How-To-Select™ Guides

How-To-Select an Obfuscation Tool for .NET™

Covers Obfuscation tools for use in obfuscating .NET application .EXE files and .DLL components

By Mike Gunderloy

Table of Contents

- Introduction
- Understanding Obfuscation
- Functionality by Category
- Related Categories
- Decision Points
- Features
- Vendors/Authors Profiled in this Guide
- Products Profiled in this Guide
- General Criteria
- Feature Tables
- Products Not Profiled in this Guide
- Glossary
- About the Guides

http://www.howtoselectguides.com/dotnet/obfuscators/
Introduction

Hello and welcome to the second in Xtras' series of How-To-Select™ Guides: How-To-Select an Obfuscation Tool for .NET™.

In this particular Guide, we've evaluated products designed to obfuscate your .NET source code. As with the other Guides in the .NET Developer Guide Series, we've written this Guide for developers using C#, VB.NET, or any other .NET language. We've included a discussion of what obfuscation does and why you might want to use it, and offered guidance to help you choose the most appropriate obfuscation product for your own needs.

The Best Product for your Needs

We bring you the info you need to make informed choices when choosing .NET components and tools for your project. Each How-To-Select Guide goes in-depth into a different product category. We offer expert selection guidance and profile every product we can find in the relevant category, limited only by the vendor's willingness to participate. Plus we'll revise each Guide biannually to keep abreast of changes.

We Inform, We Don't Decide (for you)

Rarely is there one product that meets everyone's unique needs but that doesn't stop others from selecting a single "best" product as "editor's choice," but not here. Our How-To-Select Guides instead focus on the product categories to help you quickly become a category expert and ensure our Guides are useful long after written. We'll help you sort out use cases for various product types, cover vendor support and licensing policies, and make sure you understand what each product profiled can (and can't) do for your project. We profile both free and open source software alongside commercial offerings so we don't artificially limit your options.

Our Goals

We set forth several goals for our How-To-Select Guides. We wanted them to be:

- Clear & Concise  - Quick to read and easy to comprehend.
- Accurate & Unbiased  - Completely Defensible among leading experts with as little bias as humanly possible.
- Exhaustive, Thorough, & Complete  - Covering all options including commercial, shareware, freeware, open source, and even coding techniques; all decision points and aspects of potential concern; and including all currently available products in the category.

The Perfect Guide Includes You!

We know our How-To-Select Guides can save you time and help you avoid costly mistakes but we need your help to make them even better:

- Subscribe - Subscriptions are free, so sign up on our website to receive each Guide as it's released.
- Participate Online - Ask questions and make suggestions online at our forums located at http://forums.howtoselectguides.com.
- Email Feedback - Send your questions, comments, or suggestions to feedback@howtoselectguides.com
- Tell Others - If you know .NET developers, send them a link to http://www.howtoselectguides.com and suggest they subscribe too.
- Mention in Newsgroups - When you answer questions in developer newsgroups and forums about components and tools, mention and include a link to our Guides.
- Blog about a Guide - If you blog and are inspired by our Guides, mention and include a link to that Guide.
- Ask Questions - If our guides don't answer a question you have, ask it in the forum so one of our authors and your fellow .NET developers can answer the question for you.
- Answer Questions - Answer questions asked in our forum by other .NET developers, which also helps others with the same question.
Obfuscation Tools for .NET

- Make Suggestions - If you notice anything we can improve about our Guides, post a suggestion in our forums.
- Offer to Write - If you are a category expert, or want to become one, and you are a good writer willing to meet deadlines, offer to write one of our future Guides.
- Encourage Vendor Participation - Tell vendors being included in relevant Guides is a prerequisite for considering purchase of their products.
- Support our Sponsors - Please consider supporting the companies that sponsored and/or advertised in these Guides, such as TallComponents and Xtras.Net in this Guide edition.

Thanks for reading about our new How-To-Select Guides, and we hope this and future Guides will both save you time and help you make the right selection!

Oh, and don't forget to subscribe!

- The How-To-Select Guides Team

Understanding Obfuscation

One of the key architectural features of Microsoft .NET is that all its languages compile to .NET assemblies containing CPU-independent instructions. These instructions, known as Microsoft Intermediate Language (MSIL), contrast with many other languages that generate CPU-specific instructions for the target CPU. .NET assemblies are comprised both of MSIL instructions and also metadata that describe types, members, and code references from other assemblies. At runtime, the MSIL instructions are converted to CPU-specific instructions by a just-in-time (JIT) compiler.

The use of this architecture has several huge benefits for the .NET developer. For instance, it makes possible easy interoperability for code written in differing languages, and it makes it easy for an assembly to specify exactly the version of another assembly with which it will work. But there is one major drawback for some developers as well: the MSIL and metadata in an assembly provide enough information to recover the original code. The .NET framework ships with a tool, ILDASM, that can disassemble MSIL into assembly language, and other utilities can carry the process even further, translating a .NET assembly back into a high-level language such as C# or Visual Basic .NET.

This is a drawback because it is very difficult to protect the intellectual property in an application if anyone can read the source code for the application. Developers who have spent months or years coming up with complex algorithms or workarounds for bugs in the .NET Framework, or other components, often prefer to have their methods remain secret from their competitors.

This is where obfuscators come in. The purpose of an obfuscator is to apply one or more transformations to a .NET assembly without affecting the proper functioning of the assembly, but that make it difficult or impossible to understand any source code recovered from the assembly. As a simple example, every obfuscator offers some level of member renaming. Source code where all of the objects are named things like A, B, and C instead of Customer, DatabaseConnection, and InterestCalculation, is substantially more difficult to understand.

What Should You Obfuscate?

