Microsoft

AppLocker Design Guide

AppLocker Design Guide

Microsoft uses the term "application control" to describe the approach of explicitly allowing the code that will run on a Windows host. This concept is widely referred to as "application whitelisting"[1] across the IT industry, so this latter term will be used throughout this document to avoid any potential confusion.

[1] Examples include http://www.dsd.gov.au/publications/csocprotect/top_4_mitigations.htm, http://www.sans.org/critical-security-controls/guidelines.php, and http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-53r4.pdf

Contents

Authors

Nick Torkington
Microsoft Trustworthy Computing

1 Introduction

Malicious software (or "malware") has become one of the top security threats across all segments of the computing landscape – from the largest multinational corporation down to the individual home user. The approaches to combating malware are largely reactive; with ever-growing signature databases to detect known malware, and a variety of techniques to detect "malware-like" behaviors in as-yet unclassified new malware variants. While the latter is typically marketed as "proactive detection", it still relies on analysis of past and present malware to define the behavior models that will hopefully identify future malware. The success of this approach is limited – with detection rates below (and in many cases well below) 70%[2], and a comparatively high false positive rate (which can have a high impact on productivity if key applications are affected). Combine this lack of efficacy with the costs of infrastructure and bandwidth for regular signature updates, and the performance penalties incurred by malware scanning, it is apparent that traditional malware defenses alone[3] are insufficient to combat the volume and sophistication of the modern malware threat.

The underlying problem is a lack of consistent methods for establishing trust between a computing platform and the code it is asked to run. While a number of techniques exist such as digital signing (to prove authenticity and integrity) and distribution though app stores (where apps can be vetted), these are not applied uniformly across the computing ecosystem and are not enforced by the end-user's computer.

"Application whitelisting" is an approach which attempts to address this trust issue. The organizations IT function (in consultation with the business units) defines what is trusted by the organization, and an application whitelisting product on the organization's computers enforces this policy by only allowing trusted content to run. Application whitelisting is increasingly recognized in the security community as a more effective alternative to the never-ending "arms race" between anti-malware vendors and the criminals who use malware as a tool in performing illegal activities.

In fact, the Australian Defence Signals Directorate list application whitelisting as the highest ranking control in preventing successful cyber intrusions http://www.dsd.gov.au/infosec/top35mitigationstrategies.htm. This finding is based on their vulnerability assessment work along with lessons learned from real incident responses during 2011 and 2012. They recommend deploying whitelisting initially to all high sensitivity user systems, such as PC's run by executive staff, then progressively extending whitelisting to the broader organization of users and server infrastructure.

Effective application whitelisting can both prevent an unapproved executable from running on a system and prevent an executable from being installed. It can be equally applied to end user systems and servers.

---

[2] Taken from last 3 years reports of proactive detection tests from http://www.av-comparatives.org/comparativesreviews/retrospective-test

[3] Anti-malware software is still important as part of a comprehensive, defense-in-depth strategy.

AppLocker is an application whitelisting technology that is built into business-focused editions of Windows 7 / Server 2008 R2 (and later). It requires no additional licensing, and little or no additional infrastructure to be used. Other than policy, it doesn't require distribution of any software or signatures, and the probability of false negatives or false positives is extremely low.

While the technology is free to use and fairly simple to understand, it is not trivial to deploy application whitelisting in an organization of even moderate complexity. Effective application whitelisting requires the organization to explicitly define what software is 'trusted' throughout the organization. If the organization cannot accurately express what is trusted in AppLocker policy, end users may be impacted. There are mechanisms within the AppLocker technology, such as first running in "Audit Only" mode that can help organizations in this regard.

The aim of this guide is to describe the end-to-end process for developing, testing and deploying AppLocker in an organization of any size and regardless of their security requirements, in a way that minimizes the impact on the operation of the business.

2 An AppLocker Primer

2.1 What is AppLocker?

AppLocker is a technology built into business-focused editions of the Windows platform (Windows 7 / Server 2008 R2 and later) that allows an organization to centrally manage the execution environment on their clients and servers. AppLocker implements a concept called "application whitelisting" – whereby applications, application installers and scripts are prevented from running unless they are explicitly allowed by inclusion in a set of whitelisting rules.

What Risks does Application Whitelisting seek to Control?

Application whitelisting seeks to ensure that only authorized or trusted programs can be run on the clients and/or servers it protects. This addresses risks such as:

- Preventing applications infected by malware, exploits or other malicious code from being executed and compromising the security, integrity or availability of the computer.
- Preventing unsafe applications from being installed or run that may violate corporate policy or compliance obligations.
- Preventing retired or otherwise unsupported applications from being installed or run which may lead to higher support costs.
- Preventing unlicensed software (or software used to share and distribute it) from being installed, which may expose the company to liability.

AppLocker and Access Control Lists

The Windows security model uses access control lists (ACLs) to determine what actions **subjects** can take on **objects**. In order to run executable code, the subject must have Read and Execute permissions on the executable object. The general approach taken in Windows is to allow any authenticated user to Read and Execute an executable file, and use more restrictive ACLs on the objects and data manipulated by the executable. For example, any authenticated user can run the registry editor by default, but the items they can view and change are controlled by ACLs on the individual registry keys. This approach makes sense because a user may be able to run the same executable from a volume whose security is not centrally-managed, or a file system that does not support ACLs at all. In contrast, AppLocker allows or blocks execution independently of ACLs. An executable will be blocked whether it resides on an ACL-controlled NTFS volume, or a USB drive that is not controlled by ACLs. While AppLocker should not be viewed as a replacement for ACLs, it provides complementary protection against execution in scenarios that ACLs alone cannot address.

2.2 AppLocker Policy

A computer can implement one or more AppLocker policies that are defined locally (in Local Security Policy) or centrally via one or more Group Policy Objects ("GPO") (computer policy). The **effective policy** that is actually implemented on the computer is the sum of all rules defined in Local and Group policies. Creation of rules in the Local Security Policy is generally not recommended in an enterprise environment (as subsequent changes cannot be managed centrally), but is extremely useful for developing rules on a reference computer, and importing those rules into GPOs for broad deployment. This approach will be described in detail in this document.

A possible exception to this rule is that high value computers or computers operating in high risk environments may get local AppLocker policy applied during deployment to provide immediate protection before AppLocker rules from Group Policy are applied. Use the steps in section 6.5 to export the desired AppLocker rules to XML, and the steps in section 6.7 to import them into the target computer. This should be balanced with the fact that local policy cannot be managed centrally, and may become "stale" during the lifetime of the computer.

Creation of rules in Local Security Policy requires local admin rights on the computer. Because these rules will be merged with rules defined in GPOs, it is evident that a user with local admin rights can circumvent centrally-managed AppLocker policy by creating permissive rules locally (or they could simply stop the Application Identity service). Therefore AppLocker does not provide protection against a malicious user with administrative rights to the computer. It can however enforce desired behaviors in (non-malicious) admins by requiring them to take explicit actions to work around policy – especially if event logs are centralized and monitored for AppLocker policy changes.

While AppLocker rules can be created on any business-focused version of Windows 7 / Server 2008 R2 and later, they can only be enforced on the following versions:

- Windows 7 – Ultimate and Enterprise editions
- Windows Server 2008 R2 – Standard, Enterprise, Datacenter and Itanium editions
- Windows 8 – Enterprise edition
- Windows Server 2012 – Standard and Datacenter editions

2.3 AppLocker Rule Types

2.3.1 Rule Types and Associated File Associations

AppLocker can be used to control several different types of executable content, with Windows 8 / Server 2012 building on the types available in Windows 7 / Server 2008 R2. Each rule type has their own independent operating mode (see next section) and unique set of rules. The five rule types that can be defined in AppLocker are as follows.

Executables
Refers to .exe and .com files that are launched from the local computer, removable media, or from a network location (file share, web server etc.). It also includes applications that are delivered by application virtualization (App-V). This is the file type most commonly controlled as they provide the greatest opportunity for computer compromise.

Windows Installers
Refers to traditional application packages that are installed by the Windows Installer service. It includes the installer packages themselves (.msi), and application patches (.msp). On Windows 8 / Server 2012 computers, it also includes application transforms (.mst) that specify which components of the installer package should be installed.

Windows installers install files and make configuration changes that only users with administrative rights can perform. Assuming that users are not given administrative rights, a good level of control already exists in preventing users from making unauthorized changes. Even after an application is installed, Executable rules must still be in place to allow the application to actually execute. For these two reasons, many organizations choose not to control Windows Installers using AppLocker. Alternatively, they may deem to implement a slightly restrictive policy allowing administrators to install applications as long as they have been signed by a publisher that the computer trusts.

Scripts
Refers to PowerShell (.ps1), Windows command interpreter (.cmd and .bat), VB script (.vbs) and Java script (.js) files, but does not apply to client-side scripts executed by a browser. Note that additional script types (such as Perl) cannot be controlled by AppLocker, although the scripts themselves can be prevented from running by preventing the corresponding script interpreter from running. Also note that Office applications can run Visual Basic for Applications (VBA) code in macros that are not controlled by AppLocker.

Unlike almost all executables, installers, packaged apps and DLL/OCX files, many of the several hundred scripts that ship with Windows are not digitally signed. A number of complex path or file hash rules will need to be created to allow them to run. While secondary to controlling executable files, execution of scripts should be considered for high value computers, or those exposed to significant risk. If the organization has enabled code signing for in-house developed applications and packages, the same process can be used for digital signing of scripts to allow simpler (and fewer) publisher rules to be used to allow them to run.

Packaged Apps (Windows 8 / Server 2012 only)
Packaged apps (.appx) are the new application delivery model introduced in Windows 8 / Server 2012. Applications are isolated inside individual "App containers" from each other, the operating system, and computer resources, and can only interact with them through tightly controlled interfaces. All packaged apps are vetted before publication in the Microsoft Store and must be digitally signed to verify their origin and prevent tampering.

Because of the isolation of packaged apps from the rest of the system and their tighter controls, the requirement for administrative access to install them has been removed – allowing any standard user to install any signed app (unless controlled by GPO). As packaged apps pose lower risk to the computer and other users than traditional Windows apps, an organization may choose to give their users more flexibility in installing and running these apps. This could be done by creating the default rule which allows any packaged app signed by any publisher to run. Alternatively, specific applications or applications by specific publishers only could be allowed by AppLocker rules. Also, access to the Windows Store (WinStore) and the Modern UI Control Panel (windows.immersivecontrolpanel) can be allowed or denied just like any other packaged app.

Shared Libraries and Controls
Refers to .dll and .ocx files that are shared between applications and loaded by the applications that require them.

Modern malware targets the shared code that legitimate applications load in an effort to avoid detection. To provide the best possible protection, DLL/OCX files should be evaluated by AppLocker policy. This however this creates an additional dependency in that AppLocker rules must be created that allow every such .dll and .ocx file required for the application to function. If a single .dll or .ocx file that is required by an application is prevented from loading, the application will either fail to run or will generate errors when certain tasks are performed. Consequently, checking of .dll and .ocx files is disabled by default and must be explicitly enabled before it can be used.
While organizations must perform rigorous testing to ensure that they include all shared libraries used by their applications, the procedures in this guide should allow this feature to be enabled and used to provide maximum protection against malicious code. However depending on resource availability, some organizations may choose to initially implement AppLocker without enabling this feature, and enable it once they are comfortable in deploying and maintaining it in Production.

Rule Types and Associated File Associations
Each rule type has an associated (non-extensible) set of file associations that is summarized in the following table:

Rule Type	File Associations
Executable	.exe, .com
Windows Installer	.msi, .msp (patches), .mst (transforms – Win 8 / Server 2012 only)
Script	.ps1, .bat, .cmd, .vbs, .js
Packaged App	.appx – Windows 8 / Server 2012 only
Shared Libraries & Controls	.dll, .ocx

Table 1: Rule types and associated file associations

2.3.2 Applications that Cannot be Controlled by AppLocker

AppLocker can only control applications running natively in the Win32 subsystem. It cannot control:

- **Applications running in any other subsystem (such as Posix).**
  The Posix subsystem can be prevented from running using AppLocker – either by not including it in an allow rule, or explicitly denying it. The product name is "Microsoft Windows Subsystem for Unix-based Applications" (if a Publisher rule is used), and the filename is \Windows\System32\psxss.exe (if a Path rule is used). The Posix subsystem can also be disabled in Windows by removing it from the registry value HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\Subsystems\Optional[4]
- **16-bit applications running in a Windows NT Virtual DOS Machine (ntvdm.exe).**
  The VM (and any 16-bit applications it would host) can be prevented from running using AppLocker – either by not including it in an allow rule, or explicitly denying it. The filename is \Windows\System32\ntvdm.exe. Note that this only applies to 32-bit versions of Windows, as 16-bit applications cannot run on x64.
- **Scripts interpreted by non-Windows script hosts (e.g. Perl).**
  Effective AppLocker policy (as well as least privilege) will stop third-party script interpreters from being installed. If these are required, native controls in the interpreter will need to be used to control what can and cannot run.
- **Code interpreted by other applications (e.g. macros within Office).**
  Office can be configured to only run macros signed by trusted publishers[5].

2.3.3 Operating Modes

AppLocker operates in one of three modes for each of the rule types listed above:

- **Not configured** — If no rules are present, AppLocker is disabled for that rule type. If one or more rules are present, AppLocker operates in the "Enforce rules" mode for that rule type.
  Also, if a higher precedence GPO is set to "Not Configured", the mode selection is deferred to the next highest precedence GPO.

- **Enforce rules** — The actions specified in the AppLocker rules will be enforced, and actions logged to the Windows event log.

[4] The Security Compliance Manager baselines can be used to configure this setting (http://technet.microsoft.com/en-us/library/cc677002.aspx)

[5] Refer to the security guide for the appropriate version of Office contained within the Security Configuration Manager baselines. Also, ensure that the hotfixes described in section 5.10 are installed to prevent possible circumvention of AppLocker policy.

- **Audit only** The actions specified in the AppLocker rules will **not** be enforced, but events will be logged to the Windows event log indicating whether those actions would have been allowed or blocked based on the rules defined.

"Audit only" mode is used to develop and test the AppLocker rules without impacting normal operations. Computers will typically operate in this mode until sufficient confidence is reached that the AppLocker rules support all legitimate use cases.

After validation of the AppLocker rules, computers are changed to "Enforce rules" mode where actions that are not explicitly allowed (through rules) are blocked. Allow / block decisions continue to be logged, allowing on-going monitoring to enable rules to be tuned, or to spot suspicious behavior.

You need to be aware of how AppLocker behaves under certain conditions:

- Choosing "Enforce rules" or "Audit only" mode has no effect for that type until at least one rule is created.
- If in the "Not defined" mode when one or more rules are created, the computer will operate in "Enforce rules" mode for that type. Therefore, you should always explicitly choose "Audit only" mode for that type before you start developing rules.
- An enforcement setting of "Enforce rules" or "Audit only" in a GPO will overwrite the enforcement setting in local policy. If multiple GPOs configure the enforcement mode, the last to be applied (using normal group policy precedence) takes effect.
- If in "Enforce rules" mode for EXE files and one or more rules exist, APPX operates in "Enforce rules" mode even if no APPX rules have been defined. The result is that APPX packages will be unable to run until explicit APPX rules have been created.

2.4 AppLocker Rules

AppLocker rules describe the **Action** that a **Subject** can take on an **Object**.
Each AppLocker rule is made up of the following elements. These are described in detail in the remainder of this section:

- Name
- ID
- Description
- Subject
- Action
- Object (with optional Exceptions)

2.4.1 Rule Name, ID and Description

Rule Name
The rule name is a human-readable label that is displayed in the user interface. The name should be as descriptive as possible, without making it overly long (where it won't be easily viewable in the user interface).

Rule Name Prefix
While not a requirement, every rule name should have a prefix. This prefix should be used to identify the application it is controlling – such as "IE9: or "Office2013:". Prefixes are extremely useful for the following reasons:

- When rules are sorted by name, all rules affecting components of an application will be grouped together.
- The "effective policy" on a machine may be made up of rules from multiple GPOs as well as (although not recommended) the local policy. If rules in GPOs have known prefixes, it is easy to tell which GPOs are applied on a machine.

Rule ID
While not exposed through the user interface, each rule also has a unique identifier which is a random hexadecimal number that uniquely identifies each rule. These are used during merge and replace operations and allow AppLocker to match source and destination rules even if some of the properties are different.

Rule Description
Descriptions are optional but allow additional information to be associated with individual rules – such as specific reasons for its creation, contact details of the rule author or application owner, or any other details that cannot be expressed in the rule name. Use of detailed descriptions is highly recommended to assist in traceability back to original rule requirements.

2.4.2 Rule Subjects

Each AppLocker rule applies to a **single** subject – either an individual user or more commonly, a Windows security group. If multiple individual users or multiple security groups should be granted access to the same object, multiple rules will need to be created (or a group created that contains all of these groups / users).

By default, rules apply to the built-in "Everyone" group; which matches any user trying to run the specified object.

Note that AppLocker rules have no effect on files running as SYSTEM, such as the Windows kernel, critical services and kernel-mode drivers.

2.4.3 Rule Actions

Each AppLocker rule can have one of two actions:

- Allow
- Deny

Unlike many rule-based access control products, the ordering of AppLocker rules is not significant. "Deny" rules are always processed first, and so take precedence if a conflict occurs with an "Allow" rule. If a Subject / Object pair does not match any "Deny" rules, "Allow" rules are processed until a match is found, and then the action is allowed. If no "Allow" rules are matched, the action is denied. This is referred to as "default deny" mode and is the only mode that AppLocker can operate in – objects are **implicitly** denied unless **explicitly** allowed.

