AppInit DLLs in Windows 7 and Windows Server 2008 R2

May 15, 2009

Abstract

This paper provides information about the AppInit\_DLLs infrastructure for the Windows® family of operating systems. It provides guidelines for application developers to ensure that any applications that depend on the AppInit\_DLLs mechanism function correctly when run in Windows 7 or Windows Server® 2008 R2.

The AppInit\_DLLs infrastructure enables applications to load arbitrary DLLs into all user-mode processes. The most common use of this mechanism is API hooking. The AppInit\_DLLs infrastructure has been changed for Windows 7 and Windows Server 2008 R2 to help improve the integrity and visibility of code that is running on these systems. This paper includes information about the new code signature requirement that has been introduced for AppInit\_DLLs. It also includes information that is related to the following message that is observed in Event Viewer:

Custom dynamic link libraries are being loaded for every application. The system administrator should review the list of libraries to ensure they are related to trusted applications.

This information applies to the following operating systems:
 Windows 7
 Windows Server 2008 R2

References and resources discussed here are listed at the end of this paper.

The current version of this paper is maintained on the Web at:
 <http://www.microsoft.com/whdc/driver/install/AppInit-Win7.mspx>

Disclaimer: This is a preliminary document and may be changed substantially prior to final commercial release of the software described herein.

The information contained in this document represents the current view of Microsoft Corporation on the issues discussed as of the date of publication. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication.

This White Paper is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, AS TO THE INFORMATION IN THIS DOCUMENT.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, email address, logo, person, place or event is intended or should be inferred.

© 2009 Microsoft Corporation. All rights reserved.

Microsoft, Authenticode, Visual Studio, Windows, Windows Server, and Windows Vista are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

Document History

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Date | Change |  |  |  |
| May 15, 2009 | Added note in Abstract about Event Viewer message |
| April 22, 2009 | First publication |

Contents

[Introduction 4](#_Toc229998935)

[AppInit\_DLL Functionality in Windows 7 and Windows Server 2008 R2 4](#_Toc229998936)

[Configuration 4](#_Toc229998937)

[DRM-Protected Processes 5](#_Toc229998938)

[Windows 7 Upgrades 5](#_Toc229998939)

[Windows 7 5](#_Toc229998940)

[Windows Server 2008 R2 5](#_Toc229998941)

[Blocked Process List 5](#_Toc229998942)

[System Event Log Entry 6](#_Toc229998943)

[Developer Best Practices 6](#_Toc229998944)

[Code Signature Overview 6](#_Toc229998945)

[Detailed Code Signature Information 9](#_Toc229998946)

[Getting Started with Code Signing 9](#_Toc229998947)

[Overview of the Code-Signing Tools 9](#_Toc229998948)

[Where to Find the Code-Signing Tools 11](#_Toc229998949)

[How to Run the Code-Signing Tools 12](#_Toc229998950)

[How to Prepare the Signing Computer 12](#_Toc229998951)

[How to Test-Sign a DLL Binary File 13](#_Toc229998952)

[Step 1: Create a Test Certificate by Using MakeCert 13](#_Toc229998953)

[Step 2: Install the Test Certificate in the Trusted Root Certification Authorities Certificate Store 16](#_Toc229998954)

[Step 3: Test-Sign a DLL Binary File with an Embedded Signature 17](#_Toc229998955)

[Step 4: Verify the Test Signature 17](#_Toc229998956)

[How to Install and Verify a Test-Signed DLL 18](#_Toc229998957)

[Step 1: Install the Test Certificates on the Test System 18](#_Toc229998958)

[Step 2: Enable the Test-Signing Boot Configuration Option 20](#_Toc229998959)

[Step 3: Enable Code Integrity Event Logging and System Auditing 21](#_Toc229998960)

[Step 4: Copy and Install the Test-Signed DLL to the Test Computer 22](#_Toc229998961)

[Step 5: Verify that the Test-Signed DLL Is Operating Correctly 22](#_Toc229998962)

[How to Release-Sign a DLL Binary File 22](#_Toc229998963)

[Step 1: Obtain and Install an SPC 23](#_Toc229998964)

[Step 2: Obtain and Install a Cross-Certificate 25](#_Toc229998965)

[Step 3: Release-Sign a DLL File with an Embedded Signature 26](#_Toc229998966)

[How to Install and Verify a Release-Signed DLL 28](#_Toc229998967)

[Step 1: Disable the Test-Signing Boot Configuration Option 29](#_Toc229998968)

[Step 2: Enable Code Integrity Event Logging and System Auditing 29](#_Toc229998969)

[Step 3: Restart the Test Computer 29](#_Toc229998970)

[Step 4: Copy and Install the Release-Signed AppInit DLL to the Test Computer 29](#_Toc229998971)

[Step 5: Verify that the Release-Signed AppInit DLL Is Operating Correctly 29](#_Toc229998972)

[Feedback 29](#_Toc229998973)

[Resources 30](#_Toc229998974)

# Introduction

One of the key reasons for the success of the Windows® platform is its inherent extensibility. Not only does the Windows operating system provide a platform for developing and executing a variety of applications and scenarios, Windows also provides infrastructures that allow applications to extend the core functionality of the operating system. One example of this extensibility is an infrastructure known as AppInit\_DLLs.

The AppInit\_DLLs infrastructure provides a mechanism that lets an arbitrary list of DLLs (AppInit DLLs) be loaded into each user-mode process on the system. Today, only a small set of legitimate applications use this mechanism. Unfortunately, a larger set of malware use this mechanism. Applications and malicious software both use AppInit DLLs for the same basic reason, which is to hook APIs.

The AppInit\_DLLs infrastructure provides an easy way to hook system APIs by allowing a custom DLL to be loaded into the address space of every interactive application. After the custom DLL is loaded, it can hook a well-known system API and implement alternate functionality.

Using AppInit DLLs can unintentionally cause system deadlocks and performance problems, because they are loaded during the initialization of user32.dll. We do not recommend that you load a DLL during the initialization of another DLL. For more information about problems that can occur when you use AppInit DLLs, see ”AppInit\_DLLs should be renamed Deadlock\_Or\_Crash\_Randomly\_DLLs” in one of the blogs on the MSDN® Web site.

To help improve system reliability, performance, and visibility into the origin of software, the AppInit\_DLLs mechanism has been updated for Windows 7 and Windows Server® 2008 R2 to include a new code-signing requirement.

