

The unbearable lightness of tampering .NET apps

As many of you probably already know .Net apps are not distributed as a binary code but uses a specialized format called 'MSIL' which is an abbreviation for Microsoft Intermediate Language.

The idea behind the MSIL format is to provide a machine independent distribution format that promotes the ability to run .Net apps cross platform. This is done by the CLR (the .Net runtime engine), specifically by a 'just in time' compiler (a.k.a jitter). The CLR hands the MSIL code to the jitter which in turn compiles the code to its native format, then the code in its native format is handed to the CPU for execution.

Unfortunately this important virtue make .Net apps vulnerable to malicious reverse engineering attacks. In fact it's so easy to tamper that one might consider whether taking advantage of the framework is worthwhile taking into account the risk involved in exposing its code to a potential hacker.

Lutz Roeder's Reflector, Microsoft's ILDASM and other .NET decompilers are all capable of exposing the IL code stored in your assemblies to the prying eyes of a potential hacker – this includes all the metadata, external references, and so on. At this point your program can be studied, hacked, changed and reassembled (with ILASM).

A popular code protection technique that is used in the industry today is code obfuscation. Code obfuscators transform an application into one that is functionally identical but is much harder to understand. This is done by renaming meaningful symbol names such as variables, fields and method names with non-meaningful ones. Code blocks are rearranged thus making it harder to infer program logic.

While code obfuscators raise the bar for understanding program logic they have some major drawbacks.

First, they don't rename public method names since such methods can be called from other assemblies referencing the obfuscated assembly.

This means that inter-assembly calls can be traced easily when looking at the MSIL code of an obfuscated assembly. Licensing routines can be traced down easily and removed from the source code thus bypassing the program's licensing mechanism.

A second drawback is the issues obfuscation tools introduce when reflection is used. Methods calls that were performed through the usage of reflection are likely to fail once the application has been obfuscated, this happens since the method has been renamed by the obfuscator but the call site still refers to the method by its original name.

Sure, many obfuscators allow the user to define the methods that shouldn't be renamed by it but this raises the overhead on the R&D team as well as the QA team that now has to cope with bugs introduced as a result of obfuscating the code.

A third drawback is the ability to trace bugs once they are reported from the field. The ability to recover stack dump information through the usage of `System.Exception.StackTrace` method is essential for tracing down the source of bugs once the application has been deployed. Imagine that a user sends you a bug report with stack dump information that reads as follows:

`System.NullReferenceException: Object reference not set to an instance of an object.`

```
at a.a()  
at a.b()  
at a.c()  
at a.d()  
at a.a(Object A_0, EventArgs A_1)  
at System.Windows.Forms.Control.OnClick(EventArgs e)
```

This tells very little about the source of the problem, and can make R&D wonder where the problem originates from.

The fact of the matter is something that must be understood very well by companies interested in protecting their intellectual property: It doesn't matter which obfuscator your product was mangled with. If it compiled to a .NET EXE or DLL, ILDASM will disassemble it and all your work is exposed to the viewer in CIL. In this domain, you remain utterly unprotected.

CliSecure is the only technology that offers real protection for your intellectual property, in contrast to an obfuscator it makes it impossible to reconstruct your program logic.

This is done by encrypting the IL code stored in your assemblies, during execution the code is handed to the jitter in its decrypted form just before compilation occurs. The CliSecure execution engine assures that at most a single method will reside in memory in its decrypted form in any given time. This makes it impossible to reconstruct your original assembly even through the usage of memory dump tools.

Through the use of a command-line utility CliSecure takes your assemblies as input and applies a security envelope on them. The application is then distributed to the client with an additional library component.

The protection scheme employed by our product generates valid .NET assemblies, Class, method, and other symbol names are kept intact thus ensuring that your code behaves exactly the same as the original. The hosting environment is not restricted by any means, this makes CliSecure the perfect choice for component developers, Asp.net applications as well as winforms based applications.

For more information about 'CliSecure' you are welcome to log on to our web site on <http://www.secureteam.net>