Control access to software using Software Restrictions in Group Policy

By Archiveddocs

Archived: 2026-04-05 22:47:41 UTC

Security

Application Lockdown with Software Restriction Policies

Chris Corio and Durga Prasad Sayana

At a Glance:

- How software restriction policies work
- Inventorying applications in your environment
- Creating and enforcing policies

When IT professionals look to reduce the total cost of ownership, or TCO, of their desktop machines, there are two key strategies that often come to mind. The first one is to get your desktop

users' accounts out of the Administrators group. And the second one is to limit the applications that the users can run. Approaching these problems can be quite a challenge in an enterprise environment, but Windows Vista® offers some technologies that can help you to achieve these goals.

Windows Vista, and its User Account Control (UAC) feature, has made a giant step toward helping IT professionals run their enterprise users as members of the Users group (Standard Users). UAC changed the default security context for all applications to be that of a User rather than an Administrator. This migration to the Users group is still a formidable task, but as the industry adjusts to this new paradigm, the task will get easier over time.

After analyzing the challenges of moving users to the Users group, or sometimes during this process, many administrators catalog what applications their users need to run and consider the steps necessary to allow only these applications. The software restriction policy feature is designed to help IT pros do just that.

You simply specify what applications are allowed to run and then deploy the policy using Group Policy. Enforcing such a policy throughout an entire enterprise can reduce TCO, as this lockdown will limit issues that are related to unsupported applications. (And you can also use software restriction policies in some interesting and very extreme ways, as we discuss in the sidebar "The Bare-Bones Software Restriction Policy.")

How Software Restriction Policy Works

Software restriction policy aims to control exactly what code a user can execute on a Windows Vista machine. You, the administrator, create a policy that defines what can (or cannot) be run in your environment. This policy is

then evaluated whenever and wherever code may be executed. This includes during process creation, in a call to ShellExecute, and when a script runs. (We'll look at this in more detail in a moment.)

If it is determined that an application is allowed to execute, the application will start. If, however, it is determined that an application is not allowed to run, the app is blocked and the user is notified. For example, if you tried to run Solitaire from the Start menu and it wasn't allowed, you'd receive a dialog similar to the one shown in **Figure 1**.

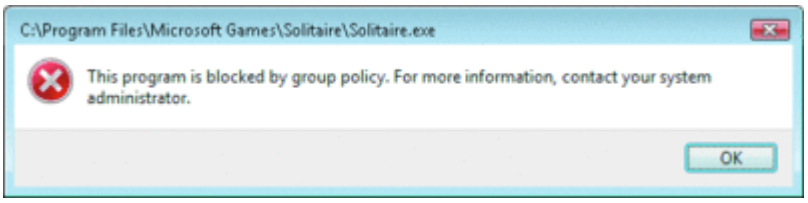

Figure 1 **A dialog appears when an application is blocked** (Click the image for a larger view)

The UI for defining a software restriction policy is exposed in the Group Policy Object Editor (GPOE), and this is where the lockdown policy is authored. There are a variety of methods for defining what code can and can't run. Once the policy is complete and tested, you can deploy it.

Defining the Software Restriction Policy

The first major decision you will make, a decision that will dramatically affect how software restriction policy will work in your environment, is to choose a default rule type. Software restriction policies can be deployed in one of two modes: Allow List or Deny List. Basically, you're choosing whether you want to create a policy that describes every application that is allowed to run in your environment or a policy that defines every application that cannot run.

In Allow List mode, the default rule within your policy is Restricted and will block all applications that you don't explicitly allow to run. In Deny List mode, the default rule is Unrestricted and restricts only the applications that you have explicitly listed.

Deny List mode, as you might guess, is an unrealistic approach if you are seeking a broad TCO reduction and the security benefits resulting from an application lockdown. Creating and maintaining an extensive list that blocks all malware and other problematic apps would be next to impossible; therefore, we recommend that software restriction policy be implemented in Allow List mode, which means a default rule of Restricted.

Inventorying Apps in Your Environment

If you're going to design a policy that will specify what applications can run, you need to determine exactly which applications are required by your users. The software restriction policy feature offers an advanced logging feature with a very simple policy to understand exactly what applications are running in your environment.

