Protect .NET code from reverse engineering?

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

Viewed 279k times

527

Obfuscation is one way, but it can't protect from breaking the piracy protection security of the application. How do I make sure that the application is not tampered with, and how do I make sure that the registration mechanism can't be reverse engineered?

Also it is possible to convert a C# application to native code, and Xenocode is too costly.

C# provides lot of features, and is the ideal language for my code, so writing the whole codebase again in C++ is out of the question.

Secure certificates can be easily removed from the signed assemblies in .NET.

c# .net obfuscation reverse-engineering

Share

Improve this question

Follow

edited Feb 24, 2013 at 8:47

community wiki

6 revs, 5 users 53%
Peter Mortensen

@Andreas: This is awesome!! I'm going give a try. Anyone using it? – Jack Apr 5, 2015 at 17:32

1 @Jack it's for window store apps only. There is no timeline for desktop apps(as far as I can tell). – Tyler Liu Dec 27, 2015 at 5:26

If you want native without archaic C++, use Delphi. The ease of .Net came from Delphi anyways. – William Egge Jan 4, 2017 at 16:09

stackoverflow.com/questions/25133111/xamarin-code-security/… solved! – Nick Kovalsky Jun 24, 2019 at 20:41

Looks like native forever and no needs in obfuscators; Looks like net core RT workable solutions; soon all apps will go to .net core; codeproject.com/Articles/5262251/… learn.microsoft.com/en-us/archive/msdn-magazine/2018/november/… not tested maybe with old win .net sdk possible do similar. PLS wote my answer to go it up; looks like native compile better any free obfuscators or any other options; – user1005462 Oct 4, 2020 at 14:43

38 Answers

Sorted by: Highest score (default)

1 2 Next

705

You can't.

There are steps you can take to make it a *little* more difficult, but ultimately any executable on the local machine is crackable. Eventually, that code has to be



converted into native machine code and every application that is runnable is vulnerable.

What you want to do is just make it difficult enough to crack to make it not worth peoples' trouble.

Some suggestions I have for you to help protect your application:

- **Obfuscate** your code. Dotfuscator has a free edition and comes with Visual Studio.
- Use public/private key or **asymmetric encryption** to generate your product licenses. This ensures that only *you* can generate your license codes. Even if your application *is* cracked, you can be sure that they won't be releasing a key generator for your application, because it is impossible to reverse the key generating algorithm.
- Use a **third-party packer** to pack your .NET executable into an encrypted Win32 wrapper application. Themida is one of the better ones. This stops people from reflecting your application in .NET Reflector and makes it a pain to unpack for reversing.
- Write your own **custom packer**. If the third-party packers are too expensive, consider writing your own. Sometimes custom packers can be very effective, because there aren't well published methods on how to unpack them. The tutorial *How to*

write your own packer gives a ton of good information on writing your own Win32 packer.

Ultimately though, if people want your application cracked they will. Look at all the commercial software out there that has a vast amount of resources to protect their applications and yet they are cracked before the applications are even released to the public.

A skilled reverse engineer can fire up IDA-Pro and slice through your application like butter no matter what you do. A packed application can be unpacked and obfuscation only prevents it from making it a walk in the park. All your hard work with your complex license code can be undone with a single byte patch.

You just need to accept that there is a very real chance people are going to pirate your software. There are some people who are *never* going to pay for your application no matter what and these are the people you don't need to worry about.

There are however, many businesses out there who would never risk a lawsuit and happily buy software licenses and many computer users who either don't want to risk it, find it wrong or are not tech savvy enough to pirate. These are your true customers, and you should focus your efforts on providing them with a good user experience and ignore the people cracking your software.

I've had my application pirated before, and I took it as a personal affront. Here I was, a small-time developer,

pouring my heart and soul into an application and these people had the gall to pirate from me?! They were taking money directly from my pocket!

I immediately added in a bunch of draconian DRM code and attempted to sabotage any person using an illegitimate or cracked copy. I should of course have been working on making my application better instead of trying to stop the inevitable. Not only that, but I was hurting my *true* customers will all these extra protections I was putting in.

After a long battle I realized I was fighting the tides and all this time wasted was for naught. I took out all the phone-home code except for the barebones license functions and never looked back.

edited Feb 24, 2013 at 9:07

community wiki
13 revs, 5 users 68%
Simucal

72 So +1 that it's almost +2. I wish more people would finally get that you simply can not protect your software against a determined attacker. – Bombe Feb 3, 2009 at 8:16

110 You have reached software protection nirvana: it's not about adding more protection, it's about focusing on the product and making it so good that people WANT to pay for it. And for those that pirate it, they would have never paid anyways

so it's as if they never existed. – Arthur Chaparyan Feb 3, 2009 at 8:59

6 @Arthur Chaparyan, I agree. It took a long time to get here but I finally have seen the light. I went down the road of more restrictive protections and battling the crackers. I learned all I could about reverse engineering in an attempt to prevent my own. I finally figured out the right ideology – mmcdole Feb 3, 2009 at 9:03

55 Hell I'd have been honored to find out someone thought my software was worth pirating... – Erik Forbes Feb 4, 2009 at 20:04

11 When you start relying on the sales of your software for a major part of your income it changes things. It feels like someone is stealing from you. I get what you are saying though. I was shocked when I first found cracks for my software on torrent sites. – mmcdole Feb 4, 2009 at 20:20

270

You can't fully secure any application (managed or not). If systems like the Playstation and iPad can get cracked — where the vendor even controls the hardware — what hope does your app have? Thankfully, you don't really want to. In my opinion, you need to secure your application just enough that someone can't *accidentally* pirate your product, *and no more.*

For example, if you use a per-machine license, it shouldn't just work when you install it on a new second machine. You'll want a good error message to prevent extra support calls, but don't spend extra time making it too hard to work around and don't hit users over the head with it.

