GEN11518

Introducing Secure Development Methods and Concepts for AutoCAD Plug-ins

George Varghese
Autodesk, Inc

Davis Augustine
Autodesk, Inc

Tekno Tandean
Autodesk, Inc

Learning Objectives

- Learn how to use AutoCAD security settings such as Secureload, Trusted Path, Search Path, Safemode, and Plugin loading mechanisms.
- Learn how to use secure coding practices such as compiler switches, insecure functions, API changes and the Fortify tool.
- Learn how to use the CAD manager utility to lock down the application, and use the best security practices for CAD managers.
- Learn the best security practices for developing AutoCAD plug-ins.
- Learn about code signing, and how to sign executable such as ARX/CRX/DLL, and LISP files.

Description

AutoCAD software has been the target of hacker attacks for industrial espionage. Many of these hackers have been primarily driven to steal intellectual property. There are a number of steps that the AutoCAD Security Team has implemented to enhance the security in AutoCAD 2016. These features help to avoid AutoCAD from being a threat vector. They include locking down the folders from which plugins are loaded, loading plugins from trusted publishers and developing plugins that are secure.

Your AU Experts

George Varghese has been with Autodesk, Inc., for over 15 years, working in the AutoCAD Group, and primarily in the AutoCAD Software Development Group. He worked on AutoCAD software features such as the Tool Palette, Table, Ribbon, and others. Lately, he has been working on the security features in AutoCAD, dealing with digital signatures. He worked on the feature to be able to digitally sign LISP files. ~~LISP~~ files are the primary threat vectors used by attackers to target AutoCAD software users. He worked on the "Trusted Publisher" feature that helps users discern the plug-ins that come from a trusted source. He has given presentations on threat modeling to a number of internal Autodesk groups. He has given presentations on cryptographic algorithms and x.509 certificates to an Autodesk audience.

Davis Augustine has been a bit twiddler in the AutoCAD Software Development Group for over 15 years, working on various parts of the product, mostly in the core and acdb areas. He has been involved in security for the past couple of years, including the areas of secure APIs, path searching, static analysis, compiler and linker switches, and buffer overrun prevention.

Tekno Tandean has been a code monkey for the last 18 years for the AutoCAD Software Development Group, working on C++, COM, and .NET APIs. He was pulled into an AutoCAD security related incident few years back, and has been contributing to make the AutoCAD ecosystem safer by hardening AutoCAD code, and raising awareness of secure development.

AutoCAD security settings

The SecureLoad System Variable

This AutoCAD system variable (sysvar) can have values 0 through 2, with the default being 1. When set to 0, AutoCAD does not check for modules being signed at load time, nor does it check for them being on the TrustedPath list. Mode 0 is not recommended. Mode 1 (the default) causes a prompt on attempts to load executables from outside the TrustedPaths, allowing the user to approve of the loading of the executable. With Mode 2, there is no prompt and the load fails. CAD Managers may want to lock down this sysvar, to prevent users from running unsafe executables.

Special Runtime Folders

The AutoCAD process has a concept of three logical folders (directories) at runtime:

- Current (Working) Folder: maintained by the operating system, can be changed at runtime, such as by the File Open dialog. See the **WorkingFolder** sysvar.
- Start-In Folder: the original Current Folder in effect when AutoCAD started. See the **StartInFolder** sysvar.
- Document Folder: the location of the drawing file currently being edited. See the **DwgPrefix** sysvar.

Executable Files

AutoCAD treats executable files differently from data files. In general, executables are not loaded from the three special runtime folders (Current, StartIn, Document). The two main types of executable files in the AutoCAD are LISP (.LSP, .FAS or .VLX) and DLLs (.ARX, .CRX, .DBX, .HDI or .NET assemblies). Other file types considered executable are .EXE, .MNL, .SCR, .DVB, and acad.rx. In a future release, .DCL files will also be treated as executables.

When searching for executable files to load, AutoCAD 2016 and above will generally not search in the three special runtime folders mention above, because that can lead to “binary planting” attacks. This behavior can be overridden, however, with the **LegacyCodeSearch** sysvar introduced in AutoCAD 2016.

The TrustedPaths System Variable

This AutoCAD sysvar defines a “white list” of folders from which it is safe to load executables. Note that this is different from the support path. The support path specifies which folders to look in when searching for various files to load, including menus, fonts, executables, etc. The TrustedPath sysvar does not specify which folders to look in, it just specifies which folders are safe from which to load executables, once AutoCAD has found an executable in that folder.

An exception to this behavior in AutoCAD 2016 and earlier involves Heidi driver (HDI) files. For these, the TrustedPaths list is searched.

Safemode

This mode is turned on with a command line switch, and is a readonly sysvar. When set to 1, AutoCAD does not load executable files. Use this to help fix AutoCAD’s installation when it has been infected by malware. But how do you clean out the malware from within safe-mode AutoCAD?

Security Best Practices for Users and CAD managers

Operating system related best practices

A more secure environment can be created by following the principles of layered security and defense in depth[1]. The operating system provides infrastructure to support a secure environment, which should be employed.

