.NET Intellectual Property Protection

The Merits of "True Encryption"

Summary

Software copy protection is an important concern of developers who wish to ensure their products are not copied or used without authorization, which results in profit losses.

While some software protection companies utilize obfuscation to deter hackers, encryption is a much stronger method of protecting .NET applications. Encryption is the best way to protect software from unauthorized usage.

This white paper explains the differences between obfuscation and encryption and details why encryption is a much stronger form of protection of .NET programs.

Introduction

Microsoft's newest platform for development, .NET, poses some especially challenging problems for manufacturers and vendors, who must continually guard against illegal duplication of their software.

.NET code is wonderfully easy to effectively reverse-engineer. One can decompile the code, crack the license, change the copyright name or simply see how the application hits the database. Reverse engineering based on compiled code is possible but for hackers, reverse engineering based on source is simply icing on the cake.

The good news is that you can make the process of reverse engineering harder for hackers to achieve--in fact, much harder. To be effective, software protection must deter the reverse engineering of code.

One way to achieve reverse engineering of code is to make understanding the code as difficult as possible. This is the goal of obfuscating software. A second way to achieve this goal is to make examination of the code as difficult as possible. This is the goal of encrypting software.

What Is Obfuscation?

Obfuscation is a one-way transformation of source code utilized to discourage understanding or recompilation of the code. Obfuscation transforms programs in such a way as to not affect what they do and at the same time impede reverse-engineering efforts.

Some techniques that obfuscators use include deleting comments, removing neat indentation and white space, and encoding constants in unreadable ways. Obfuscators also rename identifiers in source from their original self-explanatory names to meaningless terms that convey no information. The larger the code, the messier this gets, making the program correspondingly more difficult to reverse-engineer.

In a nutshell, the goal of obfuscating is to remove all information that is useful for a human, but not necessary for the computer, from the code. A simplified example would be to transform the easily understood line of source code

$$\text{Energy} = \text{Mass} * \text{SpeedOfLight}^2$$

to

$$V1 = V2 * V3^{2.}$$

The computer will not notice the difference in these two statements once compiled but a human will have to do much more work to figure out what is being calculated.

Why Obfuscation?

Until the release of the .NET platform, most sophisticated copy protection systems for PC software focused on encrypting the compiled software program to discourage reverse engineering. Over time, the format of PC executables became well understood, and methods of encrypting it evolved.

The .NET platform radically changed the format of the executable. Instead of compiling source code to well understood machine language, the .NET compiler compiles the source code into MSIL code, which is essentially data that the .NET system interprets and executes.

The compiled .NET code is not executed by the well-understood and documented microprocessor. It is executed by a little understood and mostly undocumented Microsoft program. To make matters worse, the compiled .NET code is easily decompiled, and the decompiled code is as understandable as the original source code.

Existing encryption copy protection techniques do not work on this type of code, and at the time of this writing, few encryption systems for .NET have evolved. Obfuscation, although not historically the strongest solution, has come to the forefront of .NET protection simply due to the absence of encryption systems on the market.

Obfuscation Falls Short of Copy Protection

While obfuscation is an inexpensive, fast, extra layer of protection from reverse engineering, it falls short as a way to protect .NET software from being copied. There are three major reasons for this.

First, protection from reverse engineering offered by obfuscation is generally not strong enough to discourage a hacker. Obfuscation will discourage the average person from looking into or even decompiling .NET code; however, hackers are used to working with decompiled machine language, which is even harder to understand that obfuscated code. For a hacker, obfuscated code is a step forward, not backward, in the hacking process.

A second problem is that obfuscation fails to hide calls to third party software. Copy protection is likely to be implemented by a third party, and the hacker will be seeking to circumvent it. To give a simple example, the software may need to call a third party like:

“IsThisProgramAuthorizedToRun()”.

Since .NET needs to know the exact name of the function to find and run it, after the program is obfuscated, the function is still easy to find and circumvent, because it will appear in the obfuscated code exactly the same:

“IsThisProgramAuthorizedToRun()”.

The third problem with obfuscation is that it does not do all of the extra, important jobs that good copy protection needs to do to discourage professional reverse engineering. It **does not**:

- prevent debuggers from being run on the program;
- detect and prevent changes to the executable code while it is running (patching);
- prevent stepping through a program one instruction at a time;
- prevent code from being examined while in file; nor does it
- prevent code from being examined while in memory.

While the above technically are not encryption tasks, some are a byproduct of it, and all are likely to be included in a copy protection system that implements code encryption.

What Is Encryption?