Another example is a time-limited trial. Don't even worry about simple things like if users can just roll back the system clock. Someone who does that knows they are breaking your license, and as long as a user *knows* when they are in violation you've done enough.

You need to do this much because users don't care about your license. Licenses are made-up things that *nobody* cares about until they need to. No one reads them, and they really shouldn't have to. Therefore the best way to tell the user where the boundaries are is if the out-of-the-box behavior for your application complies with the license. In this first case that means either failing to install or installing in trial-version mode the second time. For the latter, it might just mean checking a plain-text date in a configuration file. Either way, make sure you handle it in an elegant, helpful, and respectful manner.

So that explains what it means do just that much. But why not go any further? **Why not plug every little hole you can find?** The answer is in two parts. First, if someone will cross the ethical threshold of *consciously* breaking your license terms — even in a simple way — they'll also be willing to do something more difficult or dangerous like pull your application from a torrent site — and there is a certain amount of danger involved in running applications downloaded from untrusted sources. Making it any harder is only a minor annoyance for these users and risks causing problems with your paying customers. Keeping it simple may prevent someone from digging into your application and releasing a more comprehensive crack.

Second, you have few eyes available to look for flaws; the hackers have many, and they have more practice finding them. You only need to miss one little flaw, and your app will have the same distribution on pirate sites as if you did nothing. You have to be right every time; they only have to be lucky once. So the effort required is very high, and the likelihood of any measure of success is very low.

Ultimately, if someone wants to *pirate* your application (as opposed to just using it), and that is their main goal, they will. *There's nothing you can do to stop them.* This is the nature of software; once the files that make up your product are on a user's computer they *will* be able to do with them as they wish. This is especially relevant in managed environments like Java or .NET, but it definitely applies to native code as well. Time is on their side, and given enough time any digital security can be broken.

Since you can't stop users from pirating your product, your best course of action is to engage this class of user in a way the uses them to your benefit. It is often possible to get them working for you rather than against you. With that in mind, no matter what your application is, it's probably worth it to keep a free version that is almost completely functional and doesn't expire. The difference between even a US$1 price tag and free is huge, if for no other reason than that the customer doesn't have to trust you with their credit card. A free edition of your product will not only effectively kill pirated distribution (why risk a pirated version when you can be legitimate for the same

price?), it has the potential to dramatically expand your audience.

The result is that you may need to increase the price of the for-pay edition, so that in the end instead of 2,000 users at $20 each you have 100,000 free users, of which 500 are willing to pay $99 for the "professional" edition. This earns you more money than if you spent a bunch of time locking up your product. More than that, you can engage these free users and leverage the relationship in several important ways.

One is support. A pessimist would take this opportunity to complain about the increased cost of supporting 100,000 free users, but something amazing happens instead: your product becomes largely self supporting. You see this all the time with large open source projects that have no money for support costs. Users will step up and make it happen.

Free users generally have reduced support expectations to begin with, and for good reason. All you need to do is mark the free edition as only qualifying for community support and put up a user-moderated online forum for that purpose. Your support knowledge base is self-generating, and advanced users will shepherd those who need extra hand-holding on your behalf. Even more importantly, this will allow you to identify and correct bugs faster, ultimately improving the quality of your product and lowering total support costs. This wasn't possible before because your user base wasn't large enough, but when

you treat the free users as customers it can work very well.

Another is feedback. By watching your forum, you learn important improvement ideas that you may never have considered otherwise. This can allow you to ultimately turn more of your free users into paid users and create a more compelling product that will attract an even larger audience.

Finally, you need to consider marketing. All these free users are now fans rather than adversaries, and they will act accordingly. Not only that, but when it comes time to release your next version these users will have all gone through your approved distribution channel, rather than some other unknown mechanism. That means for your next version you start out connected with a larger, highly interested and supportive audience.

The best features to reserve for the professional edition are tools aimed at making it easy to do corporate deployment and management. A cracker won't see these as a compelling enough reason to hack it for his own use, but for a business looking to buy 300 licenses and push it out company-wide this is a must-have. Of course, the professional edition will be pirated anyway, but again: don't sweat it because you probably wouldn't be able to sell the product to those pirates no matter what you did, so it's not costing you *any* revenue.

While psychologically it can be hard to give away your product this much, hopefully you can understand how it

really is the best way to go. Not only that, it's the only way to go in the long term. I know someone is out there thinking that they don't want to do it this way. After all, they've got by just fine selling their locked-down $20 product for years. But that's just too bad, because if you don't do it this way, *eventually someone else will*. And their product will be just as good as yours, or close enough they can get away with claiming that. Then all of a sudden your pricing looks outrageous, sales drop dramatically, and there's nothing else you can do. You can opt for an additional middle tier if you must, but it's unlikely to help you.

edited Apr 15, 2014 at 15:57

community wiki
30 revs, 2 users 87%
Joel Coehoorn

1 @Learning: it kinda grew over time and passed the automatic convert to CW threshold. – Joel Coehoorn Mar 17, 2009 at 17:29

1 +1 Better than the answer I gave to the previous version of this question. @Joel Didn't know there was a limit. – pipTheGeek Jul 23, 2009 at 21:01

1 I don't agree completely with everything in here, and it's an easy +1 anyway for the extremely well thought out presentation and argument. – Beska Jul 23, 2009 at 21:23

2 Good argument, but misses a point about protecting intellectual property. If your app has some code that does

something that is somewhat complex, obfuscation can provide the difference between flat out copying and pasting code, and trying to interpret and re-engineer code so it works. This is particularly important with updates: if someone has copied your obfuscated code, when you release an update, they have to do that over again - and at the very least, it causes them more pain/expense/time. If you don't have it protected in some way, they just copy/paste again and it works. – gregmac Jan 19, 2010 at 19:28

4 Great post - it really goes into all the right details about the reasons why it's not worth bothering to write complex copy protection. Even though the question is a duplicate it's worth reopening it just for this answer. Maybe a mod can merge it? – EMP Jun 21, 2010 at 7:55

45

In my experience, making your application or library more difficult to crack hurts your honest customers whilst only slightly delaying the dishonest ones. Concentrate on making a great, low friction product instead of putting a lot of effort into delaying the inevitable.

