.NET OpenSource projects and strong named assemblies?

Asked 15 years, 11 months ago Modified 1 year, 6 months ago

Viewed 2k times

27

I am currently thinking about open-sourcing a project of mine and am in the process of preparing the source code and project structure to be released to the public. Now I got one question: how should I handle the signature key for my assemblies? Should I create a new key for the open-source version and publish it along with the other files to the SVN repository? Should I leave the key out and everyone who wants to compile the code should generate his own key?

How do you handle this? I feel a little bit uncomfortable with releasing a signature key to the public.

.net open-source strongname

Share

Improve this question

Follow

edited May 19, 2009 at 23:39

hlovdal

28.1k ● 11 ● 100 ● 175

asked Dec 28, 2008 at 12:51

Martin Carpella

12.5k ● 8 ● 42 ● 54

5 You SHOULD feel uncomfortable releasing a signature key to the public. – Cheeso May 20, 2009 at 14:31

My concern is with LGPL requirement for someone to be able to use a modified version of the LGPL library. If I don't release the key I used to sign the original LGPL build of a library I am using, the user cannot drop-in replace the library, which would violate LGPL... – Martin Carpella Feb 25, 2010 at 16:40

Related: stackoverflow.com/q/11656520/126014 – Mark Seemann Nov 11, 2013 at 12:30

4 Answers

Sorted by: Highest score (default)

24

For Protocol Buffers, I release the key. Yes, that means people can't actually trust that it's the original binary - but it makes life significantly easier for anyone who wants to modify the code a bit, rebuild it, and still be able to use it from another signed assembly.

If anyone really wants a version of Protocol Buffers which they can trust to be definitely the legitimate one built with the code from GitHub, they can easily build it themselves from the source that they trust.

I can certainly see it from both sides though. I think if I were writing an Open Source project which revolved around *security* that might be a different matter.

edited Feb 8, 2018 at 18:19

Share Improve this answer

Follow

kenorb
166k ● 94 ● 706 ● 773

answered Dec 28, 2008 at 13:23

Jon Skeet
1.5m ● 889 ● 9.3k ● 9.3k

1 But why you sign it? I understand the reasons behind strong named assemblies when the keys are private, but not in "Protocol Buffers" case. – Jader Dias Feb 24, 2009 at 1:03

1 Releasing the key seems like a really bad idea. Who brokers the version numbers? What if Joe in BigCorp Department X downloads ProtocolBuffers.sln, modifies it, assigns version 2.1.0.0. And then Shirley in BigCorp Dept Y does the same thing. Both are signed, strongly named assemblies, but they are different. ?? How hard is it for these people to sign their modified assemblies with their own keys? – Cheeso May 20, 2009 at 3:37

4 If BigCorp decides to screw things up, then they can indeed make their own lives harder. Without releasing the keys, it makes *everyone's* life harder. I'd rather make life easier for most and allow people to very occasionally shoot themselves in the foot than make life harder for everyone else.
– Jon Skeet May 20, 2009 at 5:25

2 How hard is it for them to create a keypair to sign the assembly? It's one extra step on the command line! And it's done only ONCE. Ever. This may not matter for ProtoBufs, but for something more ubiquitous, let's say an XML library, it leads to perdition. The Jar-H3LL that surrounded Java's Xerces lib is a great lesson. There were versions everywhere. You'd have apps that depended on multiple OSS projects, and they each had dependencies on different versions of Xerces. Strong naming (which Java lacks) fixes this. But only strong naming with integrity. – Cheeso May 20, 2009 at 14:23

3 @Cheeso: For Noda Time, I think we're going to have a key which anyone can build with, and then a private key which only a few committers have access to. That way everyone still gets a signed assembly by default, but the prebuilt binary is verifiably "ours". I think that's the best of both worlds. It does strike me that if the problems you describe (protecting others from themselves) are secondary concerns, then you should describe the primary *practical* concerns though.
– Jon Skeet Feb 18, 2010 at 6:21

19

Don't release the key.

You SHOULD feel uncomfortable releasing a signature key to the public. It is not the *project's signature*. It's **YOUR signature**. The integrity of the signature on the binary is maintained only if you keep your key secret. Releasing the key subverts the meaning and intent of signed assemblies and strong naming, which introduces new possibilities for errors, and thus makes every system less reliable. *Don't release the key*.