Install in Program Files

Windows has been locking down the OS since Windows Vista, implementing a number of security features, protecting the Operating System resources. One such enhancement is protecting folders such as *Program Files*. It is recommended that executable files are installed here. The files in these folders are read-only by nature; any change to the files will require administrative privileges. This practice follows the principle of Least Privilege described below, and will prevent executable files from being tampered with.

Running without admin access

This is part of the Principle of Least Privilege[2]. A non-admin user account should be used when possible to reduce the risk of the computer being compromised by malware. The damage will be limited due to the limited privileges. The malware will need to be more sophisticated to do damage in this mode, such as employing privilege escalation[3].

Common attacks like ransom ware can be prevented from spreading across the company network if each user has just enough privilege to do their work.

Turning on UAC

Turning on User Account Control[4] is the safest practice for administrators. Code running without administrator privilege is safer for the system.

The UAC slider setting is derived from Group Policy in the enterprise environment. The network administrator can configure the UAC based on the workflow, should the UAC hinder the work.

There are certain types of malware that can be prevented when the UAC is raised to its maximum settings. It is recommended that you work in this mode.

See the Windows security model[5] for additional information.

Locking security sysvars or settings using CAD manager tools

The AutoCAD security settings are stored in HKCU, which can be modified by a standard user. The best practice would be to lock down these settings to restrict them only to modification by an Administrator. The CAD Manager Control Utility (CMCU) application provides the necessary functionality.

---

[1] Layered Security and Defense in Depth
[2] Principle of Least Privilege
[3] Privilege Escalation
[4] UAC
[5] Windows security model