Share Improve this answer Follow

edited Sep 27, 2011 at 14:54

community wiki
2 revs, 2 users 67%
Kent Boogaart

A secret that you share with lots of people is not a secret. If you have secret stuff in your code, obfuscating it is no protection; it only has to be deobfuscated *once*. If you

41

have a secret that you don't want to share with your customers, then *don't share it with your customers*. Write your code as a web service and keep your super secret code on your own server, where only you can see it.

Share Improve this answer Follow

answered Feb 21, 2010 at 6:13

community wiki
Eric Lippert

1 Btw i m doing a project in which i want to activate a product "offline" also.(I made a WCF service to make activate online). In this case how u will manipulate code ? can u give me some hits ? – Piyush Feb 22, 2010 at 11:26

2 Interesting idea, butWhat course of action would you recommend to someone developing an application which must run without a data connection, such as a WP7 game? – Charlie Skilbeck Mar 15, 2011 at 17:51

Web apps also need hosting and we can not trust hosting guys, also not everyone can offord their own server with 99.9% availability. – Saqib Nov 13, 2021 at 17:17

25

Broadly speaking, there are three groups of people out there.

- Those who will not buy your software and resort to cracks, or if they don't find any, not use your software at all. Don't expect to make any money from this group. They rely either on their own skills or on

crackers (who tend to prioritize their time depending on your useful and how big your audience is. The more useful, the sooner a crack will be available).

- The group of legitimate users who will buy (pay for) your software, irrespective of what protection mechanism you use. Don't make life hard for your legitimate users by using an elaborate protection mechanism since they are going to pay for it in any case. A complex protection mechanism can easily spoil the user experience and you don't want this happening to this group. Personally, I'd vote against any hardware solution, which adds to the cost of your software.
- A minority who will resort to "unethical" cracking and will only *pay* for your software *because* its features are protected by a licensing mechanism. You probably don't want to make it exceedingly easy for this group to circumvent your protection. However, all that effort you spend on protecting your software will pay back, depending on how big this group of people is. This entirely depends on the type of software you're building.

Given what you've said, if you think there is a large enough minority who can be pushed into buying your software, go ahead and implement some form of protection. Think about how much money you can make from this minority versus the time you spend working on the protection, or the amount you spend on a third party protection API/tool.

If you like to implement a solution of your own, using public-key cryptography is a good way (as opposed to symmetric algorithms) to prevent easy hacks. You could for instance digitally sign your license (serial no, or license file). The only way to get around this would then be to decompile, alter and recompile the code (which you could make harder using techniques such as those suggested in Simucal's answer).

Share Improve this answer Follow

edited Jan 27, 2020 at 21:19

community wiki
8 revs, 3 users 85%
Mystic

Using strong cryptography to protect/verify your licences is completely useless if somebody rips out the code that aborts the application if the licence doesn't check out. :) – Bombe Feb 3, 2009 at 8:33

Agreed, but as I was saying, the protection isn't for those groups of users who will resort to using cracks (an assumption that I made will exist). – Mystic Feb 3, 2009 at 8:43

public-key cryptography = asymmetric cryptography. I think you meant symmetric. – mmcdole Feb 3, 2009 at 9:29

1 To be fair the third point is biased since it assumes that portion of people is always a minority. I'm pretty sure under certain frameworks it's a clear majority. I have online games with massively multiplayer features in mind because a) most users are kids with very low ethical standards b) the cost can

be significant to those users if it's a monthly fee that drags for months etc. – j riv Oct 26, 2017 at 4:26

20

You can't prevent people from cracking your software.

However, you can make them create cracks that will hurt your sales less. Keygenerators that can issue a valid registration code for your software are much worse than simple patches that remove registration incentives from your software. That's because a crack will work for one software version only, and will cease to work with the next software update you release. The keygenerator will continue to work until you change your registration key algorithm and that's something you don't want to do often because it will put off your honest clients.

So, if you are looking for a method to fight illegal keygenerators for your software and you do not want to use assymetric encryption because of the long registration codes this generates, you might have a look at Partial Key Verification.

Partial Key Verification makes sure that each illegal keygenerator works only for one particular release of your software. Basically what you do is to make sure that each release of your software only links with the code for checking SOME digits of the registration code. Which digits exactly is random, so crackers would have to reverse engineer many different versions of your software and combine all this into one keygenerator in order to

release a keygenerator that works for all versions of your software.

If you release new software versions on a regular basis, this leads to numerous keygenerators spread on all kinds of software piracy archives which are not working anymore. Potential software pirates usually look for a crack or keygen for the latest version, so they will likely try a few of those and give up eventually.

I've used the Partial Key Verification in my (C++) newer shareware games and it has been very effective. Before we had plenty of problems with keygenerators which we could not fight. Afterewards there were lots of cracks and some few keygenerators that worked only for that particular version of the game, but no key generator that would work with all versions. We regularly released very minor updates of the game and to render all previously existing cracks useless.

