ANTIMALWARE PLATFORM WHITEPAPER

REQUIREMENTS FOR CONSUMER ANTIMALWARE APPS ON WINDOWS 10

Revision History

Date	Version
6/30/2015	Final 1.0 for Windows 10 RTM
11/24/2015	Final 2.0 for Windows 10 – Requirements through July 1, 2016

DOCUMENT CHANGES

Date updated	Description
November 24, 2015	• Ch. 5.2 - Updated details on UPAO end to end flow, for clarification • Ch. 7 - Revised details on Antimalware App and the Windows 10 upgrade, for clarification • Ch. 8 - Included details on Windows Store app remediation process flow • Ch. 9 - Added new Requirements for Using Notifications in Windows 10 for July 2016 enforcement • Ch. 13.2 and Appendix J - Included details on the Performance Requirements and tests for July 2016. • Ch. 14.2 – Revised Antimalware Application policies and compliance definitions • Ch. 15 – Revised compliance definition for only using documented APIs for July 2016 • Ch. 20 - Added Windows 10 API deprecations

1 TABLE OF CONTENTS

2 INTRODUCTION

The Antimalware Platform (AMP) is a Windows-based effort to ensure that all Windows 10 users are protected from malware by an antimalware app that provides synchronous protection using updated signatures. This effort has two important components: support security software antimalware app developers in creating antimalware apps that protect the user from malware; and develop features and components in Windows that support user protection when the user chooses not to use a third-party solution.

To this end, it is imperative that antimalware app developers have important and up-to-date information as to how their products fit into the overall user protection effort, and that these developers are enabled to develop antimalware products that not only protect the Windows 10 user but which also run efficiently and integrally with Windows 10.

This whitepaper is a collection of platform requirements and guidelines to help antimalware app developers develop their apps to run on Windows 10. Following these guidelines will also ensure a high-quality user experience for antimalware apps installed on Windows 10.

> The requirements outlined in this document map to the requirements specified in the Windows 10 Antimalware API License and Listing Agreement.

The goals of this document are to assist security ISV antimalware app developers to:

- Protect users across a broad set of scenarios, especially the core Windows 10 scenarios and features,
- Reduce the performance impact of antimalware apps on the user's PC, and
- Improve user experiences around renewals and notifications.

> The Windows 10 performance and user experience requirements in this document apply only to consumer antimalware applications. They are also intended to be recommendations for Enterprise antimalware applications at this time.

3 HOW TO USE THIS DOCUMENT

This whitepaper is divided into two sections: introductory and detailed sections for engineering steps. Each of these sections contains (respectively) basic knowledge and appendices covering topics such as Windows Security Center, user notification, app remediation, etc. The document is designed to divide information into introductory and detailed sections so that one document can serve both management and engineering.

4 WINDOWS SECURITY CENTER REQUIREMENTS

NOTE: This section details the application development requirements for developing firewall, antivirus, and malicious software detection and removal applications that report status to Windows Security Center (WSC) in Windows 10.

Windows Security Center is the way that Windows communicates with users about the security state of their PC. In the context of this paper it is important because it informs the user if they are running an enabled and up-to-date antimalware app. Communicating the accurate protection state to WSC is required of any antimalware app that runs on Windows 7, Windows 8, Windows 8.1, and Windows 10 since inaccurate reporting will result in the user believing they are protected when they might not actually be protected. WSC also provides users with information as to how they can make their PC more secure, if the PC is not running an enabled or up-to-date antimalware app.

Since Windows 7, WSC has been integrated with the Action Center, and is used to display warning messages or notifications when the user is inadequately protected. Windows Security Center also has a public API which allows app developers to query the aggregated view of the PC's health state or retrieve specific details about the firewall, antivirus and antispyware programs that are registered. A sample that details how to use these APIs is available on MSDN.

The Windows Security Center background service (WSCSVC) is the Action Center component that monitors the PC's security settings and services (see **Error! Reference source not found.**). When an antimalware app reports a change of state using either the antimalware app APIs or a direct handle into the WSCSVC, WSCSVC will report the change to the Action Center. The user is notified via the Action Center UI in the form of a notification and/or a message in the Control Panel. The security state of each of the services and settings that WSCSVC monitors are stored using Windows Management Instrumentation (WMI).

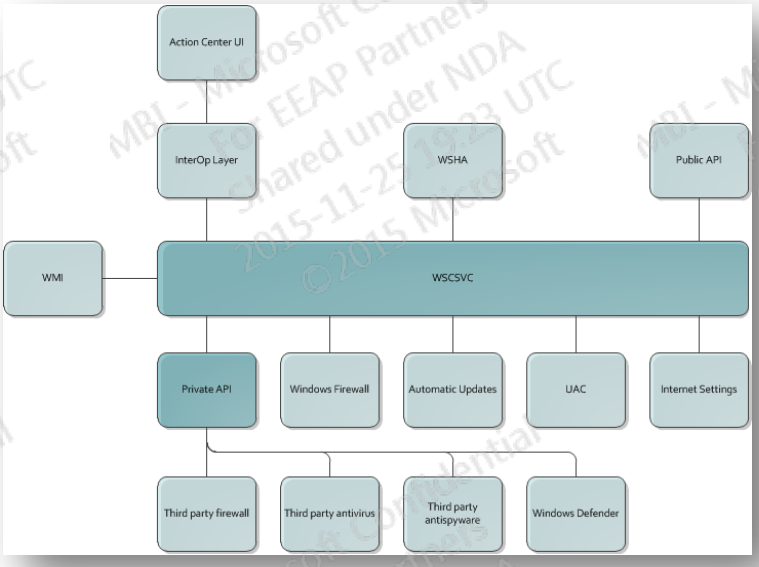


Figure 1: Structure of the Action Center and the Windows Security Center Service

4.1 REQUIREMENTS AND RECOMMENDATIONS FOR ANTIMALWARE APPS IN WINDOWS SECURITY CENTER

The table below lists what Windows Security Center requires from antimalware apps.

#	Requirement
1	Antimalware apps must report "on", "off", "snoozed" and "expired" states. *On: the antimalware app is providing malware protection. If the app provides real-time protection, then it must be enabled.* *Off: the antimalware app has stopped providing malware protection and any real-time scanning is disabled.* *Snoozed: the antimalware app has temporarily stopped protecting against malware until the next reboot or until the user explicitly changes the state.* *Expired: the antimalware app has deliberately stopped providing the user and the PC with the level of protection that would be performed with an installed and activated product including real-time protection and regular updating of signatures.*
2	In addition to the "on", "off", "snoozed" and "expired" states, antimalware apps must also report "up to date" and "out of date" states. *"Up to date" and "out of date" are Boolean flags that the app reports to notify the user of the current freshness of its malware signatures. The difference between "expired" and "out of date" is that the former is deliberate. For "expired," the software has deliberately stopped getting signatures. For "out of date" the software should be trying to get updates, but is not able to.*
3	Antimalware apps must report that they are "up to date" after signatures have been updated.
4	Antimalware apps must report that antivirus signatures are "out of date" when signatures are older than 7 days.
5	Once an antimalware app is expired, it must report an "expired" state. *However, if an antimalware app downgrades itself to a 'free' version and continues to protect the user with the level of protection that would be performed with an installed and activated product including real-time protection and regular updating of signatures, the antimalware app should not report as expired.*
6	Antimalware apps must support and properly implement the switches for /disable, /enable, /renew and /update.
7	When the app is expired and WSC calls the app with the /disable switch, the antimalware app must turn itself off and suspend real-time and background scanning for antivirus and antispyware. *If the app is not expired then it may ignore the /disable call.*
8	When /disable is called, no UI may be displayed during the /disable process.
9	When /disable is called, the app must report OFF in WSC. *If the antimalware app is registered as antivirus and antispyware, both must report as OFF.*
10	When /disable is called, the app must disable itself in less than 10 seconds.

#	Requirement
	If the app takes longer than 10 seconds, WSC will assume that disable has failed and prompt the user to uninstall the app.
11	If the antimalware app is reporting Off and expired and /enable is called, the app must report as expired in WSC.
12	When /renew is called, the app must provide a method for users to renew or update the app.
13	Once an app is renewed, it must report the correct state to WSC.
14	When antivirus and antispyware are both included in a bundled antimalware product, there must be only ONE remediation EXE provided to manage both of these components.
15	The antimalware app must NOT disable Windows Defender via registry keys or any other direct methods. Once the antimalware app registers with WSC, WSC will disable Windows Defender.
#	Recommendations
1	The antimalware provider should report an easily identifiable and distinct product name to WSC. This allows the user to reasonably identify the provider of the antimalware app in OS dialogs.
2	The antimalware provider should disclose to mmpcvia@microsoft.com the exact name used for all products registered with WSC, including any OEM version of their product in which they are in control of the product name reported to WSC.

4.1.1 REPORTING THE "EXPIRED" STATE

Microsoft requires antimalware apps to report "expired" once the app stops providing the user and the operating system with the level of protection that would be performed by an installed and activated product including real-time protection and regular updating of signatures. The app should report expired if it downgrades core protection functionality such as:

- Stopping real-time protection, or
- Stopping automatic update of virus signatures.

When the app reports this status to Windows Security Center, the user will be shown a red warning tile and asked to renew their subscription with the app. WSC allows antimalware apps to report the expired state by invoking the `UpdateStatus` API functions by setting the `productState` attribute to `WSC_SECURITY_PRODUCT_STATE_EXPIRED` (App subscription expired).

When the antimalware app has met these conditions it must report expired (`WSC_SECURITY_PRODUCT_STATE_EXPIRED`) and continue to report as expired until the remediation executable is called with the /disable switch or the user has renewed their subscription.

If the app continues to provide a full level of protection after the app subscription has expired, then it does not have to report as expired.

Signatures out of date are not considered to be expired unless the antimalware app deliberately stops automatic update of signatures for the app. A good example of this is if the user is away from their PC for a long period of time (e.g., on vacation). When the user turns on their PC for the first time after this type of

hiatus, the signatures may be out of date initially, but the user should receive all updates via the usual method soon thereafter.

4.1.2 SUPPORTING THE /DISABLE SWITCH

When an antimalware is in expired state, the app may be called to disable itself by WSC. The antimalware app must turn itself off and suspend real-time and background scanning for antivirus and antispyware. If the app is not expired, then it may ignore the /disable call.

When the antimalware app is called with this switch it must perform the following:

- Disable real-time and background scanning on the system related to malware protection.
- Ensure that the entire process is transparent to the user, meaning no UI (e.g. pop-ups, main app interface) is displayed during the disable process. The user must not be prompted to make changes or take any actions.
- Once the app has completed disabling itself, it must invoke the UpdateStatus API functions by setting the `productState` attribute to “off” (`WSC_SECURITY_PRODUCT_STATE_OFF`)**.** If the app also registers as an **antispyware** app it must also report as “off”.
- The app should disable itself within **10 seconds**. If it takes longer than this, Windows Security Center will assume that disable has failed and prompt the user to uninstall the app.
- After the app has been called with the /disable switch **and the app is still expired**, if it is called with the /enable switch or if the user tries to turn it on, the app should report back as expired to WSC.

The app should only accept the /disable call when it is in the expired state. If for some reason the app is called with /disable while the app is in another state it is not necessary to perform the actions described above.

In order to ensure that an administrative-level user on the machine is authorizing this action, the app should request user elevation before performing the disable operation. See User Account Control Recommendations (UAC) for more information on how to perform this task.

4.1.3 SUPPORTING THE /RENEW SWITCH

After the antimalware app has expired, Windows Security Center may call the remediation executable with the /renew switch if the user choses to renew the app through Action Center or some other interface. When the antimalware app is called with this switch it must perform the following:

- Provide a way for users to renew their app subscription, either via a UI or a link to an online renewal.
- Once renewed, the app must stop reporting “expired” and report the current state (“on”, “off” or “snoozed”).

Similar to the behavior required for the /disable switch, the user should be prompted to enter their credentials via the UAC when the /renew switch is called. The UAC may be called from the antimalware app in this case.

Once the user has renewed their subscription, the app should stop reporting as “Off” and report its current state as “On”. In the process Windows Defender will automatically be disabled in order to avoid potential problems that could arise with multiple antimalware apps running side by side.

4.2 ENGINEERING RESOURCES

There are three appendices included in this document that provide additional information for development of apps with WSC:

Appendix A: Develop an app to communicate with WSC
Appendix B: Best practices for developing WSC apps
Appendix C: Code signing app binaries
Appendix D: How WSC works in Windows10

5 USER PROTECTION ALWAYS ON IN WINDOWS 10

5.1 INTRODUCTION

It is important that all users are protected from malware at all times. The ''User Protection Always ON" scenario (UPAO) helps to ensure this for Windows 10 users. UPAO is built on the User Not Protected (UNP) feature work from Windows 8 and 8.1, which is also described in this white paper. The goals of this scenario are to ensure that:

1. Users are protected
2. Users are provided with the renewal of their existing antimalware app as the top user choice
3. Users are provided with clear and useful choices to get protected when their existing antimalware app expires.

This feature relies on the third-party antimalware app reporting its correct protection state to Windows Security Center (WSC). The way WSC functions and requirements for communicating with WSC are also documented in this whitepaper.

The UPAO experience has been designed to be available across all user experiences in Windows (not just via Windows Security Center in the Windows desktop), so users will be notified when they need to take action. The UPAO scenario experience behaviors are outlined in the following sections. Differences between the behaviors of Windows 8 versus Windows 8.1 are called out in separate sections.

UPAO has 3 components:

1. Antimalware apps **must make use of notifications that WSC provides** to notify users that the app may expire soon, and enable users to proactively take action to renew their subscription. Antimalware apps should exclusively use the UPAO functionality for the scenarios it supports and ensure redundant prompting does not occur due to duplicative UX in the third-party antimalware app. An antimalware application must notify the user through the API provided below about entering expiry and the renewal needed. An antimalware application must not notify the user outside of this API.
2. User protection from malware is on at all times. Windows Defender will be automatically turned ON when the antimalware app expires.
3. WSC will notify users that their antimalware app will expire soon, and encourage users to renew their antimalware subscription. Enabling Windows Defender when the user's antimalware app expires is to ensure that antimalware protection for Windows users is always on.

5.2 UPAO END TO END FLOW

Note: Day 0 is defined here as the day when the antimalware app reports its status as expired to WSC.

Phase 1: An antimalware app will expire soon. The app must call WSC to show a series of renew notifications from the OS to the user. Here is the functionality available to the app for this scenario:

- Windows notifies the user via a toast notification for up to 4 days (day -5 to day -2)
- Windows notifies user through a modal System Dialog on last day before expiry (day -1)

Phase 2: The antimalware app expires (Day 0):

- Antimalware app must disable itself soon after going to expiry
- Windows turns ON Windows Defender.
- Windows notifies the user via a toast notification for the first 2 days that this change has taken place and offers the user an opportunity to renew (day 0 and day 1).
- Windows notifies the user through a modal System Dialog on the 3rd day since expiry and offers the user an opportunity to renew (day 2)

NOTE: The notifications within Phase 2 will continue to be shown until a user has renewed the antimalware app.

Phase 3: Exception case if the expired app does not disable itself

The following flow will apply in cases where the antimalware app is expired but fails to disable itself soon after going to expiry. In this case, Windows Defender cannot be turned ON and the user is left unprotected.

- Windows will notify the user via a system modal dialog to renew or uninstall the antimalware app on day 0
- Windows will continue to notify the user as above via a system modal dialog every 24 hours till the user takes an action to get protected

NOTE: If the antimalware app is registered as antivirus and antispyware, both must report as OFF soon after going to expiry. Turning off only one but not the other will trigger uninstall flow.

In the case when a user opts to renew their antimalware app, the app should notify WSC of its new state (turned on and up-to-date). At that point WSC will turn Windows Defender OFF (if the antimalware app has previously disabled itself after going to expiry).

5.3 UPAO NOTIFICATION DETAILS

5.3.1 NOTIFYING USERS BEFORE EXPIRATION

WSC provides a new API for antimalware apps to notify users 5 days prior to the app falling into an expired state. Antimalware apps that have registered with WSC can take advantage of this API to drive their renewal rates up and help keep users protected without any interruption of the user's subscription.

An antimalware application must notify the user through the API provided below about entering expiry and the renewal needed. An antimalware application must not notify the user outside of this API.

Here are the requirements for prompting users before the antimalware app expires:

1. The app must be active, enabled and not out of date.
2. The app must be close to its expiration date.
3. The app may invoke the UPAO scenario no more than twice in a 30-day period.

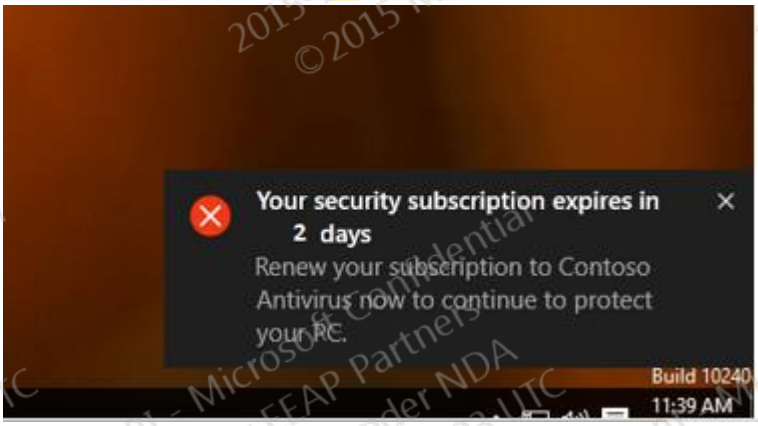
```
interface IWscAVStatus2 : IWscAVStatus
{
……
//
// This function will be called by the ISV reporting application to notify
// the user that their subscription going to expire soon.
```

```
//
HRESULT NotifyUserForNearExpiration(
[in] DWORD dwDaysUntilExpiration);
};
```

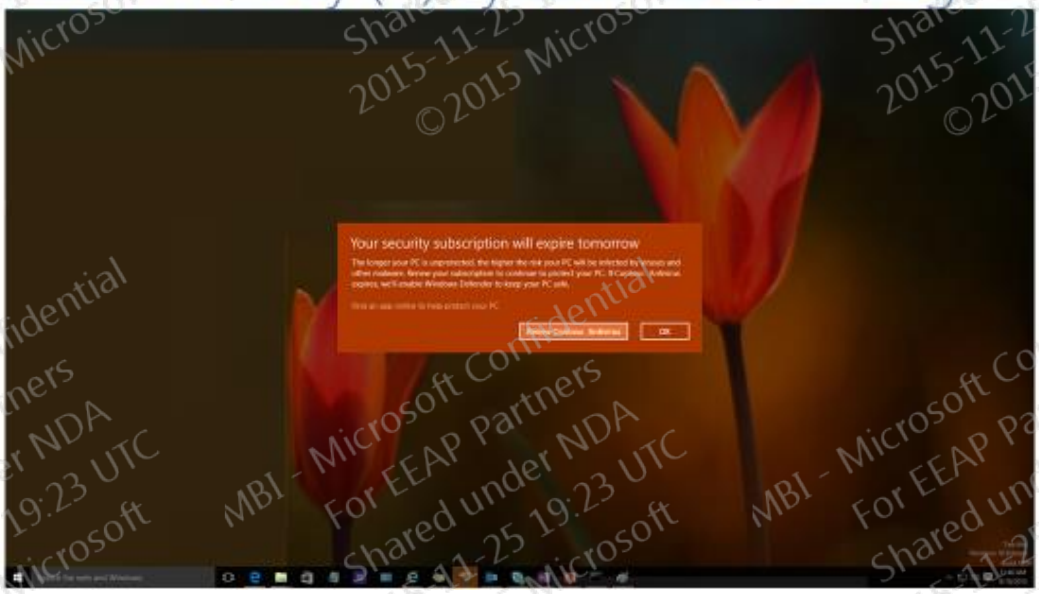
Days (-5 to -2) toast notifications	
Title	Contoso Avenger expires in N days
Message	Renew your subscription now to continue protection on your PC
User Action	**Toast Notification Action**
User clicks toast	WSC calls the Remediation EXE with '/renew' flag registered by the antimalware app.
User clicks (X) on toast	Notification is closed.
User does nothing	Toast notification will disappear after standard timeout.