It is possible to simulate a "default allow" mode by creating an "Allow" rule matching everything for each rule type, and using "Deny" rules to block specific objects. This would be useful in an organization that gives their users the general freedom to run whatever they want, except for a defined list of prohibited applications (such as peer-to-peer file sharing applications that may be used to spread infected or unlicensed content). This approach is generally not recommended, as it is impossible to identify (and hence create rules for) every instance of undesirable programs.

However some companies that collect inventory data from their computers may choose to use this approach as a means of blocking undesirable applications that show up in inventory reports. This is a reactive approach and has obvious drawbacks, but may be applicable to some organizations. While implementation of this approach can be easily inferred from this document, the remainder of the guidance in this document assumes that AppLocker is operating in the intended "default deny" mode.

Avoid "Deny" Rules

Except to enable specific scenarios such as that described above, "Deny" rules should generally be avoided. AppLocker denies everything by default, so careful consideration to the use of "Allow" rules is all that is required to enable all desired usage cases without the complexities introduced by "Deny" rules.

"Deny" rules introduce complexity because they are always processed before "Allow" rules. This is simple to understand and troubleshoot when all rules are coming from a single policy object, but becomes exponentially more complex with the number of policy objects whose rules are merged to create the "effective policy". When troubleshooting an access denied issue, you not only need to check that the user is within scope of an "Allow" rule, you also need to check that they don't belong to a group that may have been scoped in a "Deny" rule. Fortunately there are techniques to identify the rule within the effective policy that is denying access, but the granting of access through layered policy objects is greatly simplified if only "Allow" rules are used.

Another reason to avoid "Deny" rules is that it would be easy to make a change to a file that does not affect its functionality but would cause it to no longer match a "Deny" rule. This

could be as simple as changing a file's name in the case of a Path rule, or changing a binary file so that the calculated hash no longer matches that listed in the rule (both assuming that the user has permissions to do so).
While there may be valid reasons to use "Deny" rules which are perfectly acceptable as long as they are well managed, the remainder of this guide assumes that only "Allow" rules will be used.

2.4.4 Rule Objects

There are three different ways in which objects can be identified by AppLocker, and correspondingly three different rule types that are used to define them in AppLocker rules:

- **Path rules** uses the folder or file name to identify an object
- **File hash rules** uses the hash value of a file to uniquely identify it
- **Publisher rules** uses information about the publisher or file attributes for digitally signed files

Each of these three rule types have their own unique advantages and disadvantages, and are described in detail below. A table at the end of this section also summarizes these advantages and disadvantages.

Path Rules
Path rules apply to an individual file or a folder. In the case of an individual file rule, the full path and name of the file is specified.

In the case of a folder rule, the rule applies to all files with applicable extensions in that folder and **ALL** subfolders. While absolute path names can be used, variables may be used to cater for any customized folder names or drive letters. However where possible, absolute paths should be used to mitigate the risk of a variable change allowing code to run from an alternate location. The variables are unique to AppLocker, and are listed in the table below (along with their Windows system variable equivalents):

Windows Directory or Drive	AppLocker path variable	Windows environment variable
\Windows	%WINDIR%	%SystemRoot%
\System32	%SYSTEM32%	%SystemDirectory%
Windows installation directory	%OSDRIVE%	%SystemDrive%
Program Files	%PROGRAMFILES%	%ProgramFiles% and %ProgramFiles(x86)%
Removable media (CD, DVD etc.)	%REMOVABLE%	
Removable storage (USB)	%HOT%	

Table 2: AppLocker path variables

Path Rule Advantages
The advantage of path rules is that they can accommodate future software changes on the target computer far better than the other types – which is why they are used in the creation of default rules for EXE, scripts and DLL / OCX. For example, if you create a path rule for %PROGRAMFILES%, any future additions to "\Program Files", "\Program Files (x86)" or any of their subfolders will be automatically accommodated without changes to the rule.

Path Rule Disadvantages
Path rules however have serious disadvantages compared to the other two types. Most obviously, it does not align well with the policy intentions of application whitelisting. Those defining AppLocker policy may not care where a particular file is run from – they would want the same policy applied to a file regardless of which folder it resides in or whether it is local, on a file share, USB drive or DVD.

The way in which AppLocker treats subfolders of path rules is another serious disadvantage. If AppLocker only cares that a file is run from a particular location, then a user can copy any file to a writable subfolder under the parent path defined in the rule to circumvent policy. The default EXE, script and DLL / OCX rules (which will not be used in this guide) allow anyone to run anything if it resides in a subfolder of \Windows. As there are numerous standard user-writable subfolders of \Windows, these rules need to have numerous exceptions (described below) defined if they are to be effective. The number of writable subfolders increases further when you consider accounts commonly used by services – such as Local Service and Network Service. Any service running as these accounts that could be made to write content to one of these locations could be used to circumvent a path rule.

As described above, path rules can be defined down to individual executables (by full path and name). However AppLocker has no means of determining whether the file name actually represents the file intended to be allowed. This means that any executable can be renamed to match the conditions of a path rule and will be allowed to run – whether it is the legitimate executable or something completely different.
The final disadvantage is that path rules do not check the integrity of files before allowing them to run. This means that a file that is allowed to run via an AppLocker rule will run equally well if it is the intended version or has been modified for malicious purposes (e.g. by malware).
Due to these numerous, significant disadvantages, path rules should only be used when neither of the two alternative rule types are appropriate. When using path rules, care must be taken to ensure that ACLs on the respective folders (and all subfolders) prevent users from writing content to them.[6]

File Hash Rules
Each file hash rule contains the calculated hashes of one or more files. Before a file is run, its hash is calculated by Windows, and if it matches a hash in any of the rules, the corresponding rule action is applied (allow or deny).

[6] AccessEnum (http://technet.microsoft.com/en-us/sysinternals/bb897332) can be used to enumerate user-writable subfolders.

File Hash Rule Advantages

File hash rules avoid the location dependence and integrity issues of path rules. You cannot simply copy a file to a different location or rename a file to circumvent policy as you can with path rules, as the file's hash value is used to uniquely identify it. You also cannot change a single bit within a file whose hash is listed in a hash rule, as the calculated hash will be completely different from the one listed. The one-way nature of hash algorithms makes it infeasible to try to change or create a file that will hash to a specific hash value and still perform its intended function. This gives a very high degree of assurance that the file the user is trying to run is the one listed in the hash rule, and the appropriate action will be applied.

File Hash Rule Disadvantages

While providing the highest assurance of all rule types, hash rules are the least adaptable to change and hence the most expensive to maintain. Every time that a file is upgraded or patched, the hash rule must be modified to include the new hash value or the file will be prevented from running. This would require a disciplined change management process, and result in significant administrative overhead. Consider also that upgrades / patches may be applied over a period of time, and you may have numerous versions of any given executable in use across an organization – of which the hashes of all must be included in hash rules to allow them to run.

The other key disadvantage of hash rules is the sheer number of rules that need to be created, maintained and processed by the computers implementing them. While a single AppLocker rule can contain many hashes, each file needs to be uniquely identified by its hash value. There can be many thousands of eligible files on a computer, making the task of managing thousands of hash values a difficult one.

While security-conscious companies may accept the increased burden for the very tight control given by hash rules, the general recommendation is to avoid hash rules except in very specific circumstances – primarily where the files have not been digitally signed. Files that have been digitally signed should use Publisher rules (described next) which provide the same integrity benefits as hash rules but with far less management overhead.

Publisher Rules

When a file is digitally signed, a hash of the file contents and file attributes (such as the product name and its version) is cryptographically protected (signed) by a publisher. Assuming that publisher is trusted by the user's computer, the computer can validate the integrity of the file and its attributes. Similar to the mechanism used in evaluating hash rules, if the contents or attributes of a signed file is changed, it won't match the information in the signature and will fail validation.

AppLocker uses this validation mechanism in Publisher rules. Rather than matching the hash of the file to be run, Publisher rules make decisions based on any combination of:

- Publisher name (or at its most permissive, any publisher that is trusted by the computer)[7]
- Product name
- File name (as originally chosen by the publisher, not the current name of the file if it has been changed)
- File version – exact version, specified version and later or specified version and earlier

Note that the new .appx packages that run on Windows 8 / Server 2012 must be digitally signed (or Windows won't run them) and have a package name and package version instead of product name, file name and file version.

Publisher Rule Advantages

Publisher rules provide a similar level of integrity protection as file hash rules, without the management overhead. If a software vendor patches or updates their product, they typically sign the updated files with the same attributes as the previous versions – publisher, product and file name. File version would typically be incremented, but this can be accommodated in a publisher rule by the condition of "this version and above" (or you can use * to match any file version). In addition to the obvious security benefits, publisher rules can be used as a way of forcefully retiring obsolete versions that an organization no longer wishes to support, or are known to contain vulnerabilities.

Publisher Rule Disadvantages

The obvious downside to publisher rules is that they can only be used to control applications that have been digitally signed. To control unsigned files, file path or hash rules are the only alternative. The good news is that the majority of software vendors sign their code to ensure its authenticity and to avoid the negative user experience created by operating system warnings that the "publisher cannot be verified". For organizations that develop their own applications, it is a straightforward process to digitally sign their files to allow publisher rules to be able to control execution.

When choosing publisher rules, you need to keep in mind that they are only as secure as the publisher is trustworthy. If you rely on publisher rules and one of the specified publishers has their code signing certificate private key compromised, whoever compromised that certificate could use it to sign their own code[8]. AppLocker would not be able to distinguish between this and legitimate files, and could potentially be used to bypass it.

There have also been a number of high profile incidents where trusted certificate issuers have been compromised and fraudulent certificates generated[9]. Fraudulent code signing

[7] If Organization, Locality, State & Country properties are defined for a signing certificate, only these properties are used by AppLocker and all other properties (including common name) are ignored. If any of O, L, S or C are not defined, AppLocker will use the full certificate subject name.

[8] Witnessed in the Bit9 incident described at https://blog.bit9.com/2013/02/25/bit9-security-incident-update/

[9] Trusted certificate authorities include DigiNotar, Comodo and TurkTrust

certificates (that mimic legitimate publisher certificates) could be used to sign malicious code that would be trusted by AppLocker, potentially subverting AppLocker policy.

Organizations must make an assessment as to which publishers (and all certificate authorities in the publisher's trust chain) they choose to trust. If they cannot trust a publisher or any of the CAs in the chain, they must use file path or hash rules for the application in question.

Packaged Apps

Because .appx applications and installers **must** be signed and publisher rules provide the best security and manageability, publisher rules are the only type that can be used to control .appx applications and installers.

Publisher Certificate Validation

The publisher certificate that signed the files must chain to a root CA certificate that is trusted by the computer. The publisher certificate and any intermediate CA certificates are typically attached to the files, and the root CA certificate should be installed in the trusted root store of the computer.

While not required, it is strongly recommended that files are "countersigned" as well as signed. Countersigning is the process whereby part of the signature is itself signed by a Time Stamping Authority (TSA), and embedded in the signature. This proves that the file was valid (i.e. not revoked) at the time of countersignature.

The reason why countersignatures (or "timestamps") are important is that they extend and simplify the certificate validation process. If a file is not countersigned, the file will fail validation once the publisher certificate reaches its expiry date (typically 1 year). Also, revocation information must be retrieved on a periodic basis to ensure that the publisher certificate (or any of its intermediate CA certificates) have not been revoked. This would not be possible for computers that are not regularly connected to the Internet (e.g. servers), and could cause validation to fail. The requirement to check revocation information is removed for files that have been countersigned. For more information on Authenticode certificates, timestamps and the validation process, refer to http://download.microsoft.com/download/9/c/5/9c5b2167-8017-4bae-9fde-d599bac8184a/Authenticode_PE.docx.

Object Exceptions

Path and **Publisher** rules allow you to define one or more **Exceptions** to the object definition to get finer-grained control over the files that a rule matches. (File hash rules do not need exceptions because the hash of each individual file must be included in the rule – a file is excluded by simply not listing its hash).

Examples of the use of exceptions to further refine object matches include:

- **Path rule –** Allow a subject to run any executable in %WINDIR% **except** for files that reside in any of the user-writable subfolders.

- **Publisher rule –** Allow a subject to run any signed executable with product name of "Microsoft Office 2013" **except** for files that have a file name of MSACCESS.EXE. This would enable the entire Office 2013 suite, except for Access.

The conditions used to match files in exceptions can be the same or different from the rule type. For example, the examples above could exclude specific files based on their hash value rather than path or publisher information. You can even have exceptions of all three different types (path, file hash and publisher) within a single rule if that is the most efficient way to exclude files that you don't want a rule to match.

Exceptions provide an efficient means of matching exactly the objects you wish to target with a smaller number of rules without having to resort to the use of "Deny" rules.

Summary of Rule Types

The following table summarizes the advantages and disadvantages of the three different rule types. The assessment is qualitative, and the meanings of the assessed items are:

- **Flexibility** – the ability for existing rules to accommodate routine changes such as patching or upgrading to a later version.
- **Location independent** – rules continue to be enforced regardless of where a user tries to execute it from.
- **Object assurance** – the ability to accurately match an object and prevent changes to its properties (such as file name) from avoiding a match.
- **Object integrity** – rules protect the integrity of the objects they target. For example, preventing an infected file from running where a rule allows the original file to run.
- **Ease of use** – rules are easy to create and maintain and are intuitive as to their purpose.

	Flexibility (future proof)	Location Independent	Object Assurance	Object Integrity	Ease of Use
Path	High	Low	Low	Low	High
File hash	Low	High	High	High	Low
Publisher	High (Note 1)	High	High (Note 2)	High	Medium (Note 3)

Table 3: Assessment of AppLocker rule types

Note 1: assumes that the software vendor adopts a consistent naming and numbering scheme for the files they sign.

Note 2: assumes that the publisher takes adequate steps to protect their signing certificate, and depends on the trustworthiness of all CAs in the chain that issued it.

Note 3: slightly more complex due to the number of attributes that can be used in object definitions, but this complexity gives a great deal of flexibility.

2.4.5 Default Rules

Whenever you create rules in the AppLocker user interface, you are prompted to enable the default AppLocker rules for that rule type. This prevents the user from effectively locking themselves out through creation of an overly restrictive set of AppLocker rules. Consider the following scenario. A user (who is a local administrator) forgets to change the enforcement mode to "Audit only", creates one or more rules and then closes the AppLocker console. The default enforcement mode of "Not configured" behaves as if it were configured to "Enforce rules" as soon as the first rule is created. The user finds that their ability to use the computer is severely impacted by the lack of "Allow" rules to enable required functionality. Furthermore, because they closed the Local Security Policy console and they did not create a rule to allow them to start MMC.EXE, they can't go back in to add more rules.

To avoid scenarios like this, AppLocker allows the administrator to create a set of permissive **default rules** that will allow the user to interact with the operating system without restriction.
The default rules for **Executable** rules, **Script** rules and **DLL** rules (if enabled) are:

Action	Subject	Object	Exceptions
• Allow	Everyone	%PROGRAMFILES%*	No exceptions
• Allow	Everyone	%WINDIR%*	No exceptions
• Allow	BUILTIN\Administrators	*	No exceptions

For the **Windows Installer** rule type, the default rules are:

• Allow	Everyone	Signed by any publisher	No exceptions
• Allow	Everyone	%WINDIR%\Installer*	No exceptions
• Allow	BUILTIN\Administrators	*	No exceptions

For the **Packaged apps** rule type, the default rule is:

• Allow	Everyone	Signed by any publisher	No exceptions

While allowing an inexperienced administrator to safely interact with AppLocker, it does not create a secure environment. The guidance provided in this document recommends that the default rules not be used. By following the process in this document, an administrator can safely create a highly tailored set of AppLocker policies without the need for the default rules.
To reiterate, **DO NOT ENABLE THE DEFAULT RULES.**

AppLocker Events

All AppLocker events are logged to one of the following four "Applications and Services" event logs under the path **\Microsoft\Windows\AppLocker** (the last two apply to Windows 8 and Server 2012 only).

- Microsoft-Windows-AppLocker/EXE and DLL
- Microsoft-Windows-AppLocker/MSI and Script
- Microsoft-Windows-AppLocker/Packaged app-Deployment
- Microsoft-Windows-AppLocker/Packaged app-Execution

A list of relevant AppLocker event IDs can be found in the following table.