Encryption, on the other hand, is a two-way transformation that protects against reverse engineering by making the code completely unreadable to the human or the decompiler. An example as simple as the one above is more challenge to provide; however, the easily understood line of source code,

Energy = Mass * SpeedOfLight^2

would encrypt to something that, if examined with an editor, would look like:

($*%)@#H$@#^%$@$@CD$##@???@#$

Encryption is usually used on the object code that is produced by compilers. There are two mediums where encryption can be used on a program:

1. Encryption of the code in a file. This is the easiest place to use encryption and the most necessary. If the program file is not encrypted, modern day hacker tools such as reverse compilers can help hackers easily reverse engineer and circumvent any copy protection code.

2. Encryption of the code while running in the computer. This is by far much more difficult to achieve but quite important in the copy protection effort. Without this, experienced hackers can take a snapshot of the code while it is running unencrypted in memory and with some effort, write the whole program back to file unencrypted.

Encryption vs. Obfuscation

While encryption is often compared to obfuscation, their goals differ. The purposes and key differences between encryption and obfuscation are outlined below:

<u>Encryption</u>	<u>Obfuscation</u>
• **a two-way transformation** that provides strong protection for hiding data or executable code	• **a one-way transformation** of data ▪ most obfuscating transforms cannot be undone because they are one-way transformations

Encryption	Obfuscation
▪ with **encryption,** every bit of data originally in a file remains after the file is uncompressed	▪ **obfuscation** reduces a file by permanently eliminating certain, especially redundant information; ▪ when a file is uncompressed, only a part of the original information is still there, although the user may not notice it
• protects against reverse engineering	**• prevents recompilation of software code** ▪ goal is to destroy relationships that exist between compiled and source-code versions of code
▪ utilizes password or encryption key to prevent examination of code from those without access to the key	• accepts a source file and generates another functionally equivalent source file that is much harder to understand or reverse engineer • deletes comments, remove neat indentation and white space and encodes constants in unreadable ways • renames identifiers in source from their original self-explanatory names to meaningless terms that convey no information
▪ stronger than obfuscation due to inability to examine encrypted code	**▪ requires no changes to compilation or environment** ▪ does not prevent reverse engineering by very determined opponents; with sufficient persistence and know-how can be circumvented

Why Encryption?

Encryption offers very strong protection from reverse engineering. While it takes no special training for a reasonably intelligent person to make headway understanding an obfuscated program, it takes a person with intimate knowledge of encryption, the operating system, and computers to reverse engineer an encrypted program. Here's why.

To break an encryption copy protection system, the hacker is usually forced to completely understand large amounts of machine code to find and duplicate the mechanism the program uses to unencrypt the code. If the system only unencrypts small amounts of code at a time, the hacker must study and understand this system *and* the system that is used to bring the right piece of code to the processor at the right time.

While it is true that it is not possible to make this method "crack proof" (since the code must eventually be decrypted to run), there is no limit to how complicated this encryption and the extra jobs around it can be made. It can use more than one algorithm at a time, it can use different algorithms for different chunks of code, and it can unencrypt and can run small only pieces of code at a time. The variations and strategies are infinite. A programmer who is knowledgeable in this area is really only limited on the amount of time he can spend. The more time spent, the longer it takes to crack, and thus, fewer hackers will have both the time and knowledge to break the code.

The “infinite complexity” feature arises from the fact that encryption doesn’t just confuse; it can effectively prevent any examination or reverse engineering of the encrypted code. This means the hacker must first figure out how to un-encrypt the code, which can be infinitely complicated.

Since the complications that can be created are unlimited, the amount of time and energy spent creating the encryption becomes a choice based on cost–benefit analysis, rather than technical limitation.

In some cases, software revenue can be dramatically increased by a relatively simple encryption mechanism that keeps the average person from easily copying the software.

A more complicated encryption mechanism can be used on a program to make cracking take more time than it is worth. Such is often the case with specialized programs that have a smaller distribution.

In other cases, a very complicated encryption mechanism can delay the availability of an unprotected version of a program on the Internet long enough to allow the software company to make a profit.

“True Code Encryption” = .NET Protection

The key benefit to code encryption is that it can provide a meaningful level of copy protection, and the level of protection can be driven by cost–benefit analysis.

Obfuscation, on the other hand, can provide a limited reverse engineering deterrent but it cannot provide a meaningful level of copy protection. While companies utilizing obfuscation claim to protect .NET applications from illegal copying, in reality, they do not.

Conclusion

Only encryption-based systems can offer a meaningful, scalable, level of copy protection against the modern, technically evolved, hacker. To date, CrypKey (Canada) Inc. offers one of the few known encryption software protection solutions for .NET programs.

Resources & Information

CrypKey (Canada) Inc. is a leading developer of reliable, world-class software protection and license management solutions for small, medium and large enterprises.

CrypKey solutions have been commercially available since 1992. CrypKey was the first company to offer solely software copy protection solutions and is the now first to offer an encryption copy protection solution for .NET software.

For more information, contact CrypKey at 1-403-258-6274 or visit www.CrypKey.com.