Realtime publishers

The Essentials Series:
Code-Signing Certificates

How Are Certificates Used?

sponsored by

VeriSign®

by Don Jones

How Are Certificates Used? 1
Web Applications 1
Mobile Applications 2
Public Software 2
In-House Software 3
Scripts 3
Software Security 4

Copyright Statement

© 2009 Realtime Publishers, Inc. All rights reserved. This site contains materials that have been created, developed, or commissioned by, and published with the permission of, Realtime Publishers, Inc. (the “Materials”) and this site and any such Materials are protected by international copyright and trademark laws.

THE MATERIALS ARE PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. The Materials are subject to change without notice and do not represent a commitment on the part of Realtime Publishers, Inc or its web site sponsors. In no event shall Realtime Publishers, Inc. or its web site sponsors be held liable for technical or editorial errors or omissions contained in the Materials, including without limitation, for any direct, indirect, incidental, special, exemplary or consequential damages whatsoever resulting from the use of any information contained in the Materials.

The Materials (including but not limited to the text, images, audio, and/or video) may not be copied, reproduced, republished, uploaded, posted, transmitted, or distributed in any way, in whole or in part, except that one copy may be downloaded for your personal, non-commercial use on a single computer. In connection with such use, you may not modify or obscure any copyright or other proprietary notice.

The Materials may contain trademarks, services marks and logos that are the property of third parties. You are not permitted to use these trademarks, services marks or logos without prior written consent of such third parties.

Realtime Publishers and the Realtime Publishers logo are registered in the US Patent & Trademark Office. All other product or service names are the property of their respective owners.

If you have any questions about these terms, or if you would like information about licensing materials from Realtime Publishers, please contact us via e-mail at info@realtimepublishers.com.

How Are Certificates Used?

Code-signing certificates are used in an ever-growing number of ways, but all of these uses have two things in common: They seek to guarantee that content hasn't been tampered with, and they seek to positively identify the people who produced the code.

Web Applications

We're not talking about simple HTTPS encryption, here: Web applications have grown ever-more complex. Web components written in Java, Microsoft's .NET Framework, Adobe Air, Adobe Flash, and other languages all run code that could potentially do a great deal of damage—or at the very least, convince users to provide sensitive information to the wrong party. Web browsers are becoming increasingly suspicious of unsigned code, and many newer Web browsers will, by default, simply refuse to execute it.

The use of code-signing certificates in this scenario can be confusing because the same scenario might well use a Web server certificate (commonly called an SSL Certificate) to create an authenticated, encrypted communications channel between the Web server and the client browser. The differences are:

- When software—whether Java, ActiveX, or something else—is downloaded to the client browser, that software should be signed by using a code-signing certificate. Modern browsers usually make this a requirement. Newer versions of Microsoft Internet Explorer, for example, are configured by default to not download and run unsigned ActiveX controls—meaning Web pages containing these controls may not display properly, may not function properly, and may cause error or warning messages to be displayed to the user.
- The communications between that software and the Web site will normally be in clear text, unless a Web server certificate is used to create an encrypted communications channel. Thus, even if the software itself is signed, someone could intercept the communications sent back to the Web server if another certificate isn't used.

Mobile Applications

Mobile platform operators—think cell phones, here—are terrified of the prospect of malicious code running *on a radio communications device.* Think of the damage that could be done to the telecommunications infrastructure if some hacker was able to get malicious code onto a cell phone! For that reason, nearly every mobile platform will refuse to run code that hasn't been digitally signed. Does that guarantee the safety of the code? Of course not, but it does guarantee that any malicious code can be tracked back to its author, who can be dealt with accordingly.

Mobile platforms generally have a *very* limited list of Certification Authorities (CAs) that they trust. In some cases, you need to obtain a code-signing certificate from the mobile platform manufacturer, as is the case with Apple's iPhone. In other cases, you might be directed to a specified commercial CA who has been authorized to issue certificates recognized by that particular device.

Public Software

Public software is perhaps one of the first uses of code-signing certificates that we're all familiar with. Because it's often distributed (or at least updated) via the public Internet, code signing helps ensure that the identity of the publisher is known and verified, and that the code itself hasn't been tampered with since it was released.