Event ID	Level	Event message	Description
8000	Error	Application Identity Policy conversion failed. Status <*%1*>	Indicates that the policy was not applied correctly to the computer. The status message is provided for troubleshooting purposes.
8001	Information	The AppLocker policy was applied successfully to this computer.	Indicates that the AppLocker policy was successfully applied to the computer.
8002	Information	<*File name*> was allowed to run.	Specifies that the .exe or .dll file is allowed by an AppLocker rule.
8003	Warning	<File name> was allowed to run but would have been prevented from running if the AppLocker policy were enforced.	Applied only when the **Audit only** enforcement mode is enabled. Specifies that the .exe or .dll file would be blocked if the Enforce rules enforcement mode were enabled.
8004	Error	<*File name*> was prevented from running.	Access to <*file name*> is restricted by the administrator. Applied only when the **Enforce rules** enforcement mode is set either directly or indirectly through Group Policy inheritance. The .exe or .dll file cannot run.
8005	Information	<*Script or MSI*> was allowed to run.	Specifies that the script or .msi file is allowed by an AppLocker rule.
8006	Warning	<*Script or MSI*> was allowed to run but would have been prevented from running if the AppLocker policy were enforced.	Applied only when the **Audit only** enforcement mode is enabled. Specifies that the script or .msi file would be blocked if the **Enforce rules** enforcement mode were enabled.
8007	Error	<*Script or MSI*> was prevented from running.	Access to <*file name*> is restricted by the administrator. Applied only when the **Enforce rules** enforcement mode is set either directly or indirectly through Group Policy inheritance. The script or .msi file cannot run.
8008	Error	AppLocker disabled on the SKU.	AppLocker policy has been applied to a Windows edition that does not support it. Refer to section 2.2 for a list of supported editions.
8020	Information	<Packaged app> was allowed to run.	Added in Windows Server 2012 and Windows 8. Logged when an application is **executed**.

8021	Warning	<Packaged app> was allowed to run but would have been prevented from running if the AppLocker policy were enforced.	Added in Windows Server 2012 and Windows 8. Logged when an application is **executed**.
8022	Error	<Packaged app> was prevented from running.	Added in Windows Server 2012 and Windows 8. Logged when an application is **executed**.
8023	Information	<Packaged app> was allowed to run.	Added in Windows Server 2012 and Windows 8. Logged when an application is **installed**.
8024	Warning	<Packaged app> was allowed to run but would have been prevented from running if the AppLocker policy were enforced.	Added in Windows Server 2012 and Windows 8. Logged when an application is **installed**.
8025	Error	<Packaged app> was prevented from running.	Added in Windows Server 2012 and Windows 8. Logged when an application is **installed**.
8026	Error	No packaged apps can be executed while Exe rules are being enforced and no Packaged app rules have been configured.	Added in Windows Server 2012 and Windows 8.
8027	Warning	No Packaged app rule configured.	Added in Windows Server 2012 and Windows 8.

Table 4: AppLocker events

3 Process Overview for Deploying AppLocker

This section describes a process that can be used to design, deploy and maintain application whitelisting using Microsoft AppLocker. By its nature, application whitelisting is a potentially disruptive control that needs to be carefully planned and supported so that it provides the right balance between security and productivity. Implemented without due care, whitelisting may adversely impact the productivity of the business, drive up support costs and create resistance to its continued use. However if realistic control objectives are defined and whitelist maintenance is integrated into the software support lifecycle, organizations can expect a significant improvement in their security posture against malware and targeted malicious threats.

The process for deploying application whitelisting into an organization follows the Microsoft Solutions Framework (MSF) methodology. This describes a five phase process progressing from solution envisioning and requirements gathering through design, testing and into deployment. As the actual "deployment" of AppLocker is trivial and the ongoing maintenance of AppLocker is so critical to its success, the "Deploy" phase is extended to "Deploy / Operate" to address this critical aspect.

The phases are summarized as follows and are described in detail throughout the remainder of this document:

1. **Envision**
   Determine the objectives for using AppLocker in the organization and the scope governing its use. Also identify assumptions and risks that will guide the remainder of the AppLocker project.
2. **Plan**
   Perform a detailed analysis of the computing environment – including computer roles, user roles and applications to be controlled. Determine the AppLocker policy strategy and the strategy for deploying it by using Group Policy. Determine how AppLocker will be monitored after the deployment, and the processes for supporting users and maintaining policy. Finally, determine the AppLocker deployment and communications plans.
3. **Develop**
   Create AppLocker rules on reference computers for the operating system and all applications. Test and refine the rules until they are ready for formal testing, and then export the rule sets to XML.
4. **Stabilize**
   Configure centralized monitoring of AppLocker events, import AppLocker policy into GPOs, and deploy (in "Audit only" mode) to target computers. Perform detailed validation and user acceptance testing until sign off can be achieved.

5. **Deploy / Operate**

   Change AppLocker to "Enforce rules" mode to complete the deployment of AppLocker and transition into the "Operate" phase for ongoing support and maintenance.

The phases of the AppLocker design and deployment process are illustrated in the following diagram.

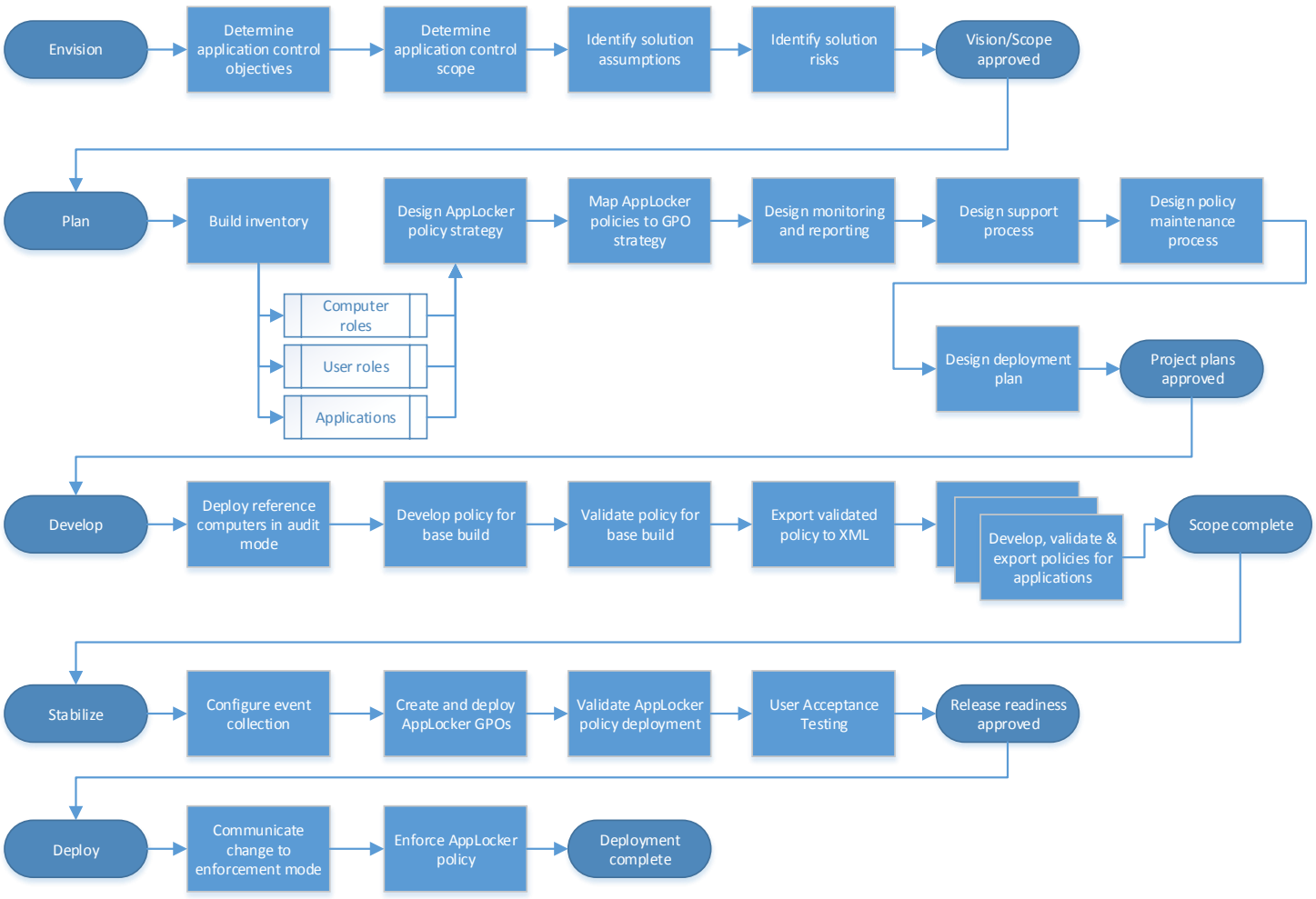


Figure 1. AppLocker Design and Deployment Process

4 Phase 1 – Envision

*"**What**, **Why**, Who, Where, When, How"*

4.1.1 Overview of the Envision Phase

The purpose of the Envision phase is to determine the goals and objectives that will drive the AppLocker design and deployment process. A clear statement of the business benefits expected will provide justification for the project, and a clearly defined scope will set the boundaries for the remainder of the project. Without clearly stated goals and scope, there will be ambiguity during the design process – resulting in the need for rework or an end result that does not meet the needs of the business.
The following list summarizes the activities in the Envision phase of the AppLocker deployment project:

- Determine the high-level objectives and motivation for deploying AppLocker.
- Determine the scope of the AppLocker deployment:
    - What are the different computer roles that will implement AppLocker?
    - What are the different user roles on these computers and what are their unique requirements?
    - Use the above to determine global and role-specific application control objectives for the organization.
- List the assumptions under which the AppLocker project will operate.
- Identify risks – both to the AppLocker deployment project as well as risks to the business's operations from deploying AppLocker.

4.1.2 Application Control Objectives

Describe the high level objectives and motivation for deploying AppLocker within the organization. These statements should be at a high level and specify the policy intent rather than attempting to go into specifics (which will be done during the Plan phase of the project).

Examples of high level control objectives may include:

- Reduce the risk of malware infection on managed clients by restricting execution to known and supported applications and utilities.
- Prevent the introduction of vulnerabilities by the installation of unsafe software or software that may not be covered by patching.
- Prevent the use of software that impacts productivity or risks exposing the company to liability from unlicensed software.
- Enforce standardization across computer roles to defined applications and specific application versions in an effort to improve supportability.
- Audit the usage of applications across the organization without enforcing control.

- Prevent administrators from installing applications where the publisher cannot be verified or is by an unapproved publisher.
- Prevent Windows 8 users from installing or running packaged apps. (Unless otherwise configured via Group Policy, standard (non-administrative) users can install packaged apps from the Microsoft Store.)
- Maintain a "blacklist" of applications or publishers than cannot run, as opposed to the default behavior of whitelisting things that can run. This is less secure than the default option, but may be desirable in specific cases.

4.1.3 Application Control Scope

Describe the intended scope of the AppLocker deployment.
It would be typical to start with high value or high risk assets first, and to progressively deploy AppLocker to lower value / risk assets over time. The scheduling of AppLocker design and deployment will be determined during the Plan phase, however describe the desired end state of the deployment and any prioritization of assets that should occur.

4.1.4 Computer Roles

Since AppLocker policies are applied to computers (as opposed to users), start by defining the distinct computer roles that will implement AppLocker, and what the control objectives are for each.
In the context of this document, a **computer role** is the set of computers that will implement a common base AppLocker policy. They should all have been built from a common standard operating environment (SOE) image, and would typically have the same additional applications installed. It is however possible that some computers within a role may have more or less optional applications installed compared to their peers. This does not require the creation of a new computer role (as the number of combinations could result in a very high number of roles), but it does require that the **resultant** AppLocker policy contain a superset of rules covering all possible applications. This is described in detail in the Plan phase.
As computer roles are driven by the applications they are expected to run, they typically align to a business unit or computing function. Examples may include "Finance Department Clients", "Public Kiosk Computers", and "HR Department Clients". While the focus of this document is on client computers, it could equally include "Domain Controllers" or "DMZ Web Servers" if AppLocker is used to control server roles.

AppLocker requires a direct tradeoff between security and flexibility / cost to operate. Determine what this balance should be for each computer role. Doing this early will provide guidance to the design team when considering design alternatives. Consider a qualitative scale such as Prioritize Security / Balanced / Prioritize Flexibility, or a more granular scale if appropriate.

Consider which control objectives apply across all AppLocker-protected computer roles, and which are specific to a computer role. If you're considering deploying AppLocker on servers, decide which server roles will be targeted first and how those server roles are distributed within your infrastructure.

When considering the control objectives (for each computer profile or to be applied globally), consider each of the rule categories that AppLocker can control – as you may not wish to control all categories on all computer roles. Refer to the considerations for each rule type in section 2.3.1 for more information.

4.1.5 User Roles

Consider different types of users on each identified computer role, since AppLocker rules can grant different access to different users. For example, you may wish to only allow Finance users to run Finance-specific applications in cases where a computer may be shared by different roles.

You may have task-specific administrative roles that logon to a computer profile and wish to limit their access, yet allow more flexibility to the highest tier of administrators to provide them the flexibility to troubleshoot issues.

4.1.6 Global and Role-specific Application Control Objectives

Examples of global and role-specific application control objectives may include:

- [Global] Prevent administrators from installing applications where the publisher cannot be verified (i.e. installers that are unsigned).
- [Global] Allow any user to run any installed packaged app, but do not allow them to access the Windows Store.
- [Finance] Prevent Finance computers from running any application that is not installed as part of their standard operating environment (SOE). Additionally, only allow Finance users to run Finance-specific applications.
- [Kiosk] Prevent Kiosk computers from running any application other than those required for computer start up, management and monitoring, and the Kiosk application.

4.1.7 Assumptions

Capture the assumptions that are in place for the use of AppLocker in the organization. The process described in this guide has the following assumptions, however these may be altered or new ones added to suit the needs of the organization:

- There is a mandate to control application usage and assets are managed according to organizational policy. If users have an expectation of being able to download and run anything they choose, controlling application usage may cause friction.
- The organization has relatively few standard images (or "builds") that all client and server roles are based on. If computers are built on an ad-hoc basis and don't follow a standard deployment pattern, it will be very difficult to define and test a manageable set of AppLocker policies that will function as expected.
- The organization is able to define a list of known applications that are authorized to run. If this list cannot be determined or agreed, it is likely that user productivity will be impacted (as the unknown applications will be prevented from running).
- AppLocker can only be used on the following operating system platforms:
    - Windows 7 - Ultimate and Enterprise editions
    - Windows 8 - Enterprise edition
    - Windows Server 2008 R2 - Standard, Enterprise, Datacenter and Itanium editions
    - Windows Server 2012 - Standard and Datacenter editions
- While software restriction policies (SRP) that were introduced with Windows XP can be used in conjunction with AppLocker on modern Windows platforms, SRP will not be used. While performing a similar role, AppLocker is vastly superior and will be used exclusively.
- The organization has resources to design, deploy and then support AppLocker (i.e. there is a cost associated with the increased level of control).
- Users that have local Admin rights will be able to circumvent AppLocker policy if they so choose. This would be done by adding permissive rules in the local security policy of a computer that would allow them to run additional content. This does however require deliberate actions on the part of the user to circumvent AppLocker control, and will be recorded in the event log of the computer (which can be centralized and audited).
- AppLocker provides the appropriate restrictions on types of executables, installers, scripts and libraries, noting the limitations described in section 2.3.2.

4.1.8 Risks

Define the risks and mitigations of the AppLocker deployment. Include both risks to the AppLocker project's success, as well as risks posed by AppLocker to the operation of the organization's systems and processes.
Examples of **project** risks may include:

- Lack of definition or consensus on computer roles and the applications they require make translation into AppLocker policies difficult.
- Inadequate resources to design, test and deploy AppLocker.
- Inadequate resources to extend the deployment of AppLocker beyond the initial target computers.

Examples of **operational** risks may include:

- Business units are used to a high degree of autonomy in the applications they run, and centralized control will create resistance.
- User productivity will be impacted if application requirements are not adequately defined or AppLocker policies adequately tested.
- Inadequate resources and operational processes to maintain the AppLocker policies as new applications and versions are introduced.

5 Phase 2 – Plan

*"What, Why, **Who, Where, When**, How"*

5.1 Overview of the Plan Phase

The Plan phase is where the high level control objectives are transformed into specific control actions that will be implemented within the constraints imposed by the environment (e.g. the OU and security group structures).
The following list summarizes the activities in the Plan phase of the AppLocker deployment engagement:

- Build an inventory of all computer roles that will implement AppLocker.
- Build an inventory of all user roles that will use those computers.
- Build an inventory of all applications that will be used across all computer roles identified above.
- Align AppLocker policy deployment with the software deployment strategy used by the organization.
    - Design the AppLocker policy strategy.
    - Design the AppLocker policy deployment strategy.
- Design the ongoing monitoring and reporting strategy.
- Design the AppLocker support process.
- Design the AppLocker policy maintenance process.
- Design the AppLocker deployment and communication plans.
- Plan for deployment of AppLocker hotfix

5.2 Build Inventory of Computer Roles

Create a spreadsheet of all computer roles that will implement AppLocker rules and include the following columns for each. Computer roles were described in section 4.1.4:

- Computer role name.
- Computer role description / purpose.
- Control objective(s) for this computer role (from section 4.2).
- File type(s) to be controlled (executables, windows installers, scripts, packaged apps, shared libraries/controls).
- (Optional) Weighting of security vs. flexibility / cost (e.g. Prioritize Security / Balanced / Prioritize Flexibility). This weighting can be used during the Develop phase to assist in tradeoff decisions between the competing priorities.
- Operating system.
- Base build – list the SOE version and any special configurations that define this role.
- Computer role owner – this is useful to enable follow up to clarify role details.

- How computers in this role can be identified for the purpose of targeting GPOs to them – e.g. they all reside in a specific OU, they all belong to a security group, they have a unique naming convention, they share some unique attribute of their AD computer accounts.

5.3 Build Inventory of User Roles

For each of the computer roles identified above, list the user roles that will use each either interactively or via a remote desktop connection. Consider not only end-user roles (e.g. Finance Users) but also consider the different administrative groups that will logon to each role (e.g. Helpdesk Users, Level 2 Support Users).
Add the following columns to the computer role spreadsheet to capture the following information:

- User role name.
- Security group that maps to this role.
- Comments on what they should / should not be able to do on computers in that role.