On a sample set of machines within your environment, deploy software restriction policy with the default rule set to Unrestricted and be sure to remove all other additional rules. The plan is to enable software restriction policy but not allow it to restrict applications; instead, you are using it to just monitor what is being run.

Next, create the following registry value in order to enable the advanced logging feature and set the path to where the log file should be written:

```
 "HKLM\SOFTWARE\Policies\Microsoft\Windows\Safer\
CodeIdentifiers"
String Value: LogFileName, <path to a log file>
```

Now, when an application is run and the software restriction policy evaluated (it is evaluated even though it is allowing everything to run), an entry is written to the log file.

Each log entry includes the caller of the software restriction policy and the process ID (PID) of the calling process, the target being evaluated, the type of software restriction policy rule that was hit, and an identifier for the rule. Here's a sample entry written when a user double-clicks on notepad.exe:

```
 explorer.exe (PID = 3268) identified
C:\Windows\system32\notepad.exe as Unrestricted using
path rule, Guid =
{191cd7fa-f240-4a17-8986-94d480a6c8ca}
```

This log file represents every piece of executable code that software restriction policy will check when it is enabled and set to block applications. That means that you must decide for each entry in the log file whether or not it should be accounted for in your Allowed List. Note that you'll see several binaries being checked that are part of Windows® and required for the system to work.

The logging technique that we describe here offers a clear way for you to understand exactly what applications the software restriction policy will encounter in your environment. But it is not the only way to accomplish this task.

The Inventory Collector that is included as part of the Microsoft® Application Compatibility Toolkit 5.0 gives you the capability to inventory the applications that are being used in your environment. This tool offers a variety of different ways for you to discover what applications are installed in your environment, and it also consolidates the results into a central database.

Authoring Additional Rules

Now that you have a list of applications that must be allowed to run in your environment, you are ready to create the actual rules that will permit these applications to run. The software restriction policy feature uses two ways to identify policy—one is based on the cryptographic properties of an application (such as its hash), and the other defines a Trusted Path or folder in which trusted applications should reside.

Figure 2 shows where you would add rules to allow the applications to run in the Software Restriction Policies node of the GPOE (gpedit.msc). The most straightforward way to define applications in your environment is to create a hash rule for every single binary you encountered during the logging phase.

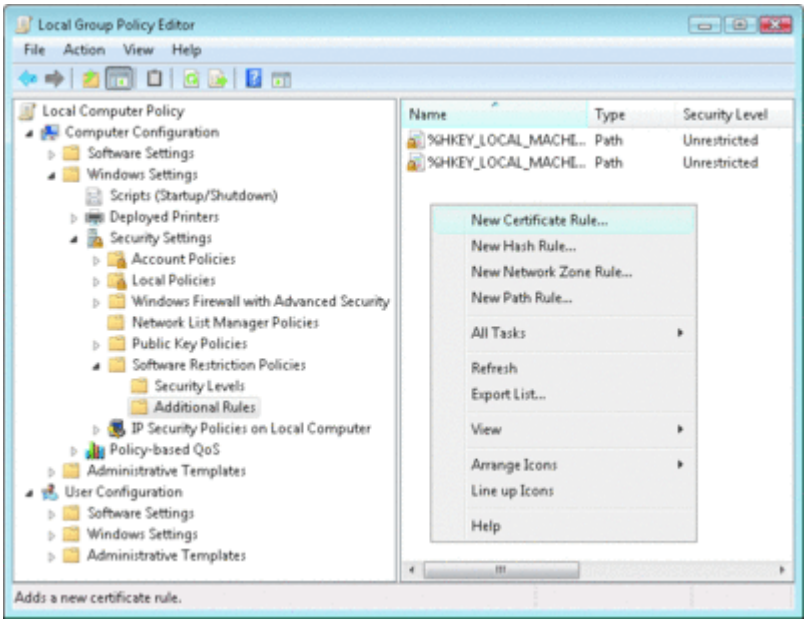

Figure 2 **Using gpedit.msc to author software restriction policies** (Click the image for a larger view)

Since the hash is a unique value that is returned for a particular set of bits, each binary in your policy is going to have a different hash. This approach is particularly secure and will only allow the specific binaries in your policy to run.