As a general rule, you should obfuscate any .NET assembly where you do not want to make the source code available along with the assembly itself. There can be several reasons for making this decision. You might simply want to protect your intellectual property because of the substantial investment it represents. You might want to protect a licensing routine from inspection by hackers. You might want to hide SQL statements or server names. In most cases, it will be simplest to obfuscate your entire .NET application. In other cases, you may decide that the bulk of your application consists of routine code of no particular interest, and that the code you care about is located in only a few assemblies. In those cases, you may choose to obfuscate only the critical assemblies, which can make your build process quicker and easier. ASP.NET applications seldom need obfuscation, because visitors to an ASP.NET-based Web site do not download the actual .NET assemblies, only the HTML output from those assemblies. But if you’re selling a commercial ASP.NET application for others to install on their own servers, you might want to obfuscate the application to protect your own programming investment.

Potential Downsides of Obfuscation

Although obfuscation is a valuable technology in many circumstances, you do need to be aware of some potential pitfalls:

- Obfuscation can break code that depends on reflection, serialization, or remoting.
- Obfuscation can make diagnosing and debugging problems in your code more difficult.
- Obfuscation adds another step and another potential error source to your build process.

Obfuscation in Visual Studio .NET

If your obfuscation needs are minimal, and you’re a Visual Studio .NET user you may not need to purchase a product at all. That’s because Visual Studio .NET includes a copy of the Community Edition of Dotfuscator for .NET, an obfuscator from PreEmptive Solutions. This obfuscator is targeted at students and freeware authors, and supports basic entity renaming and removal of unused metadata, but no advanced obfuscation features. You’ll find a review of the Professional Edition of Dotfuscator later in this Guide, and PreEmptive has a comparison chart online showing what’s in their various editions.
Functionality by Category

Most of the software in this Guide fits the definition we've just given of an obfuscator, but we've also included some closely-related tools. All have the protection of your application's source code as their primary purpose.

- Obfuscators
- Compilers
- Linkers
- Encrypting Launchers

Obfuscators

Obfuscators work by transforming a .NET assembly into an equivalent .NET assembly that is more difficult to understand when disassembled or decompiled. Some obfuscators use ILDASM to disassemble the assembly and operate on the disassembled MSIL, and then use ILASM to reassemble the result. Others operate directly on the assembly without any ILDASM/ILASM round trip. Obfuscators apply a variety of different techniques to make the disassembled source code more difficult to understand.

Compilers

Compilers work by removing the bulk of the MSIL code from a .NET assembly, replacing it with compiled native code targeted at a particular platform and a MSIL stub to call the native code. This prevents decompilation by removing the MSIL and the metadata on which a decompiler operates. Using a compiler ties your assembly to a particular CPU architecture, removing the potential cross-platform nature of .NET. This may become an important consideration as Microsoft releases .NET for 64-bit CPUs, or if you choose to use the Mono open-source implementation for Linux. In contrast, if your assembly remains in the original MSIL, it will be compiled at runtime by .NET's own Just-In-Time (JIT) compiler, which can optimize each assembly for the target CPU. Using a compiler also means that you lose some of the benefits of managed code, such as the Common Language Runtime's safe code execution environment.

Linkers

Ordinarily, a .NET assembly depends on code from a variety of assemblies that are shipped by Microsoft. These assemblies, known collectively as the .NET Framework, make up the basic functionality of .NET. These assemblies include things like Windows Forms, file processing functions, networking code, and so on. A linker can pull just the parts of the .NET Framework that your application uses into your application's own assembly, and in so doing, lets you obfuscate those parts of the Framework as well as your own code. This can make it more difficult for others to determine what your code is doing. However, it is possible that the use of a linker in this way violates the Microsoft license terms for the .NET Framework.

Encrypting Launchers

Encrypting launchers employ a two-step strategy for protecting source code. First, they encrypt your assembly and its resources. Second, they wrap the encrypted assembly in an executable application. At runtime, this application decrypts your assembly directly to memory, where it is then executed normally by the .NET Common Language Runtime. With this strategy, the code on your hard drive is no longer native .NET code and thus cannot be deciphered with the usual tools for exploring .NET assemblies.

Related Categories

Obfuscators are one type of tool for protecting your intellectual property, but they are not the only type. You should consider some other options when you're developing a strategy for protecting your intellectual property. This section of the Guide lists these other categories of tools. We plan to cover most of these related categories in future How-To-Select Guides.

- Software Licenses
- Installers
- Licensing Components
- Source Code Formatters

Software Licenses

It probably seems strange to think of something as abstract as a software license as an alternative to a product such as an obfuscator, but in fact the two are closely related. Depending on your licensing terms, it may not make sense to spend time obfuscating your code. In the extreme case, if your application is available under an open source license, you are already committed to making full source code available. In this case, there's obviously no point in applying an obfuscation tool to your code.

Even if you don't give the source code away, you may find that the protection offered by a software license is sufficient. If you're developing custom software for small markets, for example, you may choose to negotiate individual terms with each
customer. In this case, you can dicker with each customer about whether to sell source code rights and how much to charge. If you know your customers intimately and can trust them, code obfuscation is less important.

Installers
Some installers can convert a .NET application into a single tamper-resistant executable application. By blocking debugging and decompilation this achieves the same goals as obfuscation. Depending on the method employed, this application may be transformed back into managed code at runtime, or it may execute as unmanaged code.

Licensing Components
In addition to protecting your source code, you may also be concerned about protecting your application itself from unauthorized copying and redistribution. For that purpose, you should investigate licensing components. Such components are designed to address the problem of unauthorized copying by requiring the entry of a serial number or other identifying information to allow continued use of an application.

Source Code Formatters
Rather than obfuscate the compiled assembly, you may choose to obfuscate a copy of your original source code and then compile the obfuscated code. This approach has the benefit of offering you precise control over the look of the decompiled, obfuscated code.

Decision Points
Purchasing an obfuscator can represent a major investment, and learning to use it effectively can require a major effort. Making a purchase of this magnitude should involve careful evaluation of the alternatives. In this section of the Guide, we'll offer advice about factors that should influence your decision.