5.4 Build Inventory of Applications

Build an inventory of applications that can be used across all of the computers that will implement AppLocker policy. At this early stage, you don't need to analyze the binaries or application components – just get a complete listing of all applications. Collect the following information for each application:

- Application name.
- Application type (traditional or packaged app (Windows 8 / Server 2012 only).
- Install type – part of base build, or deployed post-build.
- Publisher (if signed).
- Version (or versions) supported.
- Application owner – this is useful to enable follow up to clarify application usage or configuration.
- The computer roles (from 5.2 above) that the applications will be allowed to run on.
- The user roles (from 5.3 above) that will be allowed to run the applications (if not Everyone).

Use the information in the following subsections for guidance on collecting the application inventory.

5.4.1 Inventory Installed Applications

An important task in the Planning phase is to identify all applications that are installed in the base build of the SOE. While we will use AppLocker PowerShell cmdlets to enumerate exactly what executable content exists on the computers at a later stage, time can be saved by listing installed applications and deciding whether (and who) should be allowed to run them. This method is also useful for running against a sample of machines, as what is actually installed is often different from what is expected to be installed.
Any inventory tool can be used to collect the list of installed applications on the reference computers. Microsoft have a number of products that collect inventory information, and if these are in use, the information will be readily available:

- System Center Configuration Manager – software inventory feature
- Asset Inventory Services (AIS) that is part of the Microsoft Desktop Optimization Pack (MDOP) http://technet.microsoft.com/en-us/windows/hh826069
- Windows Intune http://www.microsoft.com/en-us/windows/windowsintune/pc-management.aspx

If none of these or third-party inventory tools are in use, use PowerShell to enumerate this information. (Many of the PowerShell commands in this guide require that the PowerShell console has been elevated to "Run as administrator". This should be used for all examples in this guide.) For example, you can use the following command (all on a single line) to enumerate all installed applications on a reference computer and output it to a file named InstalledApps.csv (which can be opened directly in Excel):

```
Get-ChildItem "HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall" | ForEach-Object { Get-ItemProperty $_.PSPath } | Select-Object DisplayName, DisplayVersion, Publisher, InstallLocation | Sort-Object -Property DisplayName | Export-Csv InstalledApps.csv –NoTypeInformation
```

This will display 32-bit applications on 32-bit versions of Windows and 64-bit applications on 64-bit versions of Windows.

To display 32-bit applications installed on 64-bit versions of Windows, substitute the following into the registry path above:

```
HKLM:\SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Uninstall
```

These commands can be run interactively on each machine, or you can use PowerShell remoting to aggregate this information from multiple computers. In order to do this, the Windows Remote Management service must be running on each target computer, with a listener configured and a Windows Firewall exception created. This can be done via Group Policy, or can be done interactively by running **winrm quickconfig** on each target computer that you will query. The following command is an example of how to collect this information from two remote clients PC01 and PC02:

```
Invoke-Command PC01, PC02 –Command { Get-ChildItem "HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall" | ForEach-Object { Get-ItemProperty $_.PSPath } | Select-Object DisplayName, DisplayVersion, Publisher, InstallLocation } | Sort-Object -Property DisplayName –Unique | Export-Csv InstalledApps.csv –NoTypeInformation
```

5.4.2 Inventory Optional Applications

There are typically more applications used by an organization than may be deployed on the reference computers. If inventory data is used to compile the list of applications, any applications that are available for installation but have not been installed will be missed. This information can be found in the tool that deploys software across the organization (e.g. System Center Configuration Manager), or may be compiled from other sources such as documentation.

5.4.3 Code Signing of Custom Applications and Installers

Publisher rules offer a very high degree of security without sacrificing operational flexibility. For most organizations, they should be the default choice for controlling all signed code. While the organization cannot control the signing behavior of application vendors, they have full control over the signing of internally produced code. Organizations that are considering to deploy AppLocker and who develop their own applications (or build their own MSI installer packages) should strongly consider developing the ability to sign their code.
While code signing is beyond the scope of this engagement, the following requirements and considerations should be discussed:

- At least one code signing certificate must be obtained, and ideally stored on a smart card to prevent exposure of the private key. This certificate can be purchased from a commercial Certificate Authority, or obtained from an internally-owned PKI (e.g. built from the Active Directory Certificate Services (AD CS) role that is a free component of Windows Server). In practice, additional code signing certificates for pre-production code are often used for code before it is ready for release.
- Signed files should be time stamped by a Time Stamping Authority (TSA) – a process referred to as countersigning. If this is not done, signature validation will fail when the code signing certificate expires, or if revocation information by the CA that issued it cannot be retrieved (relevant to computers that don't regularly access the Internet).
- The root CA certificate(s) that the code signing certificate and time stamping certificates chain to must be trusted by all computers that will implement AppLocker and will run code signed and stamped by these certificates.
- The code signing certificate, time stamping certificate, Issuing CA certificates and any intermediate CA certificates must be attached to files, installed or downloadable by the clients.

- Revocation information must be present on the client or downloadable for all certificates (except for the root CA certificate(s).

Refer to the document Code-signing Best Practices for more information.
If the organization has the capability to sign and countersign code that they produce, the following guidelines should be adhered to in order to maximize their efficient use in AppLocker rules:

- The "Publisher" is defined by the code signing certificate. At its simplest, a single AppLocker Publisher rule allowing any code signed by that publisher can allow all in-house signed applications, installers and scripts to run.
- If a signed file's "Product name" or "File name" attribute is used in an AppLocker rule (in addition to "Publisher"), ensure that subsequent revisions of these files use exactly the same names. This will prevent the need to edit Publisher rules when newer versions are deployed.
- If a signed file's "File version" attribute is used in an AppLocker rule (in addition to "Publisher", "Product name" and "File name"), ensure that subsequent revisions of these files use incrementing numbers. Publisher rules allow you to specify ≥ a given "File version", and by incrementing the version number, prevent the need to edit Publisher rules when newer versions are deployed.

5.5 Align AppLocker Policy with Software Deployment Strategy

Organizations typically adopt a single (or small number of) standard operating environments (SOEs) in order to achieve a degree of standardization. The SOE will consist of a specific OS version and service pack level, with a specific configuration. It will have all of the necessary agents and applications for management and monitoring of the SOE, as well as security software like anti-malware. It will also often include supporting applications (such as compression utilities and PDF readers), and may include productivity applications (such as Microsoft Office) if it is required on all client computer roles. The configuration just described is commonly referred to as the "base build", and is deployed as a single unit. Applications are then layered on top of the base build to create computer roles specific to a task or business unit. This includes task or business unit-specific applications, but it also may include newer versions of applications present in the base build. For example, a Windows 7 SOE may have been initially deployed with Internet Explorer 8, but Internet Explorer 10 is pushed out as an upgrade package.

AppLocker policy should generally align with the software deployment methodology – using a single policy to enable applications in the base build, and then individual policies specific to each deployment package. AppLocker rules from the base policy and the application-specific policies will be merged to create an effective policy suitable for the software that is installed on any given machine.

Note that there may be multiple versions of the base or application-specific policies to suit the requirements of different computer roles. For example, some computer roles may permit Internet Explorer while others do not. Rather than using a single base policy (which allows Internet Explorer) and then layering another policy with "Deny" rules on the computers that shouldn't run it, a copy of the base policy should be taken and the "Allow" rules for Internet Explorer removed. As long as a suitable naming convention is adopted and the differences properly documented, this will lead to a simpler deployment to manage in the long term. (Alternatively, Internet Explorer may be removed from the base rules and treated as an individual application – allowing a common set of base rules to suit both roles).

Similarly there may be a need for multiple application-specific policies. Consider a suite of products like Microsoft Office. There may be no need (or no license) to run Access on some computers, while it is required on others. In this case, two AppLocker policies should be created, with only one allowing Access to run.

5.5.1 Design AppLocker Policy Strategy

Using the inventory of computer roles, user roles and applications generated from the previous sections, start designing the AppLocker policy strategy. This involves defining how many policies are required and what they will contain, and creating a suitable naming convention to avoid ambiguity.

Monolithic vs. Multiple, Layered Policies
The approach for dividing rules between AppLocker policies typically uses one of two different methodologies.

"Monolithic" Policies
With this methodology, the organization attempts to add all rules for all possible applications into a small number of "monolithic" AppLocker policies (typically, one per computer role or one per business unit).
Advantages:

- Reduces the number of policies that must be deployed and managed.

Disadvantages:

- A change in a rule for one business unit could have unintended consequences for another business unit that uses the same computer role but has different application needs. This impacts the flexibility of the policy to adapt to changing requirements.
- The number of rules and resulting size of the GPO can get quite large.
- The number of rules applied on each computer (a lot of which are for software that is not installed) could get very large, and may impact application launch times.
- You may need to support multiple versions of applications in the same policy, which could get complex.

Multiple, Layered Policies

At the opposite end of the spectrum, rules for every application are contained in their own AppLocker policies. A method for deploying the right policies to the right computers (based on the software installed) must be devised, and these policies will merge on the target computers to create their "effective policy" (which can be enumerated using PowerShell cmdlets).

The advantages and disadvantages are essentially reversed compared to monolithic policies.
Advantages:

- Flexibility to accurately target specific applications (or application versions) without impacting other applications.
- Exceptions are much easier to manage. If two business units use the same application but have different constraints, it is much easier to maintain two separate policies than try to meet all requirements in a single rule set.
- Easy to create new policies for new applications and retire existing policies as applications are retired.
- Policies are small, highly targeted and therefore easy to comprehend.
- The effective policy could be much smaller, as only rules for installed software are applied.

Disadvantages:

- Results in a large number of AppLocker policies to manage.
- Creates the need to synchronize software and AppLocker rule deployments. If an application gets deployed without the corresponding AppLocker rules being applied, the newly installed application won't run.
- Creates the need to target the right policies at the right computers. This targeting can be done via the OU hierarchy and can be further filtered by security group and/or WMI filters. (You could potentially use a WMI filter to detect an application and therefore apply the corresponding AppLocker policy, however you would have to wait for GPO processing to complete before a newly installed application could be launched).

Recommended Approach

While the above discussion describes the two opposing approaches in designing AppLocker policies, the optimal strategy is usually somewhere in between (typically favoring multiple policies).
The monolithic approach works well when you combine multiple components that are deployed and updated as a single unit into a single policy. An example of this would be the base operating system, management tools and utilities. Components that are likely to be upgraded without a full update to the operating environment are candidates for moving into separate policies, so that the necessary changes to AppLocker rules to accommodate this don't impact the base build. Another example of combining components into a single policy

is with application suites (such as Microsoft Office) and applications that are effectively treated as suites due to dependencies between them.
The complexity introduced by having multiple AppLocker policies can be lessened by the fact that multiple AppLocker policies can be merged into a single GPO – leaving less units of policy that must be targeted at the different computer roles. This will be addressed in the next topic.
Create an initial list of AppLocker policies that will be created – ideally one for each base build plus one for each application (and possibly version) or set of related applications. You may need to create multiples of these if different business units have conflicting requirements that cannot be reconciled within a single policy (e.g. by using different security groups in the rules).
Devise a naming convention for the AppLocker policies that makes it easy to identify its purpose and target business unit (if required). AppLocker policies also allow descriptions to be attached, and these should be used to further describe their purpose, contact details of the policy owner etc.

5.5.2 Design AppLocker Policy Deployment Method

Map AppLocker Policies to GPOs
After AppLocker policies have been defined, the next step is to map these to GPOs and determine how they will be delivered to the appropriate computers. While AppLocker rules can be present in the same GPO that delivers non-AppLocker settings to a computer, a common practice is to maintain separate GPOs for each function, so dedicated GPOs would be created to deliver AppLocker rules.
For larger AppLocker policies (such as a base build policy), it usually makes sense to import a single AppLocker policy into a GPO.
For smaller, application-specific AppLocker policies (which are typically far more numerous), it often makes sense to merge multiple AppLocker policies into a single GPO. This is true when the computers that will receive these GPOs have all of the applications installed that are addressed by the AppLocker policies. The limiting factor is the inherent loss of flexibility that this entails. Consider the following when deciding to merge multiple AppLocker policies into a single GPO:

- Do all computers that receive the GPO have the same requirements for each application? If you can't express an application's requirements in a single policy with the aid of different rules for different subjects, the conflicting AppLocker policies for the affected application should be imported into separate GPOs.
- Will the ability to manage AppLocker rules be delegated to the application owner? The ability to delegate management of GPO settings can only be done at the GPO level, not the rule level within a GPO.
- Are new applications frequently deployed, or existing applications frequently updated or retired? If so, there is a case for keeping them separate to reduce the amount of GPO downloading and processing. This also reduces the potential impact of an incorrect change impacting multiple computers or applications.

Target AppLocker GPOs

When the GPOs that will contain AppLocker policy have been determined, the next step is to determine how they will be targeted to the required computers.

Group Policy processing is done in the following order. AppLocker rules from multiple GPOs (and local policy) will merge to create an "effective policy" that is implemented on the computer. The AppLocker enforcement mode for a particular rule type is set by the last (highest precedence) policy:

- **Local machine policy** – this should only be used to develop the AppLocker rules which are subsequently imported into Group Policy (see the Develop phase below). Since by default AppLocker blocks everything except that which is explicitly allowed, adding rules at this level can undermine the policy intent, as the effective policy will be more permissive than the Group Policies alone would allow.[10]
- **Site** – it would be unusual to apply AppLocker policy based on a computer's physical location.
- **Domain** – AppLocker policy applied at the domain level will affect all computers joined to the domain (including domain controllers) unless filtered by security group or WMI query or the OU containing computers is blocking inheritance.
- **OU hierarchy** – policies can be applied at multiple levels of an OU hierarchy – with the policies linked to the OU containing (or closest to) the computers being applied last (and therefore having the highest precedence). Multiple GPOs can be linked to a single OU, with a separate order of precedence determining the order of application. This is the most common method for applying AppLocker policy.

With the GPO processing hierarchy in mind, determine the appropriate locations for linking the GPOs that were defined in the previous step. Use the following guidelines:

- Link to OUs at a sufficiently high level so that inheritance flows the rules down to computers that need them, but not so high that you need to filter GPOs to prevent them from applying to computers that shouldn't receive them.
- If the OU structure doesn't promote the use of inheritance to flow rules to the computers that need them without affecting computers that don't, link GPOs to the OUs containing the desired computers. A single GPO can be linked to as many OUs as is required.
- If an OU contains computers that don't have the same policy requirements, you will need to use security group and / or WMI filtering. This should be used sparingly as it adds additional complexity to the comprehension of policy inheritance. It also requires that computers are added / removed from the appropriate groups (in the

[10] High value computers or computers operating in high risk environments may get local AppLocker policy applied during deployment to provide immediate protection before AppLocker rules from Group Policy are applied. This should be balanced with the fact that local policy cannot be managed centrally, and may become "stale" during the lifetime of the computer.

case of security group filtering), or have attributes that will return the correct WMI query results (in the case of WMI query filtering).

Dedicated GPO for AppLocker Enforcement

Every GPO that contains AppLocker policy also has an enforcement mode setting for each of the AppLocker rule types. For each type, it can be "Not configured", "Enforce rules" or "Audit only". Rather than trying to determine which is the highest precedence GPO affecting any given computer and defining the enforcement mode there (bearing in mind that the highest precedence GPO affecting one computer might not be the highest precedence GPO affecting another), a good practice is to create a dedicated AppLocker enforcement mode GPO. This GPO is linked to the OU (or OUs) containing all computers within a given computer role, and is given the highest precedence. It should only configure the enforcement modes – not define any rules. This ensures that computers will always adopt the desired enforcement mode – regardless of how any other GPOs are configured.
There are two ways this could be achieved:

- Create two GPOs – one that puts all rule types into "Enforce rules" mode and the other putting all rule types in "Audit only" mode. Link one or other of these GPOs to the OU containing the target computers to enforce the corresponding mode. The benefit of this approach is that only two GPOs are required – regardless of the number of OUs containing computers.
- Create an AppLocker enforcement mode GPO for each child OU containing computers that will implement AppLocker. Use a naming convention for the GPOs to indicate which OU it will link to. The benefit of this approach is that the rule types can be set independently (e.g. put Exe rules into "Enforce rules" mode while leaving the other three rule types in "Audit only" mode).

There are a number of additional configuration items that are related to AppLocker that could potentially be configured in this GPO:

- Configure the **Application Identity** service with a startup type of "Automatic". This is required for AppLocker.
- Configure the **Windows Remote Management (WS-Management)** service with a startup type of "Automatic". This is required to forward events to a remote event collector.
- Configure the URL of the remote event collector as described in section 7.2.2 below.
- Configure the custom error URL as described in section 5.7 below.

5.6 Design the Ongoing Monitoring and Reporting Strategy

An important component of an AppLocker deployment is the ongoing monitoring and reporting of AppLocker related events. This includes deciding what events to collect, how to collect them, how to use them, and how to store or archive them. AppLocker event information is typically used for one or both of the following reasons:

- Proactive detection of legitimate application use cases being blocked. These can be triaged to determine whether changes need to be made to the AppLocker rules.
- As a detective control to identify unauthorized or suspicious activity on the monitored computers.