Of course, there are some drawbacks to this approach. For instance, your environment could easily have several thousand binaries. It could be difficult to author all of these rules in the software restriction policy UI, and as the number of rules gets particularly large, performance can be affected. In addition, each application update in your environment will require one or more new hash rules to be deployed in the environment. Updating such a large policy as applications are updated can be a huge burden.

Fortunately, this burden can be avoided because there are two other ways to identify rules that make it easier to use software restriction policies in your environment. Venturing further down the cryptographic security route, you can create a rule that will allow any binary that is signed by a particular certificate to run.

Doing this helps to simplify the maintenance of the policy list since, when an application is updated, the new binaries will generally be signed by the same certificate as the previous ones. However, if you don't want a previous version of the binary to run in your environment, you'll need to add a Restricted hash rule in order to prevent the file from being allowed.

By default, the evaluation of certificate rules is disabled for software restriction policies. There are two reasons behind why this had to be done.

First, the certificate rules in the software restriction policies are defined by what is in the system's Trusted Publishers store. Because the Trusted Publishers store is used for purposes other than just software restriction policy rules, this requires additional time and consideration when it is used for the software restriction policies feature.

The second reason is that in order to determine whether a file's signature is valid, you must take a hash of the file and compare it to the signature information. Hashing a file is a very expensive operation—the entire file must be read off of the disk and processed to calculate this hash.

In order to enable certificate rules, navigate to the Software Restriction Policies node and select Enforcement Object in the results pane. Then double-click to open its properties dialog and select the enforce certificate rules radio option.

The other common way to identify code is to use the path of the code on the local machine. This is an effective and efficient technique, but it has one drawback—you must be careful to ensure that the security settings are set properly on the folder.

If you add a particular path rule, and this path allows users to write files there (to the desktop, for example), they would be able execute anything they want just by putting the executable in that folder. However, if your users are not in the Administrators group, they generally don't have the ability to modify anything in the Program Files or Windows directories. This means that if all your applications are in the Program Files directory and your users aren't Administrators, then you should look to path rules for having a very simple and efficient policy.

Path rules offer some other features that make it more appropriate for some environments. It allows for wildcards and lets you use environment variables to make it easier to define rules that are portable in your environment—after all, %systemdrive% might not be c:\ for every user.

As for performance and maintenance, this is probably the most hassle-free way to identify code. Path rules are certainly something to consider, but you need to be aware of the additional security considerations.

Network Zone Rules

Software restriction policy does include a rule type called Network Zone rules, although this type of rule is being deprecated. The original intention of these rules was based on the idea that the source of particular executable code could be identified and trusted, and thus the code would be allowed to run. Unfortunately, this is particularly difficult to do and, as a result, never worked particularly well. Currently, this type of rule is not enforced in the majority of places on the software restriction policy entry points.

In cases where most applications are installed to the %Program Files% directory but there are other executable files that are installed elsewhere and signed by a specific certificate, it may make sense to use rules of different types. A few hash rules, a couple of path rules, and you'll find you've got the right policy for yourself.

Just keep in mind that there is an order in which the rules are processed (as shown in **Figure 3**). Certificate rules are the most specific, hash rules are next, then path rules, and then path rules with wildcards in them. So, if a piece code is identified by a hash rule and a path rule, the security level of the hash rule will take priority.