There seems to be an open source .NET framework for Partial Key Verification, although I have not tried it.

edited Sep 14, 2009 at 8:09

community wiki
2 revs
Adrian Grigore

1 like the idea, you can also use different passwords for assymetric encryption in different releases. – Priyank Bolia

Feb 3, 2009 at 9:45

Interesting idea, but what exactly is the problem with a long registration code, anyway? Nowdays nobody would enter it by hand anyway - everyone would copy and paste it, so whether it's 10 characters or 100 characters shouldn't make any difference. – EMP Jun 21, 2010 at 7:39

4 @Evgeny: That's only true if your users are power users. We have been creating shareware / casual games for many years and I can tell you that most of our users can't copy and paste. The registration window even comes with a manual on how to copy and paste, and some even don't it after reading it. – Adrian Grigore Jun 22, 2010 at 10:06

Wow! OK, well, you obviously have more experience than me in this, so I can't argue, I can only say I'm surprised. But I would say that if they don't know how to copy and paste then you *should* make the code 200 characters long, so that they learn a highly useful general computer skill. :) – EMP Jun 22, 2010 at 23:12

1 @Evgeny: Even with the short registration codes, we still got a lot of e-mails from people who have mistyped their codes and therefore thought that the code can't be valid because they would *never* make a mistake like this several times in a row. I prefer to leave the IT teaching to other companies... :-) – Adrian Grigore Jun 23, 2010 at 8:10

16

- Use online update to block those unlicensed copies.
- Verify serial number from different modules of your application and do not use a single function call to do the verification (so that crackers cannot bypass the verification easily).

- Not only check serial number at startup, do the verification while saving data, do it every Friday evening, do it when user is idle ...
- Verify application file check sum, store your security check sum in different places.
- Don't go too far on these kind of tricks, make sure your application never crash/get into malfunction while verifying registration code.
- Build a useful app for users is much more important than make a
  unbreakable binary for crackers.

Share Improve this answer Follow

edited Jun 26, 2009 at 11:36

community wiki
5 revs, 2 users 94%
rIPPER

14

You can..

~~Microsoft SLP Services~~InishTech's Software Potential offers the ability to help protect code without affecting the functionality of your applications.

UPDATE: (Disclosure: I work on Eazfuscator.NET) What makes ~~*Microsoft SLP Services*~~*Software Potential* different is the ability to virtualize the code, so you definitely *can*. Several years passed since the question was originally

asked; today there are more products available that also work on a similar basis such as:

- Agile.NET
- Eazfuscator.NET

Share Improve this answer Follow

edited Oct 21, 2014 at 11:21

community wiki
7 revs, 4 users 36%
ogggre

2 pricing my friend, buying a licensing software from Microsoft is too costly for normal ISV – Priyank Bolia Feb 3, 2009 at 9:36

I've tried it, it works very well but it only encrypt the code inside a method not the whole assembly or project, so a cracker can easily alter the flow of the program with IL injection – Mohsen Afshin Feb 27, 2013 at 7:54

@ogggre If adding vendor links, you really need to disclose your connections in the post. Also the currently available version of SLPS (which *I* work on :D) does support generics. Naturally all solutions have their individual pros and cons that only an eval can properly contextualise for people – Ruben Bartelink Oct 19, 2014 at 15:09

@MohsenAfshin I don't understand what you're saying - the point is that you need to protect any methods where the addition/removal/changing of IL would represent a licensing breach. Because virtualizing things cannot be free, it simply doesn't make sense to 'magically protect it all' as you're suggesting. Back to the key point: The aim of SP's Protection is to prevent IL changes on methods that you select on the

basis that they are sensitive (plus generally some others as noise to avoid planting a **you need to crack this bit here -->** sign) – Ruben Bartelink Oct 19, 2014 at 15:16

1 @RubenBartelink I agree with you. Unfortunately this thread is way too big with several pages of content. At first I wanted to add a new answer but StackOverflow suggested that it is better to extend the existing one. So I did. Hope my small piece of information is useful. Thanks for an update on generic support in SLPS and your corrections. – ogggre Oct 21, 2014 at 11:28

10

.NET Reflector can only open "managed code" which basically means ".NET code". So you can't use it to disassemble COM DLL files, native C++, classic Visual Basic 6.0 code, etc. The structure of compiled .NET code makes it very convenient, portable, discoverable, verifiable, etc. .NET Reflector takes advantage of this to let you peer into compiled assemblies but decompilers and disassemblers are by no means specific to .NET and have been around as long as compilers have been around.

You can use obfuscators to make the code more difficult to read, but you can't exactly prevent it from being decompiled without also making it unreadable to .NET. There are a handful of products out there (usually expensive) that claim to "link" your managed code application into a native code application, but even if these actually work, a determined person will always find a way.

When it comes to obfuscation however, you get what you pay for. So if your code is so proprietary that you must go to such great lengths to protect it, you should be willing to invest money in a good obfuscator.

However, in my 15 or so years of experience writing code I've realized that being over-protective of your source code is a waste of time and has little benefit. Just trying to read original source code without supporting documentation, comments, etc. can be very difficult to understand. Add to that the senseless variable names that decompilers come up with and the spaghetti code that modern obfuscators create - you probably don't have to worry too much about people stealing your intellectual property.

edited Feb 24, 2013 at 10:58

community wiki
2 revs, 2 users 75%
Josh Einstein

10

If you want people to able to run your code (and if you don't, then why did you write it in the first place?), then their CPU needs to be able to execute your code. In order to be able to *execute* the code, the CPU needs to be able to *understand* it.

Since CPUs are dumb, and humans aren't, this means that humans can understand the code as well.

There's only one way to make sure that your users don't get your code: don't give them your code.

This can be achieved two ways: Software as a service (SaaS), that is, you run your software on *your* server and only let your users access it remotely. This is the model that Stack Overflow uses, for example. I'm pretty sure that Stack Overflow doesn't obfuscate their code, yet you can't decompile it.