- **Try Before You Buy**
- **Consider Vendor’s Other Products**
- **Version Support**
- **Obfuscation Methods**
  - Entity Renaming
    - Naming Conventions
    - Overloaded Renaming
    - Include and Exclude Members
    - Declarative Obfuscation
  - Control Flow Obfuscation
  - Removal of Unused Members
  - String Encryption
  - Breaking ILDASM
- **User Interface**
- **Strong Name Support**
- **Debugging Support**
- **Watermarking**

Naturally, not all of these factors will apply to every application.

**Try Before You Buy**
The first thing to keep in mind is that no matter how much information we pack into this Guide, there's no substitute for trying the software in your own environment and on your own application. No one knows your own requirements, or the quirks of your own tools and components, better than you do. Fortunately, just about every vendor in the .NET tools and components business provides a trial version of their software for free evaluation. We recommend that you use this Guide to locate the obfuscators that seem like a good fit for your application, and then download copies to evaluate them in your own environment. With obfuscators, you'll find that evaluation copies normally obfuscate some but not all of the members in the target .NET assembly. This is not enough to interfere with evaluation, but generally precludes you from using an evaluation copy in a commercial application. Of course, you should follow the licensing requirements scrupulously in any case.

Trying before buying is especially important with obfuscators because they make such basic changes to your shipping code. Most obfuscators employ a variety of methods to hide your source code, and some of these methods may at times affect the proper functioning of your application. It's important to verify that your obfuscator of choice can be used successfully with your application. Fortunately, obfuscator vendors are willing to work with customers to identify appropriate option settings and help avoid potential issues.

**Consider Vendor’s Other Products**
In many cases, the obfuscator that we're reviewing is only part of a vendor's product line. Some vendors provide a much broader line of developer components and tools.

If a single vendor can supply multiple components that you need, you may benefit from reduced licensing fees by buying the components together in the form of a suite. This can also cut your learning curve by giving you products that follow similar conventions and that work well together.
Version Support

As of mid-2005, you might be working with .NET 1.0, .NET 1.1, a beta version of .NET 2.0, the .NET Compact Framework, or the Mono implementation of the .NET Common Language Runtime (a cross-platform open source implementation available for Windows and Linux). Although some obfuscators can handle assemblies targeted at any of these environments, others have limitations such as not being able to handle .NET Compact Framework applications or not yet being able to manage .NET assemblies. You’ll need to verify that your vendor can match all of your target environments, or that they have a plan for doing so before your planned ship date. Note that although we looked at which products support the current .NET 2.0 beta, all the obfuscator vendors we contacted promise to support .NET 2.0 when it officially ships in November. We’ll formally evaluate support for .NET 2.0 and 64-bit compilers in a future edition of this Guide.

Obfuscation Methods

Different obfuscators offer a wide variety of different methods for protecting your source code. Choosing an obfuscator is largely a matter of deciding which of these methods you want to apply to your own code. The feature charts in this Guide will help you understand which obfuscators support which methods. In this section, we’ll discuss the available obfuscation methods together with their pros and cons to help you make an informed decision.

Entity Renaming

The most basic method of code obfuscation, supported by all of the obfuscators we’ve looked at, is entity renaming. “Entity” refers to just about anything that can have a name in .NET: namespaces, classes, methods, properties, fields, and enums are all entities. As all developers know, good programming practice dictates naming your entities in such a way as to make your code easy to read and debug. For example, a method that returns the next available date for a conference room would probably have a name such as ConferenceRoom.GetNextAvailableDate.

When you build a .NET assembly, the entity names from your source code are preserved in the assembly’s metadata, and can provide valuable clues for anyone trying to understand the code. Such names don’t matter to the .NET Common Language Runtime when it goes to execute your code. From the point of view of the CLR, ConferenceRoom.GetNextAvailableDate might just as well be named ABC.DE. During the process of entity renaming, an obfuscator systematically goes through your assembly and renames all of the private entities in the assembly to short, nonsensical, or obscure names.

- **Naming Conventions** - Some products offer you a measure of control over the new names applied to entities. You may be able to choose from predetermined schemes, such as C# keywords or characters in an obscure Unicode range, or you may be able to supply your own list of replacements in an XML file or other format. Using confusing names can provide an extra measure of protection to humans trying to understand decompiled code.

- **Overloaded Renaming** - If an obfuscator supports overloaded renaming, it will apply the same name to multiple entities, as long as this is allowed by .NET’s overloading rules. For example, in a C# application, GetNextAvailableDate() and ReserveRoom(int hours) could both be renamed to ABC even though the two methods have nothing to do with one another, because they have different parameter lists. Heavily overloaded renaming can help make code much more difficult to decipher. Some obfuscators can even overload on a method’s return type, which is legal in MSIL but not in VB .NET or C#. This makes it impossible to decompile the MSIL to those high-level languages, even though it will still run fine.

- **Include and Exclude Members** - In general, it should be safe to rename private members, but not to rename public members, but there are many exceptions to this rule. For example, if you know a particular public member will never be used by another application (perhaps you only made it public for ease of unit testing) then you probably would like your obfuscator to rename that member. On the other hand, private members that are used in serialization or remoting, or that are accessed through remoting, cannot be renamed without breaking your application. Members that are accessed via reflection also cannot be renamed without breaking the calling application. To deal with these situations, an obfuscator must offer some means of including and excluding particular members from the renaming process.

- **Declarative Obfuscation** - No one knows better than the developer writing the code whether a particular entity should be obfuscated or left with its original name. Some products support a declarative obfuscation syntax that lets the developer tag methods, classes, and other entities using a .NET attribute to dictate whether and how they should be obfuscated. This allows for repeatable obfuscation that is well-documented directly in the source code.