This paper describes the new code-signing requirement and provides best practices for developers to follow when they use AppInit DLLs.

# AppInit\_DLL Functionality in Windows 7 and Windows Server 2008 R2

Beginning with Windows Vista®, the AppInit\_DLLs infrastructure is disabled by default. This default behavior remains unchanged in Windows 7 and Windows Server 2008 R2.

## Configuration

The behavior of the AppInit\_DLLs infrastructure is configured by a set of values that are stored under the **HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows NT \CurrentVersion\Windows** key in the registry. These registry values are described in Table 1.

Table 1. AppInit\_DLLs Infrastructure Registry Values

|  |  |  |
| --- | --- | --- |
| Value | Description | Sample values |
| LoadAppInit\_DLLs(REG\_DWORD) | Value that globally enables or disables AppInit\_DLLs. | 0x0 – AppInit\_DLLs are disabled.0x1 – AppInit\_DLLs are enabled. |
| AppInit\_DLLs(REG\_SZ) | Space -or comma-delimited list of DLLs to load. The complete path to the DLL should be specified by using short file names.  | C:\PROGRA~1\Test\Test.dll |
| RequireSignedAppInit\_DLLs(REG\_DWORD) | Require code-signed DLLs. | 0x0 – Load any DLLs.0x1 – Load only code-signed DLLs. |

## DRM-Protected Processes

AppInit DLLs are not loaded into any DRM-protected process. This behavior cannot be configured. For more information about protected processes, see “Protected Processes in Windows Vista” on the WHDC Web site.

## Windows 7 Upgrades

When you upgrade a Windows Vista system to Windows 7, the AppInit DLLs that are listed in the registry do not migrate to the Windows 7 registry. Also, when you upgrade from an earlier version of Windows to Windows 7, the AppInit DLL image files are not copied into the new Windows 7 operating system directories.

## Windows 7

We recommend that you code-sign all DLLs that the AppInit\_DLLs infrastructure loads into Windows 7. But for application compatibility, by default Windows 7 loads all AppInit DLLs, regardless of whether these DLLs are code signed. Nevertheless, we recommend that you digitally code-sign your AppInit DLLs to help improve the reliability and security of Windows 7 and to prepare for more stringent code-signing requirements in future versions of Windows.

The **RequireSignedAppInit\_DLLs** registry value controls whether AppInit DLLs must be code signed. In Windows 7, this value is set to 0 (load any DLLs) by default.

## Windows Server 2008 R2

By default, all DLLs that Windows Server 2008 R2 loads by using the AppInit\_DLLs infrastructure must be code signed. In Windows Server 2008 R2, the **RequireSignedAppInit\_DLLs** registry value is set to 1 (load only code-signed DLLs) by default.

## Blocked Process List

AppInit DLLs are not loaded into the following security critical processes:

* Windows Defender.
* Windows Software Licensing service.
* Microsoft Hyper-V critical processes (vmms.exe and vmwp.exe).

You cannot configure this behavior.

## System Event Log Entry

If an application enables AppInit DLLs, Windows logs a warning in the System Event Log. The event log entry includes a list of the DLLs that are loaded by using the AppInit\_DLL mechanism. You can view this list on the **Details** tab in Event Viewer. Wininit logs this warning one time for each boot session. Table 2 shows the fields of the event log entry when you view the entry in Event Viewer.

Table 2. System Event Log Entry Fields in Event Viewer

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Provider | Level | Event ID | Channel | Message |
| Wininit | Warning | 11 | System | Custom dynamic link libraries are being loaded for every application. The system administrator should review the list of libraries to ensure that they relate to trusted applications. |

# Developer Best Practices

The following are best practices for software developers who create AppInit DLLs:

* Code-sign AppInit DLLs.

Software developers should code-sign any AppInit DLLs to help improve the reliability and security of Windows 7. Later versions of Windows will load only code-signed AppInit DLLs and will not include a registry key to disable this requirement.

* Run DLLs only in required processes.

The AppInit\_DLLs mechanism loads the specified DLLs in all user-mode processes on the system. If an AppInit DLL must provide API hooks only to specific processes, then the DLL should call the **GetModuleFileName** function from within **DllMain** to retrieve the name of the process in which it is loaded. If the process is not a process that requires the API hooking that the AppInit DLL provides, then the DLL should simply return from **DllMain**.

* Only call APIs that are exported from kernel32.dll in the DLL's initialization routine.

Because AppInit DLLs are loaded very early during process initialization, they should use only APIs that are exported from kernel32.dll in their initialization routines.

# Code Signature Overview

Code signing uses digital signatures to verify the integrity of software applications and components. A valid digital signature identifies the software's publisher and verifies that the software has not been modified since it was signed.

A digital signature assures users that the software that they are installing or running is genuine. Software that is code signed cannot be anonymous. By providing the identity of the publisher of AppInit DLLs in the digital signature, end-users and IT professionals can communicate with the software vendor if their AppInit DLL causes system performance or reliability issues.

**Note:** Code signing does not provide any guarantee of the quality or functionality of software. Code-signed software can still contain flaws or security vulnerabilities or can be malicious in nature.

In Windows 7 and Windows Server 2008 R2, if the AppInit\_DLLs infrastructure is enabled and the **RequireSignedAppInit\_DLLs** registry value is set to 1, then only code-signed AppInit DLLs are loaded. In this situation, if a DLL in the **AppInit\_DLLs** registry value has not been code signed or has a problem with its digital signature, the operating system does not load the DLL.

The digital signature of a DLL is usually embedded in the binary file (the *.dll* file). When the digital signature is embedded in the binary file, Windows 7 and Windows Server 2008 R2 verify the signature only the first time that the DLL is loaded. The digital signature is cached after the DLL is loaded the first time. If the DLL is subsequently reloaded into the same process, the operating system does not verify the digital signature again.

To code-sign a DLL and embed the digital signature in the binary file, the following steps are required:

1. Link the DLL with the **/integritycheck** linker option. The **/integritycheck** option is supported by both Microsoft® Visual Studio® 2005 and Visual Studio 2008, including the free Express Editions.