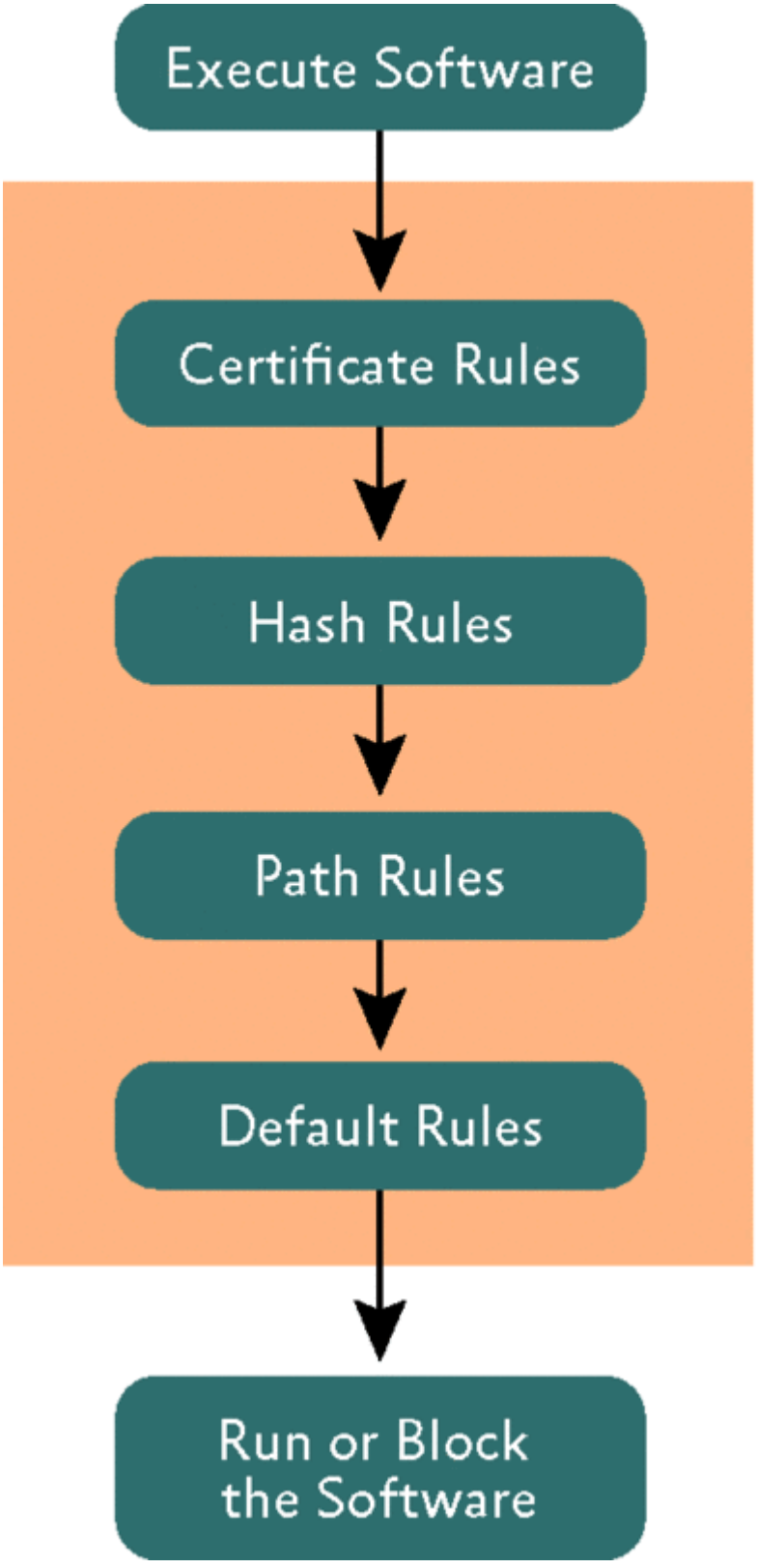

Figure 3 **Rule processing order** (Click the image for a larger view)

Policy Enforcement

The software restriction policies feature provides a breadth of coverage on the system being secured. The idea is that any location from which code can be executed should be integrated with the software restriction policy and, in

turn, check the policy to see if the executable code is allowed to run.

While there are numerous places that check the software restriction policy, the most straightforward entry point is CreateProcess. During CreateProcess, the policy is checked to determine whether or not the binary that represents the application is permitted to execute. The policy check is done by the SaferIdentifyLevel API, which is publicly documented. The general process is illustrated in **Figure 4**. (We'll discuss SaferIdentifyLevel in more detail in a moment.)