Control Flow Obfuscation

Some obfuscators attempt to make decompiled code harder to understand by modifying the original code. For example, if you’ve developed your .NET application in a dynamic way with loops that are logically equivalent but more complex appearing by using the rules of logic. An obfuscator might also insert GoTo statements and labels to turn your neat logic into “spaghetti code” to further confuse human readers.

Removal of Unused Members

An obfuscator can also evaluate your application’s entire source code to determine whether there are any methods that are actually unused, and if so, remove them from the obfuscated assembly. This situation can easily arise if you’ve developed code with a code generator or used classes from a general-purpose utility library. For example: you might have collection classes with the full range of methods that any collection should support, but only use a small fraction of those methods in your application. An obfuscator can remove the unused code, making the remaining code harder to understand and shrinking the program as a beneficial side effect.

String Encryption

String constants and literal strings in your .NET source code appear unchanged in your compiled .NET assemblies. Such strings can provide valuable clues for reverse engineering your code. For example, an attacker trying to break a licensing routine would first focus attention on strings having to do with licensing to locate the appropriate section of code. Obfuscators can make this more difficult by encrypting the strings in the .NET assembly. This is accomplished by inserting a decryption routine into the assembly and calling the decryption code at runtime to return the original strings.

Breaking ILDASM

ILDASM is the MSIL disassembler that is a part of the .NET Framework SDK. Some obfuscators inject code into the obfuscated assembly that is designed to break ILDASM so that it won’t open the assembly at all. This technique may work for particular versions of ILDASM, but you should not regard it as a permanent fix. Microsoft has a history of revising ILDASM with each .NET release so that it opens assemblies that were previously unopenable. Also, there are several other .NET disassemblers available, and it is unlikely that any obfuscator will prevent all of them from opening an obfuscated assembly. Obviously, this becomes a cat and mouse game as obfuscators try to outsmart decompilers, and vice versa.
Obfuscation Tools for .NET

Obfuscation Features
These are features that apply broadly across many types of software.

**General Features**
- **Managed Code** - The product is 100% managed code, as opposed to unmanaged code or a managed code wrapper around unmanaged code.
- **Integrated Help** - The product provides a help file integrated directly with the Visual Studio .NET help file.
- **External Help** - The product provides an external help file.
- **Documentation** - The product provides documentation beyond a manual or code samples.
- **Peer Support** - There are peer support options such as newsgroups or discussion boards available for the product.
- **Vendor Support** - The vendor provides direct support options for the product, either as part of the purchase price or as a separate support contract.
- **.NET Framework 1.0 support** - The product includes a version that works with the .NET Framework 1.0.
- **.NET Framework 1.1 support** - The product includes a version that works with the .NET Framework 1.1.
- **.NET Framework 2.0 support** - The product includes a version that works with the .NET Framework 2.0.
- **Compact Framework support** - The product includes a version that works with the .NET Compact Framework.
- **Mono support** - The product includes a version that works with the Mono implementation of the CLR.

**Obfuscation Features**
- **Entity renaming** - Obfuscates code by renaming entities such as namespaces, types, methods, properties, and fields.
- **Control of Naming Conventions** - Allows the user to specify the scheme that will be used to generate new names for code entities.
- **Overloaded Renaming** - Uses the same name for different members that have different signatures.
- **Control Flow Obfuscation** - Rearranges control flow structures to make the application's logic harder to follow.
- **Removal of Unused Members** - Removes members that are not used from the IL.
- **String Encryption** - Encrypts strings so that they do not appear in plain text in the IL.
- **Handles Satellite DLLs** - Automatically obfuscates satellite DLLs that are part of an obfuscated executable.
- **Include/Exclude Members** - Allows the developer to select which members should be obfuscated.
- **Rules-based Configuration** - Set rules to determine which members to obfuscate rather than selecting them individually.

**User Interface**
The obfuscators we looked at offer a variety of different user interfaces. Depending on your organization's workflow, you may find one user interface more appropriate than another. Obfuscators that offer a standalone graphical user interface are convenient when you want to experiment interactively with different obfuscation settings and see the results. An obfuscator that integrates with Visual Studio .NET is useful if you work in the Visual Studio .NET interface on a routine basis. When you're ready to make obfuscation a part of your build process, you'll probably want an obfuscator that can be called from a custom MSBuild task or as a command line tool for easy automation. Most obfuscators offer more than one user interface so you can use them in a variety of different settings.

**Strong Name Support**
If you're using strong names to protect the integrity of your .NET assemblies, be aware of the way that obfuscation interacts with strong naming. Strong naming uses cryptographic techniques to verify that an assembly has not been modified after it was compiled. However, obfuscation by definition modifies an assembly and thus will invalidate any strong name. The general solution to this problem is to use the technique of delay signing, which lets you compute and apply the final cryptographic signature after the obfuscator has finished its work. Some obfuscators offer support for automatic delay signing, which makes the process of obfuscating strong-named assemblies much simpler.

**Debugging Support**
Obfuscation creates special problems when it comes to debugging code. If an obfuscated assembly crashes and produces a stack trace, the stack trace will contain the obfuscated symbols rather than the original symbols, making it difficult to determine what happened. Some obfuscators provide tools to translate an obfuscated stack trace back into the original member names, which makes it easier to work with stack traces from obfuscated assemblies. Other obfuscators fix up the metadata in an obfuscated assembly so that it can be used together with the original source code in an interactive debugging session. This can be important when you're trying to track down errors in the obfuscation process itself.

**Watermarking**
Watermarking is not itself an obfuscation technique, but it's worth mentioning because several of the products in this Guide include watermarking in their feature set. Watermarking allows you to embed arbitrary information into an assembly in an encoded form. You can use this to hide copyright information, serial numbers, or other identifying information in an unobtrusive way, as an aid to tracking any violation of your software's licensing terms.

Watermarking can be beneficial for catching violators in circumstances where they disassemble or decompile your code and then reassemble or recompile your code untouched into their own assemblies, or when they simply use your assemblies directly in a manner.