To verify that the **/integritycheck** linker option was specified when a DLL was linked, run the following command in a command prompt window:

dumpbin /headers DLLname.dll

If the **/integritycheck** linker option was specified when the DLL was linked, **Check integrity** is included in the list of DLL characteristics in the output from the dumpbin utility, as shown in the following example:

c:\>dumpbin /headers test.dll

Microsoft (R) COFF/PE Dumper Version 9.00.21022.08

Copyright (C) Microsoft Corporation. All rights reserved.

Dump of file test.dll

PE signature found

File Type: DLL

FILE HEADER VALUES

 14C machine (x86)

 3 number of sections

 48F57FF2 time date stamp Tue Oct 14 22:30:26 2008

 0 file pointer to symbol table

 0 number of symbols

 E0 size of optional header

 2122 characteristics

 Executable

 Application can handle large (>2GB) addresses

 32 bit word machine

 DLL

OPTIONAL HEADER VALUES

 10B magic # (PE32)

 9.00 linker version

 1200 size of code

 E00 size of initialized data

 0 size of uninitialized data

 16E3 entry point (100016E3)

 1000 base of code

 3000 base of data

 10000000 image base (10000000 to 10004FFF)

 1000 section alignment

 200 file alignment

 6.01 operating system version

 6.01 image version

 5.01 subsystem version

 0 Win32 version

 5000 size of image

 400 size of headers

 708E checksum

 3 subsystem (Windows CUI)

 1C0 DLL characteristics

 Dynamic base

 Check integrity

 NX compatible

2. Code-sign the DLL with a Authenticode® signature.

The DLL must be signed with the **/ph** (page-hash) option. Page-hash enforcement verifies the signature on each page as it is loaded into memory. Because of this requirement, the DLL must be signed on a system that is running Windows Vista or a later version of Windows. The DLL should also be signed with the **/t** (timestamp) option to prevent the digital signature from expiring when the code-signing certificate expires. When you release sign the DLL, you must use the **/ac** option to reference a cross-certificate. This is required because the signing code path exists in the Windows kernel, which has a hard-coded root of trust. The cross-certificate is required to chain to the issuing certificate authority.

The following commands are examples of how to use the SignTool utility to sign a DLL.

* Test-signing example:

Signtool sign /v /ph /s My /n Contoso.com(Test) /t http://timestamp.verisign.com/scripts/timestamp.dll DLLname.dll

* Release-signing example:

Signtool sign /v /ph /ac MSCV-VSClass3.cer /s my /n contoso.com /t http://timestamp.verisign.com/scripts/timestamp.dll *DLLname*.dll

For more detailed information about code signing and information about how to obtain the SignTool utility and other code-signing tools, see “Detailed Code Signature Information” later in this paper.

# Detailed Code Signature Information

This section provides detailed information about how to use Windows code-signing tools to digitally sign DLL binary files that are designed for Windows 7 and later versions of Windows. This section includes the following information:

* Where to obtain code-signing tools.
* Where to obtain cross-certificates.
* How to prepare systems to use code-signing tools to build, sign, and test DLL binary files.
* Detailed examples of how to use the tools to test-sign and release-sign DLL binary files.
* How to verify test and release signatures.
* Recommendations on securing signing keys and on the release process.

## Getting Started with Code Signing

You can use several approaches to build, sign, and test DLL binary files. To become more familiar with the code-signing tools, you can perform all the examples in this paper on a single computer. However, this paper assumes separate computers for each process, which is often the best option for a production environment.

* Build computer**.**

The computer that is used to build the DLL. It should be running Windows XP SP2, Windows Server 2003, or a later version of Windows.

* Signing computer**.**

The computer that is used to sign the DLL binary files for Windows 7. It must run Windows Vista, Windows Server 2008, or a later version of Windows. The code-signing tools must be installed on this computer.

* Test computer**.**

The computer that is used to test the signed DLL. It must run Windows 7 to test the new AppInit DLL infrastructure.

### Overview of the Code-Signing Tools

This section briefly describes the following tools, which are used in signing binary DLL files:

* Capicom.dll
* CertMgr
* CertUtil
* MakeCert
* Pvk2Pfx
* SignTool
* Capicom.dll

Capicom.dll exports an API that you can use to add security that is based on cryptography to applications. Versions of SignTool that were distributed before the release of the Windows 7 Windows Driver Kit (WDK) and Software Development Kit (SDK) use Capicom.dll. If you use this version of SignTool, both Capicom.dll and SignTool must be present on the signing computer. You should copy Capicom.dll to the same directory as SignTool. The latest version of Capicom.dll is 2.1.0.2.

**Note:** Versions of SignTool that are released with the Windows 7 versions of the WDK and SDK do not require Capicom.dll.

#### CertMgr

CertMgr manages certificates, certificate trust lists (CTLs), and certificate revocation lists (CRLs). The tool has the following three functions:

* Displaying certificates, CTLs, and CRLs.
* Adding certificates, CTLs, and CRLs from one certificate store to another.
* Deleting certificates, CTLs, and CRLs from a certificate store.

The Microsoft Management Console (MMC) Certificates snap-in—which is also used during code signing—is named Certmgr.msc. Despite the similarity in names and features, it differs from the CertMgr tool.

#### CertUtil

CertUtil is a command-line tool that helps manage certificates and is included with Windows Vista and later versions of Windows.

#### MakeCert

MakeCert generates test certificates and places them in a file, a system certificate store, or both. The certificates can be either self signed or issued and signed by the Root Agent key. To test-sign AppInit DLLs, we recommend self-signed certificates.

For test signing, you should use only test certificates such as those that MakeCert generates. Generally, you should not use release certificates that are issued by a commercial third-party certification authority (CA) for test signing. For more information, see “Code-Signing Best Practices.”

#### Pvk2Pfx

Pvk2Pfx converts keys that are stored in .spc and .pvk files to the personal information exchange (.pfx) file format.

For a key to be used with SignTool, the key must be stored in the system certificate store or a .pfx file. However, some CAs use the .pvk file format to store the private key of the digital certificate and the .spc or .cer file format to store the public key. In particular, VeriSign Class-3 certificates are currently packaged as a .pvk and a .spc files. Before you use such a certificate for code signing, you must use Pvk2Pfc to convert the .pvk and .spc files to the .pfx format.

When possible, the preferred approach is to store private keys in a hardware security module, such as a smartcard. For more information about how to securely manage private keys, see “Code-Signing Best Practices.”