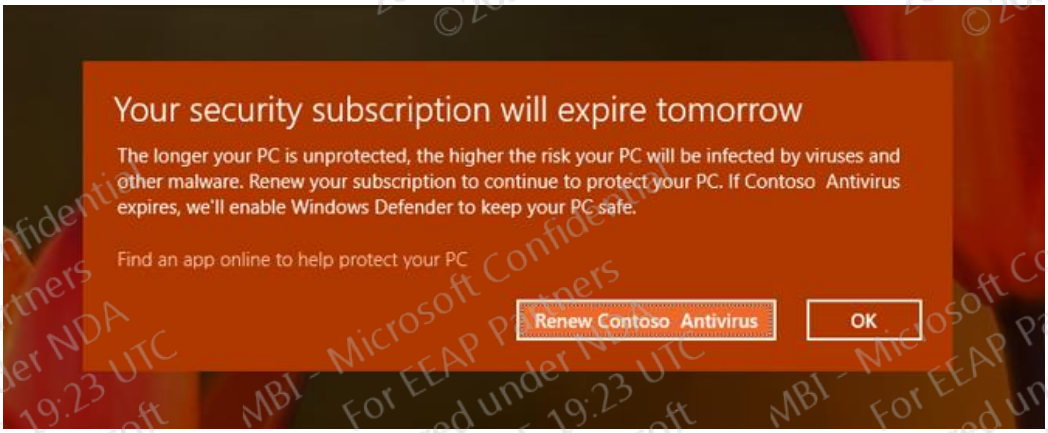
Day (-1) system modal notification	
Title	Your subscription to Contoso Avenger will expire tomorrow
Message	The longer your PC is unprotected, the higher the risk your PC will be infected by viruses and other malware. Renew your subscription to continue protection on your PC. If Contoso Avenger expires, we'll enable Windows Defender to help keep your PC protected.
Buttons	**Button Action**
Renew Contoso Avenger (this button has focus)	WSC calls the Remediation EXE with '/renew' flag registered by the antimalware app.
OK	Dialog closes. WSC will not notify the user again that their antimalware app will expire soon.

Windows 10: Day (-5 to -2) Toast Notifications



Windows 10: Day (-1) System Modal Dialog





5.3.2 ANTIMALWARE APP EXPIRATION

An antimalware app must report its state as 'expired' as per the requirements listed in Reporting the "expired" state section. Once WSC has received this expired state from the antimalware app, the app must proactively disable its real time protection as described in Supporting the /disable switch. Note that the antimalware app will not receive an explicit call to 'disable' but must always disable within a minute after reporting its state as 'expired'. This facilitates turning ON Windows Defender so that users are protected at all times. WSC waits for one minute before calling Windows Defender to turn ON when a third-party antimalware expires. If Windows Defender cannot be turned ON because an expired antimalware app has not disabled itself, then WSC will prompt the user to uninstall the antimalware app.

NOTE: If antimalware app never expires but has a model where on subscription expiry moves a premium user to a basic version then such app need not report its status as expired. AM ISV must make sure that the basic version offered meets the industry level requirements as explained for real time protection in Section 4.1.1.

Below is the step by step flow.

1. Subscription to an antimalware app expires.
2. The antimalware app reports its state as expired to WSC.
3. WSC updates its information about the antimalware as expired.
4. The antimalware app proactively disables itself without waiting for an explicit disable call from WSC.
5. WSC turns on Windows Defender to ensure users are protected.
6. Windows Defender is successfully turned ON.
7. If Windows Defender cannot be turned ON because an expired antimalware app has not disabled itself, users are prompted to uninstall the antimalware app.

5.3.3 NOTIFYING USERS AFTER EXPIRATION

Once an antimalware app has expired, Windows Defender is turned ON to ensure continuous user protection. WSC encourages users to renew their antimalware app subscription through notifications and system modal dialogs for 3 days after expiration has been reported, even though Windows Defender is ON.

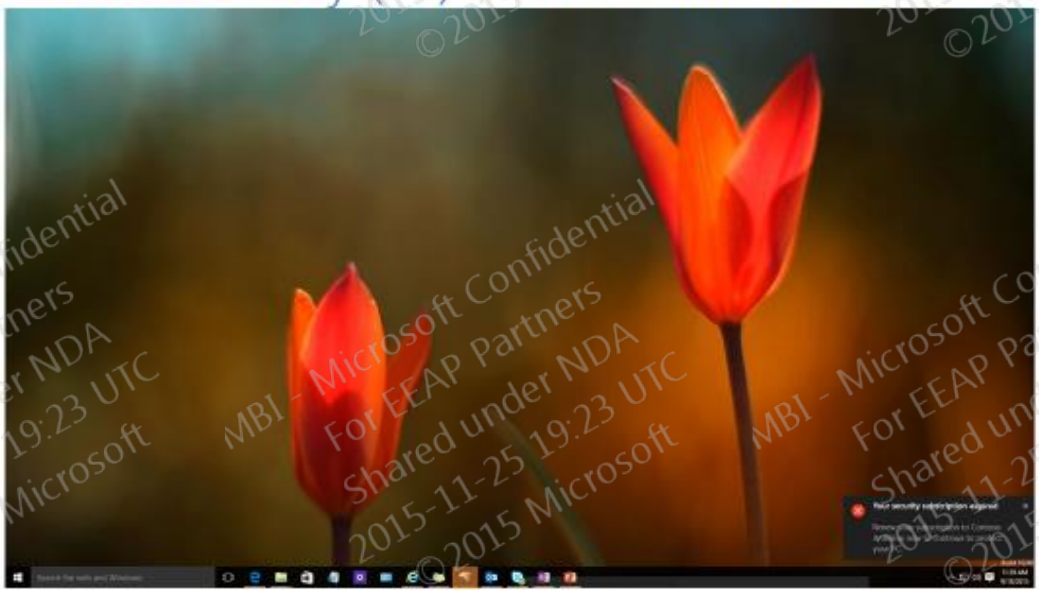
Day (0 and 1) toast notifications	
Title	Contoso Avenger expired
Message	Renew your subscription to help protect your PC. We turned on Windows Defender to help protect your PC in the meantime.
User Action	**Toast Notification Action**
User clicks toast	WSC calls the Remediation EXE with '/renew' flag registered by the antimalware app.
User clicks (X) on toast	Notification is closed.
User does nothing	Toast notification will disappear after standard timeout.

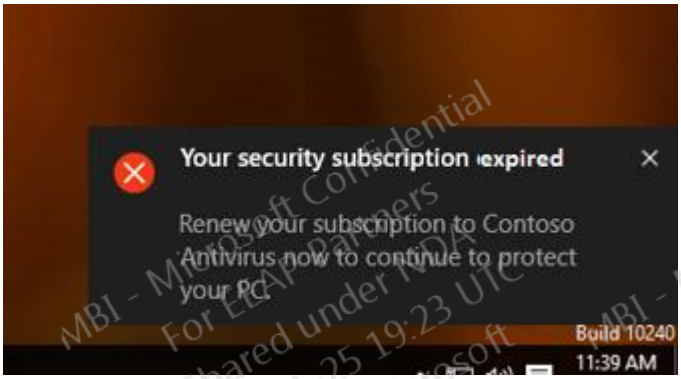
Uninstall System Modal notifications	
Title	Renew your subscription to Contoso Avenger
Text Message	Your subscription to Contoso Avenger ran out. Either renew the subscription or uninstall the security software. When you uninstall, Windows Defender will be automatically turned on to help protect your PC.
Buttons	**Button Action**
Renew Contoso Avenger (this button has focus)	WSC calls the Remediation EXE with '/renew' flag registered by the antimalware app.
Uninstall	Dialog closes. Programs and Features is opened automatically so users can manually uninstall the expired antimalware app.

Day (2) system modal notification	
Title	Renew your subscription to Contoso Avenger
Message	Your subscription to Contoso Avenger ran out. We activated Windows Defender to protect your PC. We'll deactivate Windows Defender when you renew your subscription to Contoso Avenger
Buttons	**Button Action**
Renew Contoso Avenger (this button has focus)	WSC calls the Remediation EXE with '/renew' flag registered by the antimalware app.

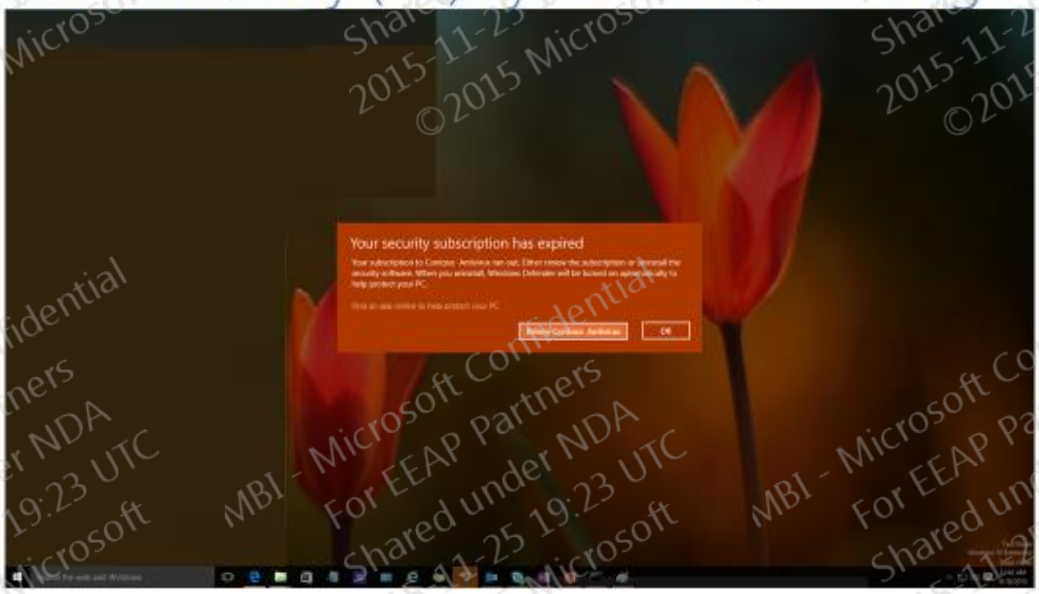
OK	Dialog closes. WSC will not notify the user again that their antimalware app has expired.

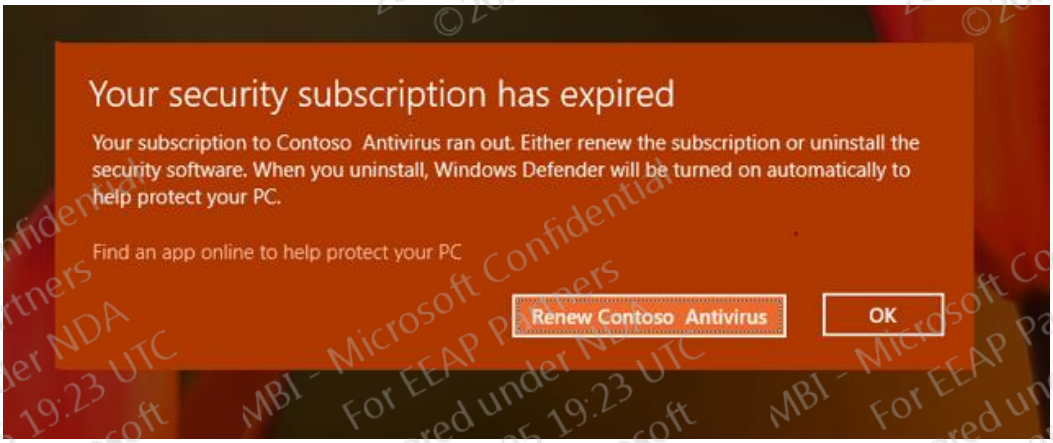
Windows 10: Day (+1) Toast Notifications



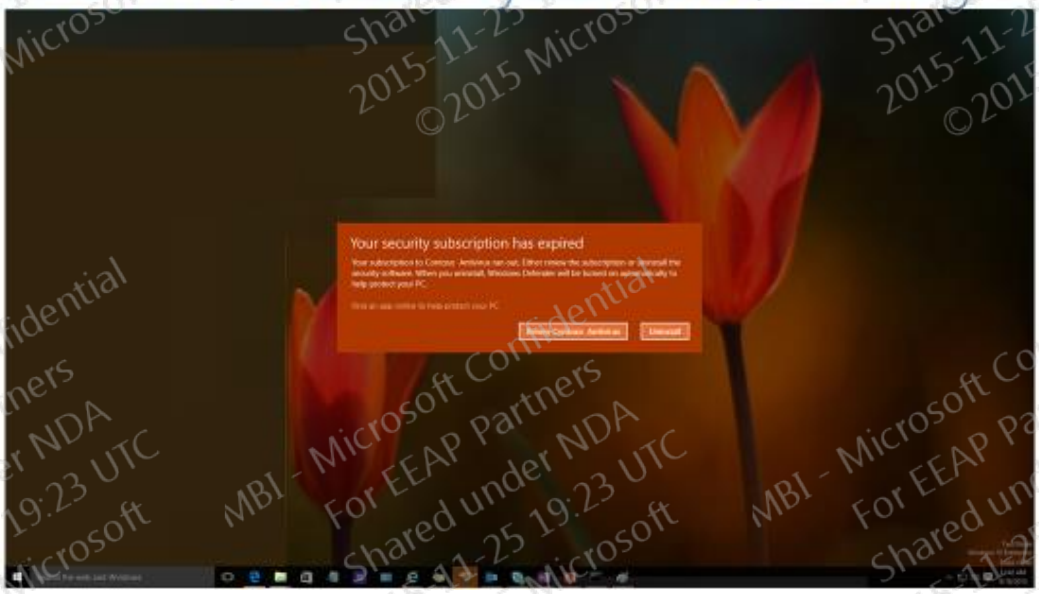


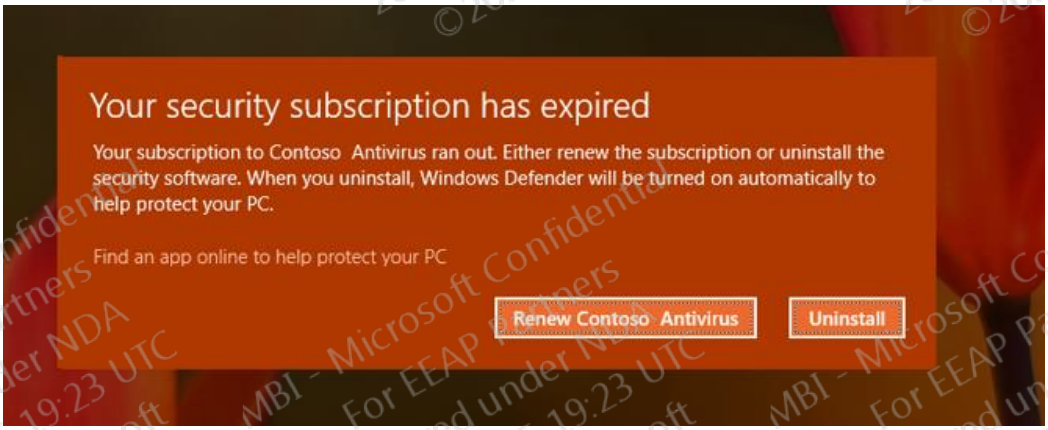
Windows 10: Day (+2) System Modal Dialog





Windows 10: Uninstall System Modal Dialog





6 WSC SUPPORT FOR OFFLINE CLEANING THROUGH INBOX WINDOWS RE

Offline cleaning is often required to remediate systems which have been deeply infected. WSC provides APIs for AM apps to make use of inbox Windows RE (Windows Recovery Environment – most Windows PCs come with Windows RE) for seamless and frictionless running of AM app's offline scanners.

6.1 OFFLINE SCANNING/CLEANING WORKFLOW:

1. WSC will accept file path of an exe (which is responsible for offline cleaning from Windows RE) from a registered AM.
2. Binary (offline scanner) whose path is passed above must be WHQL signed (signing is done through SysDev portal – same as ELAM signing).
3. WSC will pass the path to the exe (offline scanner) to Windows RE, set BCD settings to modify to boot into Windows RE and reboot the machine.
4. Upon reboot, system will boot into Windows RE, if device is encrypted by BitLocker then will automatically decrypt OS volume (if auto unlocking[1] is set in BitLocker settings), and then launch AMs registered offline scanner.
5. Offline scanner will take over, and do an offline scanning and cleaning, once done reboot the system.
6. Upon reboot system boots back into main OS.

6.2 AM'S OFFLINE SCANNERS REQUIREMENTS:

- AMs can get their offline scanner EXE signed by following the same process they follow for ELAM certification. Offline scanner EXE must be included as part of the ELAM driver package for signing.
- WSC and Windows RE both verify the WHQL signature on Offline scanners before consuming/launching the EXE.
- Requirements for AMs to get their offline scanner signed by Microsoft are:
  1. AMs must meet all the requirement that of ELAM (link to ELAM details).
  2. AM Offline Scanner binary must be a separate binary which is different from the one calling into WSC for AM registration.
  3. AM Offline Scanner binary must not violate BitLocker requirements (not provide any kind of command prompt or UI[2] before BitLocker decrypts the volume or disable BitLocker).
- Offline scanner can load or use additional dlls or exes, and need not bundle all of them into one exe. In addition, can also show UI to user for guidance and consent.
- AMs can invoke offline scanning anytime needed, but before doing it AM must provide context to user for the needed reboot into Windows RE through its UI or notifications.
- AMs can send in additional command line arguments for while launching the offline scanner EXE in Windows RE

[1] Automatic unlock is only supported on devices with TPM. If automatic unlock is not possible, Windows RE will show UI prompting the user to enter the 48-digit recovery key.

[2] The offline scanner cannot expose any UI (including a CMD prompt) that gives a user access to the file system. Progress indicator UI is acceptable.

- Only active and enabled AM are allowed to call into WSC to invoke Offline scanning.

6.3 API DETAILS

```
interface IWscAVStatus2 : IWscAVStatus
{
    //
    // This function will be called by the ISV reporting application to
    // initiate an offline scanning in Windows RE(Windows Recovery
Environment)
    //
    HRESULT InitiateOfflineCleaning(
                [in] BSTR bstrOfflineScanner,
                [in] BSTR bstrCmdLine);
……
};
```

7 ANTIMALWARE APPS AND THE WINDOWS 10 UPGRADE

7.1 BACKGROUND

In Windows 8.1, a new feature was introduced as part of the upgrade process. Incompatible apps that blocked the upgrade process to Windows 8.1 were not migrated forward to Windows 8.1. Migration implies that the app may still exist in the old image but will not be present in the newer version of the operating system. Specifically, this technology was applied for apps that block or hinder upgrade, or apps that have filter drivers or services that would crash on the newer version of the operating system. Antimalware apps generally fall into the latter category.

7.2 WINDOWS 10 UPGRADES

The migration technology introduced in Windows 8.1 is being used for Windows 10, but with some changes to the upgrade process so that users may keep their preferred third party antimalware software if a Windows 10 compatible version is available. The migration solution for Windows 10 satisfies two goals:

- Ensure that Windows 10 users are adequately protected from malware and threats.
- Enable users to maintain their third-party antimalware apps after upgrade.