Features
The following list of features is provided by category and then by specific feature.

**General Features**
These are features that apply broadly across many types of software.

- **Managed Code** - The product is 100% managed code, as opposed to unmanaged code or a managed code wrapper around unmanaged code.
- **Integrated Help** - The product provides a help file integrated directly with the Visual Studio .NET help file.
- **External Help** - The product provides an external help file.
- **Documentation** - The product provides documentation beyond a manual or code samples.
- **Peer Support** - There are peer support options such as newsgroups or discussion boards available for the product.
- **Vendor Support** - The vendor provides direct support options for the product, either as part of the purchase price or as a separate support contract.
- **.NET Framework 1.0 support** - The product includes a version that works with the .NET Framework 1.0.
- **.NET Framework 1.1 support** - The product includes a version that works with the .NET Framework 1.1.
- **.NET Framework 2.0 support** - The product includes a version that works with the .NET Framework 2.0.
- **Compact Framework support** - The product includes a version that works with the .NET Compact Framework.
- **Mono support** - The product includes a version that works with the Mono implementation of the CLR.

**Obfuscation Features**
- **Entity renaming** - Obfuscates code by renaming entities such as namespaces, types, methods, properties, and fields.
- **Control of Naming Conventions** - Allows the user to specify the scheme that will be used to generate new names for code entities.
- **Overloaded Renaming** - Uses the same name for different members that have different signatures.
- **Control Flow Obfuscation** - Rearranges control flow structures to make the application's logic harder to follow.
- **Removal of Unused Members** - Removes members that are not used from the IL.
- **String Encryption** - Encrypts strings so that they do not appear in plain text in the IL.
- **Handles Satellite DLLs** - Automatically obfuscates satellite DLLs that are part of an obfuscated executable.
- **Include/Exclude Members** - Allows the developer to select which members should be obfuscated.
- **Rules-based Configuration** - Set rules to determine which members to obfuscate rather than selecting them individually.
Declarative Obfuscation - Mark members to be obfuscated by including special attributes directly in the source code.

Generic Support - Supports .NET 2.0 Generics.

Strong Name Re-signing - Automatically applies strong names to obfuscated assemblies.

Incremental Obfuscation - Allows obfuscating an assembly whose source code has been edited while preserving the obfuscated names of members that have not been edited.

Produces Win32 Executable - Allows creating a single Win32 executable instead of a .NET assembly as the final product of the obfuscation process.

Breaks ILDASM - Protects against decompilation with older versions of ILDASM. Because of changes in ILDASM, this may not be effective against more recent versions.

### Language Support

- **C#** - Obfuscates applications developed in C#.
- **Visual Basic .NET** - Obfuscates applications developed in VB .NET.
- **Managed C++** - Obfuscates applications developed in Managed C++

### User Interface

- **Standalone GUI** - Can be executed from a standalone graphical user interface.
- **Visual Studio .NET Integration** - Can be executed from within the Visual Studio .NET IDE.
- **MSBuild Integration** - Can be called directly from a custom MSBuild task.
- **Command Line Version** - Includes a command line version for batch or automated build scenarios.

### Other features

- **Compression** - Compresses the obfuscated assembly
- **Debugging Support** - Includes support for source-code debugging directly from the obfuscated assembly.
- **Software Watermarking** - Allows you to embed a unique watermark in the obfuscated assembly.
- **Assembly Linking** - Allows linking multiple assemblies into a single obfuscated output assembly.
- **Time-Limited Trial Versions** - Can create time-limited trial versions of your applications.
- **Reports** - Produces reports on the actions taken during an obfuscation session.
- **Suite** - Part of a suite of related products such as a decompiler.

---

Vendors/Authors Profiled in this Guide

**9Rays.Net**
http://www.9rays.net
901 North Pitt Street, Suite 325, Alexandria, VA 22314
USA

**Aspose Pty Ltd**
http://www.aspose.com
41, Lily Street, Hurstville, NSW, 2220
AUSTRALIA

**Desaware**
http://www.desaware.com/
3510 Charter Park Drive, Suite 48, San Jose, CA 95136
USA

**Jungle Creatures, Inc.**
USA

**PreEmptive Solutions**
http://www.preemptive.com/
26250 Euclid Avenue, Suite 503, Cleveland, OH 44132
USA

**PV Logiciels**
http://dotnetprotector.pvlog.com/Home.aspx
50, Avenue Clemenceau, 49280 LA TESSOUALLE
FRANCE

**Remotesoft Inc.**
Products Profiled in this Guide

- 9Rays.Net
  - Spices.Obfuscator - v4.5.0.0
- Aspose Pty Ltd
  - Aspose.Obfuscator - v1.6.0.0
- Desaware
  - Obfuscating .NET: Protecting your code from prying eyes - v1st edition
- Jungle Creatures, Inc.
  - Decompiler.NET - v1.3.1.18936
  - Deploy.NET - v1.0.0
- PreEmptive Solutions
  - Dotfuscator for .NET Professional Edition - v3.0
- PV Logiciels
  - dotNet Protector - v4.0
- Remotesoft Inc.
  - Salamander .NET Obfuscator - v2.0.0
  - Salamander .NET Protector - v2.0.0
  - Salamander .NET Linker and Mini-Deployment Tool - v2.0.0
- Wise Owl
  - Demeanor for .NET, Enterprise Edition - v4.0

Aspose.Obfuscator

Version 1.6.0.0, Free
Download it here
Licensing: Freeware

Aspose.Obfuscator provides very basic obfuscation for C#, Visual Basic.NET, or JScript.NET code (managed C++ assemblies are not supported). It renames private and internal (though not protected or public) members using a simple alphanumeric naming scheme, and it overloads member names when it can. You can specify additional member names that should not be renamed via the program's user interface, but you cannot make the obfuscator rename a public member, even if you'd like to. When you're done choosing assemblies and optionally reserving additional members whose names you want preserved, a toolbar button click starts the obfuscation process. The end result is an XML report and a log file, although in our tests the log file was always empty and the XML report did not contain details on which members were renamed to which new names.