#### SignTool

SignTool is a command-line tool that signs, verifies, and timestamps files. It can be used with any Authenticode-supported file format, including portable executable (PE, which includes .exe, .dll, and .sys files), catalog (.cat), and cabinet (.cab) formats. SignTool verifies the following information about the signing certificate:

* Whether it was issued by a trusted CA.
* Whether it has been revoked.
* Whether it is valid for code signing.
* Whether it is within the certificate’s validity period.
* Optionally, whether the certificate is valid for a specific policy.

You can use SignTool for several other purposes, including the following:

* Verifying signatures for individual files in a signed catalog file.
* Verifying signatures against different Authenticode policies.
* Displaying a signature’s certificate chain.
* Displaying the SHA1 hash value of a file if the file is signed.

The version of SignTool that is available with Windows 7 displays the hash result for the specific algorithm that was used to calculate the hash, including SHA1, SHA256, SHA384, or SHA512.

* Displaying errors for files that did not verify.
* Adding and removing catalog files from the catalog database.

### Where to Find the Code-Signing Tools

The code-signing tools are available from two sources:

* The Windows SDK for Windows Vista and .NET Framework 3.0 (or later versions), which contains information and tools for developing 32-bit and 64-bit Windows-based DLLs. It is available as a free download.
* The WDK, which contains information and tools for developing Windows drivers. It is available as a free download.

The code-signing tools are available for Windows Vista and later versions of Windows.

Note: As defined in their end-user license agreements (EULAs), the tools in the Platform SDK and the WDK cannot be redistributed.

Table 1 lists the locations of code-signing and related tools. For links to these sites, see “Resources.”

Table 1. Where to Find Code-Signing and Related Tools

|  |  |  |
| --- | --- | --- |
| Tool | Source  | Additional sources |
| MakeCert | Windows SDK | WDK |
| CertMgr | Windows SDK | WDK |
| SignTool | Windows SDK | WDK |
| Capicom.dll v.2.1.0.2Capicom.dll is not required by versions of SignTool that are distributed with the Windows 7 SDK or WDK.  | Windows SDK | Download Center |
| Pvk2Pfx | Windows SDK | WDK |

### How to Run the Code-Signing Tools

Most of the code-signing tools run from the command line. The simplest way to run them is from an SDK command window. In Windows Vista, Windows Server 2008, and later versions of Windows, most of the tools require elevated privileges. You should therefore use an account that is a member of the local Administrator group on the signing computer and open the command window with elevated privileges.

To open an SDK command window with elevated privileges

1. Click Start, point to All Programs, and then click **Microsoft Windows SDK.**

2. Right-click CMD Shell and select Run as administrator.

Unless otherwise specified, this paper assumes that you run all code-signing tools in an SDK window that runs with elevated privileges.

## How to Prepare the Signing Computer

To prepare a computer for test signing or release signing, you must install the code-signing tools and the Windows SDK.

The examples in the walkthrough assume the following:

* The SDK is installed and uses a command window that has elevated privileges.
* The PATH environment variable for the SDK command window includes the directory that contains the code-signing tools.

To prepare the signing computer

1. Install the SDK on your signing computer.

2. Set the PATH environment variable for the SDK command window to the directory that contains the code-signing tools.

3. Build the DLL.

When you link the DLL, use the **/integritycheck** flag. This flag is supported by both Visual Studio 2005 and Visual Studio 2008, including the free Express Editions. The **/integritycheck** flag causes the Windows memory manager to enforce a signature check at load time on the binary file.

For more information about this flag, see “Code Signature Overview” earlier in this paper.

4. If you use a separate build computer, copy the DLL to the signing computer.

## How to Test-Sign a DLL Binary File

Test signing refers to using a test certificate to sign a prerelease version of software for use on test computers. Windows Vista and later versions of Windows support test signing DLL binary files by using either a self-signed certificate that the MakeCert utility program generates or a test certificate that was issued by any CA, including an enterprise CA. This capability lets you test a signed DLL binary file on Windows Vista or a later version of Windows before the DLL is ready for release. Such testing is important to ensure that installation procedures work as intended.

Windows Vista and later versions of Windows accept test certificates that are generated by MakeCert or test certificates that are issued by any CA, including enterprise CAs.

**Important:**  Windows 7 supports test-signed DLL binary files only for testing. You must not use them as production software or release them to customers for use with Windows 7 RC1 or Windows 7 release to manufacturing (RTM).

To test-sign DLL binary files for Windows 7 and Windows Server 2008 R2 (basic procedure)

1. Create a test certificate by using MakeCert.

2. Install the test certificate in the Trusted Root Certification Authorities certificate store on the signing computer.

3. Test-sign the DLL binary file.

4. Verify the test signature.

The remainder of this section provides an explanation of each step.

### Step 1: Create a Test Certificate by Using MakeCert

Test signing requires a test certificate. After you generate a test certificate, you can use it multiple times to test-sign DLL binary files.

To use MakeCert to create a test certificate

1. On the signing computer, open an SDK command window with elevated privileges and navigate to the folder in which you want to place the certificate. If the build and signing computers are the same, use the application’s project folder.

2. Run MakeCert as in the following example to create a self-signed test certificate:

makecert –r -pe -ss PrivateCertStore -n CN=Contoso.com(Test) ContosoTest.cer

#### MakeCert Arguments

**-r**

Creates a self-signed certificate that uses the same name for the issuer and the subject.

**-pe**

Makes the private key exportable to the signing computer.

**-ss** StoreName

Stores the test certificate in a private certificate store. This example uses a store that is named PrivateCertStore. However, you can use any name that is convenient, except for the names of the standard certificate stores such as Personal or Trusted Publishers. If the specified certificate store does not already exist, MakeCert creates it.

**-n** SubjectName

Specifies the certificate's subject name in x500 format. The simplest method is to use the “CN=*MyName*” format. For this example, the subject name is Contoso.com(Test), which is a standard fictitious name that Microsoft uses for examples. You should use your preferred subject name.

ContosoTest.cer

Specifies the name of the file in which to place the certificate. This example places the certificate in a file that is named ContosoTest.cer, which is also a Microsoft standard fictitious name. You should use your preferred certificate file name.