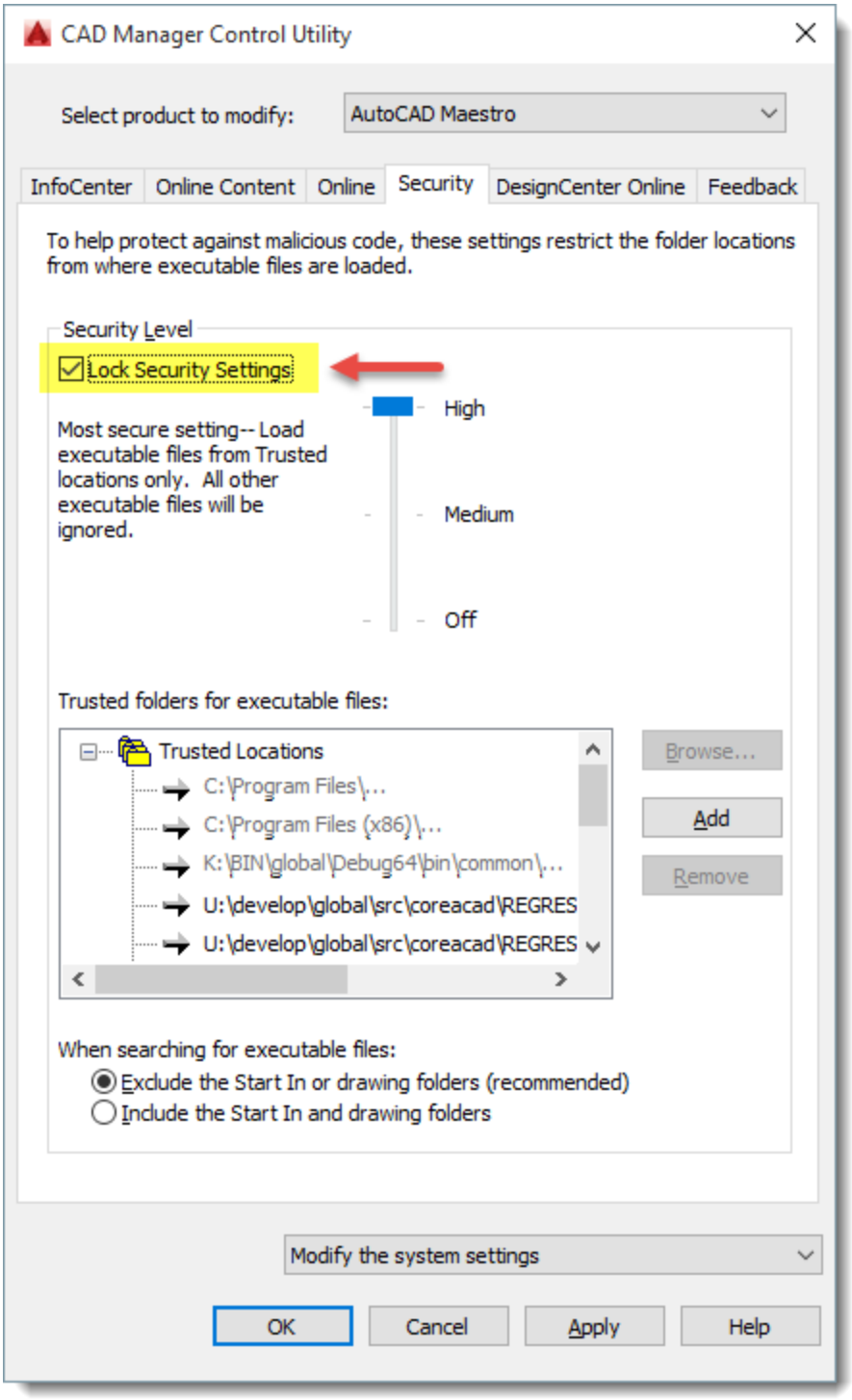

FIGURE 1: CMCU SECURITY TAB TO LOCK SECURITY SETTING

Tip: locking other sysvars

Other sysvars can also be locked like the security settings, although this needs manual intervention in the Registry. A sysvar can be locked by adding a new 'Owner' key with the value '*CADAdmin'. For example, the POLARANG sysvar can be locked as shown in the figure.

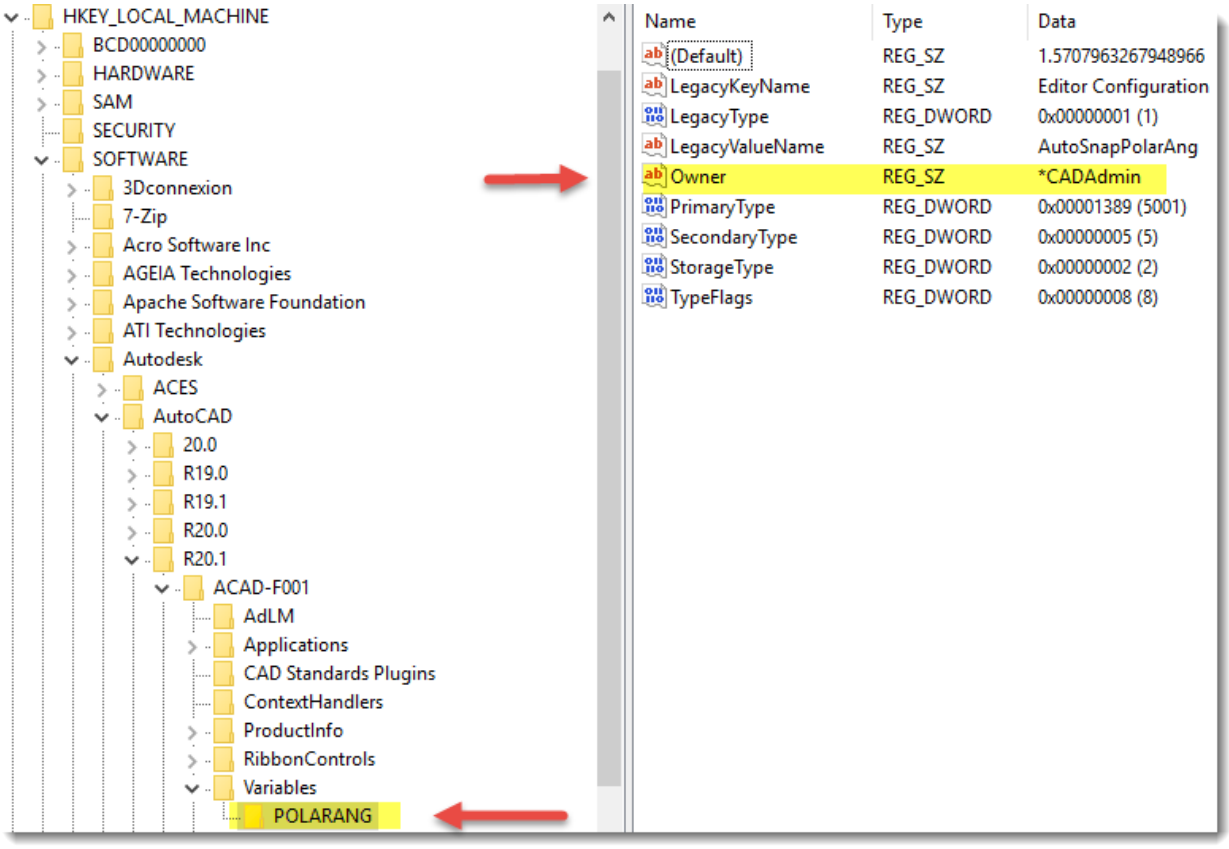

FIGURE 2: LOCKED POLARANG SYSVAR

Note that this method is not officially supported as any UI needs to honor the locked setting.

Isolate location of custom executable files

LISP files are executable files and have to be treated as such. The best practice is to put executables in pre-defined locations. The same practice can be applied to LISP files, putting them in pre-defined locations. It is much easier to audit and manage them this way.

The enterprise-wide LISP files should be shared only using read-only folders. The best practice is to sign LISP files, verify the signature, or do regular checksum comparisons to ensure that the files have not been tampered with.

Install any custom executable such as .NET DLL under the Program Files Plugin folder.