5.6.1 AppLocker Events to Collect

Section 2.5 contains a list of the four "Applications and Services Logs" that AppLocker events are written to. Organizations should choose one or more of these logs to monitor based on which AppLocker rule types they are using.
Section 2.5 also contains a table of important AppLocker events. Of most interest for collection are events relating to policy change, failure to apply policy, and prevention of an application, script or installer from running. Some organizations may also wish to collect events indicating successful execution of a file for detailed audit tracking purposes, while others will choose not to collect these events to reduce network traffic and storage requirements.

5.6.2 Collecting and Storing AppLocker Events

The method for collecting and storing AppLocker events depends on the technology already in use for collecting and monitoring Windows event logs on the target computers. Enterprise monitoring solutions such as System Center Operations Manager (SCOM) are ideal and can be easily configured to collect and report on the required events, however most organizations use such tools for server monitoring only.
If nothing is currently used to monitor Windows event logs on the target computers, consider leveraging the Windows Event Forwarding and Collection architecture that is built into Windows 7 / Server 2008 R2 (and later) computers.

Windows Event Forwarding and Collection Architecture
The Windows Event Forwarding and Collection architecture is built using Windows Remote Management (WinRM) which leverages the WS-Management implementation that is built into Windows (i.e. there are no agent or server components to install). It consists of the following components:

- **Event source** – a Windows computer that forwards all events that are defined by Subscriptions (see below) to the defined event collector. If a computer cannot connect to its event collector, events will continue to be stored in its local event logs. Once connectivity is restored, all of these stored events will be forwarded – so none

are lost (unless the event log wraps due to insufficient size). The rate of forwarding is configurable to balance latency with bandwidth use.

- **Event collector** – a Windows computer (Windows Vista SP1 / Server 2003 R2 or later) that receives and stores events forwarded from remote event source computers. Connectivity is via HTTP or HTTPS, and event sources use SSL or Kerberos to authenticate. Collectors can in turn forward to other collectors – creating a highly-scalable multi-tier architecture.
- **Subscriptions** – define the events that should be forwarded and from which event logs.
  Subscriptions can be either:
  - **Source-initiated** – the subscriptions are created on the collector and are downloaded by all event source computers that are configured to contact the collector (by Group Policy). This is the preferred method.
  - **Collector-initiated** – the collector is configured with a list of event source computers which it uses to connect to each in turn to deliver the subscriptions.

Refer to section 7.2 for steps to configure computers to forward AppLocker events to a central event collector.

5.6.3 Monitoring and Reporting on AppLocker Events

The steps in section 7.2.1 below describe how to create a subscription to forward AppLocker events of interest from computers running AppLocker to a central event collector. Events will be collected in the **ForwardedEvents** event log on the event collector.
The next task is to design a reporting system that can consume and render these events into appropriate reports to allow ongoing monitoring and management of the AppLocker deployment. A typical approach would be to host SQL Server on the event collector server, and import all events from the "ForwardedEvents" event log into SQL tables. SQL Reporting Services could then be used to author and run reports against the SQL database using an easy to use web interface. This design of this solution is out of scope for this initial version of AppLocker design guidance.
To allow administrators to get quick access to AppLocker event data before designing a proper reporting solution, the following sample script has been provided. Use this script as is, or as a starting point to developing your own scripts to report on events of interest.
The script consists of the following parts:

1. A function called **Get-AppLockerEvent** receives objects piped from Get-WinEvent and (optionally) the name and path to save results to (OutPath). If the OutPath parameter is supplied, the results are written as a CSV file to the location provided. If the OutPath parameter is not supplied, custom objects containing AppLocker event details as properties are written to the pipeline for display or further manipulation. The code for this function is available in the Appendix.

2. A variable called **$filter** will be used (by Get-WinEvent) to return only those events of interest. The filter uses XPath syntax, and the example below requires events to meet the following criteria:
    - One of the four event IDs that is generated when AppLocker allows a file to run that would have been blocked in "Enforce rules" mode (see the table in section 2.5 for the full list of event IDs).
    - The number of milliseconds back from the current time to return events for. In this example, return events from the last 86400000 milliseconds (24 hours).
    - The name of the computer that generated the events.

    Modify the filter XPath string to return the events of interest – adding or removing conditions as required. At its simplest, setting it to "*" will return all events.
3. Run Get-WinEvent against the **ForwardedEvents** log using the filter defined in point 2 above. Events matching that filter are piped into the Get-AppLockerEvent function. If a file name has been passed as a parameter to the Get-AppLockerEvent function (as in the example below), the results will be written as comma-separated values to the specified file. The CSV file can then be imported into Excel (specifying comma as the delimiter) for viewing and reporting purposes.
   If a parameter is not passed to the Get-AppLockerEvent function, custom objects corresponding to each event that contain all of the AppLocker-specific details as properties are output to the pipeline. This allows them to be displayed, or piped to other cmdlets for further manipulation (filtering, sorting etc.) As all of the relevant details are in the objects, they could also be used to populate an AppLockerPolicy object which could be piped to the Set-AppLockerPolicy cmdlet to create additional rules.

```
$filter = "*[System[(EventID=8003 or EventID=8006 or EventID=8021 or EventID=8024) and
TimeCreated[timediff(@SystemTime) <= 86400000] and (Computer='win8-1.contoso.com')]]"

Get-WinEvent -LogName ForwardedEvents -FilterXPath $filter | Get-AppLockerEvent ".\Out.csv"
```

5.7 Design the AppLocker Support Process

When AppLocker is deployed in "Enforce rules" mode, there will be cases where users are prevented from running legitimate applications. These cases will arise for a number of different reasons, and each may require a different process to resolve:

- Inadequate testing coverage of use cases results in legitimate actions being blocked. This includes the case where errors exist in AppLocker rules that were not identified in testing.
- Appropriate policies are not being applied to the appropriate computers, so the required rules are not present in the computer's effective policy. This may be because the computer is not within scope of the appropriate GPOs (or inheritance is being blocked), the computer does not belong to an appropriate security group, or does not possess the correct attributes for a WMI filter.
- A user does not belong to a particular group that is referenced in an AppLocker rule.
- New software or software updates are deployed to a computer without corresponding rules being deployed.
- Users want to be able to install or run something for legitimate business reasons, but it has not been captured in the application inventory (so is not enabled by any AppLocker policy).

The above list is not exhaustive, but covers the majority of cases. For each case, an organization should develop a process for addressing the issue that aligns with their existing endpoint support processes. This is beyond the scope of this document.

In cases where users wish to request an exception to the current AppLocker policy, an appropriate approval process must also be created (or an existing one leveraged). Administrators have the ability to customize the error a user receives when their action is blocked by AppLocker with a URL to a custom support page. This is configured through Group Policy via the **Set a support web page link** setting under **Policies | Administrative Templates | Windows Components | Windows Explorer**. The web page might host details of the organizations policies or instructions on how to request an exception. It might also contain a form for users to submit this information directly into a ticketing system or approval workflow.

The organization might also consider creation of an emergency override process to disable AppLocker in the event that serious errors in the policies are impacting the organizations operations. This is another motivating factor for the dedicated GPO for AppLocker enforcement mode described in section 5.5.2.3. If AppLocker problems cannot be rectified in a suitable time frame and operations are being impacted, that GPO can be changed from "Enforce rules" to "Audit only" until the problems can be fully resolved. As this could have a significant impact on the security posture of the affected computers, it is essential that the ICT security team be involved in the development of the emergency override process, and in the decision / reporting chain before the process is initiated.

5.8 Design the AppLocker Policy Maintenance Process

An organizations application requirements change over time, as do the management and monitoring requirements of the computers hosting those applications. AppLocker policy needs to evolve in step with these changes to provide continuing protection without blocking legitimate use cases.
The following list provides examples of scenarios that may require changes to AppLocker policies and the GPOs that deploy them to the target computers:

- Correction of policies that do not meet existing application requirements.
- Deployment of new applications.
- Patching of existing applications – always an issue if using Hash rules, however Path and Publisher rules often continue to work without modification.
- Upgrading the version of existing applications – as above.
- Retiring applications or specific versions of an application.

The organization needs to define a process for maintaining AppLocker policies over time, including answers to the following questions:

- Where will the requirements for change come from? Proactive (via analysis of centrally-collected AppLocker events or through consultation with business units) and / or Reactive (support cases from impacted users).
- What is the approval process for AppLocker policy changes?
- Which group is responsible for developing the AppLocker policy change?
- Which group is responsible for testing the AppLocker policy change?
- How does AppLocker policy fit into the regular and emergency change management processes?
- How will changes to policy be documented, tracked and communicated?
- Which GPOs will host the AppLocker policy changes, who owns them and who will import the changes?

5.9 Determine the AppLocker Deployment Plan

Determine the priority order in which AppLocker policy will be designed and deployed, and how long it will be piloted (in "Audit only" mode) before it is switched to "Enforce rules" mode.

Priority order is typically driven by the organizations need to demonstrate compliance or manage risk of a particular set of computers (such as executives or user roles required to respond to unsolicited email). Roles requiring more autonomy and flexibility (such as developers) may be deferred until later in the deployment cycle when the organization is proficient at developing and validating AppLocker policy. Depending on factors such as the urgency to reach the desired end state, the amount of resources that can be dedicated to the project, and the size and complexity of the organization, a single computer profile or multiple profiles can be developed in parallel. A carefully staged pilot and rollout sequence can also be instrumental in managing organizational resistance to this new security control. Users and administrators will be more accepting when they see it deployed successfully to other parts of the business. This will drive the definition of the rollout sequence and allocation of resources for the remainder of the project.

Given the potentially disruptive nature of application whitelisting as a security control, the ability to run in "Audit only" mode is critical to the successful adoption of this technology. All computer profiles that will be part of the AppLocker rollout will go through a Stabilization phase (described in section 7). In this phase, they will operate in "Audit only" mode for as long as it takes for the appropriate authority to be satisfied that the implementation has been sufficiently tested, and the support and maintenance processes are in place to address issues as they arise.

5.9.1 Communication Plan

An effective communication plan is critical to the success of an AppLocker deployment. Communication should be a two-way process to understand any reservations held by the business unit, and to understand any critical periods (such as end of period processing) where changes should not be introduced.

Users should be informed about the nature of the change, the reasons it is being introduced, and the timing of the change. Users should be directed to intranet sites containing details of what will and won't be blocked, as well as the support and exception management processes. Note that this URL should also be delivered by Group Policy and will be presented to the user every time an action is blocked by AppLocker.

Determine what will be communicated to users, and the frequency of communication in the lead up to AppLocker being enforced on their computers.

5.10 Plan for Deployment of AppLocker Hotfixes

5.10.1 KB2532445

Applies to Windows 7 and Windows Server 2008 R2 (RTM and service pack 1). The fix is included in Windows 8 / Server 2012.
The hotfix described at http://support.microsoft.com/kb/2532445 removes the potential for scripting features (such as Office macros) to tamper with process memory and circumvent AppLocker rules. Specifically, a number of functions[11] have flags that allow code to load irrespective of any AppLocker restrictions. This hotfix causes these flags to be ignored unless the process is running under the context of "Local System" or "TrustedInstaller".

5.10.2 KB977542

Applies to Windows 7 and Windows Server 2008 R2 (RTM only – the fix was included in service pack 1). The fix is included in Windows 8 / Server 2012.
The hotfix described at http://support.microsoft.com/kb/977542 removes the potential for non-administrative users to boot the computer into safe mode to circumvent AppLocker restrictions.

[11] For example, the LOAD_IGNORE_CODE_AUTHZ_LEVEL flag for LoadLibraryEx (http://msdn.microsoft.com/en-us/library/windows/desktop/ms684179.aspx)

6 Phase 3 – Develop

*"What, Why, Who, Where, When, **How**"*

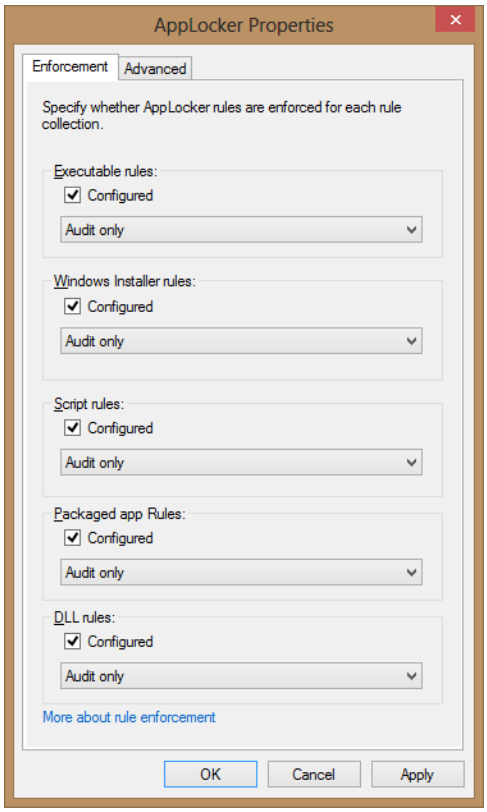
6.1 Overview of the Develop Phase

The Develop phase is where the detailed requirements from the Plan phase will be expressed as AppLocker rules and tested for validation. The following list summarizes the process, which will be described in detail in this section:

1. Deploy a base build onto a reference computer that will be used for authoring of AppLocker rules. If multiple base builds are used for different purposes (or different operating systems), deploy one of each. The reference computers can be physical or virtual.
2. Configure the "Application Identity" service to start automatically, and start the service.
3. Put AppLocker into "Audit only" mode so that the rules created don't actually block execution.
4. Auto-generate AppLocker rules against the base build for each of the file categories that will be used, and manually edit them to meet exact requirements.
5. Test all end-user and administrative usage cases, and review audit entries in the Event Log. Tune rules as required, and continue testing until there are no audit entries indicating that a desired action would be blocked.
6. Install applications, and repeat steps 4 and 5 for each.
7. Export AppLocker policies into individual XML files for later import into Group Policy.

6.2 Configure Reference Computers for AppLocker "Audit only" Mode

Open the **Services** MMC console and change the **Application Identity** service to start automatically. Start the service.
Run **secpol.msc** to open the Local Security Policy MMC console and expand **Application Control Policies**. Right-click **AppLocker** to edit its properties, and set the enforcement mode for each of the categories that will be used to **Configured** and **Audit only.** The following graphic illustrates the Properties of the AppLocker node in the Local Security Policy where the enforcement mode is configured for each rule type. Note that the DLL rule type is only displayed if enabled in the **Advanced** tab.



6.3 Create AppLocker Rules for Base Build

The next step of the Develop phase is to use AppLocker to auto-generate rules on the base build – that is the image that is deployed to new computers before additional applications are layered on top. This step should be repeated for each base build maintained by the organization. Once the rules have been automatically generated, they can be manually customized to suit the exact needs of the organization.

Later sections will describe how to build rules for individual applications. The sum of these individual policies will merge to form the "effective policy" enabling all required functionality on a given computer role.

6.3.1 Auto-generate AppLocker Rules for "Everyone"

AppLocker rules can be generated in two ways – using the Local Security Policy MMC console or by using PowerShell. In general, using the graphical console is easiest when creating a large number of rules (e.g. for the base build) as you have interactive control over the rule creation process.

PowerShell is the preferred method for creating application-specific rules, and will be used for the majority of the rule creation and validation process.

Auto-generate AppLocker Rules using the MMC Console

For each of the rule categories (Executable, Windows Installer, Script and Packaged app) right-click the category (e.g. Executable Rules) and select **Automatically generate rules**. Leave the default of **Everyone** as the user or group that rules will apply to. Additional rules for individual users or groups will be created later.

Select **C:** as the folder to be analyzed (which includes all sub-folders). This will ensure that every file on the operating system drive is analyzed. Ensure that the logged on user is a member of the local Administrators group to get maximum access for analyzing files. In any case, you will get errors on folders you don't have access to, and this is described below.

Specify a suitable label such as **SOE** that will be added as a prefix to every rule name. This will allow us to distinguish between rules that apply to the base image and those that apply to other applications.

For rule preferences, the default settings are most commonly used:

- "Publisher" rules will be created for all files that are signed.
- "File hash" rules will be created for files that aren't signed. We won't necessarily use hash rules, but the wizard lets us enumerate all unsigned files.
- The number of rules will be reduced by applying settings that will cover multiple files. In the case of Publisher rules, this means setting the file name and version to * if there are multiple files signed by the same publisher and the same product name (as is common for files that are part of the OS).

The following graphic illustrates the results of the wizard run against C:\ on a Windows 8 system.