The MakeCert command in the example does the following:

* Creates a self-signed test certificate that uses the same name for the subject name and the issuing authority.
* Places a copy of the certificate in an output file that is named ContosoTest.cer.
* Places a copy of the certificate in a certificate store that is named PrivateCertStore. Putting the test certificate in PrivateCertStore keeps it separate from other certificates on the system.

After you create the certificate and put a copy in the certificate store, you can use the MMC Certificates snap-in to view it.

To use the MMC Certificates snap-in to view a certificate

1. Click the Start button, and then click Start Search.

2. To start the Certificates snap-in, type certmgr.msc and press Enter.

3. In the snap-in's left pane, expand the PrivateCertStore certificate store folder and click Certificates.

Figure 1 shows the Certificates snap-in view of the PrivateCertStore certificate store.

|  |
| --- |
|  |

Figure 1. Viewing the certificate store by using the MMC Certificate snap-in

To view the certificate details, double-click the certificate in the right pane. Figure 2 shows an example of the details of a certificate.

|  |
| --- |
|  |

Figure 2. Viewing the test certificate

Notice that the Certificate dialog box states: “This CA Root certificate is not trusted. To enable trust, install this certificate in the Trusted Root Certification Authorities store.” This is the expected behavior. The certificate cannot be verified because Windows does not trust the issuing authority “Contoso.com(Test)” by default.

### Step 2: Install the Test Certificate in the Trusted Root Certification Authorities Certificate Store

To successfully install a test-signed DLL, the signing computer must be able to verify the digital signature. To do that, the signing computer must have the certificate for the CA that issued the package's test certificate installed in the computer’s Trusted Root Certification Authorities certificate store. Otherwise, SignTool cannot correctly verify the test-signature on the DLL.

You must add the CA certificate to the Trusted Root Certification Authorities certificate store only once. You can then use it for all test signature verification steps on that system.

In our example, the CA that issued the package's signing certificate, Contoso.com(test), is Contoso.com(test). To successfully verify a test-signed DLL, the digital certificate for Contoso.com(test) must be installed in the signing computer's Trusted Root Certification Authorities certificate store.

To use CertMgr to install a certificate in the Trusted Root Certification Authorities certificate store

1. On the signing computer, open an SDK command window with elevated privileges.

2. Navigate to the folder that contains the certificate.

3. Run the following Certmgr command line to install the Contoso.com(Test) certificate in the localMachine Trusted Root Certification Authorities certificate store:

certmgr.exe /add ContosoTest.cer /s /r localMachine root

#### CertMgr Arguments:

**/add** CertificateName

Specifies the name of the file that contains the certificate to add. For this example, the file that contains the certificate is ContosoTest.cer.

**/s**

Adds the certificate to a system store.

**/r** CertStoreLocation

Specifies the registry location of the Trusted Root Certification Authorities certificate store for the local computer, which controls who can access the certificate. In this example, the location is “localMachine root”, which means that the certificate is stored under HKEY\_LOCAL\_MACHINE and can be accessed by any user. By default, the certificate is installed in the localMachine Intermediate Certification Authorities certificate store. To restrict access to the current user, set **/r** to “currentUser”, which stores the certificate under HKEY\_CURRENT\_USER.

To view the Contoso.com(Test) certificate in the Trusted Certification Authorities certificate store, start the Certificates MMC snap-in, as described earlier in Step 1.

### Step 3: Test-Sign a DLL Binary File with an Embedded Signature

An embedded signature is a digital signature that is part of the DLL's binary file itself. You can use SignTool to create embedded signatures to both test-sign and release-sign binary files. This example uses SignTool to test-sign the sample DLL binary file, plugin.dll.

To test-sign a DLL binary file with an embedded signature

* Issue the following SignTool command to sign plugin.dll by using the test certificate that was created in Step 2:

Signtool sign /v /ph /s PrivateCertStore /n Contoso.com(Test) /t
 http://timestamp.verisign.com/scripts/timestamp.dll c:\plugin.dll

#### SignTool arguments:

**sign**

Signs the specified binary file.

**/v**

Displays successful execution and warning messages.

**/ph**

Sets page hashing. This option is available in Windows Vista and later versions of Windows.

**/s** CertStore

Specifies the certification store that contains the certificate. This example uses a certificate from the PrivateCertStore certificate store.

**/n** CertName

Specifies the certificate name. This example uses Contoso.com(Test).

**/t** URL

Adds a timestamp to the digital signature. The URL indicates the time stamp authority (TSA). This example uses the VeriSign TSA, whose URL is <http://timestamp.verisign.com/scripts/timestamp.dll>.

FileName

Specifies the name of the binary file to be signed. This example signs the file c:\plugin.dll.

A timestamp is not required, but we strongly recommend it. A timestamp lets Windows determine when a file was signed and ensures that the signature does not expire when the certificate expires.

### Step 4: Verify the Test Signature

Before you install a test-signed DLL binary file, you should use SignTool to verify the signature. However, test signatures are not trusted by default and SignTool does not verify a test signature that cannot be traced to a trusted CA. Therefore, you must direct the system to treat the test certificate as trusted so that you can verify the test signature on the binary file. To do so, use CertMgr to manually install the certificate for the CA that issued the test certificate in the signing computer’s Trusted Root Certification Authorities certificate store, as described previously in Step 2 of this walkthrough.

To verify the test signature on plugin.dll

* Use the following SignTool command line to verify the signature:

Signtool verify /pa /v c:\plugin.dll

#### SignTool Arguments:

**verify**

Directs SignTool to verify the signature of the specified binary file. For this example, the file is plugin.dll.

**/pa**

Directs SignTool to use the Authenticode verification policy when it verifies the signature.

**/v**

Directs SignTool to display successful execution and warning messages.

FileName

Specifies the file name. For this example, the binary file is plugin.dll.

## How to Install and Verify a Test-Signed DLL

This section describes how to install a test-signed DLL and verify its signature on a test system.

The following steps are required to install and verify a test-signed DLL.

1. Install the test certificates on the test system.

2. Enable the test-signing boot configuration option on the test system.

3. Enable Code Integrity event logging and system auditing.

4. Install the test-signed DLL on the test computer.

5. Verify that the test-signed DLL operates correctly.

### Step 1: Install the Test Certificates on the Test System

Before you can install and verify the test-signed AppInit DLL, you must install the following two certificates on the test system:

* The test certificate that you used to sign the application binary.
* The certificate of the CA that issued the test certificate.

