



Software Obfuscation

“To make so confused or opaque as to be difficult to perceive or understand.”

Why would anyone want to do this to medical device software? Surprisingly, it's not what you might think. Software obfuscation is a software engineering technique to make compiled software source code so confusing or opaque that no one will want to make the effort to reverse engineer it. It is routinely used in everything from highly sensitive government software to packaged consumer software. It is however shocking in its relative absence from the medical device industry. Considering the criticality of protecting patient information, the inordinate number of patent disputes and the increasingly global nature of the medical device industry, shocking is perhaps an understatement. According to a recent survey by PreEmptive Solutions, a provider of code obfuscation tools 7% of all “life sciences” companies utilize software obfuscation. Given that medical device companies are a sub-set of life sciences, that percentage is even lower.

Software obfuscation is a method for protecting a medical device or life sciences company's Intellectual Property (I.P.). Many medical device companies have invested millions of dollars in R&D to create breakthrough software applications and systems. They've paid millions more in legal fees to obtain patents on this software. Yet only a handful of these companies have spent a single dollar on a well known software engineering technique known as software obfuscation. Software obfuscation is a set of processes, which are performed by software developers using sophisticated software tools such as Dotfuscator by PreEmptive Solutions, which makes reverse engineering of important areas of source code exceptionally difficult. It is the use of a well known technology designed to protect a company's software.

Even if a medical device's software is protected by a US patent, the market and more importantly, the competition is increasingly global. Protecting patents in the US and Europe is expensive and software patent disputes drain executive time and financial resources. Without implying any nefarious intent upon the offending party, it is theoretically possible that they obtained the software methods of the patent holder by a simple process of reverse engineering. It is a simple process because the patent holder allowed it to be exceedingly simple. By not utilizing code obfuscation, the offending party was able to see exactly how the patent holder performed every software function and wrote different code that performed the same functions. Rarely is a software patent infringement case based on an exact copy of the patent holder's software.



Full Spectrum Software

Generally software patent disputes are related to “substantially similar” software technology. There is no question that some percentage (perhaps a large percentage) of software patent disputes could be avoided entirely if the patent holder used software obfuscation to protect their code.

Although extremely rare, one of the most infamous cases of software infringement involving an exact copy of software was Apple Computer vs. Franklin Computer. Franklin Computer was attempting to market an “Apple compatible” personal computer. In that landmark case, Franklin Computer had indeed copied Apple’s software, line for line. In very short order, a software expert was able to show the jury the exact same code side by side. Apple won the case summarily and Franklin Computer went out of business in record time.

In emerging markets, US patents are difficult and sometimes impossible to defend. Here it is not merely desirable, but essential that the software be protected by code obfuscation. So how does software obfuscation protect a company’s source code?

To illustrate the concept of code obfuscation, there is a very simple and well known snippet of source code that tells the computer to print the words “Hello World.”

It looks something like this: `System.out.println("Hello World!");`

If this code were obfuscated it would contain a series of complex mathematical equations, such that only if the equations were calculated perfectly would the computer print “Hello World.” In order to reverse engineer this extraordinarily simple one line command, a hacker would need to spend hours figuring out the equation, doing the math to arrive at the correct number and then he might have a chance of reverse engineering this line of code.

Note that the code can still be reverse engineered, but it is extremely time consuming. Now imagine a very large base of source code with very large sections of that code base obfuscated. Code obfuscation is not a magic bullet that makes it impossible to reverse engineer and essentially “steal” critical functionality from a medical device maker. What it does do is make it exceptionally difficult and extremely time consuming to steal code. In the best case scenario, a medical device maker makes their code so difficult to reverse engineer that the copycat company decides it will be less expensive and time consuming to write their own (and hopefully inferior) software.

There are some basic concepts in understanding code obfuscation. The first is that there are two basic types of software languages. For example, software written in C or C++ is transformed into machine code through a compiler which produces a binary file. Reverse engineering a binary file is extremely time consuming and a technically difficult process and much of the information about the high level design and structure of the original source is irreversibly destroyed in the compilation process.