The Windows 10 upgrade experience uses an updated upgrade executable ("upgrade.exe") that is capable of triggering the installation of an up-to-date and compatible version of the antimalware app through a file that contains user settings/license information. An incompatible antimalware app (e.g., on a user's Windows 7 PC) is not migrated forward to Windows 10; however, the user license info and upgrade executable (capable of installing the Windows 10-compatible antimalware app after upgrade) is migrated. After the user upgrades to Windows 10, the antimalware app can retrieve the compatible version of the antimalware app and apply the stored user data (settings/license information).

Antimalware apps that are compatible with Windows 10 will be carried forwarded to Windows 10 during the OS upgrade.

The upgrade.exe process is available for antimalware apps that are not compatible with Windows 10 and are removed before the Windows 10 upgrade for compatibility reasons. Incompatible antimalware apps that have a Windows 10 compatible version, and supply an upgrade.exe, can provide users with the compatible version of the app after upgrade is complete. If there isn't a Windows 10 compatible app available, then Windows Defender will be enabled during upgrade to ensure that users have adequate protection after they have completed the upgrade to Windows 10.

The following table provides details on the upgrade state for compatible and incompatible apps based on the current AV status of the app.

AM Status	If the AM software is compatible[3] with Windows 10 prior to upgrade	If the AM software is Incompatible with Windows 10 prior to upgrade
Active (On)	AM software survives OS upgrade process intact and maintains the current state.	The upgrade.exe and .cab file are migrated forward. Once the OS upgrade process has completed, upgrade.exe is invoked to install the Windows 10 compatible version (post upgrade.)
Expired	AM software survives OS upgrade process intact and maintains the current state.	AM software is removed. The AM software does not get migrated forward if it is in the expired state.
Snoozed	AV software survives OS upgrade process intact and maintains the current state.	The upgrade.exe and .cab file are migrated forward. Once the OS upgrade process has completed, upgrade.exe is invoked to install the Windows 10 compatible version (post upgrade.)
Off	AM software survives OS upgrade process intact and maintains the current state.	AM software is removed. The AV software does not get migrated forward if it is in the off state.
Out-of-date	AM software survives OS upgrade process intact and maintains the current state.	The upgrade.exe and .cab file are migrated forward. Once the OS upgrade process has completed, upgrade.exe is invoked to install the Windows 10 compatible version (post upgrade.)

There may be edge cases when an antimalware app does not survive the upgrade process or the upgrade.exe process does not work as intended due to unresolved software issue(s). It is incumbent on the antimalware app developer to ensure that it has tested the upgrade experiences for its application from Windows 7 to Windows 10, Windows 8 to Windows 10, and Windows 8.1 to Windows 10 and have resolved all blocking issues.

7.3 PROCESS GUIDELINES AND REQUIREMENTS

The following process outlines how the update and upgrade will work on a user's PC. A high level development timeline to achieve this process is outlined in the Timeline section.

7.3.1 STEP 1: UPDATE UPGRADE EXE DOWN-LEVEL (BEFORE UPGRADE)

The third-party antimalware app developer creates an "upgrade" executable ("upgrade.exe") and distributes it to its end users who are running previous versions of Windows (i.e., Windows 7 SP1 or Windows 8.1) so that the antimalware app is ready when a user chooses to upgrade to Windows 10. Antimalware apps will not receive an event to perform this update. This does not change the existing requirement that antimalware apps honor the disable call from Windows Security Center (WSC) when the antimalware app is in the expired state.

[3] "Compatible" here means that the app runs on Windows 10, does not block upgrade, does not cause the system to fail or crash, and this app meets the criteria explained in Section 13 and the timeline explained in Section 14.3 of this Whitepaper.

Only antimalware apps that are enabled will be migrated. Antimalware apps that are expired, off, or disabled (as reported down level) are not eligible for migration using this process.

Upgrade.exe and Upgradedata.cab must be placed under a prescribed folder structure that WSC can locate:

- An "upgrade" executable for the antimalware product to update itself to the Windows 10-compatible version, placed in the following path : `c:\Program Files\Common Files\AV\<Antimalware app name that is registered in WSC>\Upgrade.exe`
- A *.cab file that contains user license information and settings needed to preserve a user's license and PC settings information that should be migrated to Windows 10, placed in the following path: `c:\Program Files\Common Files\AV\<Antimalware app name that is registered in WSC>\userdata.cab.`
- Any other files that are required for upgrade may be placed in this folder, there is no size limitation on the folder but we recommend that this be as lightweight as possible.

Note: To clarify any ambiguity or confusion between resolution in 32 bit OS and 64 bit OS, use system variable %CommonProgramFiles% to locate "`c:\Program Files\Common Files\`"
For e.g.: `%CommonProgramFiles%\AV\<Antimalware app name that is registered in WSC>\Upgrade.exe`

Requirements for creating the folder structure and upgrade.exe:

1. As part of the file path, the name of the folder must be an exact match of what is registered in WSC as the product name.
   i. If multiple locale names exist as product name to support localization then the localized product name that is registered in WSC must be used.
   ii. If product name contains any characters that are not supported as valid characters for folder name such as \/"*?"<>| then such characters must be filtered out while naming the folder with the product name.
2. Antimalware apps must manage creating, collecting and applying user data directly from the *.cab.
3. Upgrade.exe must be signed with the same certificate (exact same certificate as WSC remembers identity of the app through hash of the certificate only) and the same way of page hash signed as antimalware app's Reporting EXE (and often same as Remediation EXE).
4. Upgrade.exe must be generic and product agnostic. The only purpose this executable serves is to fetch the newest (compatible) version of the antimalware app after the Windows 10 upgrade. It is the responsibility of antimalware ISVs to ensure compatibility is tested prior to downloading any new versions of the app.

Recommendations:

1. Userdata.cab should only contain user data/settings and be encrypted—please ensure that user data is protected at all times.
2. Only signed and verified code and data should be loaded from this folder.

7.3.2 STEP 2: UPGRADE

The user starts the upgrade process. The incompatible app is not migrated forward, but the documented files (see steps 1 & 2) are migrated forward to Windows 10.

7.3.3 STEP 3: UPDATE ANTIMALWARE APP

After the user has completed the Windows 10 out-of-box experience ("OOBE"), and after the user[4] logs in for the first time, Windows 10 will show a toast notification within 3-5 minutes after OOBE and then again 1 hour after OOBE. And then once the user clicks it, it will invoke the upgrade.exe process so that the antimalware app can update itself, with a flag specifically set for this purpose. If user misses the notification or dismisses the notification, then Windows 10 will show the notification again after the next reboot until user has taken an action (Action of clicking the toast to reach upgrade.exe). This is done for all users on the device until at least one of the users on the device has taken action.

The upgrade.exe will be called with the /upgrade switch. For example:

```
Upgrade.exe /upgrade
```

`/upgrade:` This is a new switch that is added for the post-upgrade scenario. When called with this switch, the app should detect the OS version and download the compatible binaries for the OS. The antimalware app should be installed and registered with WSC.

The antimalware app upgrade.exe should retrieve the Windows 10-compatible version of the app, and apply user data from userdata.cab. The user must be prompted for UAC during this process.

After the app has been installed, it must register with WSC (see Resources for more information). If the third-party antimalware app is invoked when there is no network available, or for whatever reason the download fails, then the upgrade.exe must retry at a later time. Windows 10 will not invoke this executable again.

7.3.4 REQUIREMENT: NO ADDITIONAL SOFTWARE TO BE BUNDLED BY UPGRADE.EXE

If you use the process described above to upgrade your antimalware application to Windows 10 (i.e., through the "upgrade.exe" process), you may not bundle or install any additional features or software with your antimalware application, such as, toolbars, adware, web browsers, or any other separate applications. Rather, upgrade.exe should be used only to re-install the versions of the antimalware application(s) that were on customers' systems before the upgrade to Windows 10. Other features or software that may have been bundled with your antimalware application and installed on customers' systems will be migrated forward if they are otherwise compatible with Windows 10. Upgrade.exe must not be used as a vehicle to install software other than the Windows 10 compatible version of the antimalware application that was previously installed and provided active protection on the user's device.

7.4 TESTING THIS SCENARIO

[4] If the user is a not the same as the original logged in user before upgrade, the first toast notification will only show up in the Notification Center, without displaying on the Desktop.

It is important that antimalware app developers test the upgrade scenarios for Windows 10, including support for migration from:

- Windows 7 RTM & Windows 7 SP1 -> Windows 10
- Windows 8 RTM -> Windows 10
- Windows 8.1 (including all subsequent updates) -> Windows 10

Antimalware app developers are expected to run full test suites when testing their app functionality, including the end-to-end upgrade scenarios. Following are the test exit requirements:

- The app will not block the Windows 10 upgrade experience
- The app test results post upgrade will match that of a pre upgrade experience
- If any issues are found through testing, antimalware app developers are asked to file bugs through their SysDev account.

7.5 UX ENGINEERING GUIDELINES

Antimalware app developers are required to follow these documented UX engineering guidelines to ensure users will have a consistent user experience on Windows 10.

```
upgrade.exe /upgrade
```

When launching or calling upgrade.exe with the '/upgrade' flag, the antimalware app must display additional information to the user via its custom UI regarding the need for downloading the Windows 10 compatible version of the antimalware app. This is required only for the upgrade.exe scenario; and is not required for downloading other updates to antimalware applications. applications This way, context is established for users to acquire the Windows 10-compatible antimalware app binaries. Any UI shown by the antimalware app at this point should contain a minimum of 2 buttons for user action.

- One button to continue download and install new binaries
- A second button to dismiss the dialog and remind users later about the action needed.

The second button is recommended, as there is no way for Windows 10 to remember to invoke upgrade.exe if the download fails, a network connection does not exist, or if the user does not have admin rights to install the app, etc. Antimalware apps should include a method to retrigger the installation flow if download fails or the user chooses to be reminded of this action later or if antimalware is not yet ready with a Windows 10 compatible version.

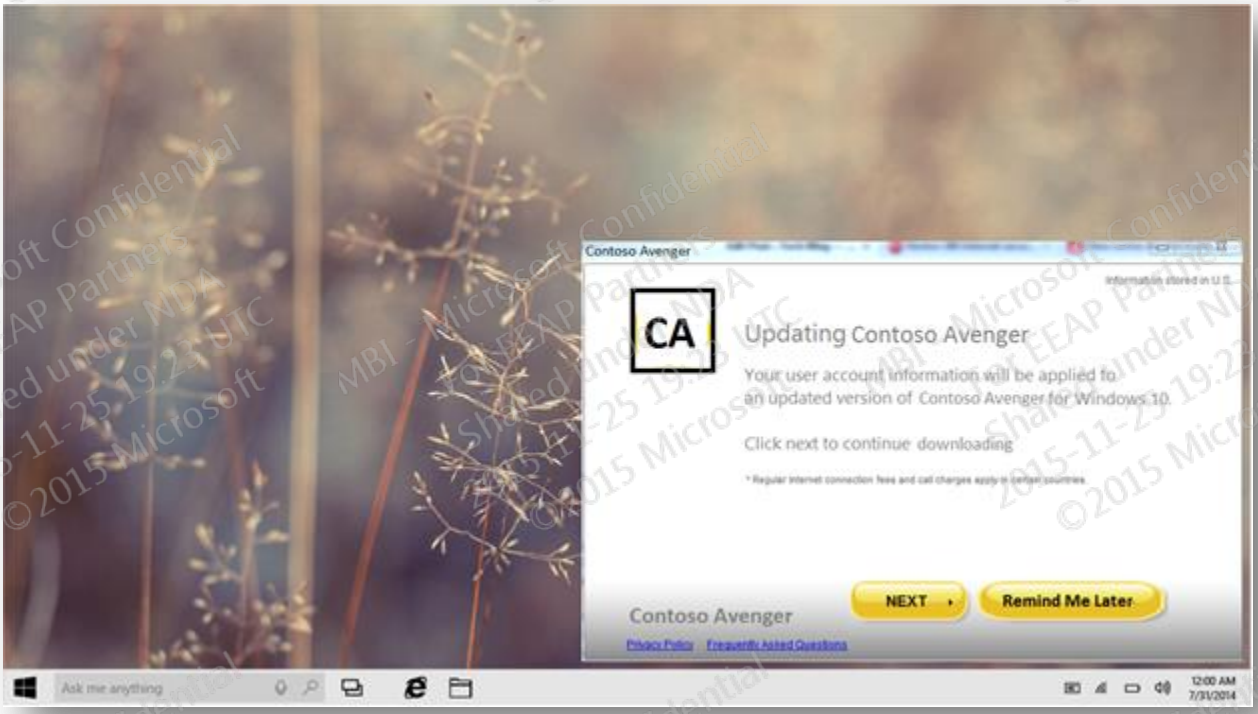


Figure 2: Sample antimalware app UI for triggering Windows 10-compatible app download

7.6 RESOURCES

- Some adjustments have been made to the switches used by antimalware apps for Windows 10. These are documented in the Appendix under Using Switches. Note that the '/upgrade' switch has been added for Windows 10.
- Antimalware apps must be registered with WSC. This is documented in Register with Windows Security Center. This section details how the antimalware app should be registered with Windows during setup or first run.

7.6.1 REGISTER WITH WSC

Once the Windows 10-compatible antimalware app components have been downloaded and installed, the antimalware app is required to register with WSC. During upgrade to Windows 10, the previous version of the app was removed (with the exception of upgrade.exe and usersettings.cab), and the WSC registration was cleaned during upgrade.

7.7 TIMELINE

The following table contains a high level development schedule that should be followed.

Milestone	Tasks/work needed & ownership
60-90 days before a new Windows 10 release	• Antimalware app developers should ensure that they are testing their antimalware apps on the latest release of Windows 10, including the upgrade process with upgrade.exe. • Antimalware app developers should ensure that bugs are filed via Sysdev during this process. Be sure to list supported version(s) of antimalware apps. • Antimalware app developers should also test compatible antimalware apps downlevel if the app is compatible across multiple versions of Windows • Microsoft and antimalware app developers will collaborate on test results, bug fixes, etc.
30 days before a new Windows 10 release	• Antimalware app developers ensure that downlevel PCs are prepared for upgrade, if migration files (upgrade.exe and userdata.cab) are used. • Microsoft will ensure that app removals do not include these migration files.
New Windows 10 release	• Users on previous versions of Windows may see notifications that they can upgrade to the latest release of Windows 10. After upgrade, users will see an additional notification that prompts them to install the compatible version of their AM product.

8 WINDOWS STORE APP REMEDIATION REQUIREMENTS

Removing an infection from an app is not always a straightforward process, and in many cases it is simply not possible to return the app back to its pre-infected state. The result, while good from a security point of view, is a broken app and a poor experience for the user.

Windows helps to solve this problem by providing a complete end to end remediation experience that can be leveraged by antimalware apps. By properly leveraging this platform functionality, antimalware apps can provide the user with the great level of protection they provide today as well as an equally great user experience. When an antimalware app removes an infection from a Windows Store app, it will use a mechanism to notify the platform, and the next time the user launches the app, they will be taken through an experience which ultimately results in the app being re-acquired and fixed.

This document provides guidance on how to notify the platform if malware has infected a Windows Store app as well as basic information about the remediation process for Windows Store apps. The intent is to provide antimalware app developers with an overview of how the process works, as well as specific examples for interacting with APIs that are provided for properly remediating Windows Store apps.

For more information see the code samples in Appendix E: App remediation code samples.

8.1 OVERVIEW

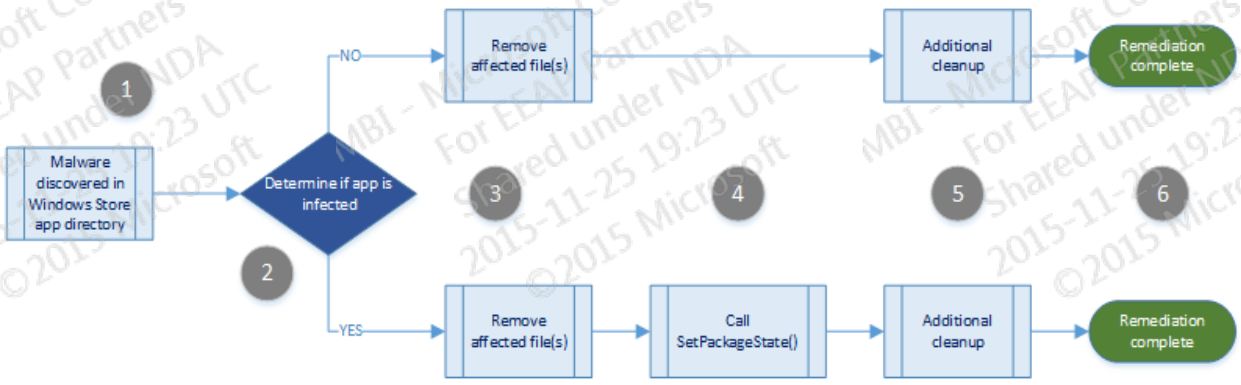


Figure 3: Windows Store app remediation flow

8.2 WINDOWS STORE APP REMEDIATION PROCESS FLOW

The process flow, as referenced in Figure 3:

1. Malware is detected on the local machine, in the Windows Store app package root (%programfiles%\WindowsApps).

2. Determine the list of Windows Store apps which should be examined
    a. Use PackageManager.FindPackages() API to determine all packages on the system (for all users)
    b. Use PackageManager.FindUsers() API to determine the state of the package for each user by looking at the PackageUserInformation.InstallState property
    c. If the PackageInstallState property is either Staged or Installed, the package should be examined further. Otherwise it should not be (app maybe currently installing, removing or paused).
3. Determine if a Windows Store app is infected.
    a. There is an extra file
        i. Validate the Block Map via its signature.
        ii. Determine if the suspect file is in the Block Map.
        iii. If the file is not in the Block Map, it should be removed.
    b. There is a missing file
        i. Validate the Block Map via its signature.
        ii. Determine if there is a missing file based on the Block Map.
        iii. If there is a missing file, then the Windows Store app is infected.
    c. There is a modified / infected file
        i. Validate the Block Map via its signature.
        ii. Determine if the suspect file is in the Block Map.
        iii. If the file is in the Block Map, yet has a different hash, then the Windows Store app is infected.
        iv. The file should be removed.
4. If a Windows Store app is infected, call SetPackageState() to disable the app and prepare the customer for remediation.
    a. The package identity for calling SetPackageState() is the name of the folder containing the app.
5. Complete any additional clean up tasks, and update Action Center.
6. Remediation completed (customer remediation will happen when customer next interacts with the affected app).

8.2.1 ADDITIONAL INFORMATION

Files that are identical between different packages which were acquired from the same source will not be duplicated on disk. This means that when two packages acquired from the Windows Store share the same file, only the first instance of the file is actually present on disk. All subsequent files are hard links to the first instance.

> When deleting an infected file, ensure that all instances of the file are removed, and SetPackageState() is called for each affected package.

Starting in Windows 8.1, there are two new package type in addition to the basic Windows Store app packages which are referred to as “Resource Packages” and “Bundle Packages”. These should be treated the same way as any other package when calling SetPackageState(). The package name for calling SetPackageState() is still the name of the folder containing the app.

The Resource Packages and Bundle Packages will not show up in the standard IPackageManager enumeration. You will need to use one of the methods in IPackageManager2. This is applicable only if you are using the package enumeration instead of hard-coding to the “Windows Apps” folder path.

Starting in Windows 10, Windows Store apps can be installed to other volumes available to the system. The supported FileSystems are Fat32, exFat and NTFS.

Packages which are installed to other volumes are still visible from the package root location on %systemdrive%\Program Files\WindowsApps via Junctions. Furthermore, packages under these locations will also be encrypted by the Encrypted File System (EFS).