Aspose.Obfuscator is a free product and is no longer being updated. Support is via a fairly quiet online forum at the Aspose Web site, which is also where the documentation is hosted.

Decompiler.NET

Version 1.3.1.18936, $550 per CPU
Download it here
Licensing: Proprietary, per CPU

Decompiler.NET offers obfuscation almost as an interesting side-effect. The application starts out as an extremely effective decompiler, able to take any .NET assembly and turn it into high-quality C# or IL code (with partial support for Visual Basic .NET or Delphi as well). This decompiler also loads as a dockable Visual Studio .NET add-in, which is itself very useful. Atreeview lets you drill into the assembly piece by piece, inspecting the code for classes and members as you go along. But it's a short step from there to obfuscation. Click over to the Decompiler Assembly tab, check the Obfuscate checkbox, and click the Decompile button. Decompiler.NET then goes to work decompiling the entire assembly to make C# source code from it, with the private symbols obfuscated. Pull that source into a new .NET solution, compile it, and you have your obfuscated assembly. This approach requires a bit of extra work: you'll need to set references and handle things like the build type for image resources by hand. But on the plus side, you have a chance to tweak the obfuscated source code before you build the final assembly. This also means you can debug through the obfuscated assembly easily, because you have the obfuscated source code to work with. Comments in the obfuscated source make it easy to see the original names of the various members involved.

The decompiler here does quite a bit of code streamlining, which may have the effect of making the obfuscated code more difficult for human beings to follow. Booleans are simplified and optimized and conditionals may be merged and reorganized. Dead code is removed, and local variables that are only used once will be eliminated as well. By default, Decompiler.NET automatically factors out the bodies of public members into obfuscated internal members and updates
Obfuscation Tools for .NET

Dotfuscator for .NET Professional Edition

Version 3.0, starting at $395
Download it here
Licensing: Proprietary, per developer

If you're a Visual Studio user, you're probably already familiar with the Dotfuscator name - because Microsoft has chosen the Dotfuscator Community Edition to include as a basic obfuscation solution in the Visual Studio box. For this Guide, we looked at their top-of-the-line Professional Edition (there's also an Intermediate Standard Edition), which offers much more control and many more features than the Community Edition. You can refer to PreEmptive's online comparison chart for more information about which features are in which edition. You can try out a trial version of the Professional Edition by visiting the Web site as well.

Dotfuscator includes symbol renaming, code-flow obfuscation, and string encryption to make it harder to understand your code. The renaming overloads members, including overloading on method return type and field type, for maximum confusion. The overloading algorithm that Dotfuscator uses to find the optimal amount of overloading to do is patented. Unused code is pruned, and multiple assemblies can be linked into a single output assembly, which can be automatically signed with a strong name. You can specify which members to obfuscate using very flexible and fine-grained rules, as well as with declarative attributes. .NET 2.0 support is included in this version, including support for Friend assemblies and Generics.

Dotfuscator keeps track of the name mapping that it generates in an XML file. This has several benefits. First, it can use this file to perform an incremental obfuscation, keeping obfuscated names stable when re-obfuscating an edited assembly. Second, it can use the file to decode an obfuscated stack trace, making it possible to see what happened when an obfuscated application failed in the field. For in-house debugging, Dotfuscator outputs PDB files that let you step through the obfuscated code while viewing the original source code. You can invoke Dotfuscator in a variety of ways, including from a standalone GUI, from an integrated Visual Studio project type, from an MSBuild task, and from the command line.

Dotfuscator includes the ability to "watermark" assemblies, embedding character information in them as a way of identifying your intellectual property; a command-line tool can extract these watermarks. Everything is very configurable, and the application didn't have any trouble with any of the assemblies that we threw at it in testing. Registered users get access to a private support site with a variety of resources for one year; extended support contracts are available for purchase.

dotNet Protector

Version 4.0, $395
Download it here
Licensing: Proprietary, per developer

The product dotNet Protector is designed to protect executable .NET assemblies by compressing and encrypting them, and
Obfuscation Tools for .NET

Obfuscating .NET: Protecting your code from prying eyes

Version 1st edition, $39.95
Licensing: Proprietary, per reader

Obfuscating .NET is a combination e-book and open-source obfuscator. The Desaware QND Obfuscator is itself issued under the Mozilla Public License, but in order to get the download location, you need to buy a copy of the book. Readers are invited to contribute improvements to the code back to the project, but there doesn't seem to be any sign of that happening yet. The QND Obfuscator itself takes a simple and unique approach to the problem of obfuscation. It simply finds every private symbol in the string heap in the PE file manifest, and changes them all to the "$" symbol. You can use attributes to prevent this from happening for particular symbols if you need to, but otherwise you effectively overload everything - indeed, to the point where the resulting executable violates the CLI specification (as author Dan Appleman points out in the e-book). This means that code decompiled from the obfuscated assembly has zero chance of compiling, but at the same time, you're depending on undocumented behavior to keep your code running. Our test assemblies seemed to work fine, but this is something to be aware of.

The greatest value here may be the technical discussion of writing the obfuscator, as well as the annotated source code showing how it works with the internals of the PE file. Of course, if this particular obfuscation strategy works for you, it's also nice to have the source code to use and improve. There's a detailed discussion in the book of its pros and cons as well. There is no support available.

Salamander .NET Obfuscator

Version 2.0.0, $799/5 developers
Download it here
Licensing: Proprietary

Remotesoft's Salamander .NET Obfuscator provides both a GUI environment (Remotesoft .NET Explorer) for exploring and obfuscating .NET applications and a command-line version of the obfuscator for easy including in automated build processes. The obfuscator offers a high degree of customization and the GUI makes it easy to adjust your settings until the obfuscated application works properly. When you've got everything set properly, you can save an XML configuration file to drive the command-line utility. Remotesoft .NET Explorer also has many other uses: it serves as a disassembler and decompiler, it can map obfuscated member names to their unobfuscated counterparts and vice versa, and it can show you the details of both .NET metadata and PE file headers, among other things.

