

Tamper Notification

The Fourth Dimension of Enterprise Obfuscation

Publication Date: March 8, 2007

Abstract

This white paper enumerates the essential characteristics of an enterprise obfuscation solution including recent innovations that enable obfuscated applications to self diagnose tampering and report back to both application publishers and consumers.

Specific topics include:

- A summary of obfuscation technical capabilities and considerations,
- Application lifecycle workflow and process requirements,
- Guidance as to when enterprise obfuscation can be an effective IT control, and
- Extending obfuscation from reverse engineering prevention to tamper detection and notification

The three dimensions of enterprise obfuscation

Reverse engineering has become a common practice for support, training and debugging of .NET and Java applications. However, unmanaged access to application source code can also pose material risks including Intellectual Property (IP) theft, application vulnerability exposure and software piracy. For those organizations where these risks must be managed, there is really only one option; obfuscation.

At its core, obfuscation is defined as a collection of transformations that are applied to compiled applications that make reverse engineering materially more difficult for people and machines but do not alter the behavior of the obfuscated application. However, the heightened emphasis on application security, compliance and development best practices has rendered this definition inadequate. A complete definition has three dimensions; technological, as development process and as an IT control

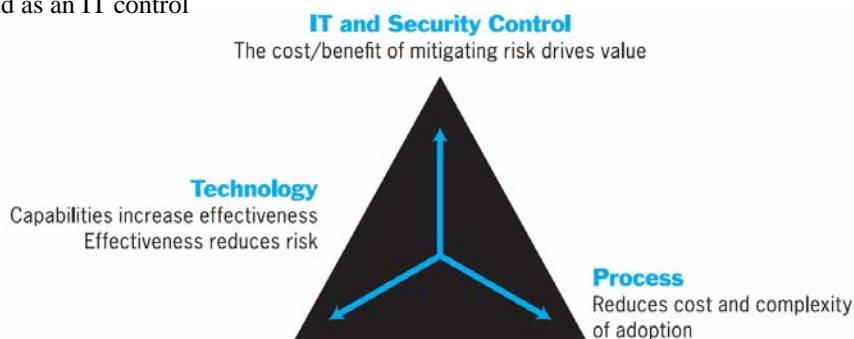


Figure I: The three dimensions of enterprise obfuscation

Technology

Obfuscation transformations fall into a number of categories including;

- Renaming: altering the names of methods, variables, etc. to make source code more difficult to understand. Strong renaming algorithms use overloading to reuse names forcing every line to be analyzed.
- Control flow obfuscation: logic and flow are re-expressed making translation into valid C# (or any other language) impossible. Sophisticated approaches provide different levels to strike the right balance between obfuscation and performance.
- String encryption: strings such as login prompts, SQL queries, etc. are encrypted and decryption function calls are injected into the instruction stack before the string is needed.
- Other: there are numerous other techniques including metadata stripping, application watermarking, etc. that raise the bar for reverse engineering above what is required to reverse engineer native code (such as C or Cobol).

Process

The process of obfuscation has emerged as the pivotal element in driving the viability and the value of obfuscation. Without a well-defined and integrated obfuscation process, the complexities and risks introduced by obfuscation may ultimately outweigh the perceived benefits that it promises. Obfuscation can complicate debugging, patch generation and management, distributed development practices and the reuse of libraries, components and web services. However, tight integration with development platforms (such as Visual Studio), the inclusion of tools and utilities that can unwind and/or reuse obfuscation transformations and lastly the integration with operations management platforms (such as Microsoft's Operations Manager) can mitigate these potentially costly side-effects.

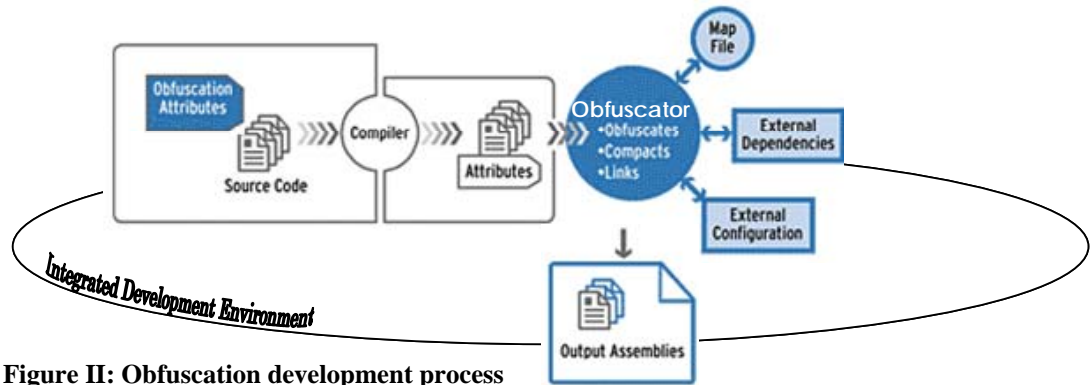


Figure II: Obfuscation development process

Figure II demonstrates a modern approach to enterprise obfuscation. The developer best practice is to have developers indicate where obfuscation transformations may or may not be appropriate. Reflection may confuse renaming transforms, high performance algorithms may call for less aggressive control flow settings, etc. Obfuscation transformations are applied after the build and do not require access to or in any modify source code. During the obfuscation transformation, additional services such as compaction (stripping of unused code to reduce size) and linking (combining multiple DLLs into one to simplify distribution) can also be applied.