Full Spectrum Software

In contrast, software designed for the .Net Framework (or Java) is compiled into an architecturally neutral language known as the “common intermediate language”, or CIL or IL. .NET code is essentially an “open book” and this is the trade-off that allows interoperability between languages, machines and platforms. In effect, many of the advantages of .NET, and they are significant, also preserve much of the information about the original source code. Microsoft recognized this vulnerability when they released .NET and chose to include an obfuscation tool from Preemptive Solutions.

<http://www.PreEmptive.com/>

Code Obfuscation Technology

The goal of obfuscation for the medical device and life sciences industries is to prevent a specific category of access control violations, specifically reverse engineering to gain access to source code. In order to be effective, obfuscation must make reverse engineering materially more difficult without altering the software’s performance or legitimate access control profiles. In practice, obfuscation is a general term that refers to a collection of techniques that meet the dual requirements mentioned above. Obfuscation works by removing the context that humans and de-compilers use to understand the functionality of a program. Consider the following two versions of the same email message, responding to the question, “How many tickets should I buy?”

```
From: Mark
To: Bill
Subject: RE: How many tickets should I buy?
Bill, I already bought 15, so all we need are 7 more.
Look forward to seeing all of you tonight as we agreed.
```

Non-Obfuscated

```
From: Mark
To: Bill
7
```

Pseudo-Obfuscated

Both questions can be delivered and answer Mark’s question, but the “obfuscated” version has been stripped of its entire context. An unauthorized reader would not know that this is a reply to a question or that Mark is being asked to buy 7 tickets or that 15 tickets have already been purchased and that the entire group is meeting later that night. In its simplest form, this is how obfuscation works – by applying multiple transformations that strip away information and alter structure to make the code less understandable to humans and de-compilers while still preserving integrity and behavior.

Code Obfuscation Transformations

Technically, code obfuscation transformations fall into a number of categories including:

- **Identifier Renaming:** altering the names of methods, variables and other code components to make source code more difficult to understand. Strong re-naming algorithms use a technique called “overloading” to re-use names, forcing a potential hacker to have to analyze every single line of code. For example, “GetImage” becomes “a”. Advanced re-naming capabilities overload these new names on a massive scale by tracking the scope of each identifier and re-using names on multiple identifiers where there is no overlap. This increases security because every identifier must be individually analyzed since every use of “a” may in fact be an entirely different identifier re-named to the same new name.
- **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. This is a critical factor in medical device and clinical software as the most sophisticated algorithms are often optimized for the highest possible speed. Yet they are often the most sensitive and valuable algorithms which a company may employ. Developers must decide how much, or how little obfuscation to apply to speed sensitive routines. Control Flow obfuscation transforms the original high level source flow into a sequence that closely mirrors the original source code’s sequence. While it is a logically equivalent sequence it has the intended side effect that the new sequence will not de-compile back to the original source. Often, de-compilers will output incorrect code, crash or simply terminate.
- **String Encryption:** strings such as login prompts, SQL queries, etc. are encrypted and decrypted function calls that are injected into the instruction stack before the string is needed. For the medical device and life sciences industries, this is another critical factor in protecting patient or sensitive clinical information.
- **Other:** there are numerous other techniques including meta-data stripping and application watermarking that significantly raises the bar for reverse engineering far above what is required for other types of native code. For example, on the .NET platform, some properties, events, method parameter names and custom attributes can be stripped out, removing relevant information about the development process and developer intent without impact impacting execution.

Transformations as Collections

From a technical perspective, obfuscation 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, one needs to take it one step further. A complete definition actually has three dimensions, in that code obfuscation is a technology, a development process and an IT control.

The technology offers increasing effectiveness such that effectiveness reduces risk. The development process, utilizing best practices, reduces cost and complexity of adoption. And finally IT and security control mitigate risk and thereby driving overall value.

In reality, there is another dimension and that is tamper notification. This is particularly important to medical device and life sciences companies. Tamper notification is another aspect of code obfuscation that helps companies know when someone has attempted to hack or reverse engineer their code. Tamper notification does depend on the device being networked and having access to the internet. While there are still some stand-alone devices and software thieves would know enough to disconnect the internet access, tamper



Full Spectrum Software

protection is still an important security issue for patient information protection or other high value data.

