.NET Assemblies protection

Sequence of protection

Two console utilities are used for the automatic protection of .NET assemblies:

Utilite	Purpose
CodeObfuscator.exe	MSIL simbol obfuscation and string encryption
CodeProtect.exe	MSIL code encryption

You can use the **CodeObfuscator.exe** and **CodeProtect.exe** utilities collectively or use one of them separately. However, a specific sequence for using the utilities must be followed – first the application needs to be obfuscated and only then proceed with code protection:

- 1st stage of protection: obfuscation and encryption of .NET assemblies
- 2nd stage of protection: code protection of .NET assemblies

Principle of protection

A twostage protection approach is used for the automatic protection of .NET assemblies with each stage performing its own tasks in the overall process:

- MSIL code symbol obfuscation
- Transfer of a portion of MSIL-code into a protected storage

This concept permits a significant increase in the overall level of .NET protection, since the widely distributed .NET reverse engineering tools (ildasm, reflect or.net, etc.) become useless.

It is predicated upon the fact that most of .NET assembly's code is stored in the protected native container, which in turn is protected by both pseudocode (software) and the dongle's functionality (hardware solution).

When MSIL code previously encrypted by a hardware algorithm is called, the dongle itself is addressed first for decryption purpose and only after the execution of code begins.

Limitations of protection

1. Assemblies with mixed code and multimodular assemblies are not supported.
2. See the limitations for autoprotection of executable Native files.

Guardant dongle is a highly effective mean of software/hardware protection. It allows building protection of virtually any level of complexity and tamperproofness.

Use the symbol obfuscator and MSIL-code protection utility. It is also reasonable to use the following options:

- **Symbol obfuscation**. It allows obfuscating the application code, which when coupled with other measures significantly increases the level of application protection. Use the obfuscation for all *.exe and *.dll assemblies in the application, except for the third-party applications and applications signed by digital signature.
- **String encryption**. Ties the obfuscated application to the Guardant dongle and encrypts the string constants in the protected assemblies. It is recommended to use this option if there are no logical elements, significantly depending on the speed of string constants.
- When the **public interface obfuscation option** is used a complete obfuscation of all the assembly occurs. However be careful, the use of this option is possible only with the looped system (all assemblies are obfuscated in one session and no methods of obfuscated assemblies from other applications are not used). In most of the cases it is safe to conduct the obfuscation of public interfaces for exe-assemblies (except for the cases of using the Reflection technology on the assembly itself or exporting types for other applications).
- **Use the exclusions file**. Use ExclusionUtility.exe utility in the Developers' Kit to generate this file. Generally, the developer needs to have a clear understanding of types, methods and properties that need to be included as exclusions for the symbol obfuscator. As a rule, these are all the language elements that can be used from within. You should pay attention to the use of Serialization, Reflection, Data Binding technologies.

You have to remember the following when using the MSIL-code automatic protection utility:

- Wisely use the **option indicating the percentage of protected methods**. The large and more complex the protected application, the less is the necessity to indicate the percent. Generally, you may set 100 percent for the most simplistic assemblies not sensitive to the execution speed, containing 2-3 types with 10-15 methods each. For large projects the % should be lowered to 10.

- Do not protect assemblies, types and methods **sensitive to the execution speed**. Remember that MSIL-code transfer to the encryption area and execution of it on a virtual machine may in some cases slow down its execution (especially during the first call). Further calls may also incur some overheads, therefore, the less the size of the method, the more obvious is the execution speed loss.
- Use the **option and utility for setting exclusions**. Exclude from protection all small methods not containing logic and holding commercial value.
- When working together with symbol obfuscator you need to use **MAP-file creation option** (see **Of the present documentation for details**).
- Exclude all language elements based on **asynchronous data transfer**. MSIL-code protection of such elements may lead to unpredictable results.