The other way is the appliance model: instead of giving your users your code, you give them a computer containing the code. This is the model that gaming consoles, most mobile phones and TiVo use. Note that this only works if you "own" the entire execution path: you need to build your own CPU, your own computer, write your own operating system and your own CLI implementation. Then, and *only then* can you protect your code. (But note that even the *tiniest* mistake will render all of your protections useless. Microsoft, Apple, Sony, the music industry and the movie industry can attest to that.)

Or, you could just do nothing, which means that your code will be automatically protected by copyright law.

edited Feb 24, 2013 at 11:08

community wiki
2 revs, 2 users 74%
Jörg W Mittag

9

Is it really worth it? Every protection mechanism can be broken with sufficient determination. Consider your market, price of the product, amount of customers, etc.

If you want something more reliable then go down the path of hardware keys, but that's rather troublesome (for the user) and more expensive. Software solutions would be probably a waste of time and resources, and the only thing they would give you is the false sense of 'security'.

Few more ideas (none is perfect, as there is no perfect one).

- AntiDuplicate
- Change the language, use the nice tricks that the authors of Skype used
- License server

And don't waste too much time on it, because the crackers have a lot of experience with the typical techniques and are few steps ahead of you. Unless you want to use a lot of resources, probably change the programming language (do it the Skype way).

Share Improve this answer Follow

edited Feb 24, 2013 at 8:49

community wiki
3 revs, 3 users 46%
Jeff Atwood

Don't forget that it is quite possible to attack the software part of the hardware lock. – Magnus Hoff Feb 3, 2009 at 8:07

Yes, that's true, the only real option would be to have the application partially implemented in hardware (some weird mix of software-VHDL application for example). This would also be crackable though... – Anonymous Feb 3, 2009 at 8:14

What about dongles that implement a public/private key strategy. Only the private key of the dongle can decrypt the application and run it. – mmcdole Feb 3, 2009 at 8:28

That's what the hardware key usually does. But you can either attack the dongle - clone it, or the software responsible for talking with the dongle (circumvent, disable, etc).
– Anonymous Feb 3, 2009 at 8:32

2 In my case it really WAS worth it. After I implemented Partial Key Verification and changed registration key scheme for an existing product, sales went up in significant manner. All software can be cracked, the question is just how high you raise the bar for the casual software pirate. – Adrian Grigore Feb 3, 2009 at 9:15

9

Apart from purchasing protection, you (or your developers) can learn to copy-protect.

These are ideas:

At first, try to write a program that writes itself to console. That's a famous problem. Primary purpose of this task is to practice writing self-referencing code.

Second, you need to develop a technology that will rewrite some code in a way dependable on other methods' CIL.

You may write a virtual machine (yet in .NET). And put some code in there. Ultimately, the virtual machine runs another virtual machine which runs the code. That's for a part of rarely-called functions for not to slow the performance too much.

Rewrite some logic into C++/CLI, and mix managed code with unmanaged. This will harden the disassembling. In this case, do not forget to provide x64 binaries too.

Share Improve this answer Follow

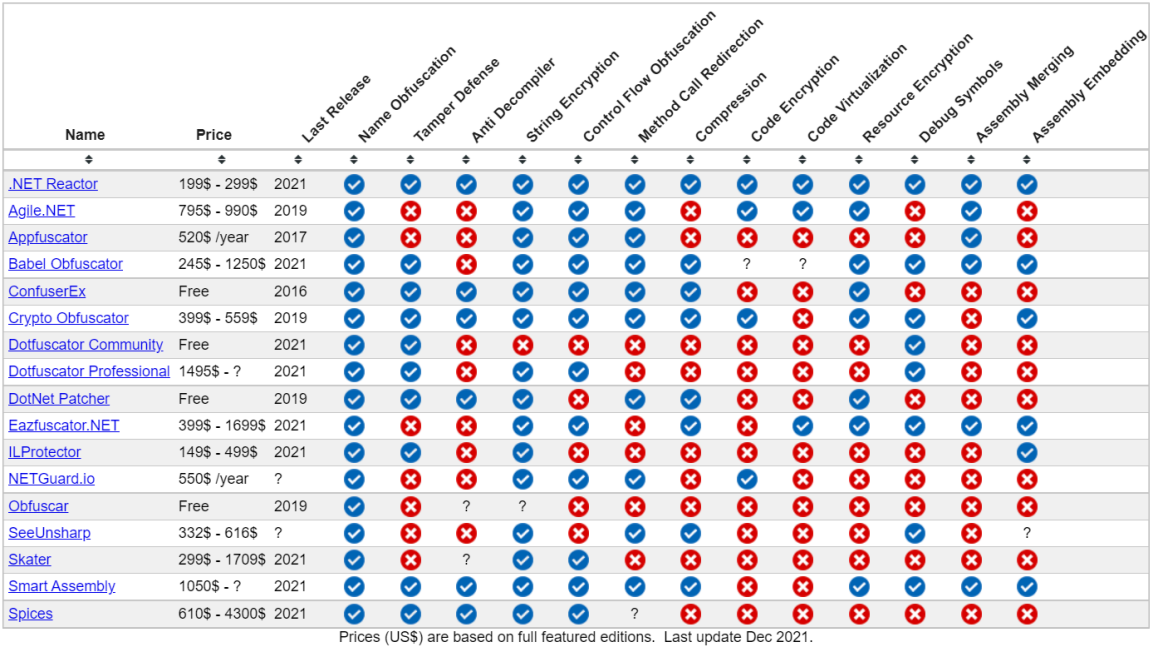
edited Feb 24, 2013 at 9:16

community wiki
2 revs, 2 users 74%
modosansreves

didn't get can you explain in detail. – Priyank Bolia Feb 4, 2009 at 15:54

9

There is a detailed comparison sheet for several .NET obfuscation tools:



Name	Price	Last Release	Name Obfuscation	Tamper Defense	Anti Decompiler	String Encryption	Control Flow Obfuscation	Method Call Redirection	Compression	Code Encryption	Code Virtualization	Resource Encryption	Debug Symbols	Assembly Merging	Assembly Embedding
.NET Reactor	199$ - 299$	2021	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Agile.NET	795$ - 990$	2019	✓	✗	✗	✓	✓	✓	✗	✓	✓	✓	✗	✓	✗
Appfuscator	520$ /year	2017	✓	✗	✗	✓	✓	✓	✗	✗	✗	✗	✗	✓	✗
Babel Obfuscator	245$ - 1250$	2021	✓	✓	✗	✓	✓	✓	✓	?	?	✓	✓	✓	✓
ConfuserEx	Free	2016	✓	✓	✓	✓	✓	✓	✓	✗	✗	✓	✗	✗	✗
Crypto Obfuscator	399$ - 559$	2019	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓	✗	✓
Dotfuscator Community	Free	2021	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗
Dotfuscator Professional	1495$ - ?	2021	✓	✓	✗	✓	✓	✗	✗	✗	✗	✗	✓	✗	✗
DotNet Patcher	Free	2019	✓	✓	✓	✓	✗	✓	✓	✗	✗	✓	✗	✗	✗
Eazfuscator.NET	399$ - 1699$	2021	✓	✗	✗	✓	✓	✗	✓	✗	✓	✓	✓	✓	✓
ILProtector	149$ - 499$	2021	✓	✓	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✓
NETGuard.io	550$ /year	?	✓	✗	✗	✓	✓	✓	✗	✓	✗	✗	✗	✗	✗
Obfuscar	Free	2019	✓	✗	?	?	✗	✗	✗	✗	✗	✗	✗	✗	✗
SeeUnsharp	332$ - 616$	?	✓	✗	✗	✓	✗	✓	✓	✗	✗	✗	✓	✗	?
Skater	299$ - 1709$	2021	✓	✗	?	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗
Smart Assembly	1050$ - ?	2021	✓	✓	✓	✓	✓	✓	✓	✗	✗	✓	✓	✓	✓
Spices	610$ - 4300$	2021	✓	✓	✓	✓	✓	?	✗	✗	✗	✗	✗	✗	✗

Prices (US$) are based on full featured editions. Last update Dec 2021.

Screenshoot taken from obfuscators.io

Share Improve this answer Follow

edited Oct 4, 2022 at 14:53

community wiki
2 revs, 2 users 82%
InputOutput

8

Unfortunately, you are not going to run away from this. Your best bet is to write your code in C and P/Invoke it.

There is a small catch-22, someone could just decompile your application to CIL and kill any verification/activation code (for example, the call to your C library). Remember that applications that are written in C are also reverse-engineered by the more persistent hackers (just look at how fast games are cracked these days). Nothing will protect your application.

In the end it works a lot like your home, protect it well enough so that it is too much effort (spaghetti code would help here) and so that the assailant just moves onto your next door neighbor (competition :)). Look at Windows Vista, there must be 10 different ways to crack it.

There are packages out there that will encrypt your EXE file and decrypt it when the user is allowed to use it, but once again, that is using a generic solution that has no doubt been cracked.

Activation and registration mechanisms are aimed at the 'average Joe:' people who don't have enough tech savvy to bypass it (or for that matter know that they can bypass it). Don't bother with crackers, they have far too much time on their hands.

edited Feb 24, 2013 at 9:09

community wiki
2 revs, 2 users 57%
Jonathan C Dickinson

1 If you really bother outsourcing your registration code into a dll, you should make sure that the DLL has to be different with every new version of your software. Otherwise your making it even easier for people to crack your software. All they'd need to do is crack your DLL once and use that for all later versions. Even end users could do this once they find an old cracked DLL, and this is even worse than putting the

registration mechanism in your managed code.
– Adrian Grigore Dec 1, 2009 at 11:57

8

Yes. It is true. .NET code is extremely easy to reverse engineer if the code is not obfuscated.

Obfuscation will add a layer of annoyance to people trying to reverse engineer your software. Depending on which version you get, you'll get different levels of protection.

Visual Studio includes a version of Dotfuscator. Since it's a bundled version, you're definitely not getting the strongest obfuscation possible. If you look at their feature lists, you'll see exactly what you're missing (and exactly what the application will do to make your code more secure).

There are a couple of other free or open source .NET obfuscators out there (but I can't comment on the quality or the various methods they use):

- Obfuscar
- Babel.NET

In the end, nothing is perfect. If somebody really wants to see how your software works, they will.

Share Improve this answer Follow

edited Feb 24, 2013 at 11:09

community wiki
3 revs, 2 users 74%
Justin Niessner

11 It's not just "if they really want to see how your software works, they will". If they care, they can probably guess without looking. 99.9%+ of software doesn't have any magic pixie dust algorithms. The hard part of programming isn't some special secret technique. It's just getting all the parts to line up and work. – Ken Jun 22, 2010 at 13:24

9 @Ken - Shhhh! You can't let the rest of the world know that most of the time we're not using magic pixie dust algorithms. – Justin Niessner Jun 22, 2010 at 13:27

9 Justin: Does Love count as a magic algorithm? Love is what makes *my* programs special. I don't think you can disassemble Love. – Ken Jun 22, 2010 at 13:49

Babel.NET is no more free. You can find a list of most common obfuscators (free and commercial ones) here. – InputOutput Feb 2, 2020 at 16:35

8

Well, you cannot FULLY protect your product from being cracked, but you can maximize/enhance the security levels and make it a little bit too difficult to be cracked by newbies and intermediate crackers.