A good analogy is that if you invest in fire prevention (reverse engineering) don't you also want fire detection (reverse engineering detection)? Tamper protection and the communication logic is delivered as part of the obfuscator in a runtime library. This logic is injected into the compiled and obfuscated application after the build process. The obfuscator uses linking to weave the additional DLLs into the final binary and uses compaction to minimize any effect on the application itself. The result is a single, agentless executable that is able to diagnose its own tampering and report back to its developers.

Process

Most importantly, layered on top of these three dimensions of technology is a well defined process. The process of obfuscation is a pivotal element in the overall value and viability of obfuscation. Without a well defined and highly integrated obfuscation process, the complexities introduced by obfuscations can outweigh its benefits. Clearly, obfuscation can complicate debugging, patch generation and management, distributed development practices and reuse of libraries, components and web services. The key to success is tight integration with the development platform, such as Visual Studio, the inclusion of tools and utilities that can unwind and/or re-use obfuscation transformations and integration with operations management platforms such as Microsoft's Operations Manager.

In a "best practices" scenario, developers decide and document where obfuscation transformations are appropriate. For example, reflection may confuse re-naming transforms, ultra-high performance algorithms or functions may call for less aggressive control flow settings or the elimination of string encryption and the most sensitive code, i.e. the most valuable code, may call for the most aggressive obfuscation settings.

Obfuscation transformations are applied after the build and do not require access to the 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.

In addition, all transformations must be captured (and previous transformations re-used as appropriate) to support the many software development scenarios outlined above (debugging, patch management, etc.). Again, it is very important to note that all of these processes must be documented and embedded within the IDE (Integrated Development Environment) to ensure continuous automation, complete transparency and most importantly, software quality.

Obfuscation can be defined as a compensating control to manage risks stemming from unmanaged source code distribution. For the medical device industry, those risks can be defined as theft of intellectual property, the potential for unauthorized use or theft of patient information and increasingly, the likelihood of system attack.

The full integration of obfuscation controls into a company's software development environment includes the complete documentation of the obfuscation process, an analysis of the specific risks that are being managed and the training and communication within the software and SQA teams that ensure that obfuscation will be applied appropriately (e.g. not over, or under used) It also includes the consistent use of obfuscation within an integrated development and QA environment while employing best practices at all times.



Full Spectrum Software

The Importance of Best Practices

Obfuscation transformations are, by design, intended to profoundly alter the way a binary appears. Effective obfuscation offers significant security and performance benefits. However, obfuscated binaries do not discriminate; they are as opaque to “good guys and good programs” as they are to “bad guys and bad programs.”

Anticipating and accommodating the expected consequences of including obfuscated binaries into a complete application development lifecycle ensures that obfuscation benefits will be maximized and that any associated complexity or risk is minimized. A very brief summary of best practices include, at minimum:

- Declarative obfuscation controls: Developers know their code the best. Using the XML-defined attributes they can specify which transformations to apply, how they should be applied and where within an application to apply them. Declarative obfuscation supports an arbitrary level of granularity enabling the maximum degree of obfuscation with the minimum degree of side effects.
- Configure and apply obfuscation transformations: SCM (Source Code Management) or build managers configure and execute the final obfuscation transformations. These capabilities should be extensible to include support for unit testing and continuous integration processes.
- De-Obfuscation for efficient debugging: A distributed capability that can un-wind obfuscated binaries providing an effective connection back to the original source code. Debugging is rarely effective when restricted to build machines. This is critical in providing diagnostic support for defects found in the field.
- Incremental obfuscation transformation: The ability to obfuscate patches that both prevent access to source while maintaining compatibility with previously obfuscated and distributed binaries.
- Validation of the obfuscation tool: As with any other tool or process that is used in the development of controlled software, validation of the selected tool is required. Integration with preferred IDE's and deployment flexibility to support today's distributed work environments are the final elements of an effective obfuscation process and best practices.

Technical information supplied by Preemptive Solutions and re-printed with permission. For readers interested in learning more about code obfuscation, please click on this link to learn more about Preemptive Solution's products. <http://www.PreEmptive.com/>

Full Spectrum Software, Inc. is a software development and Testing V&V provider for the Medical and Scientific Industries. Contact Ken Carson, kcarson@FullSpectrumSoftware.com (508) 620-6400