Regular scan of AutoCAD LISP files

Customer reports indicate that the most prevalent malware for AutoCAD is LISP based. We have done a lot of work in this area such as preventing the auto-loading of LISP files, which was used as a form of binary planting.

Audit all AutoCAD LISP files, especially those in non-admin protected folders. It is a good practice to ensure that there are no LISP files that have been tampered with or no newly introduced malware LISP files.

We have created a tool that can help with this process. The tool can generate baseline checksum files for AutoCAD-related executables for subsequent comparison.

The tool has checksums of unsigned executable files that we have shipped so far, and it can be used as base checksums.

Security Level

Level – High
The process allows user-created executables to be loaded from a trusted folder. The SecureLoad system variable is set to 2, which allows executable files to be loaded only if their location is in the trusted locations specified in the TRUSTEDPATHS system variable. The CAD manager can create a trusted path for the user. The user can create the executable and sign it with a self-signed certificate. The signed executable is then copied to the trusted path. The CAD admin can run an audit tool to audit the executable in the trusted folder. Any executable that is not signed or has been tampered with can flag the errors and report them.

Level – Medium
The process allows signed and unsigned executables to be loaded from outside a trusted folder. The SecureLoad system variable is set to 1, which loads executable files only if their location is in the trusted locations specified in the TRUSTEDPATHS system variable, and displays a warning during load requests from executable files that are not in trusted locations. Users can load signed and unsigned executables from untrusted locations. Users will need to discern the executable that they trust. The signed executable will display the publisher information which can be used by the user. The unsigned executable will display the publisher as unknown. Users will need to be extra careful in loading these executables, and must ascertain that these executable are safe by doing some research, such as searching for them on the Web, to check whether they are in anyway associated with malware. User-created executables can be loaded in this mode, and the warning dialog box will be displayed when they are loaded from an untrusted location.

Level – Low
The process allows executable files to be loaded without warning. The SecureLoad system variable is set to 0, which loads executable files without warning. This option maintains legacy behavior, but is not recommended. Users can load signed and unsigned executables from untrusted locations, and no warning is displayed. This was added to allow users to slowly transition to a more secure load mechanism. This can be used only if the system is fully locked down.

Safemode
This mode can be used for a quarantine situation. In this mode

- LISP is disabled.
- The Netload command, which loads .NET assemblies, is disabled.
- ARX load is disabled.
- Appload to load an executable will cause an error.
- Startup suite app will not load.

Secure coding practices

Data Execution Prevention (DEP)
This is the msvc linker's /NxCompat switch. It is needed only when linking exe files (not dlls). It's turned on by default.

Address Space Layout Randomization (ASLR)
This is the msvc linker's /DynamicBase switch. It is used on both dll's and exe's, and is enabled by default.

Buffer Overflow Detection
This is the msvc compiler's /GS switch, and is enabled by default.

SDL Switch
The msvc compiler's /SDL switch turns on extra runtime checking and does additional runtime safety things such as clearing pointers after deleting them.

Control Flow Guard
This is a newer technology added to msvc which keeps track of valid function addresses at runtime and checks against that list when jumping through function pointers. It has some performance impact and has not yet been enabled in AutoCAD.

Digital Signature

Code Signing

Code signing is the process of digitally signing the executable. Here executable refers to portable executable files (.exe, dll, etc.) and scripts such as LISP files. The digital signature is like an electronic security mark affixed to the executable. The digital signature has the details about the publisher, the independent entity who can guarantee the publisher's identity. It also has the cryptographic checksum which is used to verify that the content has not been tampered after signing. Among the best practices is to digitally sign the executable that is shipped for distribution. A valid signature however does not mean that the files are harmless. But the advantage is that the user can decide if they trust the publisher before executing it.

x.509 Certificate

The x.509 certificate has the following information.

- Version number of the certificate, the current version is 3.
- Certificate serial number, the unique serial number is specific to a Certificate Authority (CA).
- Signature, it includes the algorithm used for signing.
- Issuer name, the certificate authority name.
- Validity period of the certificate.
- Subject name, in this case the publisher name.
- Subject public key info, the public key of the publisher.

Then there are optional parameters such as subject unique id, the issuer unique id. Starting with version 3, additional information is stored in the certificate using extensions. Some of the extensions are mandatory while others are optional. A cryptographic checksum of the above data is created, and the hash that is obtained is encrypted using the private key of the CA. This is the signature that is embedded in the certificate.

A certificate can be legally signed by the private key corresponding to the public key contained within it. These certificates are called self-signed certificates. The CA signs the publisher certificate, but who signs the CA's certificate, it is self-signed. All the trusted root CA certificates are self-signed. Here is a snapshot of a self-signed certificate.