But bear in mind nothing is uncrackable, only the software on server side is well protected and cannot be cracked. Anyway, to enhance the security levels in your application, you can do some simple steps to prevent some crackers "not all" from cracking your applications.

These steps will make these crackers go nuts and maybe desperate:

- Obfuscate your source code, obviously this will make your source code look like a mess and unreadable.
- Trigger several random checking routines inside your application like every two hours, 24 hours, one day, week, etc. or maybe after every action the user take.
- Save your released application's MD5 checksum on your server and implement a routine that can check the current file MD5 checksum with the real one on you server side and make it randomly triggered. If the MD5 checksum has been changed that means this copy has been pirated. Now you can just block it or release an update to block it, etc.
- Try to make a routine that can check if some of your codes (functions, classes, or specific routines) are actually have been modified or altered or even removed. I call it (code integrity check).
- Use free unknown packers to pack your application. Or, if you have the money, go for commercial solutions such as Thamida or .NET Reactor. Those applications get updated regularly and once a cracker unpack your application, you can just get a new update from those companies and once you get the new update, you just pack your program and release a new update.
- Release updates regularly and force your customer to download the latest update.

- Finally make your application very cheap. Don't make it too expensive. Believe me, you will get more happy customers and crackers will just leave your application, because it isn't worth their time to crack a very cheap application.

Those are just simple methods to prevent newbies and intermediate crackers from cracking your application. If you have more ideas to protect your application just don't be shy to implement them. It will just make crackers lives hard, and they will get frustrated, and eventually they will leave your application, because it's just doesn't worth their time.

Lastly, you also need to consider spending your time on coding a good and quality applications. Don't waste your time on coding complicated security layers. If a good cracker wants to crack your application he/she will do no matter what you do...

Now go and implement some toys for the crackers...

edited Feb 24, 2013 at 11:16

community wiki
2 revs, 2 users 65%
Robin Van Persi

There's Salamander, which is a native .NET compiler and linker from Remotesoft that can deploy applications

5

without the .NET framework. I don't know how well it lives up to its claims.

Share Improve this answer Follow

answered Feb 4, 2009 at 20:04

community wiki
Matthew Olenik

> The answer is quite easy: It is a pity, that most answers talk about obfuscation. Obfuscation is good for stopping the very first (Reflector-like) try to look in your code, that's all. That is not bad. And of course there is nothing which stops real hackers who understand assembly code, besides writing a SAAS application (even then they can try to hack your server). But there is more, there are tools like Salamander, .NET Reactor an other mentioned, who provide (maybe) nearly the same security form uncompiling as a C++ compiled Win32 .exe . Which of those tools is best, I cannot judge yet. – Philm Aug 6, 2013 at 20:23

5

If Microsoft could come up with a solution, we will not have pirated Windows versions, so nothing is very secure. Here are some similar questions from Stack Overflow and you can implement your own way of protecting them. If you are releasing different versions then you can adopt different techniques for different version so by the time first one is cracked the second one can take over.

- *Managing features on a license basis for a C++ application*

- *Secure a DLL file with a license file*
- *Licensing / protection software?*

Share Improve this answer Follow

edited May 23, 2017 at 12:18

community wiki
3 revs, 2 users 48%
Shoban

5

.NET Reactor

Update

Jared pointed out that de4dot claims to be able to decompile it.

> .NET Reactor provides complete protection for your sensitive intellectual property by converting your .NET assemblies into unmanaged processes which cannot be understood as CIL, and which no existing tool can decompile. Hackers have no access to any intelligible form of your source.
>
> Powerful and flexible, the .NET Reactor licensing features allow you to enforce your license conditions and protect your revenue stream by using hardware and software locks. The license manager can build trial or permanent licenses, in

> a matter of seconds. A fully documented software development kit (SDK), complete with examples, allows you to call the licensing system directly from your code, allowing you to create custom extensions to the licensing system.

Share Improve this answer Follow

edited Jun 20, 2020 at 9:12

community wiki
2 revs
loraderon

3 I tried to contact those guys with some question, I had about their product, but they never replied. Did your tried their product. I have gone with smart assembly and both their product and support is very good. But as I already said in the question obfuscation is one way, but not full proof.
– Priyank Bolia Feb 3, 2009 at 16:20

1 I had some issues with their product earlier and then I asked some question regarding high resolution icons in groups.google.se/group/net-reactor-users and I got a reply and a fix, but now it seems like they are hard to get hold of. To bad - it's a great product and I'm still using it – loraderon Feb 3, 2009 at 20:22

2 If "no existing tool can decompile", why is it listed as a supported obfuscator/packer on the de4dot features page?: bitbucket.org/0xd4d/de4dot – Jared Thirsk Nov 17, 2013 at 2:43

Probably because it's an old statement and they haven't updated their webpage. I am no longer a user of any obfuscation tool. – loraderon Nov 18, 2013 at 9:53

de4dot is a very powerful tool. I have tried it against several obfuscators and it makes a great job. However, it wasn't able to handle .NET Reactor (v6.0) protected files. Maybe de4dot isn't up to date. – InputOutput Feb 2, 2020 at 16:46

4

Here's one idea: you could have a server hosted by your company that all instances of your software need to connect to. Simply having them connect and verify a registration key is not sufficient -- they'll just remove the check. In addition to the key check, you need to also have the server perform some vital task that the client can't perform itself, so it's impossible to remove. This of course would probably mean a lot of heavy processing on the part of your server, but it would make your software difficult to steal, and assuming you have a good key scheme (check ownership, etc), the keys will also be difficult to steal. This is probably more invasive than you want, since it will require your users to be connected to the internet to use your software.