If you are testing the DLL on the same system that you used for test signing, the certificates are already installed. Skip this step and go to Step 2.

In this example, both the test certificate and the CA certificate are the same. You created the test certificate, Contoso.com(Test), earlier in “Step 1: Create a Test Certificate” and used it to sign the Plugin.dll binary file. The certificate is packaged in a file that is named ContosoTest.cer.

You can install the certificates by using CertMgr from the Windows SDK or by using Windows Explorer.

To use CertMgr to install the test certificates

1. Copy ContosoTest.cer to the test system from the system on which you generated it.

2. Open an SDK command window with elevated privileges.

3. If you have not already done so, install Contoso.com(Test)in the localMachine Trusted Root Certification Authorities certificate store by running the following command:

certmgr.exe /add ContosoTest.cer /s /r localMachine root

4. Install Contoso.com(Test)in the localMachine Trusted Publishers certificate store by running the following command:

certmgr.exe /add ContosoTest.cer /s /r localMachine trustedpublishers

#### CertMgr Arguments:

**/add** CertificateName

Adds the certificate in the specified file to the specified certificate store. For this example, the file that contains the certificate is ContosoTest.cer.

**/s**

Adds the certificate to a system store.

**/r** CertStoreLocation

Specifies the registry location of the certificate store for the local computer, which controls who can access the certificate.

To add the test certificate to the Trusted Root Certification Authorities certificate store, use the location “localMachine root”, which means that the certificate is stored under HKEY\_LOCAL\_MACHINE.

To add the test certificate to the Trusted Publishers certificate store, use the location “localMachine trustedpublishers”.

 To use Windows Explorer to install the test certificates

1. Copy ContosoTest.cer to the test system from the system on which you generated the test certificate.

2. Open Windows Explorer and browse to the directory that contains ContosoTest.cer.

3. Right-click **ContosoTest.cer**, and then click **Install Certificate**.

4. When the Certificate Import Wizard opens, click **Next**.

5. Select **Place All Certificates in the Following Store**, and then click **Browse**.

6. Select **Trusted Root Certification Authorities**, and then click **OK**.

7. Click **Next** and then click **Finish**. If Windows cannot verify the origin of the certificate, it displays a Security Warning dialog box. You must click **Yes** to confirm that you want to install the certificate.

8. Right-click **ContosoTest.cer**, and then click **Install Certificate**.

9. When the Certificate Import Wizard opens, click **Next**.

10. Select **Place All Certificates in the Following Store**, and then click **Browse**.

11. Select **Trusted Publishers,** and then click **OK**.

12. Click **Next**, and then click **Finish**.

You should verify that the certificates are correctly installed in the appropriate certificate stores on the test system by using the MMC Certificates snap-in, as discussed in “Step 1: Create a Test Certificate,” earlier in this paper.

### Step 2: Enable the Test-Signing Boot Configuration Option

By default, Windows cannot verify test-signed AppInit DLLs even if the CA that issued the test certificate is installed in the Trusted Root certification store. However, the Windows boot loader includes a test-signing boot configuration option that, when enabled, enforces the signing of AppInit DLLs, but allows the test certificates to chain to any root CA.

To manually enable the Windows boot loader's test-signing option, you must use BCDEdit on your test computer. BCDEdit is the boot configuration editor for Windows Vista and later versions of Windows.

To enable the Windows boot loader test-signing option

1. Open a command window with elevated privileges.

2. Use the following BCDEdit command to enable test signing:

bcdedit.exe /set TESTSIGNING ON

#### BCDEdit Arguments:

**/set**

Directs BCDEdit to set a boot entry option value.

**testsigning on**

Enables the Windows boot loader's testsigning option.

3. Reboot the test system.

When the BCDEdit option for test signing is enabled, Windows does the following:

* Displays a watermark with the text “Test Mode” in all four corners of every monitor, to remind you that test signing is enabled.
* Requires each DLL that is loaded by the AppInit\_DLLs infrastructure to have a digital signature.

For more information on BCDEdit options, see ”BCD Boot Options Reference.”

To verify that test signing is enabled, use BCDEdit to list the current boot options.

To verify that test signing is enabled

* Run the following BCDEdit command:

bcdedit

Figure 3 shows an example of using BCDEdit to enable and verify test signing.



Figure 3: Using BCDEdit to enable and verify test signing

### Step 3: Enable Code Integrity Event Logging and System Auditing

Code Integrity is the Windows component that implements digital signature verification. It generates system events that are related to digital signature verification and logs the information in the Code Integrity log. The Code Integrity operational log view shows only digital signature verification error events. The Code Integrity verbose log view also shows successful signature verification events.

The data in the logs can help you troubleshoot installation problems, so you should enable code integrity logging when you install test-signed or release-signed AppInit DLLs.

The following procedure enables Code Integrity verbose event logging, which lets you view all successful Windows signature verification events. It can be used for both test-signed and release-signed AppInit DLLs.

To enable Code Integrity verbose event logging

1. Start the system Event Viewer.

Click Start, right-click Computer, and then select Manage to start the Control Panel **Computer Management** application. Event Viewer is under the **System Tools** node in the left pane.

You can also run Eventvwr from a command window with elevated privileges.

2. Click the triangle to the left of Application and Services Logs to display the various logs, and then expand the nodes under Microsoft\Windows\CodeIntegrity.

3. Click the CodeIntegrity node.

4. If both an **Operational** and a **Verbose** node do not appear under **CodeIntegrity**, right-click CodeIntegrity, click View, and then make sure that Show Analytic and Debug Logs is checked. This adds the Verbose node**.**

5. Right-click the Verbose node and select Properties.

6. On the General tab of the Properties dialog box, check the Enable Logging option.

In addition, you can use the Auditpol tool to enable system event records, which include Code Integrity image verification failure events.

To enable the audit policy to generate system event records for failed operations

* From a command window with elevated privileges, enter the following Auditpol command:

Auditpol /set /Category: **"**System**"** /failure:enable

Figure 4 shows an example of enabling the system category audit policy.



Figure 4. Enabling security audit policy

### Step 4: Copy and Install the Test-Signed DLL to the Test Computer

After the system has restarted, copy the test-signed DLL to the test computer and install it.