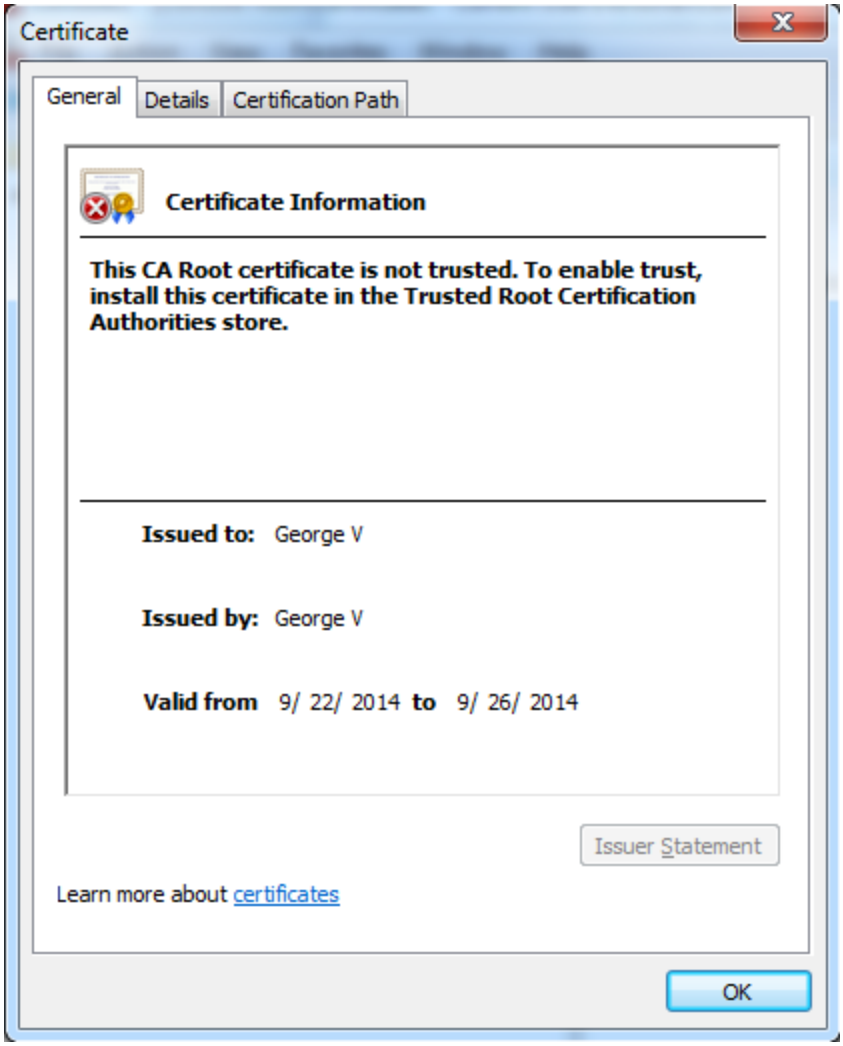

FIGURE 3: CERTIFICATE DIALOG

Getting a cert

A publisher who wishes to digitally sign an executable will need to acquire an x.509 certificate. This certificate is provided by a CA that guarantees the identity of the subject, the subject an x.509 lingo in this case is the publisher. A publisher requests a CA to provide a certificate. The CA subjects the publisher to various tests to prove the authenticity before issuing the certificate for a fee. This does not prevent rogue publishers from presenting themselves as legitimate businesses and obtain certificates. It is always prudent for the user to scrutinize any publisher before trusting them.

The trusted root CA certificate ships with the browser; it can be viewed in the certificate store. Here is a screen shot of the trusted root certificate. In IE, go to Tools > Internet Options > Content > Publishers, and click the Trusted Root Certification Authorities tab.