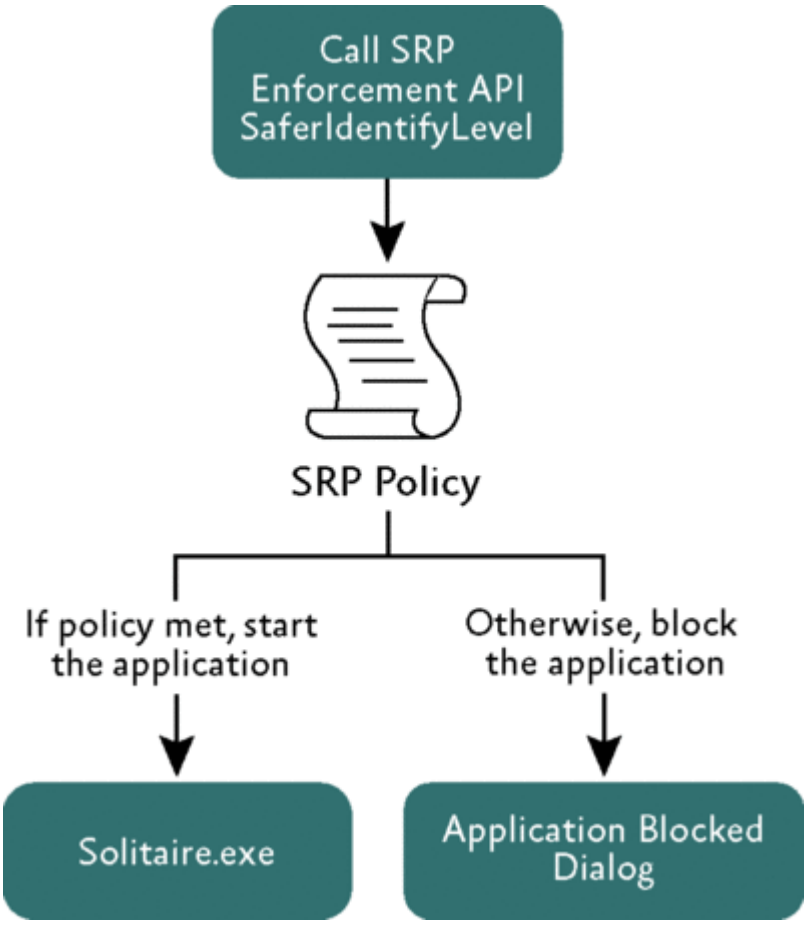

Figure 4 **Using SaferIdentifyLevel to determine whether a binary can be executed** (Click the image for a larger view)

After CreateProcess, the next most commonly used API where software restriction policy is enforced is ShellExecute. This is the API that is called when the user clicks on an application in the Start menu or double-clicks something on the desktop.

ShellExecute can be called on a wide variety of file formats. In cases such as a .txt file, calling ShellExecute on the file doesn't actually result in the file being executed—technically, of course, the file is opened. For this reason, software restriction policy contains a list of executable file types so that it can control what types of files are checked when ShellExecute is called. You can customize this list of executable file types in the software restriction policy UI.

Beyond CreateProcess and ShellExecute, there are two other key integration points: LoadLibrary and script hosts. LoadLibrary is obviously an important place to check executable code, but unfortunately LoadLibrary presents

some special constraints.

Most applications have one executable and several DLLs that are loaded. And there are typically lots of applications running on the system. This means that LoadLibrary would require a lot of checking of the policy. Depending on your policy for identifying code, this could be a very expensive entry point to enforce—imagine checking the hash of every DLL loaded on the system, which includes hashing the binary and then comparing it to a list of perhaps thousands of hashes.

This functionality is disabled by default, but it can be enabled manually. To do this, navigate to the Software Restriction Policies node in gpedit.msc and double-click on Enforcement. After that, select the All software files radio button.

As we mentioned, software restriction policy is integrated with most script hosts in the system. This includes cmd, VBScript, Cscript, and JScript®. These entry points, as well as the others, use the primary software restriction policy enforcement API: SaferIdentifyLevel.

The SaferIdentifyLevel API determines whether a specified executable should be allowed to run by looking up the identification information in the relevant software restriction policies. This is a publicly documented API. Third-party script hosts and executable environments can and should use it in order to integrate with software restriction policy so the policy can determine whether a piece of executable code should be allowed to run.

Running as a Standard User

A not-so-well-known feature of the software restriction policy is its ability to filter the privileges of certain applications when they're launched. This functionality was introduced in Windows XP, but it wasn't exposed in the software restriction policy UI until Windows Vista.