### Step 5: Verify that the Test-Signed DLL Is Operating Correctly

If the DLL does not work correctly, check the Code Integrity log for signature verification errors.

## How to Release-Sign a DLL Binary File

Release signing identifies the publisher of a DLL binary file that runs on Windows 7 or Windows Server 2008. To release-sign a binary file, you use an Software Publisher Certificate (SPC) and a related cross-certificate. Release signing requires the same code-signing tools as test signing, and the signing computer must be prepared as described in “How to Prepare the Signing Computer” earlier in this paper.

This section provides a walkthrough of the release-signing process, with a focus on the use of the tools. The actual process and infrastructure for release-signing varies from company to company. For more information about release-signing, see “Code Signing Best Practices.”

In summary, the procedure to release-sign an application binary file is as follows:

1. Obtain an SPC for signing application binaries and install it on the signing computer.

2. Obtain and install the related cross-certificate.

3. Release-sign the DLL binary file by using an embedded signature.

The rest of this section explains each step.

### Step 1: Obtain and Install an SPC

Release signing requires an SPC from a commercial CA. Follow the CA's instructions to obtain an SPC and install the private key on the signing computer. For a list of cross-certificate SPC CAs, see “Microsoft Cross-Certificates for Windows Vista Kernel Mode Code Signing.”

The examples in the release-signing section use a fictitious certificate from one of the SPC CAs that was issued to Contoso.com. This certificate is installed in the Personal certificate store.

Important: You should protect your release-signing certificate files with strong passwords. For more information, see “Strong passwords: How to create and use them” on the Microsoft Web site.

#### Convert the SPC to the .pfx Format

Release-signing certificates must be in the .pfx format. However, some certification authorities store the digital certificate's private key in a .pvk file and store the public key in an .spc or .cer file. For example, VeriSign Class‑3 certificates are currently packaged as a.pvk and a .spc files. If you have such an SPC, you must convert the .pvk and .spc files to the .pfx format for secure portability of the certificate and private key.

Use the Pvk2Pfx tool to convert .pvk and .spc files to the .pfx format.

To use Pvk2Pfx to convert .pvk and .spc files to the .pfx format

* The following Pvk2Pfx command line converts a .pvk file and an .spc file to a .pfx file:

Pvk2pfx -pvk private.pvk -pi pvkPassword1 -spc public.spc -pfx

 pfxcert.pfx -po pfxPassword1 -f

#### Pvk2pfx Arguments:

-pvk PvkFileName

Specifies the input .pvk file. For this example, the file is private.pvk.

-pi PassWord

Specifies the .pvk file's password. For this example, the password is pvkPassword1.

-spc SpcFileName

Specifies the input SPC file. For this example, the name is public.spc.

**-pfx**

Specifies the output .pfx file. For this example, the name is pfxcert.pfx.

**-po** OptionalPassword

Specifies the .pfx file's password. For this example, the password is pfxPassword1. If you do not specify a .pfx password, Pvk2Pfx assigns the .pvk file's password—as specified by the -pi argument—to the .pfx file.

**-f**

Overwrites an existing .pfx file.

#### Import the SPC into the Personal Certificate Store

Versions of SignTool that were released before Windows 7 do not support the use of .pfx files. To release-sign an AppInit DLL on such a system, you must import the certificates and key that are stored in the .pfx file into the signing computer's local Personal certificate store. This requirement is caused by a conflict in adding cross-certificates to the signature when you use a certificate from a .pfx file.

To import a .pfx file into the Personal certificate store, you can use either the Certificate Import Wizard or the CertUtil command-line tool.

To use the Certificate Import Wizard to import a .pfx file into the Personal certificate store

1. Start Windows Explorer and navigate to the folder that contains the .pfx file.

2. Double-click the .pfx file to open the Certificate Import Wizard.

3. Follow the procedure in the Certificate Import Wizard to import the certificate into the Personal certificate store.

To use CertUtil to import a .pfx file into the Personal certificate store

1. Open a command window with elevated privileges and navigate to the folder that contains the .pfx file.

2. Use the following command to import the .pfx file:

certutil –user –p pfxPassword1 –importPFX PfxFile

#### Certutil arguments:

**-user**

Places the certificate in the “Current User” Personal store.

**-p** password

Specifies the file's password. For this example, the password is pfxPassword1.

**-importPFX** Pfxfile

Specifies the .pfx file to import. For this example, the file is *PfxFile*.

**Note:** In Windows 7, you can sign a binary directly from a .pfx file instead of importing the certificates into the system certificate stores. The Windows 7 version of SignTool supports the simultaneous use of the **/ac** and **/f** switches with the **sign** command. By using these switches together, you can specify both a cross-certificate and a signing certificate on a single command line without importing the signing certificate into the certificate store. Therefore, you no longer are required to add and later delete the certificates and private keys from the system certificate and key stores. To determine the correct cross-certificate to use, run the CertUtil tool.

#### View the SPC's Properties

After the SPC is installed in the Personal certificate store, you can use the MMC Certificates snap-in to view the certificate's properties.

To use the MMC Certificates snap-in

1. Click Start, and then click Start Search.

2. To start the Certificates snap-in, type Certmgr.msc and press Enter.

3. Select the Personal certificate store.

4. Click the **Certificates** node and double-click the name of the SPC to open its **Certificate** dialog box.

5. On the Details tab of the Certificate dialog box:

* Select Subject from the list of fields to highlight the certificate's subject name. This is the subject name that you use with the procedures later in this paper.
* Find the **Issuer Name** and **Thumbprint** for the CA that issued the SPC and record those values for later use in determining which cross-certificate is required.

**Tip:**  After you finish the signing process, remove the certificate and private key from the system. You can use the MMC Certificates snap-in or the CertMgr command-line tool for this task.

To delete a certificate

1. Open the MMC Certificates snap-in and browse to the certificate that you want to delete.

2. Right-click the certificate, point to **All Tasks**, and then click **Export**.

3. On the wizard **Welcome** page, click **Next**.

4. Select **Delete the private key if export is successful**, and then click **Next**.

5. Complete the rest of the wizard.

6. Delete the certificate that you just exported.

### Step 2: Obtain and Install a Cross-Certificate

Release signing requires a cross-certificate in addition to an SPC. Each CA that provides SPCs has one or more associated cross-certificates, which are available from Microsoft.