Further, all transformations should be captured (and previous transformations reused as appropriate) to support the many development scenarios outlined above. Again, all of this should be embedded within the integrated development environment to ensure automation, transparency and quality.

IT control

Obfuscation can be defined as a compensating¹, detective² control to manage risks stemming from unmanaged source code distribution. These risks include an increased likelihood of system attack, theft of intellectual property and revenue loss through circumvention of usage and other metering enforcement. An obfuscation control includes the documentation of obfuscation processes and include the specific risks that are being managed and the training/communication activities that ensure obfuscation is being applied appropriately (e.g. not over- or under-used) and consistently (using approved technologies and practices).

Obfuscation is used to control the distribution of source code to communities that have access to binaries but not to original source code. In environments like .NET and Java, where extraction of source code from intermediate code is simple and well-understood, obfuscation is a common antidote to this source code access control gap.

The fourth dimension of enterprise obfuscation: *Tamper Notification*

If you invest in fire prevention – don't you also want fire detection? If an organization cares enough to invest time and resources into reverse engineering prevention, wouldn't that same organization want reverse engineering detection?

¹ "Compensating" refers to the fact that obfuscation is a control that compensates for a gap in a pre-existing control, e.g. access control for source code.

² "Preventative" refers to the fact that obfuscation is a control that prevents an incident (reverse engineering) from occurring. Controls either prevent or detect incidents.

Four simple steps can extend obfuscation to provide tamper notification services enhancing obfuscated applications to be able to self-diagnose and report on tampering whenever and wherever it may occur. Figure III illustrates these steps.

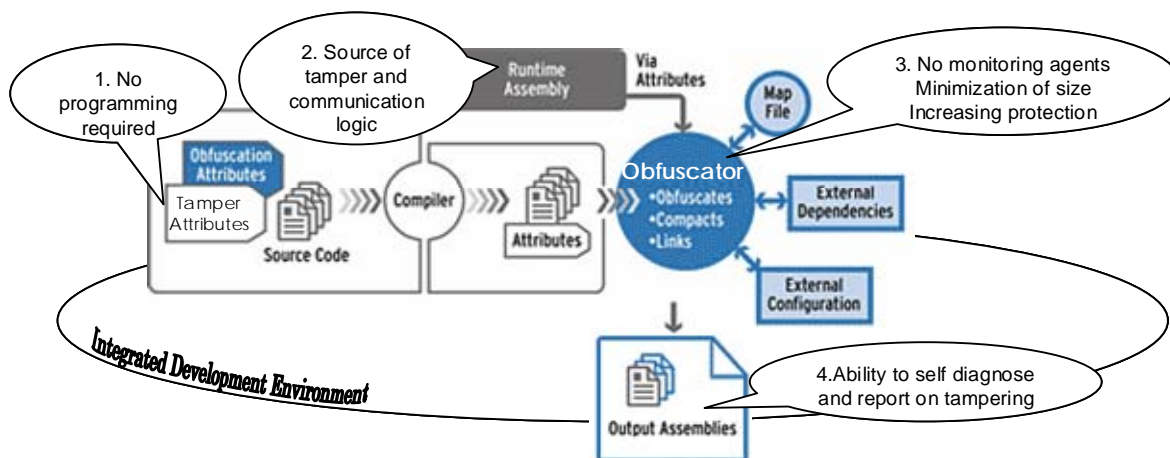


Figure III: 4D obfuscation development supporting Tamper Notification

Step 1: extend custom attributes (or annotations) to enable developers to indicate program inflection points enabling the obfuscator to instrument the obfuscated binary. There is no programming required. This reduces the effort required to implement this service and reduces training overhead.

Step 2: Tamper detection and communication logic is delivered as a part of the obfuscator in a runtime library. This logic is then injected into the compiled and obfuscated application after the build. This improves both the quality and the consistency of transmitted data. How and where this logic is injected is constrained by the developers using the attributes in step 1.

Step 3: The obfuscator uses linking to weave the additional DLLs into the final binary and compaction to minimize any potential increase in application that may result.

Step 4: The result is a single, agentless executable that is able to diagnose its own tampering and report back to both its developers and its enterprise owners.

Those who have lost sight of the fact that the organizing principle behind obfuscation is the requirement to manage risk also undervalue the importance of the obfuscation process. The result is a one dimensional view of obfuscation as a finite set of technology limited to the transformation of application binaries.

Yet, the emergence of Service Oriented Architecture (SOA) and Software as a Service (SaaS) combined with a growing recognition that the most effective IT controls connect the development lifecycle with operations management has made it possible for the first time to not only prevent reverse engineering but also to detect it if and when it should occur.