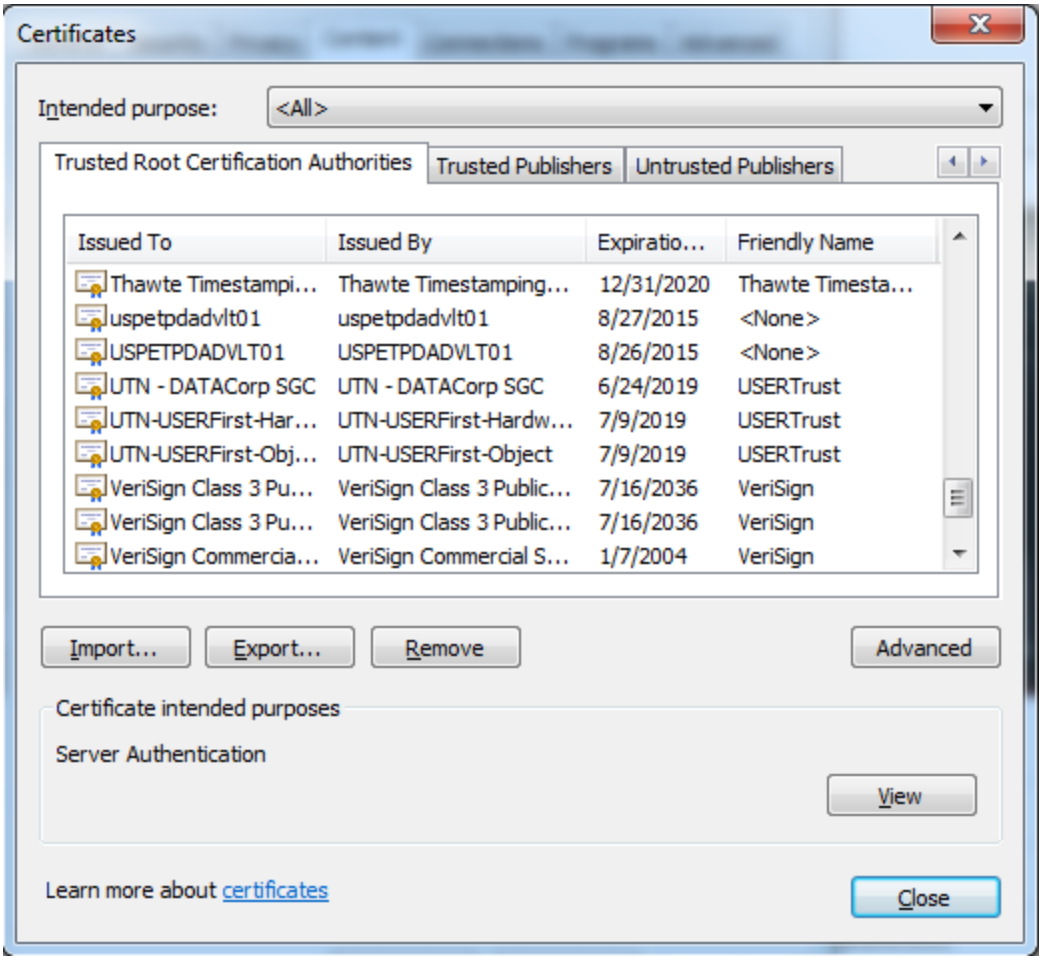

FIGURE 4: TRUSTED ROOT CERTIFICATES

Signing PE and LISP executables

Create your own certificate

You can create a self-signed certificate. This is useful for testing and also for deploying within an intranet. There are two tools from Microsoft that are available on Windows PCs for creating certificates. They can be downloaded as part of the Windows SDK from the Microsoft website, if it's not available on the system already.

Makecert.exe

This tool outputs the certificate as a .cer file.

Usage:

makecert -sv yourprivatekeyfile.pvk -n "cert name" yourcertfile.cer -b mm/dd/yyyy -e mm/dd/yyyy –r

Example:

makecert -sv myprivatekeyfile.pvk -n "CN=George V" mycertfile.cer -b 09/22/2014 -e 09/30/2014 –r

A dialog box will appear to set the password for the private key file.

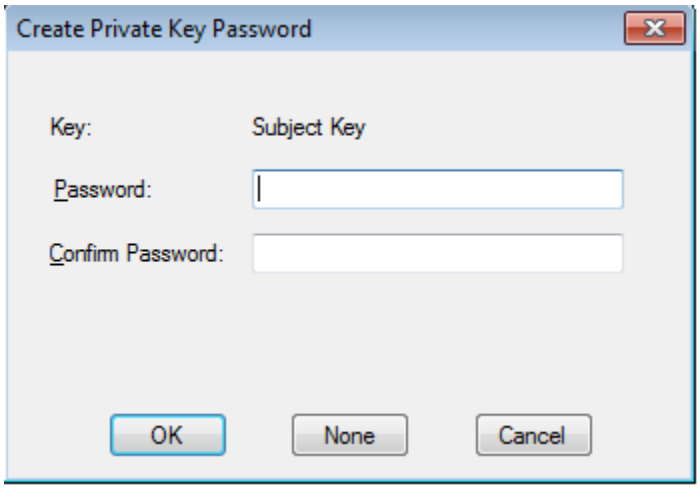

FIGURE 5: PASSWORD DIALOG

Another dialog box will be displayed to enter the password to sign the certificate. Please note that this is a self-signed certificate and so you will be signing the public key with the private key.

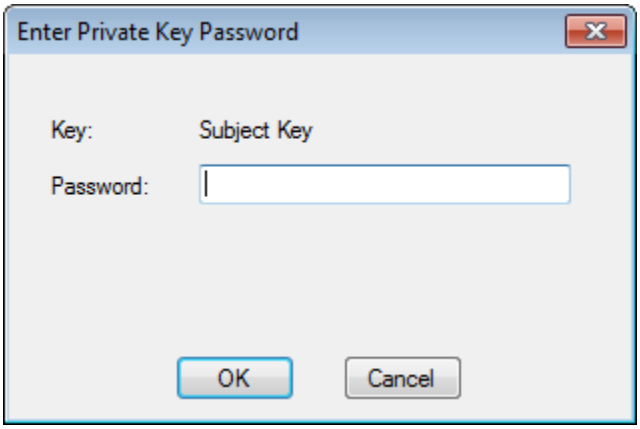

FIGURE 6: PASSWORD DIALOG

At the end of it all there will be a .cer file containing the public key and a password protected .pvk file containing the private key.

If a comma needs to be added in the common name then you will need to enclose it in double quotes as shown below.