Salamander .NET Obfuscator operates directly on your original binary exe and dll files, without altering debug and line number information. This makes source-level debugging easy; you can just drop the obfuscated assembly in place of the original, invoke the debugger, and go into your original source code. It can handle mixed-language applications, including managed C++ code, and also .NET Compact Framework assemblies. Satellite and resource assemblies are automatically obfuscated in most cases, although the manual includes instructions for manually configuring the obfuscator for special needs. In general, the manual offers good coverage of the edge cases where obfuscation can cause problems, including reflection, remoting, and serialization, and shows - with examples - how to configure your projects to avoid issues in those areas. In addition to manual configuration or using XML configuration files, there's also a complete attribute-based configuration system that lets you specify the details of obfuscation directly in your code.

Overall, Salamander .NET Obfuscator appears to work well and the user interface makes a good deal of data on your assemblies readily accessible. There's a trial download with full functionality available on the vendor's Web site. Pricing includes one year of free online support and upgrades.

Salamander .NET Protector

Version 2.0.0, $1899/5 developers
Download it here
Licensing: Proprietary

Salamander .NET Protector licenses include a copy of Salamander .NET Obfuscator, so when you purchase this product you get a superset of the Obfuscator functionality. Protector adds several new features to the mix designed to make it even harder to reverse-engineer your application. Patents go through your assemblies and replaces most of the MSIL code with native code, leaving only stubs of MSIL behind to call the native code. Thus, there's nothing left behind for tools like IDA/Reflector to get their teeth into, and you're protected from decompilation or disassembly. The tool also protects embedded strings and resources from easy retrieval, and includes hooks to let you call your own password protection or licensing code from the protected assembly for added security. Like Salamander .NET Obfuscator, Protector runs as either a command-line tool or as part of the Remotesoft .NET Explorer. There is a trial version available. Purchases include one year of support.

Salamander .NET Linker and Mini-Deployment Tool

Version 2.0.0, $489/5 developers
Download it here
Licensing: Proprietary

The Salamander .NET Linker and Mini-Deployment Tool is not itself an obfuscator, but we're including it here because it can be used as part of a successful obfuscation strategy. This tool can be used to link together multiple assemblies into a single executable - including .NET Framework assemblies. It includes the parts of the Framework that your code actually uses, reducing the size of the package that you need to deploy to your customers. More importantly, from the
Obfuscation Tools for .NET

point of view of obfuscation, this makes it possible to obfuscate calls to methods in the .NET Framework. In the general case, obfuscators cannot change the names of calls to Framework members such as MessageBox.Show, because they're external to your assembly. When you use this tool, such external members are pulled into your own code, and the resulting merged assembly can be entirely obfuscated, including the Framework members. The Linker and Mini-Deployment Tool integrates into the Remotesoft .NET Explorer shell, and is also supplied as a command-line version for use in automated build processes. There is a trial version available. Purchases include one year of support.

Spices.Obfuscator

Version 4.5.0.0, $392.95
Download it here
Licensing: Proprietary

Spices.Obfuscator is part of a suite of products from 9Rays, including Spices.Decompiler, Spices.Documenter, and Spices.Modeler. They all work together in the same graphical shell, and are also integrated into Visual Studio .NET through the optional Spices.VSIP.Net package. The obfuscator uses its own metadata engine to work directly with assemblies, supporting .NET 1.0, 1.1, 2.0 and the .NET Compact Framework, including assemblies built with managed C++ code. There's also a console version for use in automated build processes. The obfuscator renames members according to its own internal algorithm or according to a symbol map that you supply. You can also control the renaming through the user interface or via attributes in your code. Additional obfuscation features include string protection and code designed to break ILDASM and other decompilers.

Spices.Obfuscator supports incremental obfuscation through saving and reusing an obfuscation map. It includes support for strong name resigning, assembly compression, and automatic interpretation of obfuscated stack traces (just paste the stack trace into the graphical user interface and you can see the original method names). By default, it strips out debug information, but you can choose to leave debug information in the obfuscated assembly to locate problems when the obfuscation process doesn't go quite right. There's also a software watermarking implementation to let you embed information in the obfuscated assembly.

You can download a functional evaluation copy, although it only obfuscates a random selection of the members in each assembly. Spices.Obfuscator comes with an extensive help file, but the help is sometimes confusing - it's obvious that English is not the native language of the help file's author. Support is available via newsgroups or directly through e-mail.

General Criteria

What things should you consider when purchasing any type of third party development tool or component? Or when deciding between purchasing such a component and building it in-house? Here are some factors you might like to think about:

- Can you download an evaluation or demo copy prior to purchase?
- What are the limitations to the evaluation or demo copy?
- What are the company's tech support policies? How is tech support offered (phone, e-mail, newsgroup, discussion board)? During what hours? What's the average response time? Is an answer guaranteed? What does support cost? How long do you get free support? Can you purchase a support contract and what will it cost if you can?
- What are the company's return policies?
- What architecture is the product? Pure managed code, pre-wrapped ActiveX, wrapable ActiveX, non-managed code designed to be called directly from managed code?
- What other products does this vendor offer that you might need?
- Can you develop the functionality you need in a cost effective time-frame? Do you want to support the functionality after you first develop it?
- If you build yourself, how many other developers will be using it or will it just be you? If more than just yourself, do you have the time to write good documentation?
- Does the product support relevant standards where applicable, or does it use propriety implementations?
- If open source, is the project active? If not, you might end up supporting and/or enhancing it yourself.
- If commercial, how long is the vendor? How long have they been in business? How focused is their product line? How long have they supported this product? Have they ever dropped support of other products? Do they offer source code?