9 USING NOTIFICATIONS IN WINDOWS 10

9.1 INTRODUCTION

In Windows 10 an antimalware app may use different methods provided by the Windows 10 system to notify users about changes and important information. These notifications have been designed to accommodate both the desktop-style notifications and to introduce new notification types, as Windows 10 users will be working with both existing desktop apps and new universal apps.

A toast notification is a pop-up message to the user that contains relevant, time-sensitive information, and provides quick access to related content in an app. It can appear whether the user is in another app, the Start screen, the lock screen, or on the desktop. Toasts should be viewed as an invitation to the user to open or return to your antimalware app to follow up on something of immediate attention or interest.

In Windows 10, there is a new Action Center which aggregates notifications that the user has missed from apps and the system, and balloon notifications have been updated to look and feel like toast notifications and work in Action Center. With this new Action Center, a toast notification will automatically be persisted in Action Center for the user to view at a later time, if it is not clicked on by the user when it pops up.

9.2 NOTIFICATIONS AND THE WINDOWS DESIGN

Notifications can be shown to the user in a number of different ways in Windows 10 and there are specific places where it is more advantageous to inform the user of an action they need to perform. For example, the desktop is where the user typically performs system activities. To that end, any system-related issues that do not need immediate user response are more appropriately handled via a desktop-only notification.

Toast notifications, on the other hand, can be used to display a brief message to users with important information or where immediate user action is required, as this type of notification can be displayed over other apps, the lock screen, and other immersive experiences. The toast notification is designed as an invitation to the user to switch to a different context or task, and thus these notifications should be used sparingly to ensure that the user is interrupted only when necessary. Toasts will only appear for a short amount of time after which they are forwarded to Action Center. The user is in full control of their app toast behaviors—users can permanently turn off app toasts on a per app basis, if they choose to.

There are also some new notification features in Windows 10 which can be used to tailor the user experience and notify the user in a beneficial way. For example, generating a toast notification with the SuppressPopUp property set to TRUE provides the ability to log a notification to Action Center but doesn't raise a toast banner on the screen.

9.3 NOTIFICATION REQUIREMENTS

The following is a list of the Windows 10 notification requirements for antimalware apps. Antimalware apps will be required to meet all notification requirements by the set enforcement dates, which can be found under Section 14.3 (Requirements Timeline and Enforcements).

1. Antimalware applications must provide a standard, easily recognized, and readily accessible mechanism for users to minimize and close the app and its notifications.

2. Antimalware applications must provide the user a readily accessible mechanism to control the frequency of notifications shown to users. Notifications for malware detection, out of date signatures, software/subscription expiration, and those triggered by the first run of the application are excluded from this requirement.

3. Antimalware app notifications with no user-action required must not take focus from the user's active window (i.e., the notification can be visible but it cannot become the user's new active Window or interfere with the user's activities in the active window). Only notifications of malware detection or where the antimalware app blocks the user's intended action to prevent malware may interrupt the user and take focus away from the current window.

4. Antimalware app notifications must provide a clear indication (using a logo and/or text) of what application is creating the notification.

5. Antimalware app must not display notifications other than those related to malware detection when the device is in Retail Demo Mode[5].

Note: The above requirements replace and supersede the notifications requirements and guidelines in previous versions of this document.

Microsoft encourages antimalware app developers use the resources below in order to support consistent design principles and to help improve the Windows user experience:

9.4 RESOURCES

Please use this list of MSDN resources to create notifications for your antimalware apps in Windows 10.

QuickStart: Sending a toast notification from the desktop
Sending toast notifications from desktop apps sample
App user model IDs
ImmersiveMode interface documentation

[5] This requirement applies only to antimalware applications that are preloaded by OEMs on retail systems. https://msdn.microsoft.com/en-us/library/windows/apps/windows.system.profile.aspx

10 BROWSER CONSIDERATIONS FOR SECURITY SOFTWARE

10.1 INTRODUCTION

Browser performance matters to everyone. Consumers, enterprises, developers, and the technology industry share a desire for a faster and more capable web platform. How quickly a browser can load a web application, and the responsiveness of the web application after being loaded, plays a critical role in platform and browser choice.

The objective of this engineering document is to help security software vendors understand how they're impacting the performance of web applications today so that they can improve the experience for customers. This includes security vendors providing virus and malware detection and removal, password management, privacy monitoring, and related services.

For many years HTML4 web applications were bottlenecked by network speed and latency. This has changed with modern browsers, the emergence of HTML5 capabilities, and the high speed networks now available in many parts of the world. Today, client side processing of web applications plays a critical role in web performance. Nearly half of the web applications in the world are now bottlenecked by CPU time and that number will only increase.

At the same time industry hardware trends are evolving to support new form factors. Looking forward many customer scenarios will be enabled through hardware architectures that prioritize power consumption over processor throughput. There will be less CPU cycles available for the software.

The ecosystem of security software that developed around Internet Explorer over the last decade was not designed for the modern web. Most products were designed during a time when there were extra client-side CPU cycles to spare and altering the runtime patterns of the browser would not impact the customer experience.

These three industry trends are quickly converging to create a poor experience for customers.

1. CPU cycles are now critical to the loading and responsiveness of web applications
2. The number of available CPU cycles is decreasing with hardware trends, and
3. Most security software was designed to take a large number of CPU cycles.

This engineering document outlines common approaches that security products use today which impact the performance of HTML5 web applications. This document discusses both supported and unsupported approaches for integrating security software into Internet Explorer and does not sanction these approaches.

By taking the following considerations into account antimalware and related security applications can reduce their negative performance impact. This guidance applies to all Microsoft browsers on Windows platforms, including, but not limited to, Internet Explorer 9 on Windows Vista, Internet Explorer 9, Internet Explorer 10, Internet Explorer 11 on Windows 7, Internet Explorer 10 on Windows 8 and Windows 8.1, Internet Explorer 11 on Windows 8.1 and Windows 10, and additionally covers "Microsoft Edge" on Windows 10.

For more details, see Appendix F: Measuring and debugging web performance.

11 ACTIVEX CONTROL SECURITY AND PERFORMANCE CONSIDERATIONS

11.1 INTRODUCTION

Web browsing remains a large security attack vector despite advances in security and isolation boundaries. While the web platform itself is fundamentally secure, the native extensibility model around Internet Explorer provides an expansive attack surface area. Today, most Windows malware is delivered through unpatched vulnerabilities in ActiveX controls, the most common extension to the web platform. A large number of Black hole-style exploits, one of the most prevalent malware threats, exploit vulnerabilities in older ActiveX controls.

At the same time, browser performance and reliability matters to everyone. Consumers, enterprises, developers, and the technology industry share a desire for a faster and more capable web platform. How quickly a browser can load a web application, and the responsiveness of the web application after being loaded plays a critical role in platform and browser choice for customers.

Today, most security products attempt to deeply integrate with Internet Explorer to protect the user, often though unsupported and undocumented mechanisms. Some of these mechanisms include API detouring, ClassID hijacking, DLL replacement, kernel mode drivers, participating in JavaScript execution, forcing Internet Explorer to use the older compatibility mode JavaScript engine, delaying outbound network traffic, and altering Internet Explorer assumptions and runtime patterns. Many of these deep integration techniques are designed to catch the malware attack before it has infected the system. However, these techniques have a significant impact on performance, reliability, responsiveness, and power consumption for Windows customers. Security products run more complex code in more critical places than ever before.

In the March 2013 Internet Explorer Performance Lab study with Internet Explorer 10 on Windows 8 and Windows 8.1 with the top security products, we found that the top security products on average impacted the amount of time it took to launch Internet Explorer 10 and navigate to a web page by 173%, and the amount of time it took to navigate to a web page when Internet Explorer 10 had already been launched by 41%. Through our analysis, we know that this significant performance impact is due to security software using unsupported mechanisms to attempt to protect against malware attacks. We also know that browser add-ons used by security software are a large source of browser crashes and hangs. Using unsupported mechanisms to hook into Internet Explorer forces the browser to run in unsupported and untested configurations, increasingly the likelihood of crashes and hangs.

To help security software more effectively stop malware attacks from occurring and limit the impact to Internet Explorer web browsing performance and reliability, Internet Explorer now supports a new interface that will help security software make a security determination at the time that the ActiveX control is instantiated. Prior to instantiating ActiveX controls, Internet Explorer will use the synchronous IExtensionValidation interface to ask the registered security product to determine based on the content of the web page, whether or not the ActiveX control should be loaded or blocked. Based on that determination, Internet Explorer will either load the control, block the control, or block the navigation to that web page. Instead of running complex code all the time, security software will have the ability to run code at the most opportune time.

In order to more effectively stop this class of malware attacks in an efficient way that doesn't impact the performance or reliability of Internet Explorer, security products should stop using unsupported mechanisms of

hooking into Internet Explorer and use this supported and documented interface instead. The IExtensionValidation interface is supported in Internet Explorer 11 on Windows 7, Windows 8.1 and Windows 10.

11.2 SECURE AND FAST

Internet Explorer will share the context of the ActiveX control and content of the web page with security products to help them make a security determination on whether to load the ActiveX control. Based on this information, the security product will need to determine whether it is safe to load the ActiveX control, block the ActiveX control from loading, or block the entire page. Internet Explorer will call this interface individually for each ActiveX control on the page.

As this is a synchronous call, security products need to make sure that they make a security determination as fast as possible in order to minimize the performance impact of this interface. As a poorly implemented algorithm can have a large impact on Internet Explorer web browsing performance, Internet Explorer will be gathering telemetry on the amount of time it takes security products to make the security determination and will use this data to engage with security product teams directly.

We recommend that security products implement a safe list based on site reputation and known safe ActiveX control CLSID and version to reduce the performance impact on pages that are highly unlikely to contain malicious content. Security products should also remove all unsupported mechanisms for making a similar security decision as provided by this interface.

Security products should attempt to block just the ActiveX control if they determine that the page is not malicious, but malicious content may have been injected into the page. In this case, Internet Explorer will just block the control from loading, however, will render the rest of the page as would have been expected. An example of this could be that a malicious content is injected through a third party advertisement in an iframe, on an otherwise safe web page.

Security products should attempt to block the entire page if they determine that the web page is completely malicious. The following image shows the Internet Explorer blocking dialog, where Contoso Internet Security 2013 had made the blocking determination. This blocking dialog matches the Internet Explorer and Windows security experience and gives the user high confidence that Internet Explorer asked the registered antimalware software to make this security decision. Also, by giving the security product attribution in this dialog, in the case where the security product made a false positive block on a safe page, users can follow up with the security product directly to update an exception list.

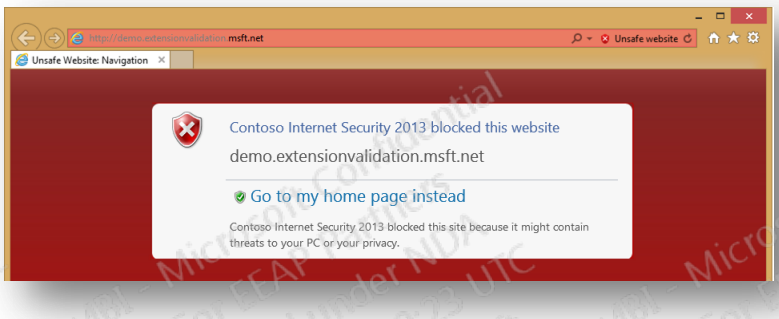


Figure 4: Internet Explorer Blocking Dialog

For more details, see Appendix G: How to use the IExtensionValidation interface.

12 AUTOMATIC MAINTENANCE

12.1 INTRODUCTION

The goal of Automatic Maintenance is to coalesce all background maintenance activity in Windows and to help easily enable third-party app developers to add their maintenance activity to Windows without negatively impacting performance and energy efficiency of the PC. Additionally, Automatic Maintenance enables users as well as enterprises to be in control of maintenance activity scheduling and configuration.

Execution of inbox and third-party maintenance activity occurs on Windows which includes Windows Update, automatic disk defragmentation and malware signature updates and scans. Additionally, enterprises frequently utilize maintenance activity such as Network Access Protection (NAP) scanning to help enforce security standards on all enterprise workstations.

Maintenance activity in Windows is designed to run in the background with limited user interaction and minimal impact to performance and energy efficiency. However, performance and energy efficiency are still impacted due to the non-deterministic and widely varied schedules of the multiple maintenance activities in Windows. Responsiveness to user is reduced when maintenance activity runs while the user is actively utilizing the PC. Users are also frequently nagged by multiple apps that ask the user to update their app and run background maintenance. Additionally, in order to configure or execute maintenance activity, users are directed to multiple experiences, including Action Center, Control Panel (System Maintenance), Windows Update, Task Scheduler MMC snap-in, third-party controls, etc.

12.2 GOALS

- Delight users by increasing system responsiveness by reducing and deferring background activities to a time when the PC is not in use.
- Help improve energy efficiency of PCs by coalescing all background maintenance activity and extend average idle duration.
- Provide predictable execution within a pre-defined time period for background maintenance activity.
- Provide mechanisms to automatically pause maintenance activity such that its execution does not impact other critical background or foreground activity.
- Enable transparency and user control over maintenance activity.
- Reduce reboots required by multiple maintenance activities.
- Enable third parties to integrate their maintenance activity with system-wide maintenance activity scheduling.

12.3 KEY PROBLEMS

Automatic Maintenance is designed to address the following problems with maintenance activity in Windows.

12.3.1 RESOURCE UTILIZATION CONFLICTS

Maintenance activity scheduled to execute when the system is idle may still inadvertently impact foreground user experience. The common problem is background activity that consumes a large portion of CPU or disk resource,

and in turn, impacts a running background activity that depends on CPU or disk throughput. Disk defragmentation scheduled during HD Media Center PVR recording is the best example. Additionally, the system may appear to be idle from a user input perspective, but presenting slides or other display content. In these cases, some maintenance activity may disrupt the foreground experience.

12.3.2 ENERGY EFFICIENCY

Most maintenance activity is executed when the system is idle and not on battery power. However, there are multiple components responsible for scheduling maintenance activity in the system, leading to randomization in execution of maintenance activity. This spreads out the workload over time, contrary to Windows' idle energy efficiency goals on both laptop and desktop systems. As a result, the system is unable to get long durations of low resource utilization. This has direct impact in reducing mobile battery life and increase in desktop energy consumption.

12.3.3 TRANSPARENCY TO THE USER

The user is typically unaware of the type, quantity and schedule of maintenance activity in Windows. As a result, users believe there is a large quantity of maintenance activity actively consuming foreground resources and negatively impacting performance and responsiveness. This may result in reduced customer satisfaction with their PC.

12.4 AUTOMATIC MAINTENANCE FUNCTIONALITY

Automatic Maintenance enables coalescing all maintenance activity to help improve idle efficiency and enable all activity to run in a timely and prioritized fashion. Additionally, Automatic Maintenance helps enable unified visibility and control over maintenance activity and enable third-party app developers to easily add their maintenance activity to Windows without negatively impacting performance and energy efficiency.

There are benefits to using this mechanism to perform periodic updates and scans:

- It is a reliable way to insure that those tasks are performance even when the system is in Connected Standby (see Section 13.4)
- It coalesces the tasks and allows good citizenship, instead of having many "schedulers" scheduling tasks independently of others at different times.
- It reduces the impact on battery life and responsiveness while the user is present.

12.4.1 FULLY AUTOMATIC

This default mode enables intelligent scheduling during PC idle-time and at scheduled-time, execution and automatic pausing of maintenance activity without any user intervention. The user has the ability to set a suitable schedule, within a predefined daily or weekly period. All maintenance activity is non-interactive and executes silently. Any required user interactions related to configuration etc., are to be outside of Automatic Maintenance execution.

The PC is automatically resumed from sleep at a time when the system is not likely to be in use, perhaps late in the evening. Full system resources at high power are utilized to complete maintenance activity as fast as possible,

allowing for increased idle efficiency when the system is otherwise in use. If the system was resumed from sleep for Automatic Maintenance, the system will be requested to go back to sleep.

12.4.2 USER-INITIATED

The user initiates execution of Automatic Maintenance on-demand to complete all maintenance activity before travel, long periods of time on battery power or to optimize for performance and responsiveness. The user understands that updating Windows with patches, applying antimalware signature updates and performing system optimization activities will help keep their system secure while they travel and connect to external networks and help increase their mobile battery life. Additionally, enthusiasts, including gamers and media professionals may optimize their systems routinely, by executing Automatic Maintenance activities.

The user will be able to view the current status of Automatic Maintenance execution, current duration, expected completion time and will have the ability to stop Automatic Maintenance execution. The user will also be able to configure attributes of Automatic Maintenance, including the automatic run schedule.

12.4.3 AUTOMATIC STOP

If the user starts interacting with the PC while Automatic Maintenance activities are underway, Automatic Maintenance will automatically stop currently running maintenance activity. Maintenance activity will be resumed when the system is idle again.

All activities in Automatic Maintenance must support stopping within a short time period (< 2 seconds) and re-starting from the point where they left off. This will allow for graceful stopping and re-starting of all Automatic Maintenance activities.

A non-intrusive notification should indicate to the user, that the activity has been stopped.

12.4.4 DEADLINE AND NOTIFICATIONS

Some maintenance activity is critical such that it must execute within a pre-defined periodic time window. If critical tasks have not had an opportunity to run within their predefined period of time, and if the deadline has passed, Automatic Maintenance will automatically start executing only the activity whose deadline was missed, at the next available system idle opportunity. This activity may impact system responsiveness and performance; therefore, Automatic Maintenance will notify the user that critical maintenance activity is executing. The bar for critical tasks should be very high so as to not inundate the user with notifications. If tasks with deadline have been missed for a long duration Automatic Maintenance will notify the user that maintenance is delayed and provide an option for a manual run of Automatic Maintenance.

Note that the goal of Automatic Maintenance is to use prescribed user notification channels, including the Windows Action Center.

12.4.5 ENTERPRISE CONTROL

Enterprise IT professionals should be able to determine when Automatic Maintenance executes on their Windows systems, enforce that schedule via standardized management interfaces and retrieve event data about the status

of Automatic Maintenance execution attempts. Additionally, IT professionals should be able to invoke specific Automatic Maintenance activity remotely via standard manageability interfaces. Status reporting should include each time Automatic Maintenance is executed, the tasks within that have been completed and notifications when Automatic Maintenance could not be executed because the user has manually paused Automatic Maintenance. IT professionals should consider moving logon scripts to Automatic Maintenance to help make the user's logon experience quicker.

For details and code samples for creating an automatic maintenance task, refer to Appendix G: Create an automatic maintenance task.

12.4.6 RECOMMENDED IMPLEMENTATION

It is recommended that antimalware uses Automatic Maintenance to perform periodic scan and signature updates in order to avoid impacting battery life and performance while the user is present. Automatic Maintenance can't enforce strong deadlines, so it is important that the antimalware solution verify that critical security-related tasks are executed on time.

Rely on Automatic Maintenance first and foremost, but also use your own existing scheduler agent or service to perform those tasks if deadlines are missed by Automatic Maintenance. The policy below is an example to illustrate a possible implementation.

- Antimalware signature updates are scheduled to be performed every day through the Automatic Maintenance mechanism, as defined that the created task (Appendix G)
- The antimalware service implements a timer that triggers X minutes after the service starts and every Y hours afterwards
    - The service checks if signatures were updated recently
    - If Z scheduled updates have been missed, the service forces updates, bypassing Automatic Maintenance

13 ANTIMALWARE PERFORMANCE REQUIREMENTS AND GUIDELINES FOR WINDOWS 10

13.1 INTRODUCTION

Antimalware apps can significantly impact user experiences when not optimized for performance. This can have a negative effect on the user's overall experiences when using a PC. It is important that all antimalware apps implement and maintain the best possible performance scenarios to ensure favorable experiences for the customer.