MakeCert -sv myprivatekeyfile.pvk -n "CN=\"George, Inc\", O=\"George, Inc\"" mycertfile.cer -b 12/18/2014 -e 12/30/2014 –r

Pvk2pfx.exe

This tool is used to create a pfx file that can be used for signing. The pfx file has the certificate, the public and the private key of the publisher.

Usage:

pvk2pfx –pvk yourprivatekeyfile.pvk -pi password –spc yourcertfile.cer –pfx yourpfxfile.pfx –po yourpfxpassword

Example:

pvk2pfx -pvk myprivatekeyfile.pvk -pi MyPassword -spc mycertfile.cer -pfx mypfxfile.pfx -po mypassword

At the end of this operation you will have a .pfx file that can be used for signing.

Signing PE and LISP executables

Digitally signing executable files

The certificate created above can be used to digitally sign the portable executable file. On a Windows PC, SignTool.exe is available from Microsoft that enables signing executable files. It can be used to sign both the managed (.NET) and native files. The sign tool takes the pfx file with the public and the private key as the input.

Usage:

signtool sign /f <PFX file name> /p <Password> FileToSign.exe

Example:

signtool sign /f mypfxfile.pfx /p mypassword MyTest.exe

Install certificate in the personal store

The certificate has to be installed in the personal certificate store for it to be available for signing LISP files. The Microsoft utility, *certmgr.exe*, can be used for installing the certificate in the personal store. It can be used to install the certificate in the UI or in the command line mode.

Run *certmgr.exe* which will display the UI, here are the steps.

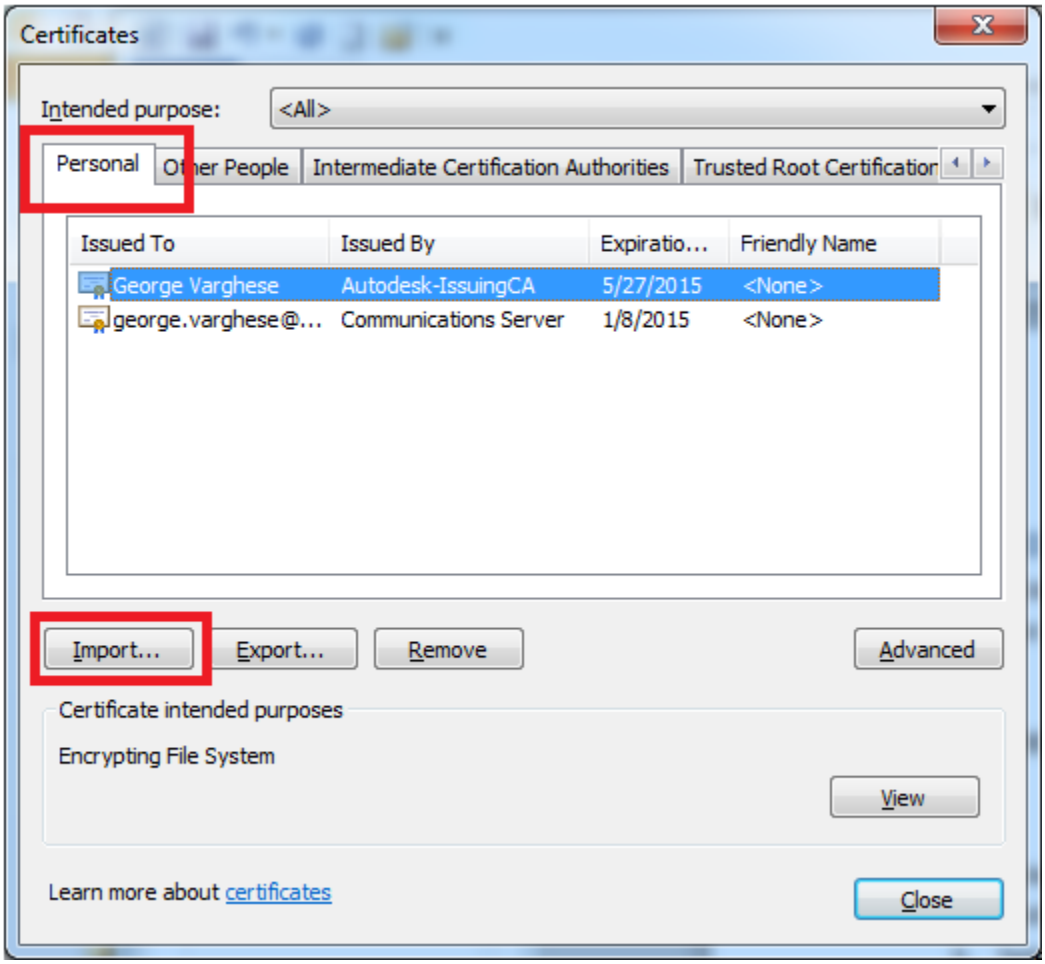

FIGURE 7: IMPORT CERTIFICATE DIALOG

1. Select the Personal tab
2. Click the Import button
3. Select the PFX file that was created

This will add the certificate to the personal certificate store.