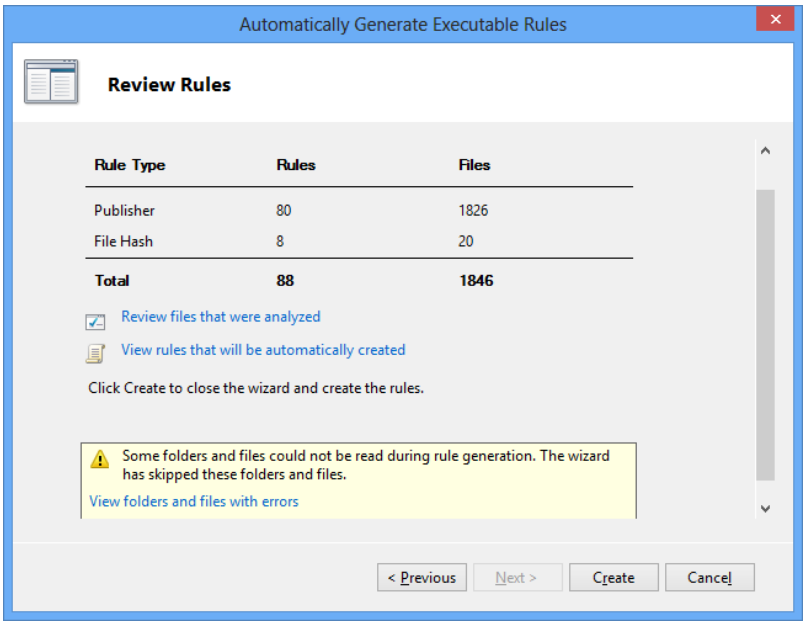


Figure 2

Select **Review files that were analyzed** and click **Publisher** to sort by that column. This allows us to see the unsigned files on the system (where the "Publisher" and "Product Name" fields are blank). We will need to determine if and how we will enable these files in the absence of Publisher rules.
Be aware of the way that the wizard treats files that are digitally signed but do not have entries for "Product name", "File name" or "File version". In this instance, the wizard will not attempt to create Publisher rules and will instead create file path or hash rules. You can work around this by deselecting the signed files in the wizard (so that file path / hash rules are not created for them), and then manually create a Publisher rule to cover these files. This is particularly relevant in the case of Script rules, as a number of scripts are signed by Microsoft, but do not have the above listed fields populated.
Leave the wizard window open and use PowerShell to get a copy of this same information in CSV format, which can be easily sorted, searched and commented in Excel. This spreadsheet is invaluable as you customize your rules in later steps. The following PowerShell command is an example of how you would enumerate all of the "Executable files" on C drive, sort by Publisher and export the result to Exe.csv.
NB. You will need to load the AppLocker module in PowerShell on Windows 7 / Server 2008 R2 before running the command. Run *Import-Module AppLocker* to load this module.

```
Get-AppLockerFileInformation -Directory C:\ -Recurse –FileType Exe | Select-Object –Property Path, Publisher, Hash | Sort-Object –Property Publisher | Export-Csv –Path .\Exe.csv –Encoding Unicode –NoTypeInformation
```

Repeat this by changing the –FileType parameter for each of the different file types that rules will be created for (**Exe**, **Script**, **WindowsInstaller**, and **Dll**).
The process for enumerating packaged apps is different. Use the example below to document packaged apps:

```
Get-AppxPackage –AllUsers | Get-AppLockerFileInformation | Select-Object –Property Path, Publisher | Sort-Object –Property Publisher | Export-Csv –Path .\AppX.csv –Encoding Unicode –NoTypeInformation
```

The resulting spreadsheets will have 3 columns (2 for packaged apps) matching those in the wizard:

- **Path** – the full path to the executable using the AppLocker-specific variables listed in the table in section 2.4.4.1.
- **Publisher** – in the format
  <Publisher's distinguished name>\<Product name>\<File name>\<File version>
  This field is blank for unsigned files.
- **Hash** – hash algorithm and value of the file. This column does not exist for packaged apps.

De-select Unsigned Files
For most organizations, hash rules are too difficult to maintain, so we will prevent auto-generation of hash rules for unsigned files and consider if and how we will enable them in a later step.

Select the top entry, hold SHIFT and select the last entry that has an empty "Publisher". Ensure that all of the entries have been de-selected. This will exclude these files from the rule creation process, and we will use the spreadsheet in a later stage to manually add rules for the files we need to allow to run. Since the above PowerShell command sorts files by Publisher, all unsigned files will be grouped at the top of the spreadsheet for easy identification.

De-select Files for which you don't want "Everyone" Rules Created
There are two reasons for doing this:

- There are files that you don't need anybody to be able to run – effectively disabling them as long as AppLocker policy is enforced.
- There are files that you only want named users or groups to be able to run.

Click a column heading (File name, Product name or Folder path) to sort by that column in order to find files that you do not want **Everyone** to run. As you de-select each file, document the condition in a new column of the spreadsheet. For each file that you deselect, enter either "Disabled" or the users or groups that can run it (bearing in mind that each AppLocker rule applies to a single user or group, so a single group that contains all of the users is preferable to avoid an excessive number of rules). Repeat this process until you have de-selected and documented all of the files that meet the above two conditions.

When you have finished de-selecting files from the list, click OK. While you can always go back and add or remove rules at a later stage, it is far easier to do this now, and allow the wizard to optimize the rules created.

Review folders that were not analyzed due to insufficient permissions
Select **View folders and files with errors**. Review the list to make sure that none of the folders contain executable content. These are typically user profile folders, system folders and the Windows side-by-side store (WINSXS) and shouldn't contain executable program files. Click OK to close the list after review.
If any of the folders do contain executable content, you will need to log on as a user with appropriate permissions and repeat the process to analyze those files.

Create the Publisher rules
Click **Create** to create the rules – answering **No** to the prompt about creating default rules. The default rules are insecure, and not required once the appropriate rules are generated. The wizard will have created Publisher rules allowing Everyone to run all of the files except those you de-selected. The next steps are to:

- Determine the best type of rules to cater for unsigned files and add them.

- Add rules to allow named users or groups (rather than Everyone) to run the files you identified on your spreadsheet.

Repeat the above process for each of the other file types (scripts, installers, and packaged apps). Note that unlike the other file types, DLL/OCX rules cannot be created using the "Automatically generate rules" wizard, although the console can be used to create rules for explicitly selected DLL/OCX files. To automatically generate rules for large numbers of DLL/OCX files, use the PowerShell method that is described next.

Auto-generate AppLocker Rules using PowerShell

The above task can be achieved using PowerShell, but you don't get the benefit of de-selecting files from a wizard to alter rule creation and optimization. This means that you will need to modify some rules after creation to get the desired outcome, however if the number of files that you would have de-selected is small, this doesn't create too much additional overhead. Using the wizard approach is preferable, but obviously is not available if you are creating AppLocker rules on a server running in "server core" mode (i.e. with no desktop shell to run MMC).

Run the same command from the previous section to create a CSV file for documentation purposes, such as:

```
Get-AppLockerFileInformation -Directory C:\ -Recurse –FileType Exe | Select-Object –Property Path, Publisher, Hash | Sort-Object –Property Publisher | Export-Csv –Path .\Exe.csv –Encoding Unicode –NoTypeInformation
```

Repeat this by changing the –FileType parameter for each of the different file types that rules will be created for (**Exe**, **Script**, **WindowsInstaller**, and **Dll**).

Note that there is a bug in the Get-AppLockerFileInformation cmdlet whereby .ocx files are not processed when –FileType is set to "Dll". To create rules for .ocx files, the Local Security Policy MMC console must be used as described in the previous section.

The process for enumerating packaged apps is different. Use the example below to document packaged apps:

```
Get-AppxPackage –AllUsers | Get-AppLockerFileInformation | Select-Object –Property Path, Publisher | Sort-Object –Property Publisher | Export-Csv –Path .\AppX.csv –Encoding Unicode –NoTypeInformation
```

Next, repeat the same command, but pipe the output to additional AppLocker cmdlets to create the policy instead of creating the CSV. For example:

```
Get-AppLockerFileInformation -Directory C:\ -Recurse –FileType Exe | New-AppLockerPolicy –RuleType Publisher –User Everyone –RuleNamePrefix SOE –IgnoreMissingFileInformation –Optimize | Set-AppLockerPolicy –Merge
```

The –IgnoreMissingFileInformation raises a warning in the PowerShell console if there is insufficient file information to create a rule of the specified type for a file (e.g. an unsigned

file where –RuleType is set to Publisher only), but allows the command to continue. You can add additional types **Path** and **Hash** to the –RuleType parameter (separated by commas) to create rules in that order of priority.
The –Merge parameter is important, as otherwise you will overwrite your enforcement setting (audit only) and put AppLocker into enforcement mode (which depending on the rules you have created, may lock you out!).
The PowerShell output shows you the folders that could not be analyzed due to lack of permissions, and the unsigned files for which rules could not be created (assuming you didn't specify Hash or Path for the –RuleType parameter).
Repeat this by changing the –FileType parameter for each of the different file types that rules will be created for (**Exe**, **Script**, **WindowsInstaller**, and **Dll**).
Note that if the AppLocker MMC console was open when rules were created using PowerShell, the console needs to be closed and reopened to display the new rules.
As for documenting the file information, creating packaged app rules uses a different syntax:

```
Get-AppxPackage –AllUsers | Get-AppLockerFileInformation | New-AppLockerPolicy –User Everyone –RuleNamePrefix SOE –Optimize | Set-AppLockerPolicy –Merge
```

6.3.2 Create AppLocker Rules for Unsigned Files

In the previous steps we auto-generated (using the console or PowerShell) Publisher rules for files that were digitally signed, and deliberately omitted creating rules for unsigned file. The next step is to review the unsigned files and determine:

- Do we need them to run at all, and if so, by Everyone or named users / groups?
- If we need them to run, can we efficiently and securely enable them via Path rules?
- If not, create Hash rules to enable them.

Path Rules for Unsigned Files
If there are a small number of executables within an application's folder, it is fairly efficient to create individual path rules for each file. Before doing this, you need to check the permissions on the file to ensure that it cannot be overwritten by users of the computer with any other arbitrary executable. This would allow users to run that arbitrary executable as it would satisfy the AppLocker path rule.

If there is a large number of unsigned files, it may be more efficient to create a single Path rule targeting a folder that contains the files (or contains subfolders that contain the files). In this case, it is critical to review permissions on that folder and **all subfolders**. If a subfolder allows users to write, they could copy any arbitrary executable into that subfolder and circumvent the AppLocker policy. Use a tool such as AccessEnum (http://technet.microsoft.com/en-us/sysinternals/bb897332) to analyze write permissions on the subfolder hierarchy.

If a user-writable subfolder (or its subfolders) don't contain any required executables, you can add that subfolder to the list of exceptions for that Path rule to prevent any arbitrary executables copied there from executing.

If a user writable subfolder (or its subfolders) do contain executables (and you are not able to change permissions on the folder), the only safe way to enable these files is via File hash rules.

File Hash Rules for Unsigned Files

File hash rules should generally be avoided unless the substantial administrative overhead can be justified by the enhanced security these rules provide. If an executable file that is protected by a File hash rule is modified or replaced by another executable, it won't match the hash in the rule and will not run. Unfortunately legitimate updates like updating with a new version or patching will also change the hash, and the hash value in the rule will need to be recalculated for it to work.

While File hash rules can contain many hashes from multiple folders, it is recommended that a new File hash rule be created for each subfolder and the subfolder clearly indicated in the rule name or description field. This will make it easier to know which hashes to update, as you may have multiple files with the same name across the computer's folder structure.

6.3.3 Create AppLocker Rules for Named Users or Groups

In a previous step we auto-generated (using the console or PowerShell) Publisher rules for the files that "Everyone" can run, and omitted files that can only be run by specific users or groups. For these files that were omitted, we documented the users / groups that are allowed to run these files in a spreadsheet. In this step, rules will be created to implement these conditions.

The process for creating these rules is similar to that for creating rules that applied to "Everyone", except that you specify a local or domain user or group instead of "Everyone". AppLocker rules apply to a single subject, so if you need multiple users or groups to run a file, you will need to create a rule for each. Where possible, use a single domain group to enable all required users to run a file.

If there are numerous files that the same subject (user or group) needs to run, use the "Automatically Generate Rules" wizard to rapidly create the required rules. Instead of starting the search from C:\, choose the highest folder in the hierarchy that contains all of the target files.

If you have a small number of rules, you can use the "Create New Rule" wizard. Select the appropriate rule type (preferably Publisher), and browse directly to the files to populate the rule attributes.

6.4 Validate AppLocker Rules for Base Build

A two stage process should be used to validate the AppLocker rules just created for the base build:

- Use the **Test-AppLockerPolicy** PowerShell cmdlet to check rule coverage and expected outcomes for specific subjects against specific objects.
- Perform end-user and administrative test cases and review the generated audit data.

Note that this testing is designed to catch as many errors as possible, but is not intended to be exhaustive. Once the rules pass this initial validation stage, they will be thoroughly tested during the Stabilize phase of the AppLocker project.

6.4.1 Verify AppLocker Rules using the Test-AppLockerPolicy Cmdlet

The **Test-AppLockerPolicy** PowerShell cmdlet is extremely useful for verifying the rule set just created for the base build.
For input you specify:

- A set of files or AppX packages (which can be listed or piped from another cmdlet).
- A user. In order to test the result that would be applied to users in specific groups, just pick any user from that group for the test. (The AppLocker cmdlet will calculate the group nesting automatically). If the –User parameter is omitted, Everyone is assumed.
- One or more comma-separated filter actions to test for which can be:
    - DeniedByDefault - no Allow rule match occurs, so the subject is implicitly denied from running the object.
    - Allowed - a rule explicitly allows the subject to run the object.
    - Denied - a rule explicitly denies the subject from running the object
    - AllowedByDefault - AppLocker has been enabled but no rules have been added to the collection. In this case, AppLocker allows everything. This is not a valid use case, so would indicate a misconfiguration.
    - <Omitted> - if the –Filter parameter is omitted, all actions are included. Pipe to Sort-Object to sort by "PolicyDecision" so that like actions are grouped together (see example 3 below).

The output is the list of objects that would be filtered in the method specified, together with the AppLocker rule that resulted in that decision (in the case of Allowed or Denied).

Example 1:
Test whether the AppLocker rules in Local Security Policy would allow any user to run specific files.

```
Test-AppLockerPolicy –PolicyObject (Get-AppLockerPolicy –Local) –Path C:\Windows\regedit.exe,
C:\Windows\System32\regedt32.exe –User Everyone –Filter Allowed | Format-Table – AutoSize
```

Example2:
Test whether the AppLocker rules in Local Security Policy would deny a specific user from running any .exe file in \System32 because there is not an explicit allow rule for any of the files, and save it to a text file. You could also add the –Recurse parameter to Get-ChildItem to create a list of all files in all subfolders of the folder you specify. This could be used to test all .exe files on C drive for a particular user. Note that you will receive errors in the PowerShell console where you don't have access to the files in a subfolder, but the command will complete for all subfolders that are accessible.

```
Get-ChildItem C:\Windows\System32\*.exe | Test-AppLockerPolicy –PolicyObject (Get-AppLockerPolicy –Local) –User
"Contoso\AliceB" –Filter DeniedByDefault | Format-Table – AutoSize | Out-File Denied.txt
```

Example 3:
In this example the –Filter parameter has been omitted, so we get a list of all of the .exe files in \System32 and the action that would be taken for the specified user by rules in the local policy (sorted by policy decision).

```
Get-ChildItem C:\Windows\System32\*.exe | Test-AppLockerPolicy –PolicyObject (Get-AppLockerPolicy –Local) –User
"Contoso\AliceB" | Sort-Object –Property PolicyDecision | Format-Table – AutoSize
```

The Test-AppLockerPolicy cmdlet has other capabilities that will be useful throughout the deployment and support of an AppLocker deployment:

- Testing can be done against policy that has been exported to XML files rather than requiring the policy to be live on a machine. This is useful for testing how new files on a computer will be handled without having to actually load policy onto that machine.
- Testing can be done against policy that resides inside Active Directory GPOs, without the policy actually being deployed to that machine. As above, this is useful for testing how new files on a computer will be handled by different GPOs without having to actually enforce policy on that machine. To do this, just use the Get-AppLockerPolicy cmdlet with the –Domain and –Ldap parameters to specify the path to the GPO.
- Testing can be done against the "effective policy" on a computer – that is the merger of all GPO policies that are in scope for that computer as well as its local policy. This is extremely useful for troubleshooting issues on computers when policy comes from

multiple sources (the reason why the rule prefix is so important as this will indicate the policy that the rule resides in). To use this, just use the Get-AppLockerPolicy cmdlet with the –Effective parameter. This will be used in the Stabilize phase of the project (see section 7.5.2 below).

- Packaged apps can be analyzed in the same way as the other file types (but only against XML files – not loaded policy).

6.4.2 Perform Usage Cases and Review Audit Data

The second step in the base build rule validation process is to run end-user and administrator usage cases on the computer (that has been put into "Audit only" enforcement mode), and then review the audit data generated.
The appropriate user and administrator accounts should logon to the computer, and try as many tasks as is practical to thoroughly test the rule set. In addition to tasks that the user should be able to perform, they should also perform tasks that they should not be able to perform once AppLocker is configured to enforce rule actions. Once these usage cases have been run, the audit data will be reviewed, and any necessary changes to the rules made.
Use the following command to search the Windows event log for events indicating that objects would have been prevented from executing if AppLocker had not been in the "Audit only" enforcement mode:

```
Get-AppLockerFileInformation –EventLog –EventType Audited –Statistics
```

Possible values for –EventType are:

- Allowed an object was allowed to run by the effective AppLocker policy. This corresponds to **Information** events with ID **8002**, **8005**, **8020** and **8023** (for Exe, MSI/Script and Packaged app-execution and Packaged app-deployment respectively)".
- Denied an object was prevented from running by the effective AppLocker policy. This corresponds to **Error** events with ID **8004**, **8007**, **8022** and **8025** (for Exe, MSI/Script and Packaged app-execution and Packaged app-deployment respectively)".
- Audited an object was allowed to run but would have been prevented if AppLocker was not in "Audit only" enforcement mode. This corresponds to **Warning** events with ID **8003**, **8006**, **8021** and **8024** (for Exe, MSI/Script and Packaged app-execution and Packaged app-deployment respectively)".