In this way, it was a precursor to Windows Vista UAC because you could use it to run an application as a Standard User even when the user was a member of the Administrators group. This is what happens when you create a rule and set the Security Level to Normal User in the Additional Rules UI.

Both UAC's token filtering functionality and software restriction policies' Normal User rules take advantage of an underlying API that implements the same behavior as the CreateRestrictedToken API. However, the overall architectural differences in the technologies are very significant. But UAC differs from software restriction policy in a couple of key ways.

First, in Windows Vista with UAC enabled, every application is started with a security token similar to a member of the Users group by default, even when the user is an Administrator. This is achievable with software restriction policy, but there is no means to start an application with the user's actual Administrator token—for instance, when the user needs to install an application. The change in default security context and ease of access to a user's full administrator token are key benefits of UAC.

The second difference is that, in the case of executables, the code itself expresses what privilege level is required in order for it to function. This is a key distinction because independent software vendors (ISVs) and developers understand the needs of their code. For example, if a control panel application needs to edit something that requires administrator privilege, it can specify that requirement in its manifest. Thus the ISV can describe the

privileges needed by an application rather than imposing a certain privilege level upon it without any means available to change this level.

For now, you should avoid using the Normal User rules unless you really understand how they work. UAC is an excellent way to help get your desktop users out of the Administrators group, and you should seriously consider simply leaving UAC in your enterprise environment.

Software Restriction Policies in Use Today

There are a number of moving parts that you need to account for when you use software restriction policy. But it's not as you might think, and, in fact, you may even be using software restriction policies today without realizing it. If, for instance, you run Parental Controls on a Windows Vista system, you are using software restriction policies to control the execution of applications.

Software restriction policy was an important technological development when it was first introduced in Windows XP. But application lockdown is really just now starting to get the attention of IT professionals.

The software restriction policy feature in Windows Vista has some rough edges that still need to be polished, but it is clear that administrators want to have this increased control over what is running in their environments. Fortunately for all of us, this technology will continue to evolve and make it easier for IT pros to manage their systems and lower the cost of running a Microsoft Windows environment.

The Bare-Bones Software Restriction Policy

One application of software restriction policy is to create a policy that locks down Windows into a kiosk mode. Microsoft actually produces a tool kit, called SteadyState™, for creating this kiosk. However, if you are only interested in locking down the applications that can run, this can be achieved more simply.

To allow the minimum set of applications required to log on to Windows Vista, create a policy that allows logonui.exe and userinit.exe to run from %windir%\system32. Most users will probably also need the following applications allowed to run:

```
dllhost.exe, rundll32.exe, control.exe (also under %windir%\system32)....
```

If you want the default Windows shell, you would also want to add %windir%\explorer.exe.

Alternatively, you could choose to boot directly into an application, such as Internet Explorer®. In that case, it would be necessary for you to list iexplore.exe instead of explorer.exe.

Another aspect of this bare-bones policy is that your users should not be in the Administrators group. This is important so that they cannot bypass the policy. However, because they can only run a very basic set of applications and do not have the privileges of an Administrator, the users will not have the privileges to install applications or maintain the system.

You will need to have some way to do that for these machines. If you intend to update and maintain these machines by logging on locally with an Administrator account, you should select the radio button that applies

software restriction policy to all users except local administrators. The policy should also include consent.exe and cmd.exe. This will allow the Administrator to launch any administration option from an Administrator cmd prompt.

Note that UAC limits the privileges of the user's security token by default to make it look like the token isn't a member of the Administrators group. Even if you set the setting above to not apply the policy to Administrators, it will still be applied to the users. It is only when you go through the UAC elevation experience that the Administrator actually gets the full administrator privileges and the software restriction policy will then not be applied.

Chris Corio Chris Corio was a member of the Windows Security team at Microsoft for more than five years. His primary focus at Microsoft was application security technologies and management technologies for securing Windows. You can reach Chris at winsecurity@chriscorio.com.

Durga Prasad Sayana Durga Prasad Sayana is a Software Design Engineer/Test on the Windows Core Security team. His primary interests are security technologies and software testing. You can reach Durga at durgas@microsoft.com.