Digitally signing LISP files

LISP files can be digitally signed using two of the in-house tools. The certificate has to be installed in the personal certificate store for it to be available for signing.

AcSignApply.exe

This is a UI tool originally developed for signing drawing files, and was extended to sign LISP files. This tool is shipped with AutoCAD. The following UI is displayed when we run AcSignApply.

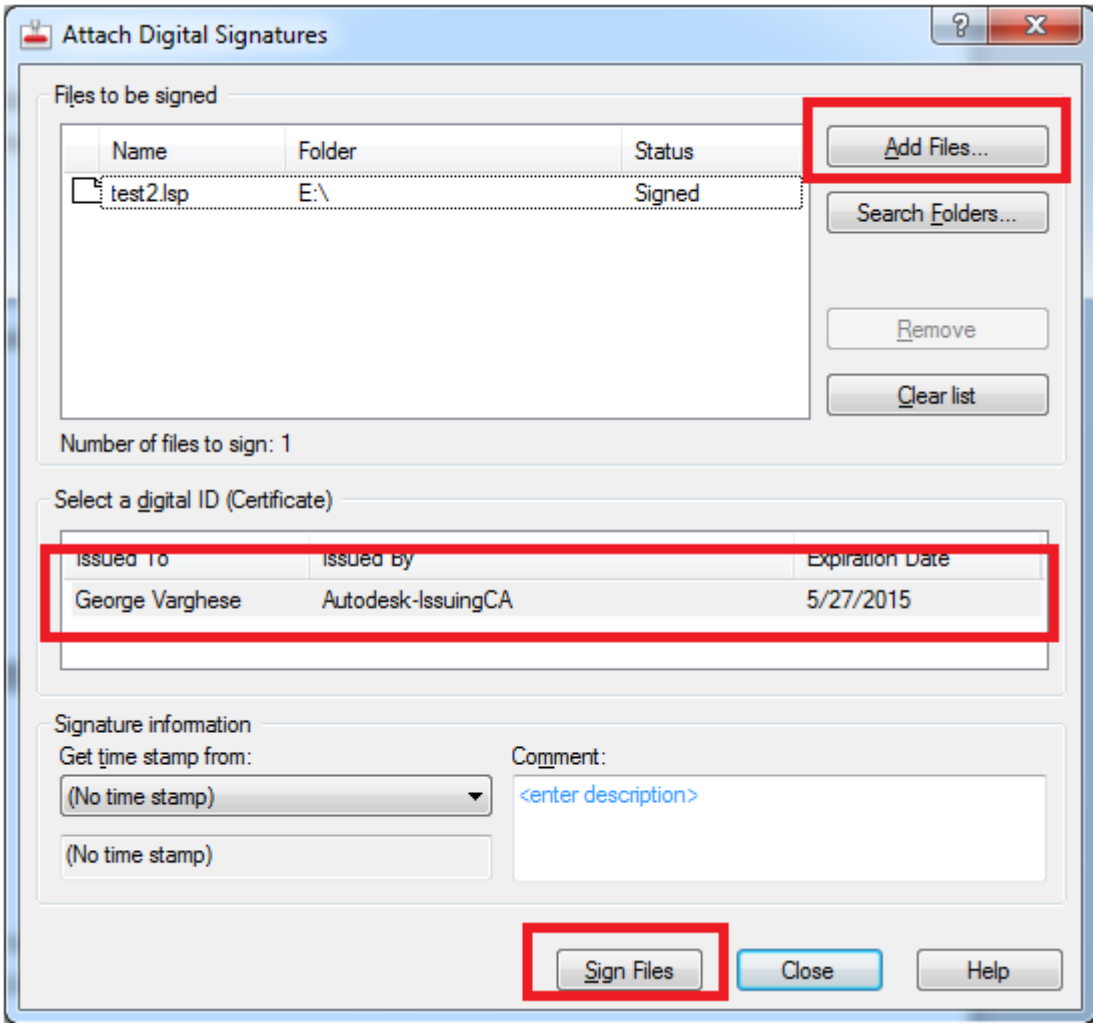

FIGURE 8: DIGITAL SIGNATURE DIALOG

1. Click the Add Files button to add the LISP files to be signed.
2. Select the LISP file to be signed in the list control.
3. Select the certificate that should be used for signing.
4. Click the Sign Files button, the LISP file will be signed and status in the list control will be updated to show the signed status.

AcSignTool.exe

This is a command line tool that is used for signing LISP files. This can be used in scripts to sign the LISP files in batch, for example at the end of a build.

Usage:

AcSignTool -sign /file:[inputfile] /cert:[certificate] /time:[timestamp] /comment:[description]

Example:

AcSignTool -sign /file:"c:\files\foo.lsp" /cert:12345678901234567890 /time:1

AcSignTool can also be used to verify the signature of a LISP file.

Usage:

AcSignTool -verify /file:[inputfile]

Example:

AcSignTool -verify /file:"c:\files\foo.lsp"

Running AcSignTool without any argument will display the Help, with the certificates installed along with the time source available for usage.