Some common performance issues observed due to antimalware apps in earlier versions of Windows include:

- An increase in overall boot read footprint by more than 100MB
- High CPU and disk usage by antimalware app processes on the critical paths of scenarios such as boot, resume, and shutdown
- 50-100% increase in time to copy local files when an antimalware app is installed on the system
- 50-100% increase in launch times of common Office apps such as Word, Excel, PowerPoint, and Outlook with an antimalware app installed on the system.

13.2 PERFORMANCE REQUIREMENTS

For Windows 10, antimalware apps will be required to meet a defined set of performance requirements by the set enforcement dates. For details on the enforcement dates, please see Section 14.3 (Requirements Timeline and Enforcements).

The performance requirements comprise of a set of assessments from the Windows Assessment and Deployment Kit ("ADK") which cover key performance scenarios such as Fast Startup, Internet Explorer[6] Startup, Internet Explorer Security Software Impact Page Display time and Windows Store App launch time. Refer to the below table for additional details.

This table outlines the performance metrics targeted by the new requirements.

ADK Assessment	Metric and units	Notes
Boot Performance (Fast Startup[7])	Total Boot [Excluding BIOS] Duration in seconds	The total time to boot the system (includes the time to desktop and post On/Off background resource consumption)
Internet Explorer Startup Performance	IE Startup Duration (User Perceived) in seconds	Includes IE process, frame and tab creation times to a blank web page.
Internet Explorer Security Software Impact	Page Display Time in seconds	Measures the time that IE takes to display initial page content after the assessment navigates to the

[6] Although Microsoft Edge is the default browser for Windows 10 consumers, the Windows 10 performance requirements assess performance impact on Internet Explorer as this is representative of Win32 application performance.

[7] On/Off Transition Performance: https://msdn.microsoft.com/en-us/library/windows/hardware/hh825330.aspx

		webpage
Windows Store App Performance	Launch time in seconds The evaluated metric will be an average of the launch time of all Bing apps on the system.	The time that it takes for the apps to open
Microsoft Edge Performance	Navigation time in seconds	The metric covers the launch time of Microsoft Edge until a webpage is displayed to the user.
Video playback battery life	Energy consumption rate in mWh/min	The metric captures the system energy consumption under an active hi-def full screen video playback workload.
File Copy	File Copy throughput in MB/s	The test will use an alternate 500MB file corpus instead of the default 2GB ADK file corpus.
OOBE duration	First Logon time in seconds	The metric covers the out-of-box first logon experience duration until the desktop is visible and usable by the user.

Pre-packaged assessment jobs with the proper configurations will be provided[8] to antimalware app developers in order to allow for self-testing, along with system preparation scripts and test documentation. The number of iterations per metric is increased to ten instead of three (in the default assessment configuration) in order to obtain a more accurate average. Outliers are identified with a basic IQR method. An iteration is rejected if it is above Q3 + 1.5*IQR.

The next table contains the matrix of systems Microsoft will use to measure and rate the performance metrics, segmented by form factor and price point. PC manufacturers may change and obsolete a device at any given time so we strongly recommend that once the device list is published, you purchase the devices as quickly as you are able to. Microsoft plans to continue testing with this set of hardware until at least July 1, 2016.

[8] The February 2016 performance requirements pre-packaged assessment jobs and test instructions for Windows 10 builds are available on Connect here: https://connect.microsoft.com/site1304/Downloads/DownloadDetails.aspx?DownloadID=57541

For the July 2016 performance requirements, the pre-packaged assessment jobs and test instructions for Windows 10 November release builds, are available on Connect here: http://connect.microsoft.com/site1304/Downloads/DownloadDetails.aspx?DownloadID=59706

	Tablet	2-in-1	Notebook	AIO
Entry		Lenovo FLEX 10-30 Intel Celeron N2807 2 GB 500 GB 5400 RPM HDD	Toshiba CL15-B1300 [9] Intel Celeron N2840 2 GB 32 GB eMMC	ASUS ET2031IUK Intel Celeron 2955U 4 GB 500 GB 7200 RPM HDD
Value	HP Stream 7 [9] [10] Intel Atom Z3735G 1 GB 32 GB eMMC			
Volume			Dell Inspiron 11 Intel Core i3-4030U 4 GB 500 GB 5400 RPM HDD	
Premium	Acer Aspire Switch 12 Intel Core M 5Y10c 4 GB 128 GB SSD		HP Spectre X360 13-4001DX Intel Core i5-5200U 4 GB 128 GB SSD	Lenovo A740 Intel Core i7-4558U 8 GB 1 TB HDD + 8 GB SSD

On each system, two test passes are executed to gather metrics: a baseline[11] and third party antimalware measurement. Each metric is then rated (**Exceptional**, **On Track** or **Strong Concerns**) independently for both test passes, based on the specific performance criteria of the tested system as described in the table below requirements due July 1, 2016[12].

		HP Stream [9] [10]		Lenovo FLEX 10		Toshiba CL15 [9]		ASUS ET2031IUK		Acer Aspire Switch 12		Dell Inspiron 11		HP Spectre X360		Lenovo A740	
Rating	**s**	Excep.	On Track	Excep.	On Track	Excep.	On Track	Excep.	On Track	Excep.	On Track	Excep.	On Track	Excep.	On Track	Excep.	On Track
Fast Startup	**s**	<20	<30	<20	<30	<20	<30	<20	<30	<15	<25	<20	<30	<10	<15	<15	<30
IE Launch	**s**	<1	<=1.5	<0.5	<=1	<0.5	<=1	<0.5	<=1	<0.5	<=1	<0.5	<=1	<0.5	<=1	<0.5	<=1
Page display time	**s**	<1.5	<1.8	<1.2	<1.5	<1.2	<1.5	<1.2	<1.5	<0.5	<0.8	<1.2	<1.5	<0.5	<0.8	<0.5	<0.8
Windows Store App launch	**s**	<2	<=3	<1	<=3	<1	<=3	<1.5	<=3	<1	<=3	<1	<=3	<1	<=3	<1	<=3
Microsoft Edge perf.	**mWh /min**	<2.5	<3.3	<2.2	<3.0	<1.7	<2.5	<2.2	<3.0	<1.5	<2.3	<2.2	<3.0	<1.5	<2.3	<1.5	<2.3
Battery life	**mWh /min**	N/A [13]	N/A	<80	<90	<80	<90	N/A	N/A	<70	<80	<110	<120	<90	<100	N/A	N/A
File Copy	**MB/s**	>45	>35	>45	>35	>45	>35	>45	>35	>120	>100	>40	>30	>110	>90	>90	>70
OOBE	**s**	<120	<150	<120	<150	<120	<150	<120	<150	<90	<120	<120	<150	<90	<120	<90	<120

[9] Windows 10 will be deployed on this hardware with compact OS. Single-instancing will also be configured for antimalware products that are preloaded by OEMs on retail systems to test the OOBE requirement.

[10] 32-bit Windows 10 will be deployed on this hardware

[11] Baseline is defined as a Windows 10 test system with a full set of OEM optimized drivers and with Windows Defender enabled as the only preinstalled Antimalware app.

[12] Starting in February 2016, we will begin the next wave of performance tests for Redstone. If there is an OS regression, we may need to adjust the performance baselines for all of the devices. This would only change the performance targets by increasing them based on the specific OS regression.

[13] There is currently an issue with the battery firmware preventing us from defining a power target for HP Stream 7. We will define as soon as the issue is mitigated.

An antimalware product succeeds if, when installed, the metric rating is the same as the baseline. If the rating changes (e.g., metric is Exceptional on the baseline and On Track with the third party solution), then a metric regression percentage is considered. The next table illustrates the decision matrix.

Baseline/Antimalware Rating	Exceptional	On Track	Strong Concerns
Exceptional	Pass	>= (20% reg. | 200ms) < (20% reg. | 200ms)	Fail
On Track	Pass	Pass	>= (20% reg. | 200ms) < (20% reg. | 200ms)
Strong concerns	Pass	Pass	>= (20% reg. | 200ms) < (20% reg. | 200ms)

A system with third party antimalware needs to **score the same rating as the system did with Windows Defender**. If the rating is the same, the antimalware passes. If the rating changes (e.g., goes from Exceptional to On Track when third party antimalware is installed), then the delta introduced by the third party antimalware – against the baseline – needs to be below 20% (or below 200ms, whichever is largest). **For an antimalware product to meet the performance requirements, it needs to pass on all eight tested system configurations for all four metrics**.

Antimalware apps will be evaluated on systems running Windows 10 build 10240 until February 1, 2016. After that date, the evaluations will occur using Windows 10 build 10586.

For details on how to setup a test environment and execute performance tests to validate that an antimalware app meets the performance requirements, please see Appendix I: How To Test and Validate Performance Requirements For Antimalware Apps.

13.3 PERFORMANCE GUIDELINES

Please see the Appendix J: Recommended performance guidelines for antimalware apps.

13.4 CONNECTED STANDBY

Windows customers are shifting towards lighter, smaller, more mobile platforms to satisfy their computing needs. As part of the shift to mobile devices, users have become increasingly concerned about battery life of their devices. Connected standby is a power paradigm first introduced in Windows 8, and enabled on systems that have specific hardware that provides support for this low power mode. Connected Standby occurs when the device is powered on, but the screen is turned off and eventually the system is in the low power state. In this power state, the system is technically always “on” (to support key scenarios like mail, VoIP, social networking, and instant messaging with Windows Store apps). It is analogous to the state a smart phone is in when the user presses the power button.

The Desktop Activity Moderator (DAM) is one of several features designed to ensure consistent, long battery life for devices that support Connected Standby. DAM was created to suppress desktop apps execution in a manner similar to the Sleep state (S3 on ACPI devices). This is done only for desktop apps because Windows Store apps are already suspended. It does this by suspending desktop app processes or throttling services across the system upon Connected Standby entry. This enables systems that support Connected Standby to deliver minimized resource usage and long, consistent battery life while enabling Window Store apps to deliver the connected experiences they promise. Windows Store apps are suspended automatically when device enters Connected Standby.

There are three key scenarios for antimalware app developers writing their apps:

1. System scan for threats.
2. Code and signature updates.
3. Real time scanning.

Developers trying to achieve system scan or updates are recommended to use the automatic maintenance API as described in Automatic maintenance functionality. Using the automatic maintenance API will allow your app to be energy efficient and will be coalesced with other Windows activity during Connected Standby.

See Appendix J: Recommended performance guidelines for antimalware apps, for detailed information on real time scanning and the impact of DAM for antimalware app developers.

13.4.1 RESILIENCY CHANGES IN WINDOWS 10

With Windows 8 and Windows 8.1, Connected Standby systems wake up periodically every 30 seconds from low power to process work and execute throttled services.

Windows 10 reduces power consumption by the OS and only wakes from the lowest power state when absolutely necessary (network notifications, OS maintenance, user wakes the system). The periodic 30 seconds wakes have been removed to improve resiliency.

Device interrupts continue to operate as normal to allow incoming instant messages, notifications, phone calls, etc. Some timers are postponed indefinitely until the system is awake or plugged into AC power to reduce the amount of power consumed by processing non-critical work when the system is “asleep” on battery.

13.4.2 MODERN STANDBY IN WINDOWS 10

The low power idle infrastructure is evolving in several ways in Windows 10. This will help expand the use of low power idle technology for market segments that were previously limited to the S3 power model.

A power model, called **Modern Standby**, allows systems that have a hard disk drive, and/or a networking adapter that doesn’t support the hardware offload requirements for Connected Standby to still take advantage of the low power idle model and instant on capabilities. In Modern Standby, the PC will not maintain a connection with the network while in low power state and incoming network notifications for Windows Store apps are not supported.

All the concepts and guidance around Connected Standby in this document also apply to Modern Standby (DAM, Automatic Maintenance, throttled services, etc.). Modern Standby is based on the same OS infrastructure as Connected Standby.

More documentation is available in the Modern Standby Introduction whitepaper.

13.5 OUT-OF-BOX EXPERIENCE (OOBE)

When customers turn on their Windows PCs the first time, or for the first time after an upgrade, OOBE (Out of Box Experience) displays. OOBE is a series of screens that require them to make crucial choices and enter info for their PC's customized experience. Customers should be able to complete this process as quickly as possible. It was previously observed that some antimalware software can introduce additional disk contention (through real time protection and file scanning) at the end of OOBE where storage might already be pegged, thus increasing the time the user has to wait to use their system.

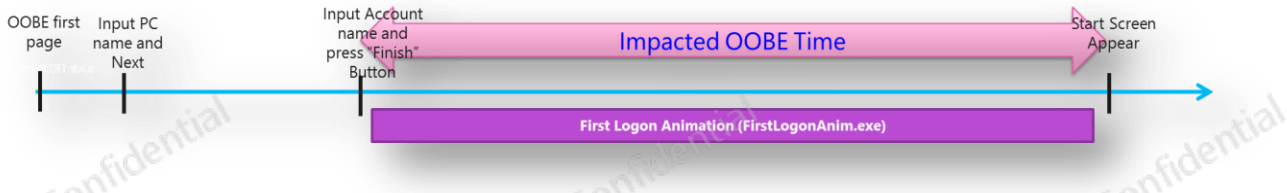


Figure 5: OOBE flow

It is recommended to only enable real-time protection after OOBE is completed. In Windows 10 you can use the OOBE completion APIs documented in the Windows Notification Framework Reference on MSDN.

13.5.1 USING THE OOBE COMPLETION NOTIFICATIONS

Windows 10 architectural changes will make it mandatory to hook into the OOBE Completed API to detect when antimalware work should begin. Current methods of determining when OOBE is completed (based on desktop switch event) may not work in Windows 10. As a results, customers could experience very long logon times due to IO contention – unless the new API is leveraged.

While this API already exists in Windows 8.1, its semantic is changing in Windows 10. It extends the window of time during which the API reports that the system is still in OOBE. It signals OOBE completion once the user has gone through both:

- Interactive machine OOBE (msoobe.exe process)
- Entire duration of the first sign-in (animation with fading colors)

In Windows 8.1, OOBE completion was signaled only after the interactive machine OOBE was completed.

The end result is that antimalware software gets signaled only when the system is out of the critical logon time window, where there's significant I/O contention.

13.5.2 RECOMMENDED IMPLEMENTATION

In order to leverage this notification service, here's the recommended implementation:

- Implemented in the antimalware monitoring service
- Call **RegisterWaitUntilOOBECompleted** with an *OOBE_COMPLETED_CALLBACK*
- The callback method will be triggered once OOBE completes
- The callback method should enable live protection (starts file scanning)

- Call **UnregisterWaitUntilOOBECompleted** once notification is received

Here's some pseudo-code

```
class OOBECompleteListener
{
    RegisterForNotifications()
    {
      // If OOBE hasn't completed yet then register callback

      fSetupComplete = false
      OOBEComplete(fSetupComplete)

      if (!fSetupComplete)  {
            RegisterWaitUntilOOBECompleted (OobeCompletedCallback, this,
           &OOBECompleteWaitHandle)
      }
      else {
            EnableLiveProtection()
      }
    }

    OobeCompletedCallback(PVOID CallbackContext)
    {
      EnableLiveProtection()
      UnregisterWaitUntilOOBECompleted(OOBECompleteWaitHandle)
    }

    PVOID OOBECompleteWaitHandle;
};
```

14 ANTIMALWARE POLICY BOARD

14.1 INTRODUCTION

The Antimalware Policy Board (AMPB) will review antimalware apps for Windows 10 to ensure performance, quality, protection, and favorable customer experiences. If an antimalware app developer cannot meet the requirements and user scenarios as outlined in this whitepaper based on the current timelines, they may ask for an exception if they meet the following criteria.

1. They have read the Antimalware Platform Whitepaper requirements and understand the requirements that are being asked and by the deadlines set.
2. They have made demonstrable progress towards reaching the requirements and have been working closely with Microsoft to achieve the new performance requirements and user experience policies.
3. They have formally asked for an exception as defined in this document and can provide a date by which they will be compliant.

For Windows 10, there will be an exception process for antimalware app developers who have taken action to improve the performance, reliability and user experience of their apps. If they have come close within reaching the performance requirements and user experience policies and are continuing to make improvements in their products, but have not met the requirement metrics, they can ask for an exception.

For more details on the exception process see Section 14.3.3 (Exception Process).

14.2 ANTIMALWARE APPLICATION POLICIES

For Windows 10, antimalware apps will be required to meet a defined set of policies by the set enforcement dates. For details the enforcement dates, please see the next section (Section 14.3 Requirements Timeline and Enforcements). This table outlines the policies for antimalware apps on Windows 10.

User Scenario	Policy	Notes
Performance Requirements	Antimalware apps **must not unnecessarily or materially degrade the responsiveness of the Windows 10** user experience by default.	Performance requirements will be phased in by scenario as described in Section 13.2 Performance Requirements.
Upgrade.exe	Antimalware applications which do not have a compatible version of their Windows 10 app for down level operating systems **may use the Upgrade.exe process** to install the compatible version of the app post Windows 10-upgrade. Antimalware applications that elect to use the Upgrade.exe process must abide by all the requirements set out in this document for use of the process.	This approach assumes that there is a Windows 10 compatible version of the antimalware app available.

No additional apps promoted through Upgrade.exe	Antimalware apps **must not bundle or install any additional non-RTP related software** with the RTP antimalware product which is installed by upgrade.exe.	Upgrade.exe should not be used as a vehicle to install software other than the Windows 10 compatible version of the RTP antimalware product that was previously installed and provided active protection on the user's device.
Using Notifications in Windows 10	Antimalware apps must neither flood the user with notifications nor provide disruptive notifications if immediate user action is not required or necessary.	Chapter 9 contains updated requirements that supersede all previous communications regarding this requirement This requirement will be enforced effective July 1, 2016 rather than February 2016. Microsoft will evaluate the user experience of antimalware apps on Windows 10 and may request that antimalware vendors who are not working toward implementing these requirements provide a plan with agreed milestones to align their experience with these requirements.
No undocumented APIs	Antimalware apps for Windows 10 **must not use any undocumented, unsupported or private Windows APIs** or data structures.	All undocumented or unsupported APIs will need to be removed or replaced with supported APIs or filtering APIs in the antimalware app.
User Protection Always ON (UPAO)	Antimalware apps **must make exclusive use of UPAO notifications that WSC provides** to notify users that the app may expire soon, and enable users to proactively take action to renew their subscription. An antimalware application must not notify the user outside of UPAO notifications.	The requirement for exclusive use of UPAO notifications applies to the period starting 5 days prior to the app reporting an expired state and ending 2 days after expiration. Antimalware vendors may self-attest their compliance with this requirement. If it is discovered that an expired antimalware app is not promptly reporting itself as disabled to the Windows Security Center, Microsoft will exercise the enforcement options in the AMPW.

Adherence to these policies does not relieve antimalware vendors or their distribution partners of any other obligations and agreements.

14.3 REQUIREMENTS TIMELINE AND ENFORCEMENTS

In Windows 10, antimalware apps will be required to meet a defined set of performance requirements and user experience policies. The requirements will become effective in phases as specified in this document. If the antimalware app developer does not meet the requirements by the specified deadline, and has not made a case for an exception, then it will be notified of the enforcement policies in place and that will affect its antimalware products. The enforcement date for OEMs is equivalent to their Distribute date.[14]
This table shows the policies based on enforcement dates.