> **Note**
>
> I used the term "public" in this section because the same concerns that apply to commercial "for-sale" software also apply to software that is available to the public but not strictly commercial: open-source software, shareware, freeware, and so forth.

Frankly, *all* public software should be code signed. Signed software helps provide an additional level of assurance to customers, and costs relatively little compared with the other costs of public software development and distribution. Customers, were they better educated about signatures and software security, would likely refuse to run unsigned software—so it seems sensible for software publishers to take the first step and simply sign all the code they release.

Sometimes, you might not even realize that the software you're creating and distributing is a candidate for signing but nearly any kind of executable code can, and should, be signed. Web browser plug-ins, Java applications, dashboard widgets, and many other "smaller" types of software often support signing, and whenever they do support it, you should definitely take advantage of the capability.

In-House Software

Digitally signing software that is produced internally can be beneficial, too. Signing ensures that malware doesn't tamper with your software as it sits or runs on users' computers, for example.

Code signing can also be used as a form of change control within the environment. For example, as we'll cover in the later "Software Security" section, you can configure your environment to run *only* trusted (signed) code. Internal software developers can be issued code-signing certificates, which are trusted only on their development computers (often called *local certificates* or *self-signed* certificates); although those certificates will permit the developers to run, test, and debug their code, it won't allow the code to be deployed to the production environment because those local certificates aren't trusted by the larger environment. Instead, a manager or other change control person holds a code-signing certificate that *is* trusted by the production environment and is responsible for code signing any software that is deemed ready for production. This use of certificates helps enforce quality assurance processes, for example, by preventing un-reviewed or unapproved code from being easily deployed in production.

Scripts

Certain forms of scripting have often created security vulnerabilities. Microsoft's VBScript and Windows PowerShell are two examples, particularly in the case of VBScript that has in fact been used to create numerous annoyances and viruses.

Interestingly, code-signing certificates have been able to secure these scripting languages for years now, although many scripters are unaware of the benefits. Both Microsoft's Windows Script Host and Windows PowerShell (as well as other, similar technologies on both Windows and other platforms) can be configured to only run *trusted* scripts—that is, scripts that have been digitally signed by means of a certificate issued from a trusted CA. Again, the signature itself doesn't attest to the safety of the script. Rather, it proves the identity of the script's author and proves that the author's original script code is intact—so if the script performs a malicious action, the author can be identified and held responsible.

Software Security

Software security is an aspect of code signing that really applies to commercial and in-house software, but in a unique way. Think for a moment about how we combat malicious software today: We typically have an array of anti-software: antivirus, anti-spyware, anti-whatever. These pieces of software are essentially blacklists, meaning they look for software that is known to be malicious, and try to stop it from running. There are several problems with the blacklisting approach:

- The people producing malicious software are, by definition, always a step ahead of the anti-software people. That means the anti-software people are always going to be reacting, and trying to react faster and faster.
- Anti-software programs consume computing resources, meaning the act of protecting your computer makes it slightly less powerful.
- Anti-software programs have to work at a fairly low level of the operating system (OS), meaning they have ample opportunity to negatively impact the entire computer if they're not well written.
- Blacklisting places the decision-making process on the anti-software company—*they* get to decide what's trusted and what's not, rather than the software publisher and user working together.

The reason we need all this anti-software, of course, is that most computers are configured to run whatever software we throw at them, and because very few users can accurately determine whether a piece of software is safe.

But that doesn't *have* to be the case, and it won't always *be* the case. In Windows XP and Windows Server 2003, for example, Microsoft introduced a new component of Group Policy called Software Restriction Policies (SRP). With SRP, you can change the computer's default mode from executing *any* software to executing *no* software. You then create *exceptions* to that default mode. For example, you might specify individual applications that are allowed to run—all of a sudden, you don't need that anti-software as much! SRP will only allow designated software to run—a concept called *whitelisting.* The problem, of course, is that individually designating every application in your environment isn't practical. Instead, you might tell SRP to allow all applications *that are digitally signed,* specifying a trusted CA. Instantly, every application that has been signed by a certificate from that trusted CA will be allowed—and nothing else will run. This is essentially making a regular computer behave like a mobile device in terms of software security.