Note: Be careful with very large vendors that are not focused in the area of your interest. Large vendors have a bad habit of becoming interested in developer tools yet quickly drop support when they realize how hard it is to make money selling components and tools to developers. Exceptions are when the developer tools are their core competency or support their strategic direction. Conversely, don't discount small vendors if they have been in business for a while and have shown a proven ability to focus and provide quality products.
Feature Tables

Click here to view the Feature Tables.

Products Not Profiled in this Guide

These are the products appropriate for this Guide but for one reason or another we did not cover. In some cases, we simply didn’t locate the products in time to include them in this edition of the Guide, and hope to add them to the next revision. In others, they declined to participate and refused to send us a license for evaluation. We list these products and their URLs here so you can research them on your own if you like.

*Dynu .NET Obfuscator*
http://www.dynu.com/
The vendor did not respond to our requests for a review copy of the software.

*LSW IL-Obfuscator 2.0*
http://www.lesser-software.com/
The product is still in beta testing and not yet ready for review.

*XenoCode*
http://www.xenocode.com/
The vendor did not respond to our requests for a review copy of the software.

Glossary

- **Disassemble**
  To convert MSIL into assembly code.

- **Decompile**
  To convert compiled code (such as compiled MSIL) into a high-level language such as C# or Visual Basic .NET.

- **ILASM**
  The MSIL Assembler tool, which converts MSIL assembly code into compiled MSIL. It is included with the .NET Framework SDK.

- **ILDASAM**
  The MSIL Disassembler tool, which converts compiled MSIL code into MSIL assembly code. It is included with the .NET Framework SDK.

- **Microsoft Intermediate Language (MSIL)**
  The universal language of .NET assemblies, into which all high-level .NET languages are compiled. The .NET Common Language Runtime (CLR) uses Just-In-Time compilation to convert MSIL to machine code when you execute a .NET assembly.

- **Obfuscate**
  To transform a .NET assembly in such a way that it remains valid MSIL while making the disassembled or decompiled version more difficult to understand.

- **Overloaded Renaming**
  To use the same name for two or more members of the same assembly during obfuscation.
About the Guides

The Guides Process
We begin the process of creating a Guide begins by identifying a product category we believe will be of interest to many .NET developers. Next, we identify every product we can find that fits into the category. We then invite commercial products to participate and require them to supply our writer(s) with a fully-licensed copy of their product(s). In the future we also plan to require a small research fee to help cover paying our writers, not to exceed US$500, but for this first guide and probably the next several we waived that fee. In addition when applicable open source and public domain product exists, we include them in our Guides and waive the participation fees for these products.

Editorial Policies regarding Vendors
At the beginning of the process, vendors are invited to submit their opinion of what a developer would need to know in order to select a product in the category. Our writer(s) then take that information and do their own research to produce an initial draft of a Guide. We then give vendors an opportunity to review this initial draft and to submit factual corrections to the sections regarding their own products before publication. The final draft of the Guide is then prepared by our editorial team and submitted to the printer. We do not allow vendors to provide input to final version of the Guide before it is published. In additional, our policy is that once a vendor agrees to participate in a Guide, our decision on how to cover their product and what to say about it is final.

Funding the How-To-Select Guides
For the first few guides we made available only a single exclusive sponsorship to vendors in the Guides. These vendors supply their own promotional material for the sponsorship section as advertising content. On an ongoing basis we will solicit advertising in the Guides and allow us to to fund the level of research needed to make our Guides able to meet the needs of the most demanding developer.

Feature Charts
The feature charts in this Guide represent our best effort to pick out some of the most important features for this class of software and to indicate which products support which features. It's impossible to include every feature of every product in these charts, and it's impossible to capture the subtle differences between products with such crude delineations.

Editorial Policy regarding Published Prices
Many of the products that we cover have quite complex pricing structures. For example, there may be separate prices for developer, redistribution, workgroup, and server licenses, for individual and quantity purchases, for named and floating licenses, and for perpetual or annual licenses. There may also be separate support or upgrade fees. In addition, many vendors offer discounts when you purchase their products through a reseller, or make promotional pricing available under certain circumstances.

Our policy is simple: we list the basic undiscounted list price for the least expensive fully-functional version of each product that we include in a Guide. You should always check directly with the vendor, or with your reseller of choice, for the most complete and current pricing. All prices are in U.S. dollars unless otherwise indicated.

Reader Feedback
While we strive to make each How-To-Select Guide as complete and correct as possible, we recognize that nobody's perfect. Products change, people invent new techniques for solving old problems, and sometimes we just make mistakes. That's why we review each Guide on a regular basis. And that's also why we invite your feedback to help make the next edition of this Guide even better than this one.

How-To-Select Guide Reader Forums
One way to provide feedback is to participate in our forums. You can visit our forum for the How-To-Select Guides at:

http://forums.howtoselectguides.com

Any content related to our Guides is welcome, but we're especially interested in the following areas:

- What questions do you have that aren't answered in the Guide? We'd love our best to answer them!
- What new scenarios are you finding in your own work that we ought to include in the next edition?
- What other products should we consider including in the Guide?
- What areas of the Guide need correction or clarification?

In addition, we'd like you to upload your own "How-To-Select Case Studies." If you evaluate two or more of the products in this Guide, please take a few minutes to write up your scenario, what you learned, and why you chose the product that you did. Your fellow developers will thank you!

E-mail Feedback
You can also send your feedback directly via e-mail. If you've got feedback on any of our content, or just want to get in touch to tell us what you thought of our coverage, we'd love to hear from you. If you're willing to have your feedback reposted to the online forum, you can contact our writers and editors at feedback@howtoselectguides.com. If you prefer, you can use feedback-private@howtoselectguides.com to send private feedback that we won't repost to our forum. We regret that we cannot promise a personal reply to every comment, but we will carefully consider all feedback in preparing the next edition.