# encrypting a .Net application and assemblies

Asked 15 years, 10 months ago Modified 4 years, 2 months ago

Viewed 10k times

6

I have an encryption/copy protection question.

I'm writing an application for a company that uses a dongle. Please don't tell me that software protection is useless, or that I should just let it fly free into the air, or that any time I spend doing this is a waste; this isn't a philosophical question about the validity of software protection, more like a how-to.

As I understand it, the first step in cracking a dongle-protected piece of software is to remove all the calls to the dongle from the code (ie, patch the executable). Also as I understand it, I can create 'strong names' in .NET in order to protect the application and the assembly, as explained in this MSDN article.

Is strong naming enough to ensure that my application can't be easily patched? Or do I need to use some sort of encryption library? If I need to use a library, which one, or where can I get information about setting this up?

The next step, of course, is to put important algorithms on the dongle. I realize that these are just speed bumps to the dedicated cracker, but as our market share grows, the

speed bump will help us get to the point where the sting of piracy is not so keenly felt (I hope).

Thanks!

.net encryption dongle

Share

Improve this question

Follow

edited Jan 4, 2012 at 19:38

Josh Darnell

11.4k ● 9 ● 39 ● 66

asked Jan 30, 2009 at 1:43

mmr

14.9k ● 29 ● 97 ● 148

## 5 Answers

Sorted by: Highest score (default)

8

Assembly strong naming was never designed to protect against an attacker who is in control of the machine. From the msdn entry on delay signing:

> The following example turns off verification for an assembly called myAssembly.dll.
>
> ```
> sn –Vr myAssembly.dll
> ```

The design goal of strong names is to provide name uniqueness and to protect *the user* (not the publisher) against an attacker. If the user wants to disable all strong

name checks, or maybe even strip out your signature and re-sign the assembly with his own key then there is technically speaking nothing to prevent him from doing so.

Simply loading your assemblies from an encrypted file is also not very useful because the decryption code itself cannot be encrypted and is therefore an easy target for reverse engineering.

As mentioned by other posters, what you are looking for is **obfuscation**. You probably already have such a tool: Visual Studio (at least 2005 and 2008) comes with the community edition of PreEmptive Solutions' Dotfuscator. Microsoft also has its own "Software Licensing and Protection Services" product.

Obfuscation has some technical disadvantages however:

- it may complicate your build process. You need an unobfuscated and an obfuscated build, because the latter is not debuggable.
- I like to have an error dialog for unexpected exceptions where the user can click "copy details" and send me a mail with some technical information including the stack trace. With obfuscation however, you can forget about getting anything useful from Exception.StackTrace.
- if your code makes use of reflection then there is a good chance that things will break in the obfuscated

build, because internal type and member names are not preserved.

Share Improve this answer Follow

edited Jan 30, 2009 at 3:33

answered Jan 30, 2009 at 2:55

Wim Coenen

66.7k • 14 • 161 • 253

So basically, I could make the release build obfuscated, and the debug build clean, and be able to still develop then? Can obfuscation cause changes to the code that are significant (ie, behavior changes)? – mmr Jan 30, 2009 at 16:55

Obfuscation techniques are designed to not change the code behaviour. But as I said, it becomes hard to guarantee this if the code is inspecting itself through reflection. Fortunately reflection should still work if you only use it on public/protected (i.e. not internal) types and members. – Wim Coenen Jan 30, 2009 at 18:36

I'm pretty sure I'm not using reflection, unless it's being done without my knowledge. So this looks like the right first step. – mmr Feb 1, 2009 at 17:58

No, obfuscation just changes the names of functions & variables. Encryption is another thing altogether. – kristianp Mar 25, 2013 at 4:15

4

Signing your assembly will make it impossible to alter it without altering the signature, and hence its reference. The consequence of this is that a (strong named)

reference to the assembly will fail to resolve against the altered version. And that's guaranteed against ridiculous odds.

That doesn't solve your problem, though. Not completely, anyway. If you pack your dongle calls, say, into a strongly named assembly, then reference that assembly from your application, the application will not work without your unaltered assembly, and hence not without the dongle. But the application itself can be altered!

Another means available to you is obfuscation. There's a free version of an obfuscator shipped with Visual Studio, which can be upgraded to industrial strength. Obfuscation renders code incomprehensible without altering it's behaviour, and hence presents a real barrier to reverse engineering.

I'd say the solution lies in some clever combination of these two techniques.

And that's the extent of my knowledge, I'm afraid. Someone else will have to provide the actual answer here (and it's probably embarassingly much shorter than mine ;-)



answered Jan 30, 2009 at 2:04

Tor Haugen

**19.6k** ● 9 ● 47 ● 64

that's a big help to my understanding, but I still need to know the nitty-gritty details. Thanks! – mmr Jan 30, 2009 at 2:07

2

If they are patching your executable a strong name does not help. It will however help you ensure that a dll you reference is the correct version and has not been tampered with.
you might check Salamander orpreemptive for obfuscation.
Encryption you might look at Assembly Lockbox, CodeVeil, or ThinApp

Share Improve this answer

Follow

edited Jun 22, 2015 at 16:59

Brad Larson
**170k** ● 45 ● 400 ● 572

answered Jan 30, 2009 at 2:06

Aaron Fischer
**21.2k** ● 18 ● 78 ● 117

Why thinapp? It looks like it's something that will make it run on another OS-- won't that cause significant slowdowns? – mmr Jan 30, 2009 at 2:28

Because ThinApp represents a line of virtualization tech that I could remember off the top of my head. Speed wise I can't say I only played with small apps in it. but it will provide the tamper proofing you are looking for. – Aaron Fischer Jan 30, 2009 at 3:19

1

If you want to use a dongle and encrypt your program, this article may be useful to you: http://www.gironsec.com/blog/2012/02/dongles-how-do-they-work/

Here is a pertinent quote from that article:

> There are 2 ways to implement a dongle. The right way and the wrong way. The right way would be to encrypt your programs and store the encryption key on the dongle and decrypt at run time depending on whether the device is connected or not.

Share Improve this answer

Follow

edited Sep 30, 2020 at 19:59

ardila

1,285 ● 1 ● 13 ● 24

answered Mar 25, 2013 at 4:19



kristianp

5,895 ● 39 ● 62

-1

To some extent, Obfuscation may be useful for you. But it is not a 100% secure way. IN fact, there is not any 100% secure way to protect any software module. At its best, obfuscation merely makes it time-consuming to reverse-engineer a program. .NET Reflector is a reverse engineering tool which regenerates the source code from any .NET assemblies. Using Dotfuscator will make it

difficult for the intruder to understand the original source code but one can effort to have a good estimation of that source code if the reward is big.

Share Improve this answer

Follow

answered May 9, 2011 at 5:34

Ruchir

1

2 Hi @Ruchir-- as I said, "Please don't tell me that software protection is useless, or that I should just let it fly free into the air, or that any time I spend doing this is a waste; this isn't a philosophical question about the validity of software protection, more like a how-to." – mmr May 9, 2011 at 13:32