Share Improve this answer Follow

answered Mar 16, 2009 at 17:12

community wiki
rmeador

5 what if the server is down or users don't have internet access always, your method will simply frustrate the customers by having so many frequent round trip to the internet servers just to use an app. – Priyank Bolia Mar 17, 2009 at 9:37

I agree completely. That's why I said "this is probably more invasive than you want..." in my last line. I was just offering the OP the best solution that I thought was technically feasible, not the best approach to keeping customers happy :) – rmeador Mar 17, 2009 at 14:21

In the first quarter of 2010 (several months after this answer was written), the game development company Ubisoft tried this, and apparently the load on the server-side components was so big that the games were unplayable. Overall impression of the SW: "hassle to install, cannot be used offline, invasive *and unreliable*". So, should you decide that server-side processing is the way to go, make sure you can actually scale to demand. – Piskvor left the building Jun 21, 2010 at 7:11

3

Anything running on the client can be decompiled and cracked. Obfusification just makes it harder. I don't know your application, but 99% of the time I just don't think it's worth the effort.

Share Improve this answer Follow

answered Jun 22, 2010 at 13:13

community wiki
Kendrick

3

It's impossible to fully secure an application, sorry.

Share Improve this answer Follow

edited Feb 24, 2013 at 9:17

community wiki
2 revs, 2 users 50%
Fredou

3

Obfuscate the code! There is an example in *Obfuscating C# Code*.

Share Improve this answer Follow

edited Feb 24, 2013 at 11:05

community wiki
2 revs, 2 users 67%
ligaoren

2

Bear in mind that 99%+ of your users aren't going to be interested in examining your executable to see how it works.

Given that so few people are even going to bother trying and that most obfuscators can be worked around, is it worth your time and effort?

You'd be better off investing the time into improving your product so that more people want to use it.

Share Improve this answer Follow

answered Jun 22, 2010 at 13:16

community wiki
ChrisF

2

Just to add a warning: if you are going to use obfuscation, check that everything still works! Obfuscation might change things like class- and method-names. So if you use reflection to call certain methods and/or classes (like in a plugin-architecture) your application could fail after obfuscating. Also stacktraces might be useless to track down errors.

Share Improve this answer Follow

answered Jun 22, 2010 at 13:20

community wiki
Hans Kesting

If it's written in .NET and compiled to CIL, it can be reflected. If security is a concern and obfuscation is to be avoided, then I recommend writing your application using a non-managed language, which is, by nature, harder to reverse engineer.

2

Share Improve this answer Follow

edited Feb 24, 2013 at 8:51

community wiki
2 revs, 2 users 67%
Peter Mortensen

> How to make sure that the application is not tampered with, and how to make sure that the registration mechanism can't be reverse engineered.

2

Both have the same very simple answer: don't hand out object code to untrusted parties, such as (apparently) your customers. Whether it's feasible to host the application on your machines only depends on what it does.

If it isn't a web application, maybe you can allow for SSH login with X forwarding to an application server (or Remote Desktop Connection, I guess, for Windows).

If you give object code to nerdy type persons, and they think your program might be fun to crack, it *will* get cracked. No way around it.

If you don't believe me, point out a high-profile application that hasn't been cracked and pirated.

If you go with the hardware keys, it'll make production more expensive and your users are going to hate you for it. It's a real bitch to crawl around on the floor plugging and unplugging your 27 different USB thingies because software makers don't trust you (I imagine).

> There are packages out there that will encrypt your EXE and decrypt it when the user is allowed to use it

Of course, the way around it is to crack the "can-I-use-it" test so that it always returns true.

A nasty trick might be to use the byte values of the opcodes that perform the test somewhere else in the program in a dirty way that'll make the program crash with high probability unless the value is just right. It makes you linked to a particular architecture, though :-(

edited Feb 24, 2013 at 9:12

community wiki

3 revs, 3 users 58%
Jonas Kölker

doesn't a crash point is easy to debug, and override the code in .NET to bypass that check. Also how will you change the opcodes in .NET, can u elaborate on this? – Priyank Bolia Feb 3, 2009 at 9:00

Oh. I had C tricks in mind; say, take the address of the validation function, add up the 10 first bytes in that char array (cast the function pointer); pick any function f, and store [the address of f minus the previous sum] in fptr. Always call f as *(fptr + that sum). Precompute "that sum" – Jonas Kölker Feb 4, 2009 at 8:38

2

Just make a good application and code a simple protection system. It doesn't matter what protection you choose, it will be reversed... So don't waste too much time/money.

Share Improve this answer Follow

edited Feb 24, 2013 at 9:17

community wiki
2 revs, 2 users 50%
knoopx

2

When it comes to .NET, if you're releasing a Windows Forms application (or any application where the client has the Portable Executable file), it's able to be cracked.

If you want to stick with .NET and want to minimize the chance of having your source code taken, then you may want to consider deploying it as an ASP.NET application across a webserver, instead of making it a Windows Forms application.

Share Improve this answer Follow

edited Feb 24, 2013 at 10:38

community wiki
2 revs, 2 users 60%
Peter Mortensen

2

Frankly, sometimes we need to obfuscate the code (for example, register license classes and so on). In this case, your project is not free. IMO, you should pay for a good obfucator.

Dotfuscator hides your code and .NET Reflector shows an error when you attempt to decompile it.

Share Improve this answer Follow

edited Feb 24, 2013 at 11:03

community wiki
2 revs, 2 users 78%
Peter Mortensen

Highly active question. Earn 10 reputation (not counting the association bonus) in order to answer this question. The reputation requirement helps protect this question from spam and non-answer activity.