Scenario	Detailed Scenario	Enforced by July 29, 2015	Enforced by February, 1 2016	Enforced by July 1, 2016
Fast Startup	Total boot duration	-	√	√
App Launch	Internet Explorer startup performance	-	√	√
Responsiveness	Internet Explorer Security Software impact	-	√	√
App Launch	Windows Store apps performance	-	√	√
Browsing	Microsoft Edge performance	-	√	√
Energy Efficiency	Video playback battery life	-	-	√
File Operations	File Copy	-	-	√
OOBE	OOBE duration[15]	-	-	√
User Experience	Upgrade.exe[16]	√	√	√
	No additional apps promoted through Upgrade.exe	√	√	√
	Using Notifications in Windows 10	-	--	√
	No undocumented APIs	-	-	√
	User Protection Always On	-	√	√

14.3.1 ANTIMALWARE APP SUBMISSION PROCESS FOR TESTING

For Windows 10, an antimalware app must be tested by both the antimalware app developer and Microsoft to ensure that it is Windows 10 compatible. Once the antimalware app developer believes they have a Windows 10 compatible version that meets all of the performance requirements and user experience policies as specified in the

[14] "Distribute" and "Distributed" means when an OEM System or a device leaves the control of an OEM Party or Company's authorized third-party installer. Distribute and Distributed includes shipment to any OEM Party affiliate, Channel Partner, or End User.

[15] OOBE requirement will only be applicable to products that are preloaded by OEMs on retail systems

[16] Upgrade.exe is the required solution for AV antimalware app developers to use who decide to update their AV apps post-Windows 10 upgrade.

section above, based on the enforcement timelines, they should submit a bug via SysDev and attach the version of the Windows 10 compatible app for testing purposes.[17] The antimalware app submitted must be tested prior to submission to ensure it meets the following criteria:

- The antimalware app will not block the Windows 10 upgrade experience
- The antimalware app test results post upgrade will match that of a pre upgrade experience
- The antimalware app meets the new performance requirements and user experience policies based on the enforcement timelines
- All known bug issues are addressed prior to submission.

Once the antimalware app is submitted, it will be tested for both compatibility and performance. Upon receipt of the SysDev request, the app will be provided to appropriate Microsoft teams to ensure that it meets the effective requirements.

Applications that do not register as virus protection with Windows Security Center will not be tested.

Antimalware apps can be submitted for testing starting at Windows 10 RTM. However, antimalware apps can only be tested for performance testing on a bi-monthly basis[18]. The performance team will only accept app submissions for performance testing twice a month on the 1st–3rd and the 15th–17th of every month.

For requesting performance testing please log a SysDev bug with the following details:

1. Title contains [Perf Testing], App Name and Version[19]
2. Issue type == Documentation
3. 1 AM app per bug mWh/min
4. Provide the AM app version by attaching it to the bug
5. Activation code if the product needs to be activated for testing
6. Any app preparation steps that are needed (other than signature updates and full system scans)
7. Any special instructions for OEMs on how to preload the application in the Windows image (only applicable to products that are preloaded by OEMs on retail systems)

If you do not have access to SysDev, please email the above details to: AMPerfReq@microsoft.com

After the antimalware app is submitted, it will be reviewed for performance testing and will be categorized as either “passed” or “failed” within 5 business days from the time of the submission.

If for any reason the antimalware app fails either the compatibility or performance testing, the antimalware app developer will be provided with a summary of the reasons for the failure via the SysDev bug. Additional guidance on how to improve the apps compatibly and/or performance will also be attached to the bug in order for the developer to make the recommended fixes and then they can resubmit the antimalware app via a SysDev bug.

---

[17] Each Windows 10 compatible antimalware app should be submitted via a separate SysDev bug.

[18] Unlike AppCompat testing which is hardware agnostic, performance testing is hardware dependent which requires testing on a variety of hardware configurations so submission for performance testing is limited to twice a month.

[19] App Name and Version in the SysDev bug title must be the same as it would appear on a customer’s device.

Antimalware apps need to be submitted for performance testing whenever there is a major release or update of either the antimalware app or Windows 10. The following table outlines the scenarios where testing is required.

Windows 10 Release → App Version ↓	**Same**	**Minor** (e.g. monthly updates)	**Major** (e.g., TH1->TH2)
Same	No test needed	No test needed	Submit for testing
Minor (e.g. 1.1 -> 1.2)	No test needed	No test needed	Submit for testing
Major (e.g. 1.2 -> 2.0)	Submit for testing	Submit for testing	Submit for testing

14.3.2 ENFORCEMENT POLICIES FOR ANTIMALWARE APPS

Antimalware apps must meet the requirements described above. If an antimalware app does not meet, or falls out of compliance with these requirements, Microsoft may respond with the following enforcement actions:

1. The non-compliant antimalware app may not be migrated forward through future system updates offered by Microsoft.[20] This is to ensure that Windows users receive an uncompromised initial user experience as a result of future updates.
2. OEM PCs that ship with the antimalware app may no longer qualify for Microsoft marketing funds, including Windows Jumpstart[21] and other marketing programs.
3. Non-compliant antimalware apps that are listed on the Consumer Security Software Provider listing page on www.microsoft.com will be removed.[22]

Note, these enforcement policies do not prevent any enterprise or consumer from installing antimalware apps. Both enterprises and consumers will be free to install whatever antimalware apps they prefer.

Antimalware app developers are encouraged to test their antimalware apps using the performance assessments provided in the ADK and to work with Microsoft to resolve any performance issues found prior or during the app submission process.

14.3.3 EXCEPTION / WAIVER PROCESS

For antimalware app developers who have met the criteria to request an exception, they may open a SysDev bug to request an exception or waiver. All exception requests must be submitted no later than 30 days prior to the requirement deadline (enforcement date). The AMPB will review each waiver request on a case-by-case basis and

[20] Non-compliant antimalware app developers will be provided with advance notice before a block occur.
[21] JumpStart is an incentive program for OEMs that provides additional funds for marketing activities if they ship prescribed configurations on qualifying devices.
[22] Antimalware apps listed on this page must also meet the requirements of the Windows 10 Antimalware API License and Listing Agreement (to obtain the API license and remain listed).

will determine if an exception is granted. If an antimalware app developer does do not have access to SysDev, please email the above details to ecoav@microsoft.com.

Please see details on how to file a waiver request via a SysDev bug below.

1. Title contains [AMPB Waiver], App Name, Version, and Waiver request type (E.g. “User Experience” “Using Notifications in Windows 10”)
2. Issue type == Documentation
3. 1 AM app per bug
4. Provide the antimalware app waiver request.
5. What is the requirement that cannot be met?
6. What is the amount of time it would take to meet the requirement?
7. How much work has been done to meet this requirement?
8. Any other details that the antimalware app developer thinks think would be important to note.

When an exception/waiver is granted, the AMPB will follow up with the antimalware app developer to discuss the exception provided and any next steps to ensure the requirements are met for future releases.

15 USE OF DOCUMENTED WINDOWS APIS AND DATA STRUCTURES BY ANTIMALWARE APPS

15.1 INTRODUCTION

Antimalware apps can negatively impact the Windows user experience through the use of undocumented, unsupported, or private Windows APIs and data structures or by using Windows APIs and data structures in undocumented and unsupported ways (e.g. hooking an API). The negative result can include degraded performance, reliability, responsiveness and power consumption of users' Windows devices. In addition, antimalware apps that rely on undocumented, unsupported, or private Windows APIs and data structure usage risk their own performance and reliability and ultimately may be identified as incompatible with Windows 10.

15.2 REQUIREMENTS

For Windows 10, antimalware apps must meet the following requirements related to the proper use of Windows APIs and data structures. For details on the enforcement dates for this set of requirements, please see Section 14.3 (Requirements Timeline and Enforcements).

Antimalware apps for Windows 10 must not use any undocumented, unsupported or private Windows APIs or data structures. Additionally, antimalware apps must not use Windows APIs or data structures in undocumented and unsupported ways (e.g. hooking an API). Finally, if an API or data structure has been specifically documented as deprecated, it should be considered as undocumented for the purposes of this section.

A Windows API or data structure is considered documented and supported for Windows 10 if documented in any of the following locations:

- Publically documented via the Microsoft Developer Network, and/or
- Privately shared with antimalware ISVs via the Microsoft Virus Initiative or Virus Information Alliance.

Any other documentation source, including Microsoft product team blogs, are not valid for compliance with this section.

15.3 COMPLIANCE

An antimalware ISV will be considered compliant with this requirement in the following cases:

1) The antimalware ISV has submitted a list of undocumented APIs used by the product no later than December 31, 2015 via SysDev. Submissions should include detailed descriptions of the scenario or use case for which the undocumented API is being used in order for Microsoft to triage the usage of the undocumented API. Microsoft may request an antimalware app developer to provide a plan with agreed milestones for removing dependencies on undocumented APIs.

   OR

2) The antimalware app developer attests that no undocumented Windows APIs are used in the products via email to apisupport@microsoft.com no later than July 1, 2016.

Microsoft will assess the undocumented APIs used by antimalware apps for opportunities to provide supported mechanisms in future releases. Microsoft may also suggest alternative supported solutions to help antimalware app developers migrate away from the undocumented APIs. However, it remains the responsibility of the antimalware app developer to ensure compliance with this section.

For the avoidance of confusion, if an antimalware app is technically incompatible or negatively impacting the Windows customer experience severely, the application will be addressed through our existing incompatible application process, regardless of whether the application incompatibility was caused by reliance on an undocumented API or some other cause. If the incompatibility was caused by use of an undocumented API, Microsoft may also engage with the ISV in the manner detailed in this whitepaper.

16 ANTIMALWARE SCAN INTERFACE

16.1 INTRODUCTION

The Antimalware Scan Interface (AMSI) is a generic interface standard that allows applications and services to integrate with any antimalware product present on a machine. It provides enhanced malware protection for users and their data, applications, and workloads.

AMSI is antimalware vendor agnostic, designed to allow for the most common malware scanning and protection techniques provided by today's antimalware products that can be integrated into applications. It supports a calling structure allowing for file and memory or stream scanning, content source URL/IP reputation checks, and other techniques.

AMSI also supports the notion of a session so that antimalware vendors can correlate different scan requests. For instance, the different fragments of a malicious payload can be associated to reach a more informed decision, which would be much harder to reach just by looking at those fragments in isolation.

16.2 USAGE OF AMSI IN WINDOWS 10

To allow antimalware vendors to provide better protection for their customers, AMSI has been integrated into the following Windows 10 components:

- User Account Control (UAC)
- PowerShell (scripts, interactive use, and dynamic code evaluation)
- Windows Script Host (Wscript.exe and Cscript.exe) (scripts and dynamic code evaluation)

There are two types of providers for antimalware vendorsyou and they can register them as follows:

Out-of-proc UAC AMSI provider

This is for a provider that implements the IAntimalwareUacProvider interface. The provider must run out-of-proc and must be a protected server process. Of the two types of providers, this is the more complicated one to register. Registration starts with setting up the following registry keys:

HKLM\SOFTWARE\Microsoft\AMSI\UacProviders
HKLM\SOFTWARE\Classes\CLSID
HKLM\SOFTWARE\Classes\AppId

And here are examples of the above for Windows Defender:

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\AMSI\UacProviders\{2781761E-28E2-4109-99FE-B9D127C57AFE}
HKEY_LOCAL_MACHINE\SOFTWARE\Classes\CLSID\{2781761E-28E2-4109-99FE-B9D127C57AFE}
  (Default) REG_SZ Windows Defender IAmsiUacProvider implementation
  AppId REG_SZ {2781761E-28E2-4109-99FE-B9D127C57AFE}
HKEY_LOCAL_MACHINE\SOFTWARE\Classes\AppId\{2781761E-28E2-4109-99FE-B9D127C57AFE}

LocalService REG_SZ WinDefend
ServiceParameters REG_SZ

When the service registered with AMSI UAC starts, it needs to do as Windows Defender does below in HRESULT MpComServerInitialize(VOID) to register its class object with COM. The service has to be running as Protected Process Light (PPL).

```
MP_NOTHROW
HRESULT MpComServerInitialize(VOID)
{
    TRACE_CPP_FUNCTION();
    HRESULT hr = S_OK;

    //
    //  If not Windows Defender, do not register with COM. We allow this in
Dev mode for non-Defender so that
    //  we can still run tests on non-Windows Defender SKUs (Vista+).
    //

    if (!IsWinThresholdDefender() && !(MpInDevMode() && MpIsVista())) {
        return S_OK;
    }

    MP_COMMON_ENFORCE(s_ComServerShutdownEvent == NULL, PRECONDITION);
    MP_COMMON_ENFORCE(s_ComServerThread == NULL, PRECONDITION);

    hr = UtilCreateEvent(
            &s_ComServerShutdownEvent,
            TRUE,   // bManualReset
            FALSE,  // bInitialState
            NULL,
            NULL);
    if (FAILED(hr)) {
        TRACE(TL_ERROR, L"CreateEvent failed with 0x%x", hr);
        goto cleanup;
    }

    hr = UtilSimpleCreateThread(&s_ComServerThread, &MpComServerMain, NULL);
    if(FAILED(hr)) {
        TRACE(TL_ERROR, "UtilSimpleCreateThread failed. hr = 0x%x", hr);
        goto cleanup;
    }

cleanup:
    if (FAILED(hr)) {
        MpComServerUninitialize();
    }

    return hr;
}
```

In-proc AMSI provider

This is for a provider that implements the IAntimalwareProvider interface. The provider must run in-proc and is not required to be a server process. To register an in-proc AMSI provider, antimalware vendors will need to set up the following reg keys:

HKLM\SOFTWARE\Microsoft\AMSI\Providers
HKLM\SOFTWARE\Classes\CLSID

And here are examples of the above for Windows Defender:

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\AMSI\Providers\{2781761E-28E0-4109-99FE-B9D127C57AFE}
HKEY_LOCAL_MACHINE\SOFTWARE\Classes\CLSID\{2781761E-28E0-4109-99FE-B9D127C57AFE}
  (Default) REG_SZ Windows Defender IOfficeAntiVirus implementation
HKEY_LOCAL_MACHINE\SOFTWARE\Classes\CLSID\{2781761E-28E0-4109-99FE-B9D127C57AFE}\InprocServer32
  (Default) REG_EXPAND_SZ %ProgramFiles%\Windows Defender\MpOav.dll
  ThreadingModel REG_SZ Both

The COM in-proc DLL (MpOav.dll in the case of Windows Defender) needs to has a COM class similar to what's done for Windows Defender below. It follows other rules of a standard COM in-proc DLL.

```
#pragma once

#include "stdafx.h"
#include "mpievalidator.h"
#include "AmsiProvider.h"

#pragma warning (push)
#pragma warning (disable:4201)
// error C4201: nonstandard extension used : nameless struct/union
#define AVVENDOR 1
#include <msoav.h>
#pragma warning (pop)


class ATL_NO_VTABLE CComMpOfficeAV :
    public ATL::CComObjectRootEx<ATL::CComMultiThreadModel>,
    public ATL::CComCoClass<CComMpOfficeAV, &CLSID_MpOfficeAV>,
    public IOfficeAntiVirus,
    public IExtensionValidation,
    public IAntimalwareProvider
{
public:
    CComMpOfficeAV();
    virtual ~CComMpOfficeAV();

    DECLARE_NO_REGISTRY()

BEGIN_COM_MAP(CComMpOfficeAV)
    COM_INTERFACE_ENTRY_IID(IID_IOfficeAntiVirus,IOfficeAntiVirus)
    COM_INTERFACE_ENTRY(IExtensionValidation)
    COM_INTERFACE_ENTRY(IAntimalwareProvider)
END_COM_MAP()
```

```
BEGIN_CATEGORY_MAP(CComMpOfficeAV)
    IMPLEMENTED_CATEGORY(CATID_MSOfficeAntiVirus)
END_CATEGORY_MAP()

    DECLARE_PROTECT_FINAL_CONSTRUCT()

    HRESULT FinalConstruct();

    void FinalRelease();

    //============================================================================
    //-- IOfficeAntiVirus
    //----------------------------------------------------------------------------

    STDMETHOD(Scan)(_In_ MSOAVINFO *pInfo);

    // IExtentionValidation

    HRESULT
    STDMETHODCALLTYPE
    Validate(
        _In_ REFGUID clsidControl,
        _In_ LPWSTR extensionModulePath,
        _In_ DWORD extensionFileVersionMS,
        _In_ DWORD extensionFileVersionLS,
        _In_ IHTMLDocument2 *pHtmlDocumentTop,
        _In_ IHTMLDocument2 *pHtmlDocumentSubframe,
        _In_ IHTMLElement *pHtmlElement,
        _In_ ExtensionValidationContexts contexts,
        _Out_ ExtensionValidationResults *pResults
    );

    HRESULT
    STDMETHODCALLTYPE
    DisplayName(
        _Outptr_ LPWSTR *displayName
    );

    HRESULT
    STDMETHODCALLTYPE
    Scan(
        _In_ IAmsiStream *pStream,
        _Out_ AMSI_RESULT *pResult
    );

    VOID
    STDMETHODCALLTYPE
    CloseSession(
        _In_ ULONGLONG session
    );

private:

    // Critical section is used to protect the creation of the object
    CMpCriticalSection          m_aIevCS;
    AutoRef<CMpIeValidator>     m_MpIeValidator;
```

```
    // Critical section is used to protect the creation of the object
    CMpCriticalSection          m_aAmsiCS;
    AutoRef<CAmsiProvider>      m_AmsiProvider;
};

OBJECT_ENTRY_AUTO(CLSID_MpOfficeAV, CComMpOfficeAV)
```

See the following appendices for detailed information on AMSI integration with the above:

- Appendix L: Powershell's Use of The AMSI
- Appendix L: Powershell's Use of The AMSI
- Appendix N: UAC/CONSENT.EXE's Use of The AMSI API and The UX Flow

16.3 RESOURCES

Please use this list of resources to utilize the Antimalware Scan Interface for your antimalware apps in Windows 10.

Antimalware Scan Interface
User Account Control
Scripting with Windows PowerShell
PowerShell Blog

17 PROTECTING ANTIMALWARE SERVICES

17.1 INTRODUCTION

Most antimalware solutions include a user-mode service that performs specialized operations to detect and remove malware from the system. This user-mode service is also frequently responsible for downloading the latest virus definitions and signatures. This user-mode service becomes a frequent target of malware because it's the single point of failure to disable protection on a system. To defend against attacks on the user-mode service, ant-malware vendors have to add a lot of functionality and heuristics to their software. However, such techniques are not completely foolproof and tend to be error prone because they have to identify functionality that Windows performs on their service and whitelist that functionality.

In Windows 8.1, the concept of protected services was first introduced to allow antimalware user-mode services to be launched as a protected service. After the service is launched as protected, Windows uses code integrity to only allow trusted code to load into the protected service. Windows also protects these processes from code injection and other attacks from admin processes.

17.2 RESOURCES

Please use this list of resources to utilize protected services for an antimalware app in Windows 10.

Protecting Antimalware Services
Early Launch Antimalware Whitepaper

18 EARLY LAUNCH ANTIMALWARE

18.1 INTRODUCTION

As antimalware software has become better and better at detecting runtime malware, attackers are also becoming better at creating rootkits that can hide from detection. Detecting malware that starts early in the boot cycle is a challenge that most antimalware vendors diligently address.

Windows 8 introduced a feature called Secure Boot which continues to be supported with Windows 10. This feature protects the Windows boot configuration and components, and loads an Early Launch Antimalware (ELAM) driver. This driver starts before other boot-start drivers and enables the evaluation of those drivers and helps the Windows kernel decide whether they should be initialized.

18.2 RESOURCES

Please use this list of resources to utilize Early Launch Antimalware drivers for an antimalware app in Windows 10.

Early Launch Antimalware Whitepaper
Early Launch Antimalware Reference

19 SECURE ETW CHANNEL

Starting in Windows 10, there is a new secured event channel. The components log important kernel events, security events and win32 events which an antimalware application can consume to gain better insight into some of the kernel and system internal activities.

19.1 CONSUMER

Only an antimalware application running as a protected process can register to consumer events from this channel. More than one antimalware application can register to consume these events.