If the –EventType parameter is omitted entirely, all three event types are assumed.

"Statistics" indicates the number of occurrences of each event. It does not distinguish between "Denied" and "Audited" events, and adds both when displaying the count of "Denied" events.
The objects returned from the Get-AppLockerFileInformation cmdlet against the Event log can be used to create new AppLocker rules directly. The following command reviews the Event logs for actions that would have been prevented and creates either a Publisher, Hash or Path rule (in that order of precedence) and merges it into the Local Security Policy. It adds a prefix of AUDIT so it can be easily identified. Once these rules are created, they should be customized to meet the exact requirements, and the spreadsheets updated with the new information.

```
Get-AppLockerFileInformation -EventLog -EventType Audited | New-AppLockerPolicy -RuleType Publisher, Hash, Path -User Everyone –RuleNamePrefix AUDIT -Optimize | Set-AppLockerPolicy –Merge
```

Clearing AppLocker Events
Between iterations of testing, it is useful to clear the AppLocker events from the logs so that you only get the results from the current test pass. To clear one of the AppLocker event logs, use the following command from the PowerShell console:

```
wevtutil cl "<log name>"
```

Where <log name> is one of:

- Microsoft-Windows-AppLocker/EXE and DLL
- Microsoft-Windows-AppLocker/MSI and Script
- Microsoft-Windows-AppLocker/Packaged app-Deployment
- Microsoft-Windows-AppLocker/Packaged app-Execution

A quick way of clearing all AppLocker logs with a single command is:

```
wevtutil el | findstr /I "AppLocker" | ForEach-Object {wevtutil cl "$_"}
```

6.5 Export AppLocker Rules for Base Build to XML File

Once the rule set for the base build has been validated, the rule set will be exported to an XML file. This XML file can be used in a number of different ways:

- It can be kept as a backup or as an artefact in a change control database for retrieval if required.
- It can be exchanged with other IT groups to serve as a starting point for their AppLocker policies.
- Test-AppLockerPolicy can be run against it to determine the outcomes of actions by different users without the policy actually having to be loaded.
- It can be used to replace or merge with the existing rules in a computer's Local policy.
- It can be imported into a GPO for deployment to other computers.

Export All AppLocker Rules with Specified Prefix to XML

A useful approach is to save rules with a common name prefix (like SOE) to individual XML files. This allows you to create application-specific GPOs from the individual XML files, while still retaining the flexibility of merging multiple XML files into a single GPO if that is a more appropriate way to deliver policy.

The following example reads all rules with the prefix of **SOE** from the local policy, sets the enforcement mode for each rule type to "Not Configured", and saves the rules to **SOE.xml** in the user's current directory.

All of the script logic is contained within the function **Export-AppLockerRules** which is listed in the appendix. While not used in this scenario, you can also pass the –Effective parameter to the function to read rules from effective policy instead of local policy. Effective policy includes local policy rules plus rules from any GPOs that have been applied.

```
Export-AppLockerRules $pwd\SOE.xml SOE
```

Export All AppLocker Rules to XML

You can use the same function to export the entire set of AppLocker rules from the local (or effective) policy to an XML file, rather than just those with a specific prefix.

```
Export-AppLockerRules $pwd\AllRules.xml -All
```

6.6 Create AppLocker Rules for Individual Applications

The process of creating AppLocker rules for individual applications is similar to that for creating rules for the base build. The main difference being that instead of targeting the entire C drive, you target only the files and folders that the application installs (e.g. C:\Program Files\Microsoft Office).[12]
It is also important to use a unique rule name prefix (e.g. Office2013) to simplify the process of differentiating between different rules when they are merged into a single effective policy.
Repeat the following process for each application that was identified during the Plan phase:

- Generate AppLocker rules as described in section 6.3.
- Validate AppLocker rules as described in section 6.4.
- Export application-specific AppLocker rules to appropriately named XML files as described in section 6.5.

6.7 Clear AppLocker Policy from Reference Computers

Once all of the AppLocker policies have been created and exported to XML, the local security policy of the reference computers should be cleared. This will ensure that policy can be deployed and managed centrally, and the effective policy on the reference computers won't contain rules from the local policy (which can't be centrally managed). To clear AppLocker policy (delete all rules and set all enforcement modes to "Not configured"), right-click the **AppLocker** node and select **Clear Policy**.
To clear AppLocker policy using PowerShell:

```
$policy = Get-AppLockerPolicy –Local
$policy.DeleteRuleCollections()
Set-AppLockerPolicy $policy
```

Reimporting Policy from XML to Local Computer Policy
If you need to get the policies back onto the reference machine for additional development, import the XML file containing the base build policy and merge any additional application-specific policies required. Use the following example:

```
Set-AppLockerPolicy –XmlPolicy "SOE.xml"
Set-AppLockerPolicy –XmlPolicy "Office2013.xml" –Merge
```

[12] Microsoft recommends the use of Attack Surface Analyzer (http://www.microsoft.com/en-us/download/details.aspx?id=24487) to evaluate application installations for introduction of known security weaknesses, such as weak ACLs on executable files.

The first command doesn't specify the –Merge parameter so the entire local AppLocker policy (including enforcement modes) will be overwritten by the contents of the XML file. Subsequent commands use the –Merge parameter which won't change the enforcement mode of the computer (if it is different from what is in the XML file), will overwrite any rules with the same rule ID, and will add any rules that are in the XML file but not on the computer. This is useful to restore the full set of rules to a computer – from the base policy as well as for each individual application.

7 Phase 4 – Stabilize

7.1 Overview of the Stabilize Phase

The Stabilize phase is where AppLocker policies are imported into GPOs and deployed to the target computers. The computers will operate in "Audit only" mode until sufficient confidence is reached that the policies have been fully validated.
The following list summarizes the activities in the Stabilize phase of the AppLocker deployment project:

- Configure centralized AppLocker event collection.
- Deploy AppLocker policy via GPOs.
    - Create GPOs from XML files.
    - Create enforcement GPO and put in "Audit only" mode.
    - Filter GPOs using security groups and WMI filters if required.
    - Link GPOs to OU structure.
- Validate the AppLocker deployment.
    - Verify that correct GPOs and settings are being applied.
    - Test effective AppLocker policy using PowerShell.
- Conduct user acceptance testing.
- Obtain sign off for change to "Enforce rules" mode.

7.2 Configure Event Collection

Deploy or configure the solution for collection of AppLocker events as determined in section 5.6 above. Ensure that all clients have been configured to forward AppLocker events before AppLocker policy is targeted to them via Group Policy.
If a third party solution will be used for event collection, skip to section 7.3 to deploy AppLocker policy using Group Policy. If the built-in Windows Event Forwarding and Collection architecture will be used, the following subsections provide basic instructions for configuring the forwarder and collector components (although detailed design and deployment of this infrastructure is beyond the scope of this document).

7.2.1 Configure the Event Collector

The event collector is the computer that all AppLocker events will be forwarded to. While a client or server version of Windows can be used, configuration steps will vary depending on which is used. Windows Server 2012 is recommended, and the steps that follow are for that version.

Configure the Windows Remote Management Service and Listener
The following requirements must be met for the WinRM service which provides the underlying transport for the event forwarder infrastructure:

- The "Windows Remote Management (WS-Management)" service must be configured with a startup type of "Automatic".
- An HTTP (TCP 5985) or HTTPS (TCP 5986) listener must be created and configured to listen on at least one IP address that can be reached by the event source computers.
- A Windows Firewall exception must be enabled to allow the remote event source computers to connect to the listener.

There are a number of ways to achieve the above and a number of configuration settings (such as authentication type) that won't be covered in this guide.
Running the command **winrm qc** from an elevated command prompt will achieve all of the above requirements – enabling the WinRM service, creating an HTTP listener that is bound to every IP address on the server and accepting Kerberos or Negotiate authentication protocols, and creating a Windows Firewall exception for the HTTP listener.

Configure the Windows Event Collector Service
The following requirements must be met for the Windows Event Collector service which allows subscriptions to be created and managed, and accepts events from remote event source computers via the WinRM listener:

- The "Windows Event Collector" service must be configured with a startup type of "Automatic (Delayed Start)".
- The "ForwardedEvents" channel (which is where events will be stored) must be enabled.

Running the command **wecutil qc** from an elevated command prompt will achieve all of the above requirements.

Create an Event Subscription

The event subscription will be retrieved by the remote event source computers and configure them to forward the specified events to the collector. Create a new subscription (from the Event Viewer MMC console) on the event collector computer with the following properties:

- Name AppLocker
- Destination log Forwarded Events
- Type Source computer initiated
- Computer Groups "Domain Computers" (or custom group containing the computer accounts to accept events from).
- Events to collect:
  - Logged Any time
  - Event level Error, Warning, Information
  - Event logs Microsoft-Windows-AppLocker/EXE and DLL
    Microsoft-Windows-AppLocker/MSI and Script
    Microsoft-Windows-AppLocker/Packaged app-Deployment
    Microsoft-Windows-AppLocker/Packaged app-Execution
  - Event IDs 8000-8007, -8002, -8005, 8021-8027, -8023
    (the minus sign excludes an ID from the range. Refer to the table in section 2.5 for a list of events)
  - User <All users>
  - Computers <All Computers>
- Delivery Optimization Normal (events are batched and sent after 5 events or 15 mins)
  Minimize latency (events are batched and sent every 30 seconds)
  Minimize bandwidth (events are batched and sent every 6 hours)
- Protocol HTTP

7.2.2 Configure the Event Source Computers

Group Policy will be used to configure the clients to:

- Configure the "Windows Remote Management (WS-Management)" service with a startup type of "Automatic".
- Provide the URL of the event collector to retrieve the subscription from and forward events to. Configure **Computer Configuration | Policies | Administrative Templates | Windows Components | Event Forwarding | Configure target Subscription Manager**. Enable the setting and add a value of **server=srv01.contoso.com:5985**

7.3 Ensure Required Hotfixes are Deployed

Ensure that the hotfixes listed in section 5.10 are deployed to all computers prior to targeting them with GPOs containing AppLocker policy.

7.4 Deploy AppLocker Policy via Group Policy

7.4.1 Create GPOs from the Exported XML Files

At this stage of the process, you have an XML file containing AppLocker rules for the target computer's base build as well as an XML file for each application that can be deployed to it. The next step is to import the rules from these XML files into corresponding Group Policy Objects, so that they can be deployed to the required computers.
There are numerous ways to complete this task, and the method chosen will vary depending on who manages Group Policy in the organization, and the tools they use to do it. A recommended practice would be to check the XML files into a change tracking tool, and having the process completed by the team that manage Group Policy for the appropriate domain.
The following is an example of using PowerShell to create a GPO and then import the rules from an XML file into it. The following assumptions must be in place for the specific PowerShell commands chosen:

- The computer on which the command is being run is a domain controller or computer with the Remote Server Administration Tools installed (required for the AD cmdlets to be present).
- The computer on which the command is being run belongs to the same domain as the domain in which the GPO is to be created (if not, you will need to modify the example command to specify an alternate domain).
- The user running the command has the ability to create GPOs (members of the Domain Admins or Group Policy Creator Owners groups by default). If the GPOs will have already been created by another user, the user importing the AppLocker policy only needs to have been delegated the "Edit settings" permission for the corresponding GPO.

Create an empty GPO in the computer's domain that will host AppLocker rules for the base build. The final command will disable the user portion of the GPO as AppLocker rules apply to the computer portion only:

```
$gpoName = "AL-SOE"
$gpoComment = "AppLocker policy for base SOE"
$gpo = (New-Gpo –Name $gpoName –Comment $gpoComment)
$gpo.GpoStatus = "UserSettingsDisabled"
```

Import rules from SOE.xml into the newly created GPO.
The second command builds a string ($s) in the form:
LDAP://<domain controller>/CN={<GUID of GPO>},<distinguished path of GPO>
The third command passes that string to Set-AppLockerPolicy. As in previous examples, you can append –Merge to the end of this command to preserve existing enforcement settings and existing rules in the GPO (not relevant if you have just created a new GPO):

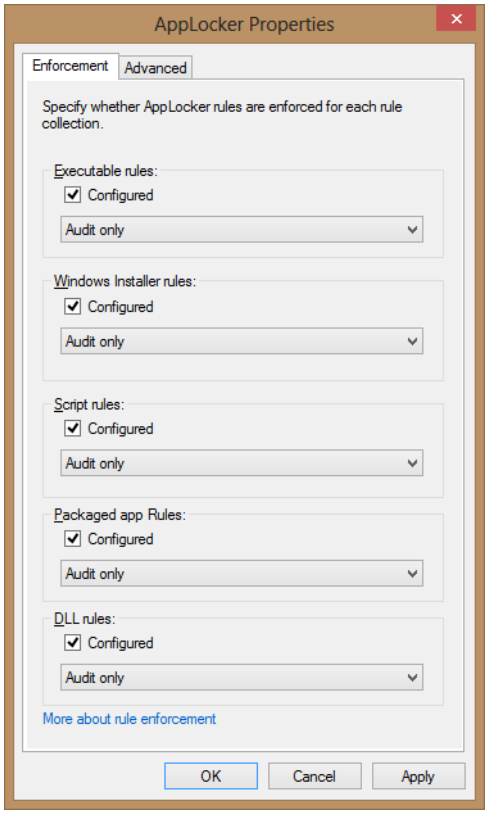
```
$gpoName = "AL-SOE"

$s = "LDAP://"; $s += (Get-ADDomain -Current LocalComputer).PDCEmulator; $s += "/CN={"; $s += (Get-Gpo -Name "AppLocker-Base Policy").Id; $s += "},CN=Policies,CN=System,"; $s += (Get-ADDomain -Current LocalComputer).DistinguishedName

Set-AppLockerPolicy -XmlPolicy "BasePolicy.xml" -Ldap $s
```

7.4.2 Create Group Policy for "Audit only" Enforcement Mode

As described in section 5.5.2.3, it is common practice to create a dedicated GPO with the highest precedence (of all GPOs containing AppLocker settings) to control the AppLocker enforcement mode on the different endpoints. The following graphic illustrates the Properties of the AppLocker node in the GPO where the enforcement mode is configured for each rule type. Note that the DLL rule type is only displayed if enabled in the Advanced tab.



As described in section 5.5.2.3, there are additional aspects of AppLocker configuration that must be managed via Group Policy. Depending on the organization's Group Policy design, these could be managed in existing GPOs, or could be added to this AppLocker enforcement GPO:

- Configure the **Application Identity** service with a startup type of "Automatic". This is required for AppLocker.
- Configure the **Windows Remote Management (WS-Management)** service with a startup type of "Automatic". This is required to forward events to a remote event collector.
- Configure the URL of the remote event collector as described in section 7.2.2 above.
- Configure the custom error URL as described in section 5.7 above.

7.4.3 Filter Group Policy Objects to Target Desired Computers

Group Policy can be filtered so that it only applies to a subset of computers that would otherwise be in scope via the Group Policy inheritance model. This is useful when the OU structure does not facilitate precise targeting of policy via inheritance alone. It is also useful when there is a requirement to pilot policy settings by applying them to an initial subset of the computers that will ultimately receive the policy.
To use security group filtering on specific GPOs:

- Create (or reuse existing) security groups and populate them with the computer accounts of the machines that should receive policy.
- Remove permissions on the GPO from the "Authenticated Users" group and grant the custom security groups access to read the GPO.

To use WMI filtering on specific GPOs:

- Create (or reuse existing) WMI query objects that will equate to TRUE when evaluated by the target computers, and will equate to FALSE when evaluated by computers that should not receive the policy (but are within scope via inheritance).
- Associate those WMI queries with the corresponding GPOs.

7.4.4 Create Group Policy Links to Deploy AppLocker Policy

The final task is to link the filtered GPOs to the appropriate OUs in the OU hierarchy. The next time that computers process Group Policy, the AppLocker policy will be applied and take immediate effect.
Link the enforcement mode GPO first to ensure that it is in place before any of the GPOs containing AppLocker rules are linked. After linking any additional GPOs to the OU that the enforcement GPO is linked to, ensure that the enforcement mode GPO still has the highest precedence.

AppLocker policy should take effect on all targeted computers within 2 hours of the GPO links being created. To cause a computer to update Group Policy immediately, run *gpupdate /force* on the local computer.
You can also leverage Windows Remote Management to perform the same action against remote computers using the following PowerShell command:

```
Invoke-GPUpdate <computer> –RandomDelayInMinutes 0 –Force
```

Where <computer> is the name of the remote computer to update policy. You can also pipe a list of computers to Invoke-GPUpdate to perform the update on multiple computers.

7.5 Validate AppLocker Policy Deployment

AppLocker policy deployment should be validated on a sample set of computers to ensure that policy has been deployed as expected. Complete the following tests on a representative sample from each computer role that has had AppLocker deployed.

7.5.1 View Resultant GPO Policy

The first validation step is to view that the correct AppLocker GPOs have been applied to the computer. From the elevated PowerShell console run the following command:

```
gpresult /scope computer /h $home\Desktop\gpresult.htm /f
```

Double-click the gpresult.htm file that was created on the desktop and click "Allow Blocked Content" at the bottom of the Internet Explorer window. Note that this command can also be used against remote computers if the "Remote Event Log Management" rule group has been enabled in the target computer's Windows Firewall configuration.

