骤 **5 --** 重新生成项目
- 步骤 **6 --** 浏览输出

观察和理解模糊处理的输出

- 观察和理解模糊处理的输出
- **步骤 1 --** 使用反汇编程序
- .NET Framework SDK 附带了一个称为 ildasm 的反汇编程序实用程序，它允许您将 .NET 程序集反汇编为 IL 程序集语言语句。
- 选择“文件”|“打开”菜单并浏览并单击“打开”，被反汇编的程序集的视图出现。
- **步骤 2 --** 反编译
- 如果您现在认为您的源代码将只能由一小群实际懂得 IL 汇编语言的技术人员访问，那么请再考虑一下。您可以做进一步的试验，使用 Reflector 和 Anakrino 之类的反编译器从我们的应用程序重新创建源代码。这些实用程序可将 .NET 程序集直接反编译为 C#、VB .NET 或托管 C++ 之类的高级语言。
- 本节我们将使用两个可免费获得的反编译器：
- Reflector for .NET，http://www.aisto.com/roeder/dotnet/
- Anakrino（GUI 版本）/ Exemplar（命令行版本），http://www.saurik.com/net/exemplar/
- Dotfuscator Professional 能够成功阻止两种主要的反编译器对经过 Dotfuscator 模糊处理的代码进行反向工程。
-

配置文件简要说明

- 版本：配置文件的版本
- 属性列表和属性：可选的属性列表节用于给出变量（称为“属性”）的定义和赋值，这些变量以后可在配置文件中使用。
- 全局节：可选的全局节用于定义在整个运行过程中都起作用的配置选项。
- “详细”、“静默”和“调查”全局选项：这些选项和相应的命令行选项相同。设置命令行选项即会在运行时设置全局选项。或者，如果设置了全局选项，且未设置命令行选项，则全局选项优先。换句话说，无法从命令行“取消设置”已设置的全局选项。
- 输入程序集列表：输入程序集列表包含想使用 **Dotfuscator** 进行模糊处理的程序集的文件名和目录。
- 按程序集选择库模式：告知 **Dotfuscator**：某个特定的输入程序集将作为库进行处理。在使用 **Dotfuscator** 进行模糊处理时，库被定义为本次运行中未被指定为输入的其他组件将会引用的程序集，无论设置的自定义排除规则如何，这都会对重命名和精简造成影响。
- 按程序集执行声明性模糊处理：**Dotfuscator** 允许对特定的输入程序集启用或禁用“声明性模糊处理”。如果未启用，**Dotfuscator** 则忽略与模糊处理相关的自定义属性。
- 输出目录：写入输出程序集的目录。
- 临时目录：指定 **Dotfuscator** 的工作目录。如果未指定，则默认工作目录为系统的临时目录。应用程序使用工作目录对输入程序集运行 **ildasm** 和 **ilasm**。将反汇编的输出和在输入程序集中嵌入的任何资源（托管的或非托管的）存储在该目录中。在处理之后，这些文件将被自动删除。
- 模糊处理属性的功能映射：功能映射用于声明性模糊处理。
- 重命名节：重命名节允许用户指定某些选项，这些选项特定于重命名、输入和输出映射文件位置以及用于从重命名中排除项的粒度更小的规则
- 重命名选项：**Dotfuscator** 允许使用几个选项，这些选项指定重命名算法处理命名空间的方式。
- 重命名排除列表：提供微调输入程序集的重命名的动态方法。
- 输出映射文件：产生所有重命名映射的日志，**Dotfuscator** 将在特定的运行过程中使用它们。
- 有关 **XML** 配置文件的说明：**Dotfuscator** 使用 **XML** 格式的文档记录配置文件和映射文件。

案例

- 略