To obtain and install a cross-certificate

1. Go to the “Microsoft Cross-Certificates for Windows Vista Kernel Mode Code Signing” Web page.

2. Use the **Issuer Name** and **Thumbprint** values that you previously recorded in “View the SPC’s Properties” to locate the appropriate cross-certificate for your SPC and download it to the signing computer. The certificate is in a .cer file that is packaged in a .zip file.

3. Extract the .cer file and copy it to your project folder.

### Step 3: Release-Sign a DLL File with an Embedded Signature

Like test signing, release signing uses SignTool to embed a digital signature in the application’s binary file itself. However, release signing requires both an SPC and a cross-certificate.

To use SignTool to embedded-sign a program file with an SPC and cross-certificate

* Run the following SignTool command line to sign Plugin.dll and timestamp the digital signature:

Signtool sign /v /ph /ac MSCV-VSClass3.cer /s my /n contoso.com /t http://timestamp.verisign.com/scripts/timestamp.dll c:\plugin.dll

#### SignTool Arguments:

**sign**

Signs the specified binary file.

**/v**

Displays successful execution and warning messages.

**/ph**

Sets page hashing. This option is valid only on Windows Vista and later versions of Windows.

**/ac** CrossCertName

Specifies the name of the cross-certificate. For this example, the cross-certificate name is MSCV-VSClass3.cer. Use a full path name if the cross-certificate is not in the current directory.

**/s** CertificationStoreName

Specifies the certificate store that contains the SPC. In this case, the argument specifies the My certificate store, which indicates the Personal certificate store.

**/n** CertName

Specifies the SPC's subject name. This example uses an SPC that has the subject name Contoso.com. To obtain the SPC’s subject name, see “View the SPC’s Properties” in “Step 1: Obtain and Install an SPC” earlier in this paper.

**/t** URL

Adds a timestamp to the digital signature. The URL indicates the TSA. This example uses the VeriSign TSA, whose URL is <http://timestamp.verisign.com/scripts/timestamp.dll>.

FileName

Specifies the name of the binary file to be signed. This example signs the file c:\plugin.dll.

To prevent the signature from expiring when the certificate expires, we recommend the timestamp. You can add a timestamp later without affecting the validity of the signature by using the SignTool **timestamp** command.

After the file is embedded signed, use SignTool to verify the signature. Check the results to verify that the root of the SPC certificate chain for kernel policy is “Microsoft Code Verification Root”.

To use SignTool to verify an embedded release signature

* Run the following SignTool command line to verify the embedded release signature in Plugin.dll:

signtool verify /pa /v plugin.dll

#### SignTool Arguments:

**verify**

Verifies the signature in the specified file.

**/pa**

Directs SignTool to use the Authenticode verification policy when it verifies the signature.

**/v**

Displays successful execution and warning messages.

FileName

Specifies the name of the binary file that contains the signature to be verified. For this example, the file is c:\plugin.dll.

Figure 5 shows an example of verifying a binary file's embedded signature by using the default Authenticode verification policy.



Figure 5. Signature verification example

**Note:** Versions of SignTool that were released before Windows 7 report success even if the cross-certificate is not present in the chain. Therefore, it is important to verify the presence of the cross-certificate in the certificate chain. However, an AppInit DLL does not load if the cross-certificate is not present.

Beginning with the Windows 7 version of the SignTool utility, SignTool no longer reports success if the cross-certificate does not appear in a certificate chain.

## How to Install and Verify a Release-Signed DLL

Installation and verification of a release-signed AppInit DLL involve the following steps:

1. Disable the test-signing boot configuration option.

2. Enable code integrity event logging and system auditing.

3. Reboot the test computer.

4. Install the release-signed AppInit DLL on the test computer.

5. Verify that the AppInit DLL operates correctly.

### Step 1: Disable the Test-Signing Boot Configuration Option

If you previously enabled the test-signing boot configuration option to install a test-signed AppInit DLL, disable the option.

To disable the test-signing boot configuration option

* Open a command window with elevated privileges and run the following BCDEdit command to disable test signing:

bcdedit.exe /set testsigning off

By default, the test-signing boot configuration option is not defined in Windows Vista and later versions of Windows. This is equivalent to setting the test-signing boot configuration option to OFF.

### Step 2: Enable Code Integrity Event Logging and System Auditing

If you have not already enabled code integrity event logging and system auditing, follow the procedure in Step 3 in the section titled “How to Install and Verify a Test-Signed DLL.”

### Step 3: Restart the Test Computer

Restart the test computer so that the changes to the boot configuration options take effect.

### Step 4: Copy and Install the Release-Signed AppInit DLL to the Test Computer

After the system has restarted, copy the release-signed AppInit DLL to the test computer and install it.

### Step 5: Verify that the Release-Signed AppInit DLL Is Operating Correctly

If the AppInit DLL does not work correctly, check the Code Integrity log for signature verification errors.

# Feedback

Comments and questions should be sent to [appinittq@microsoft.com.](appinittq%40microsoft.com.)

# Resources

#### WHDC

Boot Configuration Data in Windows Vista

<http://www.microsoft.com/whdc/system/platform/firmware/bcd.mspx>

Code-Signing Best Practices

<http://www.microsoft.com/whdc/winlogo/drvsign/best_practices.mspx>

Microsoft Cross-Certificates for Windows Vista Kernel Mode Code Signing

<http://www.microsoft.com/whdc/winlogo/drvsign/crosscert.mspx>

How to Get the WDK and the WLK

<http://www.microsoft.com/whdc/DevTools/WDK/WDKpkg.mspx>

Protected Processes in Windows Vista

<http://www.microsoft.com/whdc/system/vista/process_vista.mspx>

#### MSDN

AppInit\_DLLs should be renamed Deadlock\_Or\_Crash\_Randomly\_DLLs

<http://blogs.msdn.com/oldnewthing/archive/2007/12/13/6648400.aspx>

Windows SDK

<http://msdn.microsoft.com/en-us/windows/bb980924.aspx>

#### Other Microsoft sites

Deploying Authenticode with Cryptographic Hardware for Secure Software Publishing

<http://technet.microsoft.com/en-us/library/cc700803.aspx>

Strong passwords: How to create and use them

<http://www.microsoft.com/protect/yourself/password/create.mspx>

Working with the AppInit\_DLLs registry value

<http://support.microsoft.com/kb/197571>