For DotNetZip, I don't release the key. But here's the key point: The key does not belong to the project; it is **my key**. Many people have asked for the key so they can re-build the signed binary, but that makes no sense. I use the key to sign more than DotNetZip. Any binary signed with that key is signed by me, by definition. Any two binaries that have the same strong name using my key, are guaranteed to be identical. Releasing keys removes those guarantees, and defeats the entire purpose of strong names, and the security surrounding them.

Imagine devs choosing their own version numbers, and re-signing a modified binary with my key. Now the world would have 2 assemblies with the same strong name, but with different contents.

Imagine if I were able to sign *any* assembly with YOUR key. If you released your key, I could add any code I liked - even malicious code - and then sign it, and surreptitiously replace any "good" signed binary of yours with a "bad" one. No one would be able to tell the difference.

This is broken. Freely sharing keys eliminates any advantage to using signed assemblies at all.

If people want to modify the code in a project and then re-use the modified version in a strongly-named assembly, they can sign the modified version with their own key. It's not difficult.

edited May 20, 2009 at 14:43

answered May 20, 2009 at 3:34

Cheeso
192k ● 105 ● 484 ● 734

How do you handle the requirement for drop-in-replacements (as for instance required by LGPL)? Modified versions of your library won't load from a signed assembly unless signed by the same key. – Martin Carpella Jun 14, 2009 at 14:15

I don't understand the question. I think you are asking me, how do I enable someone else to use a different library than mine as a drop-in replacement. And the answer to that is, I don't. I don't enable that. – Cheeso Jun 14, 2009 at 21:36

2 Sorry for very late follow-up on this: LGPL *requires* that you enable someone to drop in another version of the LGPL library. If (my assumed) LGPL library cannot be replaced, it violates LGPL. – Martin Carpella Feb 25, 2010 at 16:38

FYI, My lib is not LGPL. In my opinion, LGPL is strange-upon-strange. Wikipedia says that the terms in the LGPL specifically refer to C-language linking issues, and that amendments were needed to handle LISP code. Seriously? That's just weird. That's a very broken license. – Cheeso Feb 25, 2010 at 21:18

11

I wouldn't release the key publicly. The whole point of having a signed assembly is that people can trust that you're the only one who touched the binary, and so if there is any illegitimate code added then the signing is off and people know not to trust the assembly.

Signing assemblies protects you from other people adding "bad" code to your binary and pretending it's a legitimate release.

Share Improve this answer

Follow

answered Dec 28, 2008 at 12:59

Cameron MacFarland

71.8k • 20 • 105 • 134

While I agree with you, I am not sure how to handle this in an open-source environment. Every binary distributor of the

project should then use his own code? As it could be a used as a library, this would also mean that no other people can create drop-in replacements of the components.
– Martin Carpella Dec 28, 2008 at 13:12

2 Every party who re-distributes the project should (a) redist an unmodified, signed version of the binary you produce, or (b) redist a modified version of the binary, signed with a different key - their own key, or (c) redist the source. The options are not mutually exclusive. But they important part is that the key must belong to the signing party, not to the project. Otherwise the meaning and value of signing is destroyed, as Cameron points out. – Cheeso May 20, 2009 at 14:29

I completely agree, but it violates / makes the use of LGPL libraries impossible, if my interpretation of the license is correct. – Martin Carpella Apr 15, 2010 at 9:37

0

In general .Net only cares - when loading an assembly - on the public key token, and not on the actual key and signature (besides when it comes to `InternalsVisibleTo` in which case the full public key is required).

While in order to sign a brand new assembly to have a specific public key token (and of course a public key) you will need to have access to the original key, it is however possible to modify the IL for the original assembly (either directly or by use of an IL weaver such as Fody or a tool like dnSpy) and .Net will happily load it.

In fact it appears that the IgnoresAccessChecksToGenerator tool is doing just that.

As such I don't see much value in keeping it private and I do see that many popular open source projects, such as GitHub for Visual Studio or Log 4 Net are actually keeping it public.

Also note that Microsoft recommends to check in the .snk file for open source projects:

> If you are an open-source developer and you want the identity benefits of a strong-named assembly for better compatibility with .NET Framework, consider checking in the private key associated with an assembly to your source control system.

And they clarify:

> Warning
>
> Do not rely on strong names for security. They provide a unique identity only.

Share Improve this answer Follow

edited May 30, 2023 at 19:18

answered May 30, 2023 at 19:06

yoel halb

12.7k ● 3 ● 58 ● 56