Under the **Computer Details** section are a number of subsections that contain the required information:

- **Group Policy Objects**
  - **Applied GPOs** – shows each GPO that has been applied to the computer. Expanding the GPOs will show the OU it was linked to, the security groups that can apply it and any WMI filters on the GPO.
  - **Denied GPOs** – shows each GPO that was not applied to the computer. Expanding the GPOs will show the OU it was linked to, the security groups that can apply it and any WMI filters on the GPO. It also shows the reason why the GPO was not applied.
- **Settings** – shows all settings that were applied to the computer from all of the "Applied GPOs" (the resultant set of policy), and which GPO applied each setting (the GPO with the highest precedence where multiple policies attempt to configure the same setting).

Expand **Policies | Windows Settings | Security Settings | Application Control Policies** to see each of the AppLocker rule types, their enforcement mode and the rules within them. For each item, the "Winning GPO" column indicates the GPO that set the setting.

7.5.2 Test Effective Policy

The final validation step is to use the Test-AppLockerPolicy cmdlet against the **effective policy** on sample machines to see what would be allowed and what would be blocked. The exact commands will vary depending on what AppLocker rules have been deployed, but the following examples can be used as a starting point. Refer to section 6.4.1 for more detail on the Test-AppLocker cmdlet and more examples.

Exe, Script, Installer and DLL Rule Types
This example command enumerates every .exe file on C drive, and tests whether the effective policy would block any from running for the specified user. Note that errors will be displayed in the PowerShell console for folders that cannot be accessed due to lack of permissions, but the command will complete for all files that can be accessed. Files that have been blocked by an explicit "Deny" rule (rather than being denied by default) will list the name of the "Deny" rule.

```
Get-ChildItem "C:\*.exe" –Recurse | Test-AppLockerPolicy –PolicyObject (Get-AppLockerPolicy –Effective) –User "Contoso\AliceB" –Filter Denied, DeniedByDefault | Format-Table – AutoSize | Out-File $home\Desktop\Denied.txt
```

The command should be repeated with different starting folders, different file types, and with different users to get adequate coverage of the rule set. You could also test with the filter condition of "Allowed" to see what specific users CAN run on the computer rather than what would be blocked.
NB. An issue may occur with Windows 7 computers prior to service pack 1 where application installers running under the SYSTEM account are blocked by AppLocker. This is a bug, as AppLocker should not be controlling any processes running as SYSTEM. If affected, a resolution is to create a path rule of "*" in the Windows Installer rule collection for user NT AUTHORITY\SYSTEM.

Packaged App Rule Types
Test-AppLockerPolicy usage is slightly different for packaged apps in that it needs to be tested against an XML file rather than a PowerShell policy object. This requires the additional step of calculating effective policy and saving it to XML, and then using that XML file for the testing.
Similar to the above examples, this example tests all packaged apps on the computer for a specified user and a policy action of "Allowed". The rule that allows each packaged app is also displayed.

```
Get-AppLockerPolicy –Effective –Xml | Set-Content –Encoding Unicode (“$home\Desktop\Effective.xml”)
Get-AppxPackage –AllUsers | Test-AppLockerPolicy –XmlPolicy $home\Desktop\Effective.xml –User “Contoso\AliceB” –Filter Allowed
```

7.6 User Acceptance Testing

The validation steps of the previous section validate that AppLocker policy has been successfully deployed via Group Policy on a sample set of target computers.

The final validation step is to operate the full set of target computers in their normal production setting in AppLocker “Audit only” mode. This is a critical step in the AppLocker deployment, as it is the first time that AppLocker rules will be evaluated under normal operating conditions by the intended users and administrators of the systems. This step is critical in identifying rules that do not meet identified business requirements, and scenarios that have been overlooked in the development of AppLocker policy.

This validation stage will continue with no impact on users until the appropriate authority is satisfied that the AppLocker rules have undergone sufficient testing. During this time, the processes for identifying rule change requirements, and performing the required maintenance on the AppLocker policies (developed during the Plan phase) will be invoked to ensure that they operate as intended.

Culmination of the Stabilize phase is formal sign off by the appropriate authority that AppLocker will be switched from audit-only to enforcement mode. This is typically done in a staged fashion – with different computer roles reaching sign off at different times based on the deployment plan developed in section 5.9.

8 Phase 5 – Deploy / Operate

8.1 Overview of the Deploy / Operate Phase

By the end of the Stabilize phase, all of the design work has been completed and validated, and AppLocker has already effectively been "deployed" to the target computers. The only activity to move AppLocker into production is to communicate the change to users, and change the enforcement mode from "Audit only" to "Enforce rules" mode via Group Policy at the appropriate time.
The remainder of this section will detail some of the processes and activities; both proactive and reactive, that will take place once computers are operating with AppLocker enforcing policy.

8.2 Deploy

8.2.1 Communicate Change to AppLocker Enforcement Mode

The communication plan developed in section 5.9.1 should already be in operation and users should have received numerous communications about the change in AppLocker enforcement mode on their computers. The final communication should remind them about the timing of the change as well as all of the support resources at their disposal.

8.2.2 Change AppLocker Mode to "Enforce rules"

The only task required to "deploy" AppLocker is to change the appropriate enforcement GPO from "Audit only" to "Enforce rules". As with all GPO processing, this change will be detected and applied by all computers within 2 hours (assuming they have connectivity with their domain). There is no need for any action on the target computers, and the change will be invisible to end users.

8.3 Operate

8.3.1 Proactive Operations

The objectives of **proactive operations** are:

- To identify failures that are not brought to the attention of support personnel. This may be because the user did something in error, or accepted the access denied message because they did not have the time or inclination to seek assistance.
- To identify suspicious behavior such as a user trying to run things they are not authorized to, or malware that a user is inadvertently trying to activate.
- To spot trends – such as a percentage of the user population trying to run something that is not yet accommodated by AppLocker policy.

The monitoring solution described in section 5.6.3 is the appropriate tool to use for proactive monitoring of the AppLocker solution. Ideally AppLocker events will have been imported into SQL and SQL Reporting Services is being used to report on the data. If not, the sample scripts provided in section 5.6.3 (and the appendix) can be used to mine the event data and produce reports that can be viewed in Excel.
The frequency of monitoring will depend on the value of the resources being protected and resources assigned to the task, however weekly monitoring of AppLocker events should be considered the minimum frequency.
The outcome of monitoring will be one of two potential actions:

- If suspicious or unauthorized activity is detected, an incident should be created to investigate the cause.
- If a significant number of users are being prevented from performing potentially legitimate actions, it should be reported to the change control process. Management can assess the need to accommodate the applications via additions to AppLocker policy.

8.3.2 Reactive Operations

The objectives of **reactive operations** are:

- Respond to support requests – either logged via the helpdesk or submitted by the user entering details in a web form whose link was provided in the custom AppLocker message.
- Evaluate the requests, and initiate change control to resolve the issue if required.

Identify Whether the Error is "By Design"

Determine whether the user is authorized to run the application they were trying to run. If not, they can be advised of company policy and no further action is required. The user can request access if they have a legitimate reason, and the request passed through the usual approval process.

Identify the Cause of the Error

If the user is authorized to run the application, the quickest way to identify the cause of the error is to run a query against the centralized repository of AppLocker events. The process is similar to that described in section 5.6.3 *"Monitoring and Reporting on AppLocker Events"*. The main differences are:

- A more granular filter is defined to return a smaller number of events. In the example below, we have added a condition at the start of the filter specifying the **user** that was denied access in the AppLocker event. Note that we pass the user name to a function (Get-Sid) to convert it to the corresponding SID which is referenced in the event data. The code for function Get-Sid is in the appendix.
  If the user does not know the name of their machine, we can omit the last condition as it is probably redundant.
- Different event IDs are specified that correspond to AppLocker preventing an application from running (refer to the table in section 2.5 for a list of event IDs).
- We don't pass a file name to Get-AppLockerEvent so the results are displayed in list format on the console rather than saved to a file. Assuming the filter is specific enough in selecting the events of interest, the data returned should be manageable. Bear in mind that you could pipe this to further cmdlets to do additional filtering, sorting, removal of duplicates etc.

```
$filter = "*[UserData/RuleAndFileData[TargetUser='$(Get-Sid contoso\AliceB)']] and *[System[(EventID=8004 or EventID=8007 or EventID=8022 or EventID=8025) and TimeCreated[timediff(@SystemTime) <= 86400000] and (Computer='win8-1.contoso.com')]]"

Get-WinEvent -LogName ForwardedEvents -FilterXPath $filter | Get-AppLockerEvent
```

Read the entry in the **Rule Name** property of the returned event (or events):

- If the "Rule Name" property has an entry, the file has been blocked by a "Deny" rule and the property will give you the name of the rule that denied access. Use the rule prefix to identify the GPO that it was delivered by, and investigate why this GPO has been applied (i.e. by design or in error).
- If the "Rule Name" property has a "-" symbol, it was blocked because it has not been explicitly allowed. This is the more likely case.

If you are unable to find the error by querying event data, you could use the **Test-AppLockerPolicy** cmdlet (locally or remotely) against the effective policy of the computer to find the same information as above. This is described in section 7.5.2. Alternatively you could query the local event log of the user's computer.

Initiate Resolution of the Error

The issue will be resolved by either:

- Removing the "deny" rule from the computer's effective policy. This is unlikely if the recommendation to avoid "deny" rules has been followed.
- Getting the appropriate "allow" rule into the computer's effective policy.

There are a number of common reasons why an "allow" rule does not exist in the computer's effective policy. The action to take will depend on the cause:

- The GPO has not been applied to the user's computer. Use the **gpresult** tool as described in section 7.5.1 to find out if the GPO is being filtered by WMI or security group membership. If not, check whether the GPO links to the appropriate place in the GPO hierarchy, and that inheritance is not being blocked.
- An appropriate "allow" rule has not been created. Work with the application owner to understand why the rules were not created and initiate change control to have this resolved.
- There is an error in the "allow" rule – such as the file has been updated and no longer matches the rule condition. Initiate change control to have this resolved.

9 Appendix

The appendix contains a listing of PowerShell functions that have been referenced throughout this guide. If you plan on using them extensively, copy the text from them into a file called **\Documents\WindowsPowerShell\Modules\AppLockerTools\AppLockerTools.psm1**. That way, you can simply run **Import-Module AppLockerTools** (assuming your Execution Policy allows you to run scripts) at the start of a PowerShell session, and call the functions from the PowerShell console.

9.1 Function "Get-AppLockerEvent"

The function **Get-AppLockerEvent** receives objects piped to it from Get-WinEvent and (optionally) the name and path to save results to (OutPath).
If a file name is passed as a parameter to the Get-AppLockerEvent function, the results will be written as comma-separated values to the specified file. The CSV file can then be imported into Excel (specifying comma as the delimiter) for viewing and reporting purposes.
If a parameter is not passed to the Get-AppLockerEvent function, custom objects corresponding to each event that contain all of the AppLocker-specific details as properties are output to the pipeline. This allows them to be displayed, or piped to other cmdlets for further manipulation (filtering, sorting etc.) As all of the relevant details are in the objects, they could also be used to populate an AppLockerPolicy object which could be piped to the Set-AppLockerPolicy cmdlet to create additional rules.

```
function Get-AppLockerEvent {
<#
 Input: Objects piped from Get-WinEvent and (optionally) the name and path to save results to.
 Output: If the "OutPath" parameter is supplied, the results are written as a CSV file to the location provided.
       If not, custom objects containing AppLocker event details as properties are written to the pipeline.
#>
 param ([string]$OutPath)
 BEGIN {
  if ($PSBoundParameters.ContainsKey('OutPath')) {
    $ALevents = @()
  }
 }
 PROCESS {
  $evt = [xml]$_.ToXml()
  $ALevent = New-Object System.Object
  $ALevent | Add-Member -type NoteProperty -name "Level" -value $evt.Event.RenderingInfo.Level
  $ALevent | Add-Member -type NoteProperty -name "Event ID" -value $evt.Event.System.EventID
  $dt = [DateTime]::Parse($evt.Event.System.TimeCreated.SystemTime)
  $ALevent | Add-Member -type NoteProperty -name "Date" -value $dt.ToString("dd-MM-yyyy")
  $ALevent | Add-Member -type NoteProperty -name "Time" -value $dt.ToString("HH:mm:ss")
  $ALevent | Add-Member -type NoteProperty -name "Computer" -value $evt.Event.System.Computer
```

```
    If($evt.Event.UserData.RuleAndFileData.TargetUser) {
      $sid = New-Object System.Security.Principal.SecurityIdentifier $evt.Event.UserData.RuleAndFileData.TargetUser
      $ALevent | Add-Member -type NoteProperty -name "User" -value $sid.Translate([System.Security.Principal.NTAccount])
    }
    $ALevent | Add-Member -type NoteProperty -name "File Path" -value $evt.Event.UserData.RuleAndFileData.FilePath
    If($evt.Event.UserData.RuleAndFileData.Fqbn) {
      $Fqbn = $evt.Event.UserData.RuleAndFileData.Fqbn.Split("\")
      $ALevent | Add-Member -type NoteProperty -name "Publisher" -value $Fqbn[0]
      $ALevent | Add-Member -type NoteProperty -name "Product" -value $Fqbn[1]
      $ALevent | Add-Member -type NoteProperty -name "File Name" -value $Fqbn[2]
      $ALevent | Add-Member -type NoteProperty -name "File Version" -value $Fqbn[3]
    }
    $ALevent | Add-Member -type NoteProperty -name "File Hash" -value $evt.Event.UserData.RuleAndFileData.FileHash
    $ALevent | Add-Member -type NoteProperty -name "Rule Type" -value $evt.Event.UserData.RuleAndFileData.PolicyName
    $ALevent | Add-Member -type NoteProperty -name "Rule Name" -value $evt.Event.UserData.RuleAndFileData.RuleName
    $ALevent | Add-Member -type NoteProperty -name "Rule Condition" -value $evt.Event.UserData.RuleAndFileData.RuleSddl
    $ALevent | Add-Member -type NoteProperty -name "Message" -value $evt.Event.RenderingInfo.Message
    if ($PSBoundParameters.ContainsKey('OutPath')) {
      $ALevents += $ALevent
    } else {
      Write-Output $ALevent
    }
  }
  END {
    if ($ALevents) {
      $ALevents | Export-Csv -Path $OutPath -Encoding Unicode –NoTypeInformation
    }
  }
}
```

9.2 Function "Export-AppLockerRules"

The function **Export-AppLockerRules** exports AppLocker rules to the specified XML file. It also sets the enforcement mode for each rule type to "Not Configured", as the enforcement mode will be set by a dedicated GPO as described in section 5.5.2.3.
The function takes the following inputs:

- The path and name of the XML file that will be created. Note that you can use path variables such as $pwd and $home, or enter the absolute path of the file. This parameter is mandatory and the user will be prompted for it if not supplied on the command line.
- The rule prefix (without the ":" that is automatically appended when the rule is created). If the –All switch is provided, any prefix entered will be ignored and all rules will be exported. If a rule prefix is not provided on the command line (and the –All switch is not provided), the user will be prompted for the rule prefix.

- The –Effective switch configures the function to read rules from effective policy (local policy plus any GPOs that are in scope). If this switch is omitted, rules are read from local policy only.
- The –All switch configures the function to ignore the rule prefix (if provided) and to export all rules to the XML file.

```
function Export-AppLockerRules {
<#
 Inputs:
   Path to output XML file (can use path variables). If not specified, the user will be prompted for it (specify absolute path).
   Rule prefix (without the ":"). If not specified (and the -All switch is not provided), the user will be prompted for it.
   If the -Effective switch is provided, rules will be read from effective policy. If not, rules are read from local policy.
   If the -All switch is provided, all rules will be exported. If not, only rules matching the prefix will be exported.
 Output: Rules saved to specified XML file.
#>
 param (
   [string]$OutPath = (Read-Host “Enter absolute path to XML file”),
   [string]$RulePrefix,
   [switch]$Effective,
   [switch]$All
 )
 If (!$RulePrefix -and !$All) {
   $RulePrefix = (Read-Host “`nEnter rule prefix (without the `":`") or cancel and rerun with the -All switch”)
 }
 If ($Effective) {
   [xml]$xml = Get-AppLockerPolicy -Effective –Xml
   Write-Host "`nExporting rules from Effective policy"
 } else {
   [xml]$xml = Get-AppLockerPolicy -Local –Xml
   Write-Host "`nExporting rules from Local policy"
 }
 If (!$All) {
   Write-Host "Exporting rules with prefix `"$RulePrefix`:`"`n"
   $xml.AppLockerPolicy.RuleCollection | ForEach-Object {$_.EnforcementMode = “NotConfigured”}
   $nodesToRemove = $xml.SelectNodes("//FilePublisherRule[not(starts-with(@Name,'${RulePrefix}:'))] |
//FilePathRule[not(starts-with(@Name,'${RulePrefix}:'))] | //FileHashRule[not(starts-with(@Name,'${RulePrefix}:'))]")
   $nodesToRemove | ForEach-Object {
     [void]($_.ParentNode.RemoveChild($_))
   }
 } else {
   Write-Host "Exporting ALL rules`n"
 }
 $xml.Save("$OutPath")
}
```

9.3 Function "Get-Sid"

The function **Get-Sid** takes the name of a user in the form **contoso\user** or **user@contoso.com** and returns the corresponding SID.

```
function Get-Sid {
  param ([string]$user)
  $ntaccount = New-Object System.Security.Principal.NTAccount($user)
  return [string]$ntaccount.Translate([System.Security.Principal.SecurityIdentifier])
}
```