19.2 PROVIDERS

There are 3 providers for this channel/session:

1. Kernel API audit provider: Provides events written by frequently called Kernel APIs
2. Win32 API audit provider: Provides events written by frequently called Win32 APIs
3. Security audit provider: Provides events written by the kernel and Lsass.exe. All events logged by this provider are also consumed by the EventLog service and these events are shown in the Event Viewer. Open Event Viewer and see the events under Windows Logs > Security.

19.3 RESOURCES

Please use this list of resources to utilize Secure ETW Channel for an antimalware app in Windows 10.

https://microsoft.sharepoint.com/teams/mmpc_partners/mvi/Shared%20Documents/Whitepapers/SecurityETW.pdf?d=wf44acfe962ed455da546e371559b4937

20 WINDOWS 10 API DEPRECATIONS

20.1 ANTI SPYWARE COMPONENT IN WINDOWS SECURITY CENTER (WSC) WILL BE DEPRECATED

WSC will support only antivirus but not anti-spyware. Converging antivirus and anti-spyware will simplify AM management through WSC:

- All the existing APIs will continue to work, and will gracefully ignore anti-spyware registration/status
- Antimalware applications are not required to maintain and report status for anti-spyware.
- Timeline for Change: July 2016

20.2 WDENABLE API WILL BE DEPRECATED

Windows Defender status that the calling application wants to set. TRUE enables Windows Defender. FALSE disables Windows Defender.

- This API will be deprecated.
- WSC is the only supported way to disable Windows Defender
- Timeline for Change: July 2016

See: https://msdn.microsoft.com/en-us/library/bb762466(v=vs.85).aspx

21 APPENDIX A: DEVELOP AN APP TO COMMUNICATE WITH WINDOWS SECURITY CENTER

Follow the steps outlined in this section to update or develop an app to meet the WSC requirements for Windows 8 and Windows 8.1 and Windows 10.

1. Modify the existing app's code to use the WSC APIs.
2. Link the app with the `/integritycheck` switch.
3. Add or modify the app resource file.
4. Authenticode: sign the app.
5. Test the app on Windows 8 or Windows 8.1 or Windows 10.

21.1 MODIFY THE APP'S CODE TO USE THE WSC APIS

For each app type (firewall, antivirus), requirements are outlined in the following sections.

21.1.1 FIREWALL REQUIREMENTS

Windows 8, Windows 8.1 and Windows 10 support the `IWscFWStatus` API for firewall apps to register, unregister, and send status updates. See the Windows Security Center API reference documentation for information about the `IWscFWStatus` interface.

Antimalware applications may turn OFF Windows Firewall if needed. Windows Firewall is designed to run side by side without interfering or degrading system performance so is recommended to not disable Windows Firewall when a third party firewall is registered. Antimalware applications that turn off Windows Firewall must also turn ON Windows Firewall back when user uninstalls or unregisters the antimalware or the firewall component from the third party.

21.1.1.1 REGISTER WITH WINDOWS SECURITY CENTER

The following code sample details how to register the antimalware app during installation or first run.

```
#define REMEDIATION_EXE L"%systemroot%\\Contoso\\ContosoFW.exe"
#define DISPLAY_NAME_FW L"Contoso Firewall"

IWscFWStatus pFwStatus;
BOOL fEnableNotificationsSelfReporting = FALSE;
BOOL fEnableWscNotifications = True;


hr = CoCreateInstance(
                CLSID_WscIsv,
                NULL,
                CLSCTX_INPROC_SERVER,
                IID_IUnknown,
                reinterpret_cast<LPVOID*> (&pUnknown));
```

```
hr = pUnknown->QueryInterface(IID_IWscFWStatus, reinterpret_cast<void**>
(&pFwStatus));

hr = pFwStatus->Register(REMEDIATION_EXE, DISPLAY_NAME_FW,
fEnableNotificationsSelfReporting, fEnableWscNotifications);
```

21.1.1.2 SEND A STATUS UPDATE TO WINDOWS SECURITY CENTER

The following code sample details how to send a status update for a firewall app to Windows Security Center. The app must make this API call whenever the app changes state to either On, Off or Snoozed.

```
IWscFWStatus pFwStatus;
WSC_SECURITY_PRODUCT_STATE eProductState = WSC_SECURITY_PRODUCT_STATE_ON;

hr = CoCreateInstance(
CLSID_WSCSVC,
NULL,
CLSCTX_INPROC_SERVER,
IID_IWscFWStatus,
&pFwStatus);

hr = pFwStatus->UpdateStatus(eProductState);
```

21.1.1.3 UNREGISTER WITH WINDOWS SECURITY CENTER

The following code sample details how to unregister a firewall app from Windows Security Center during uninstallation. It is required to make this API call during the app uninstall process.

```
IWscFWStatus pFwStatus;
hr = CoCreateInstance(
CLSID_WSCSVC,
NULL,
CLSCTX_INPROC_SERVER,
IID_IWscFWStatus,
&pFwStatus);

hr = pFwStatus->Unregister();
```

21.1.2 ANTIVIRUS REQUIREMENTS

Windows 8, Windows 8.1 and Windows 10 support the `IWscAVStatus` API for antivirus apps to register, unregister, and send status updates. See Windows Security Center API reference documentation for information about the `IWscAVStatus` interface.

21.1.2.1 REGISTER WITH WINDOWS SECURITY CENTER

The following code sample details how to register the antivirus app during setup or first run.

```
#define REMEDIATION_EXE L"%systemroot%\\Contoso\\ContosoAV.exe"
#define DISPLAY_NAME_AV L"Contoso Antivirus"

hr = CoCreateInstance(
                CLSID_WscIsv,
                NULL,
                CLSCTX_INPROC_SERVER,
                IID_IUnknown,
                reinterpret_cast<LPVOID*> (&pUnknown));

hr = pUnknown->QueryInterface(IID_IWscAVStatus, reinterpret_cast<void**>
(&pAvStatus));

hr = pAvStatus->Register(REMEDIATION_EXE, DISPLAY_NAME_AV, false, false);
```

21.1.2.2 SEND A STATUS UPDATE TO WINDOWS SECURITY CENTER

The following code sample details how to send a status update for an antivirus app to Windows Security Center. The app must make this API call whenever the app changes state to either “On”, “Off”, “Snoozed”, “Expired” or “Disabled”.

```
IWscAVStatus pAvStatus;
bool fProductUptoDate = false;
WSC_SECURITY_PRODUCT_STATE eProductState = WSC_SECURITY_PRODUCT_STATE_ON;

hr = CoCreateInstance(
CLSID_WSCSVC,
NULL,
CLSCTX_INPROC_SERVER,
IID_IWscAVStatus,
&pAvStatus);

hr = pAvStatus->UpdateStatus(eProductState, fProductUptoDate);
```

21.1.2.3 UNREGISTER WITH WINDOWS SECURITY CENTER

The following code sample details how to unregister an antivirus app from Windows Security Center during uninstallation. It is required to make this API call during the app uninstall process.

```
IWscAVStatus pAvStatus;
hr = CoCreateInstance(
CLSID_WSCSVC,
NULL,
CLSCTX_INPROC_SERVER,
IID_IWscAVStatus,
&pAvStatus);
```

```
hr = pAvStatus->Unregister();
```

21.2 LINK THE APP WITH THE /INTEGRITYCHECK SWITCH

Use the `/integritycheck` switch to link the app while compiling.

21.3 ADD OR MODIFY THE APP RESOURCE FILE

All apps that report status to Windows Security Center must include specific values in their resource files. If the app does not have a resource file (*.rc file format), then an *.rc file will have to be created in order for the app to work properly on Windows 8,Windows 8.1 and Windows 10.

The following version values are required:

```
VER_COMPANYNAME_STR
VER_PRODUCTVERSION_STR
```

The following is a sample resource file for a fictitious app, Contoso Fix-It.

```
#define VER_FILETYPE                VFT_APP
#define VER_FILESUBTYPE             VFT2_UNKNOWN
#define VER_FILEDESCRIPTION_STR     "Contos Fix-it!"
#define VER_ORIGINALFILENAME_STR    "IWscContoso.exe"
#define VER_COMPANYNAME_STR         "Contoso Corporation"
#define VER_PRODUCTVERSION_STR      "1.0.0.1"

#include "common.ver"
10      ICON         "contoso.ico"
STRINGTABLE DISCARDABLE
BEGIN
    101                     "Contoso Antispyware"
    102                     "Contoso Antivirus"
    103                     "Contoso Personal Firewall"
END
```

21.4 AUTHENTICODE SIGN THE APP

Special procedures are required for interoperability with Windows Security Center, so the standard code signing procedure for apps should not be used here. The following are examples of how to sign an antimalware app, app.exe, using SignTool.

The first is an example of a test-signing and the second is an example of release signing. The app must be signed with the `/ph` (page hash) switch. Page hash enforcement verifies the signature on each page as it is loaded into memory. Because of this requirement, the app must be signed on a Windows 8 or a Windows 8.1 or a Windows 10 PC. Time-stamping using the `/t` switch keeps the signature from expiring when the certificate expires. The `/ac` switch used in release-signing references a cross-certificate. This is required because the signing code path exists in

the Windows kernel, which has a hard-coded root of trust. The cross-certificate is needed to chain to the issuing certificate authority.

Test-signing (example):

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

Release-signing (example):

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

For more information on code signing, refer to Appendix C. Microsoft recommends that commercial software publishers establish a separate test code-signing infrastructure to test-sign prerelease builds of their app. Test certificates chain to a different root certificate than the root certificate that is used to sign publicly released products. This precaution helps ensure that test certificates are trusted only within the intended test environment.

Once the app binary is signed it may be copied directly onto other computers or included in an installer package for distribution to other computers.

21.5 TEST THE APP ON WINDOWS 10

It is important to note that when testing a test-signed app, Windows 10 must be in test mode. Enabling test mode is described in greater detail in Appendix C of this document.

21.5.1 VERIFY THAT THE APP IS WORKING PROPERLY ON WINDOWS 10

1. After the app has installed and registered, verify that it appears in the Windows Security Center user interface.
2. After the app updates, verify that the Windows Security Center user interface reflects that the app is current
3. Uninstall the app and then verify that it no longer is reported in the Windows Security Center user interface.
4. Test each of the remediation switches on the remediation executable (/enable, /update, /disable, /renew) and ensure that the expected behavior is seen.
5. Verify that the Windows Security Center status is updated within the user interface after selecting a **Fix** action.

22 APPENDIX B: BEST PRACTICES FOR DEVELOPING WSC APPS

22.1 USER ACCOUNT CONTROL RECOMMENDATIONS

As a rule, standard users should not be prompted with a User Account Control (UAC) dialog box. To ensure a more seamless user experience, Microsoft recommends that an app by default not require elevation for the standard user to update, enable and/or pay for a subscription. For detailed guidance on working with the UAC, refer to following resources:

- UAC website: http://msdn.microsoft.com/en-us/library/bb756996.aspx
- UAC blog: http://blogs.msdn.com/uac
- UAC samples: http://msdn.microsoft.com/en-us/library/aa970890.aspx

However, antimalware apps should require elevation for standard users when the user attempts to put the PC in an insecure state, such as snoozing or disabling the service. App developers can create apps that may provide administrators the ability to change which tasks/actions require elevation. Microsoft recommends that apps do require elevation for the standard user to turn off or snooze the antimalware app since this action puts the PC in an insecure state.

The following table provides a sample of a user event and corresponding suggested user experiences.

Event	Suggested user experience
Standard user disables, turns off or puts app into snooze state.	Require elevation for this action since the user is putting the PC at risk.
Standard user updates and/or enables or pays for a subscription.	Do not require elevation for this action(s) since the user is attempting to secure the PC.

22.2 REMEDIATION EXE

Antimalware app developers must provide the path to a code signed executable file called the remediation executable. For Windows Security Center to launch the antimalware remediation EXE, it must be code signed using a certificate that is issued from a CA in the hosted computer's root store. It is recommended that app developers code sign with a Class 3 digital certificate to provide authentication of the executable to the user. Follow the guidelines in Register with Windows Security Center to register the Remediation EXE.

22.2.1 #DEFINE REMEDIATION_EXE L"%SYSTEMROOT%\\CONTOSO\\CONTOSOAV.EXE"USING SWITCHES

Windows Security Center will call the remediation executable with a set of switches. The app is expected to perform some action based on what these switches are. The example below shows how the app will be called.

```
Contoso.exe /[application type] /[action]
```

The table below gives details about each of the switches that the program should handle.

Solution	Type	Action	Expected Behavior

Firewall	/FW	/enable	**App should switch from "Off" or "Snoozed" to "On" by setting the `productState` attribute to `WSC_SECURITY_PRODUCT_STATE_ON` and resume protecting the user.**
Antivirus	/AV	/enable	**App should switch from "Off" or "Snoozed" to "On" by setting the `productState` attribute to `WSC_SECURITY_PRODUCT_STATE_ON` and resume protecting the user.**
		/update	**App should update its signatures and once completed set `productUptoDate` to true.**
		/renew	**App must provide a way for users to renew their app subscription. For example, the app could direct the user to a web page that allows them to purchase a new subscription. Once the user has renewed their subscription, the app should switch to "On" by setting the `productState` attribute to `WSC_SECURITY_PRODUCT_STATE_ON` and resume protecting the user.**
		/disable	**App must perform and follow the following items:** **Disable real-time and background scanning on the system for both antivirus and antispyware protection.** **The disable operation must be silent, meaning any associated UI (e.g. pop-ups, main app interface) must not be displayed. The user must not be prompted to make changes or take any actions.** **Once the app has completed disabling itself, it must invoke the `UpdateStatus` API functions by setting the `productState` attribute to "off" (`WSC_SECURITY_PRODUCT_STATE_OFF`). If the app also registers as an antispyware app it must also report as "off".** **The app should disable itself within 10 seconds. If it takes longer than 10 seconds, Windows Security Center will assume that disable has failed and prompt the user to uninstall the app.** **After the app has been called with the `/disable` switch and the app is still expired, if it is called with the `/enable` switch or if the user tries to turn it on, the app should report back as expired to WSC.** **The app should only accept the `/disable` call when it is in the expired state.** **The app must request UAC user elevation before performing the disable operation. See the User Account Control section for more information on how to perform this task.**
		/upgrade	**This is a new flag for Win10. The app should detect the OS version and download binaries compatible with the OS. Then the antimalware app should install and register with WSC after install.**
Antispyware	/AS	/enable	**App should switch from "Off or Snoozed" to "On" by setting the productState attribute to `WSC_SECURITY_PRODUCT_STATE_ON`**

			and resume protecting the user.
		/update	**App must update its signatures and once complete, set productUptoDate to true.**

If an antimalware app contains multiple product offerings (for example, antivirus and antispyware), remediation for all offerings can be provided in one executable. For example, if a Contoso antispyware is both "off" and "out of date", the app developer can determine by the following switches that Windows Security Center requests enabling and updating the antispyware app.

```
Contoso.exe /AS /enable /update
```

Microsoft recommends that antimalware app developers provide users with a seamless (minimal user interaction) experience as possible to help encourage users to update, enable or renew their subscriptions when the app is out of date, turned off or expired.

22.3 WSC SWITCHES AND USER EXPERIENCES

22.3.1 APP INSTALLED, TURNED ON, AND OUT OF DATE

Windows Security Center alerts the user that the antimalware app is out of date. This alert occurs when the app is installed, turned on, its signatures are out of date and there are no other apps that are providing protection. It is important to note that in this case, the antimalware app is deliberately not providing protection.

The user will be presented with a user experience that directs him or her to the "Update now" button. When the user clicks on the "Update now" button, Windows Security Center invokes the remediation executable with the /update switch. After the update, the app must invoke the `UpdateStatus` API functions to reflect the updated status by setting the productUptoDate attribute to true and setting the `productState` attribute to `WSC_SECURITY_PRODUCT_STATE_ON` (product is enabled and real time scanning is running). NOTE: The app must not update the status before the user has actually gone through the update process.

22.3.2 APP INSTALLED, TURNED OFF, AND UP TO DATE

Windows Security Center alerts the user that the solution is turned off. This alert occurs when the solution is installed and up to date, but is turned off and there are no other solutions that are providing similar protection. It is expected that this scenario requires little user interaction and is a hands-off user experience.

The user will be presented with a user experience that directs him or her to the "Turn on now" button. When the user clicks on the "Turn on now" button, Windows Security Center invokes the remediation executable with the /enable switch. When this occurs, the antimalware app must invoke the `UpdateStatus` API functions to reflect the updated status by setting the `productState` attribute to `WSC_SECURITY_PRODUCT_STATE_ON` (Product is enabled and real time scanning is running).

22.3.3 APP INSTALLED, TURNED OFF, AND OUT OF DATE

Windows Security Center alerts the user that the antimalware app is turned off. When the user turns on the app, Windows Security Center alerts the user that the app is out of date. This alert occurs when the app is installed, turned off, its signatures are out of date and there are no other solutions that are providing protection.

The user will be presented with a user experience that directs him or her to the "Turn on" button. When the user clicks on the "Turn on" button and clicks "Update now", Windows Security Center invokes the remediation executable with the /enable and /update switch. After the update, the app must invoke the `UpdateStatus` API functions to reflect the updated status by setting the `productUptoDate` attribute to true and setting the `productState` attribute to `WSC_SECURITY_PRODUCT_STATE_ON` (Product is enabled and real time scanning is running).

The app should allow for the switches to be called in either order as in the following example.

```
Protectionapp.exe /AS /enable /update
```

OR

```
Protectionapp.exe /AS /update /enable
```

22.3.4 SCENARIO 4: APP IS INSTALLED AND EXPIRED

Windows Security Center alerts the user that the antimalware app is expired (i.e., the app is deliberately no longer providing the same level of protection to the user. The user will be presented with a user experience that directs him or her to the "Renew" button. When the user clicks the "Renew" button, Windows Security Center invokes the remediation executable with the `/renew` switch. After the user updates his or her subscription with the app, the app must invoke the `UpdateStatus` API functions to reflect the updated status by setting the `productState` attribute to either `WSC_SECURITY_PRODUCT_STATE_ON, WSC_SECURITY_PRODUCT_STATE_OFF` or `WSC_SECURITY_PRODUCT_STATE_SNOOZED`. An example of how the remediation executable would be called app would be called is shown below.

```
Protectionapp.exe /AV /renew
```

22.4 ICON GUIDANCE

Windows Security Center uses the first icon resource in the `REMEDIATION_EXE` file to display as the app branding in the Windows Security Center user interface. This is the icon with the lowest resource identifier in the file. It is recommended that the icon file support various icon sizes and pixel color depths to support a full range of monitor resolutions and accessibility settings. Windows Security Center renders the icon using `SM_CXSMICON x SM_CYSMICON` dimensions.

22.5 APPLET GUIDANCE

This section gives examples of some of the various ways Control Panel applets can be registered in Windows 10 and also how to include the registered control panel applet in Windows Security Center.

22.6 REGISTERING APPLETS

Registration of applets is required to make them appear in the Control Panel category pages as well as in the Classic View. The control panel needs the following information:

- Location of the applet: `%SystemRoot%\system32\desk.cpl`
- Display name: `Personalization`
- Icon: `%SystemRoot%\system32\shell32.dll,-14`
- Description / comment / tool tip text ("Change the appearance of the desktop.")
- Categories ("1" = Appearance and Personalization category)
- Canonical name of the applet: `Microsoft.UserAccounts`

This section gives examples of some of the various ways control panel applets can be registered in Windows 10 and also how to include the registered control panel applet in Windows Security Center.

22.6.1 RECOMMENDATIONS

When considering writing a new applet, Option 1 is recommended, as it is the most simple. Option 1 is also required if the applet requires UAC elevation since the applet must be an executable file. If the applet requires a different display name or icon depending on some kind of restriction or state then it must implement the CPlApplet API and registration as is described in Option 2. Option 2 is also viable if the goal is just to register categories for an existing applet that uses the CPlApplet API.

22.6.1.1 OPTION 1: REGISTER AS A SHELL COMMAND OBJECT

A command object is defined completely in the registry. The Windows shell has a mechanism that reads the registry entries to get item information and the command to run when the item is invoked. Create a GUID, add it in a new registry key, and add the applet GUID to `HKEY_CLASSES_ROOT`.

First, create a GUID for the applet (using a tool such as uuidgen.exe). Add the created GUID as a new registry key in the following location:

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Con
trolPanel\NameSpace\{applet guid}]
```

The default value of the registry key is the name of the applet. This default value identifies the GUID. For example:

```
(Default) = [REG_SZ]"Applet name"
```

Second, add the applet GUID to HKEY_CLASSES_ROOT under CLSID:

```
[HKEY_CLASSES_ROOT\CLSID\{applet guid}]
  (Default) = [REG_SZ]"Applet name"
```

Also under this key is the applet's localized name and description (used for the Comment column in Classic View or the tool tip in a category page). Either enter the strings directly into the registry or enter resource identifiers to enable Windows to load the strings in the currently selected language:

```
LocalizedString = [REG_EXPAND_SZ]"@path-to-applet,-resourceId"
InfoTip = [REG_EXPAND_SZ]"@path-to-applet,-resourceId"
```

In Windows 10, a developer can specify that the applet appear in only the Classic View or only in a category page or register the applet to appear in multiple category pages. In general, an applet appears in a single category page and the Classic View.

Here is a map of how categories map to their identification numbers:

```
0 = Additional Options
1 = Appearance and Personalization
2 = Hardware and Sound
3 = Network and Internet
4 = Hardware and Sound
5 = System and Maintenance
6 = Clock, Language, and Region
7 = Ease of Access
8 = Programs
9 = User Accounts and Family Safety
10 = Security
11 = Mobile PC
-1 = Show applet only in Classic View
-2 = Show applet only in Control Panel Home (Category View)
```

A comma-separated string of numbers is entered to register the category or categories in which the applet belongs. For example, the string "2, 10" means that the applet will appear in Hardware and Sound category and in Security category. The string "-1,8" means that the applet should not appear in the Control Panel Home, only in Classic View, and the Category column should show the Programs category. In order to maintain consistency, the applet should appear under the “Security” category (identification number 10).

For example:

```
System.ControlPanel.Category = [REG_SZ]"categoryId,categoryId"
```

If no category is registered, then by default the applet appears under Additional Options (“0”).

Windows 10 support canonical names for applets. When other programs invoke an applet using the `IOpenControlPanel` interface, or when an applet is opened on the command-line using control.exe, the canonical name is used. For example:

```
control.exe /name "Microsoft.Personalization"
System.ApplicationName = [REG_SZ]"canonicalNameOfApplet"
```

The convention for applets is to begin the canonical name with "<*CompanyName*>", for example, "Microsoft.UserAccounts". It helps prevent canonical name collisions, for example if there was "Contoso.Firewall" it would not collide with "Microsoft.Firewall".

To register the applet icon, the icon resource identifier is entered under in the DefaultIcon registry key.

```
[HKEY_CLASSES_ROOT\CLSID\{applet guid}\DefaultIcon]
(Default) = [REG_EXPAND_SZ]"path-to-applet,-resourceId"
```

The last item to register is the command to run when the applet is invoked. The full path to the applet should be given for better security.

```
[HKEY_CLASSES_ROOT\CLSID\{applet guid}\Shell\Open\Command]
(Default) = [REG_EXPAND_SZ]"path-to-applet and command-line args"
```

Once the applet is registered, open Control Panel to view the applet. The applet is not loaded except to get the string and icon resources.

Any app can be registered as an applet. The following example registers notepad.exe.

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Con
trolPanel\NameSpace\{0052D9FC-6764-4D29-A66F-2F3BD9E2BB40}]
(Default) = [REG_SZ]"Notepad"

[HKEY_CLASSES_ROOT\CLSID\{0052D9FC-6764-4D29-A66F-2F3BD9E2BB40}]
   (Default) = [REG_SZ]"Notepad"
   LocalizedString = [REG_EXPAND_SZ]"@%SystemRoot%\notepad.exe,-9"
   InfoTip = [REG_EXPAND_SZ]"@ %SystemRoot%\notepad.exe,-5"
   System.ApplicationName = [REG_SZ]"Microsoft.Notepad"
   System.ControlPanel.Category = [REG_SZ]"1,8"

[HKEY_CLASSES_ROOT\CLSID\{0052D9FC-6764-4D29-A66F-
2F3BD9E2BB40}\DefaultIcon]
   (Default) = [REG_EXPAND_SZ]"%SystemRoot%\notepad.exe,-2

[HKEY_CLASSES_ROOT\CLSID\{0052D9FC-6764-4D29-A66F-
2F3BD9E2BB40}\Shell\Open\Command]
   (Default) = [REG_EXPAND_SZ]"%SystemRoot%\notepad.exe"
```

It uses the localized "Notepad" string and the Notepad icon. To get these resource ID numbers, open the binary in a tool such as Visual Studio. It is registered to appear in categories 1 and 8, which are "Appearance and Personalization" and "Programs". When it is invoked it simply opens notepad.exe. It can be opened using the canonical name "Microsoft.Notepad".

22.6.1.2 OPTION 2: REGISTER IF CPLAPPLET API IS IMPLEMENTED

All applets used to be required to implement the CPlApplet function in order to interact with Control Panel. Now that Option 1: registering as a shell command is available**,** it is not recommended to use Option 2: register CPlApplet API because as it slows down the performance of the Control Panel window (the applet module needs to be loaded to get the applet property information).

One reason to use CPlApplet is if the applet has been implemented on a prior version of Windows and it would be easier to register categories for it without changing the binary to remove the CPlApplet function.

To register this type of applet, the first item to register is the location of the module.

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Control
Panel\Cpls]
AppleName = [REG_EXPAND_SZ]"full-path-to-applet-module"
```

Each module listed in this registry key is loaded and shell32.dll looks for a CPlApplet function in the module. The CPlApplet function provides display name, description, and icon information to the Control Panel window.

> Do not use the CPL_NEWINQUIRE message since it requires the applet module to be reloaded each time Control Panel is opened.

Next, category information is registered as in the following example:

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Control
Panel\Extended Properties\System.ControlPanel.Category]
ModuleFilePath = [REG_SZ]"categoryId,categoryId"
```

Registering a `REG_DWORD` with a single category number instead of a `REG_SZ` string of numbers is supported. Registering a canonical name is similar to Option 1: Registering as a shell command object.

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Control
Panel\Extended Properties\System.ApplicationName]
ModuleFilePath = [REG_SZ]"canonicalNameOfApplet"
```

22.7 INCLUDING THE REGISTERED APPLET WITH WINDOWS SECURITY CENTER

Antimalware apps that report their status to Windows Security Center for centralized monitoring may want to include a link in Windows Security Center to direct the user to an applet that provides more detailed control over the individual security solution. To do this, start with a CPL applet registered with the Control Panel with a registered canonical name (see above).

Including this applet in Windows Security Center requires a new registry key that includes the app's canonical name with the other applets listed there.

Continuing with the Notepad example, to register Notepad in Windows Security Center after following the instructions above to register it as a Shell Command Object CPL, add the following registry key.

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Security
Center\Extensibility\System.ApplicationName\Microsoft.Notepad]
```

There can be multiple keys named after applet's canonical names under `System.ApplicationName`. Each of key will be included in Windows Security Center's applet list.

22.7.1 SAMPLE ISV SERVICE FOR WINDOWS SECURITY CENTER

The provided sample service is a bare-bones service that monitors for all versions of a fictitious antimalware app called “Contoso Antivirus”. This service is provided as an example of how to use the Windows Security Center API and how to monitor for potentially status changing events on the PC.

> This code is an example of a single-purpose service that keeps antivirus status up-to-date. This service could be available for independent download for existing users to detect multiple versions of an app. Typically, an independent monitoring service should be designed to detect legacy versions as well as be compatible with future versions.

Most of the code is implemented in the `StatusSpy` class in the sample.cpp module. On startup, the service detects the current status of the Contoso Antivirus app and then calls `IWscAVStatus::UpdateStatus`. The service monitors for a registry change event and file change event and calls `IWscAVStatus::UpdateStatus` when it discovers configuration changes.

To run the sample, create the sample registry keys that the service looks for. The keys can be created using the file samplesvc.reg. Create a folder under Program Files named Contoso and copy the files AvSvc.exe, ContosoAV.exe and signature.dat (the contents of this file are unimportant) to %Program Files%\Contoso. Otherwise, if the registry keys and/or file are not present, the service will stop itself. The service can be registered with Service Control Manager (SCM) by running the following (elevated) command at the command-line (inserting the correct path):

```
sc create contosoav DisplayName= "Contoso Antivirus" binpath=
"%ProgramFiles%\Contoso\AvSvc.exe"
```

The included wscsample.msi installer package performs these tasks that are needed to run the sample code. The installer package creates the registry keys, places the binaries in Program Files, and registers the service. This MSI can also be created by using the included `WscSample.wxs` WiX script. WiX is available at http://wix.sourceforge.net. To create an MSI installer package, run ‘candle.exe WscSample.wxs’, and then call ‘light.exe WscSample.wixobj’. The WiX script shows some required steps like invoking the Unregister API on uninstall.

This example illustrates a minimal service. It does not do a few things that are highly recommended.

- Currently it just detects when a signature is updated, but it should also detect when a signature goes out of date. This could be accomplished by using a timer callback set to fire once whenever it goes out of date.
- It could also watch for events when the “Contoso Antivirus” service starts or stops (SCM events can be received using WMI) or using the Windows 10 SCM API, not just looking at a registry key that may or may not be consistent with the actual protection state.

The example illustrates a few things that are highly recommended:

- The sample uses only one thread and consumes minimal resources for monitoring.
- The sample runs in the background: It is using asynchronous events and not any inefficient polling mechanisms (although polling will be necessary if waiting for SCM events from WMI since WMI uses polling internally to obtain SCM change notifications).

- The sample only calls `IWscAVStatus::UpdateStatus` when an actual status change has been detected. Calling `IWscAVStatus::UpdateStatus` repeatedly due to events that are not due to any actual change should be avoided.
- The sample detects all legacy versions. In other words, it does not fail when it is unable to detect something that is not common across all versions.

There are two important things to remember if the decision is made to use this code as a foundation for a new service:

- Replace the GUID sample with a newly generated GUID. Make sure to use a different GUID for antivirus, antispyware and firewall if the company has these app offerings.
- Review all the comments prefixed by the word "Note" as these give valuable recommendations.

22.8 FREQUENTLY ASKED QUESTIONS (FAQ)

Here are some frequently asked questions about integrating the app with Windows Security Center for Windows 10.

Q: How do I remediate a yellow or red warning message in the spyware or virus protection pillars?
A: When Windows Security Center detects that the PC has antispyware or antivirus app that is turned on with scanning enabled and an up-to-date signature.

Q: I don't have separate antispyware apps; however, my antivirus app provides spyware protection. How do I report this protection in Windows Security Center?
A: It is acceptable to register as both an antispyware and antivirus as long as users can easily identify that the app protects them against viruses and spyware.

Q: Do I need to inform Windows Security Center when my app has been shut off or disabled?
A: Yes. Windows Security Center in Windows 10 requires that the app communicate with Windows Security Center through the `UpdateStatus` API functions. Set the `productState` attribute to `WSC_SECURITY_PRODUCT_STATE_OFF` (Protection is disabled and real time scanning is off).

Q: If the user uninstalls my app, how do I report this in Windows Security Center?
A: Unregister the app using the unregister API call.

Q: If Windows Security Center disables my app with the `/disable` switch, can it be turned on in the future?
A: Yes, if a user manually turns on the app or Windows Security Center calls the app with the `/enable` switch then the app may continue protecting the user and reporting state to Windows Security Center. However, the app must accurately report the current state of the protection status. For example, if the disabled app was previously expired and the user tries to turn it on the app must report back as expired.

Q: I have heard that Windows Security Center has a new public SDK that other apps can consume. Is this true?
A: Yes, information on Windows Security Center is available at MSDN in the Developer Notes at http://msdn.microsoft.com/en-us/library/gg537273(v=vs.85).aspx

Q: Is it okay to stop and disable the Windows Security Center service ("wscsvc") if my company provides a Security Suite?

A: No, since Windows Security Center provides services to other apps through the public SDK, it is not acceptable for app developers to stop or disable the wscsvc service.

Q: Where can I get more information on code signing?
A: Microsoft has a code signing best practices whitepaper available at http://www.microsoft.com/whdc/winlogo/drvsign/best_practices.mspx.

Q: We receive error 80070307 while trying to register our app with the Windows Security Center.
A: This is related to a linking problem. Please make sure the exe is signed with the integrity bit set (sign with `/IntegrityCheck` flag)

Q: On x64, we get a link error LNK1257.
A: Please disable the LTCG (link-time code generation) flag to get the solution to build, when compiling for amd64 on VS 2008/2010.

Q: Can I reach out to Microsoft support to help me with Windows Security Center integration?
A: Yes, here is a paid support link to request any support help you need from Microsoft Support Team towards WSC APIs. http://support.microsoft.com/oas/default.aspx?prid=15401 This is the only support pipeline available for WSC APIs has it requires NDA verification.

23 APPENDIX C: CODE SIGNING APP BINARIES

This appendix provides detailed information about how to use the Windows® code-signing tools to digitally sign binary apps that are designed for Windows 10. The section covers the following areas:

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

23.1 GETTING STARTED WITH CODE SIGNING

Several approaches can be used to build, sign, and test binary apps. To become more familiar with the code-signing tools, complete all of the examples in this paper on a single PC. However, this paper assumes separate computers for each process, which is often the best option for a production environment.

- **Build PC**.
  The computer that is used to build the app. It should be running Windows XP SP2, Windows Server® 2003, or later versions of Windows.
- **Signing PC**.
  The PC that is used to sign binary app code for Windows 10, it must be running a Windows 8 or later versions of Windows and should have the code-signing tools installed.
- **Test PC**.
  The PC that is used to test the signed appWindows 10 in order to test interoperability with the new Windows Security Center API.

The code-signing tools are available from several sources:

- The Microsoft Windows Software Development Kit (SDK) for Windows 10 contains information and tools for developing 32-bit and 64-bit Windows-based apps. It is available as a free download.
- The Microsoft .NET Framework SDK contains the required information and tools to develop managed-code apps. Like the Platform SDK, it is available as a free download.

For more information, see the Resources section.

> The tools in the Windows SDK are not re-distributable. For more information, see the end-user license agreement (EULA) for the Windows SDK.

The following table summarizes the sources for code signing and related tools. For links to these sites, see Resources at the end of this appendix.

Tool	Windows SDK	Additional sources
MakeCert	SDK	.NET SDK
CertMgr	SDK	.NET SDK
SignTool	SDK	.NET SDK

Capicom.dll v.2.1.0.2	SDK	Download Center
PVK2PFX	SDK	

23.2 CODE-SIGNING TOOLS OVERVIEW

The code-signing tools in the previous table are used for both test-signing and release-signing of binary apps. This section briefly describes each tool. The walkthrough that follows shows how the tools are used, including examples of typical command-line parameters.

23.2.1 MAKECERT

MakeCert generates digital certificates that can be used for test-signing. They can be either self-signed or issued and signed by the Root Agent key. Self-signed certificates are recommended for test-signing drivers. The test certificate can be placed in a file, a system certificate store, or both. Windows 10 accept test certificates that are generated by MakeCert for test-signing.

> Generally, certificates that are issued by a third-party certification authority (CA) to be used for production signing should not be used for test-signing. For more information, see Code-Signing Best Practices.

23.2.2 CERTMGR

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

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

23.2.3 SIGNTOOL

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

- Whether it was issued by a trusted CA.
- Whether it has been revoked.
- Optionally, whether the certificate is valid for a specific policy.

SignTool can be used for a number of other purposes, including:

- Verifying the files in a signed catalog file.
- Verifying signatures against different Authenticode policies.
- Displaying a signature's certificate chain.
- Displaying the SHA1 hash value of a file.
- Displaying errors for files that did not verify.
- Adding and removing catalog files from the catalog database.

Only some of the functionality listed above will be used in this walkthrough.

23.2.4 CAPICOM.DLL

Capicom.dll exports an API that app developers can use to add security that is based on cryptography to apps. Because SignTool uses this dynamic-link library (DLL), both files must be present on the signing PC. Capicom.dll should be copied to the same directory as SignTool. The latest version is 2.1.0.2.

23.2.5 PVK2PFX

PVK2PFX moves certificates and private keys that are stored in *.spc and *.pvk files to personal information exchange (*.pfx) files.

To be used SignTool, a key must be stored in a *.pfx file. However, some CAs use the *.pvk file format to store the private key of the digital certificate and a *.spc or *.cer file to store the public key. In particular, Verisign Class-3 certificates are currently packaged as a pair of *.pvk and *.spc files. Before using such a certificate for code signing, convert the *.pvk and *.spc files into the *.pfx format.

> When possible, the preferred approach is to store private keys in a hardware security module, such as a smartcard. For more information on managing private keys, see Code-Signing Best Practices.

23.3 HOW TO TEST-SIGN A BINARY APP

> Test-signed binary apps are supported on Windows 10 only for testing purposes. They must not be used for production purposes or released to customers for use with Windows 10.

Test-signing refers to using a test certificate to sign a prerelease version of app for use on test PCs. Windows 10 support test-signing of binary apps. In particular, it allows developers to sign app binaries by using self-signed certificates that the MakeCert utility program generates. This new capability allows developers to test signed app binaries on Windows 10 before they are ready for release-signing.

> Windows 10 accept test certificates that are generated by MakeCert or test certificates that are issued by any CA, including enterprise CAs.

To test-sign app binaries for Windows 10 (basic procedure):

1. Prepare a PC for test-signing.
2. Create a test certificate by using MakeCert.
3. Install the test certificate in the Trusted Root Certification Authorities certificate store.
4. Test-sign the app binary file.

> When following these directions, use an account that is a member of the local Administrators group on the PC where the signing tools are installed.

23.3.1 STEP 1: PREPARE THE PC FOR TEST-SIGNING

To prepare a PC for test-signing, install the signing tools.

The examples in the walkthrough assume that the SDK is installed and uses an elevated SDK command window. The window's PATH environment variable includes the directory that contains the code-signing tools. An elevated command window is one that runs with administrative privileges.

To open an elevated command window

- Click the **Start** button, point to **All Programs** and click **Microsoft Windows SDK**.
- Right-click **CMD Shell** and select **Run as Administrator** from the shortcut menu.

23.3.2 STEP 2: CREATE A TEST CERTIFICATE BY USING MAKECERT

Test-signing requires a test certificate. After a test certificate is generated, it can be used multiple times to test-sign app binary code. The driver package's project folder should be used for test-signing.

Consider, for example, this MakeCert command-line:

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

This MakeCert example command does the following:

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

Parameter	Description
-r	Creates a self-signed certificate with the same issuer and subject name.
-pe	Makes the certificate's private key exportable to the signing machine.
-ss ***StoreName***	The certificate store name that stores the test certificate **`PrivateCertStore`**.
-n ***X500Name***	The certificate subject's X500 name, which is Contoso.com(Test).
ContosoTest.cer	The certificate's output file name.

After the certificate is created and a copy is put in the certificate store, the Microsoft Management Console (MMC) **Certificates** snap-in can be used to view it.

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

1. Click the **Start** button and click **Start Search**.
2. To start the **Certificates** snap-in, type **Certmgr.msc** and click **OK**.
3. In the snap-in's left pane, expand the **PrivateCertStore** certificate store folder and double-click **Certificates.**

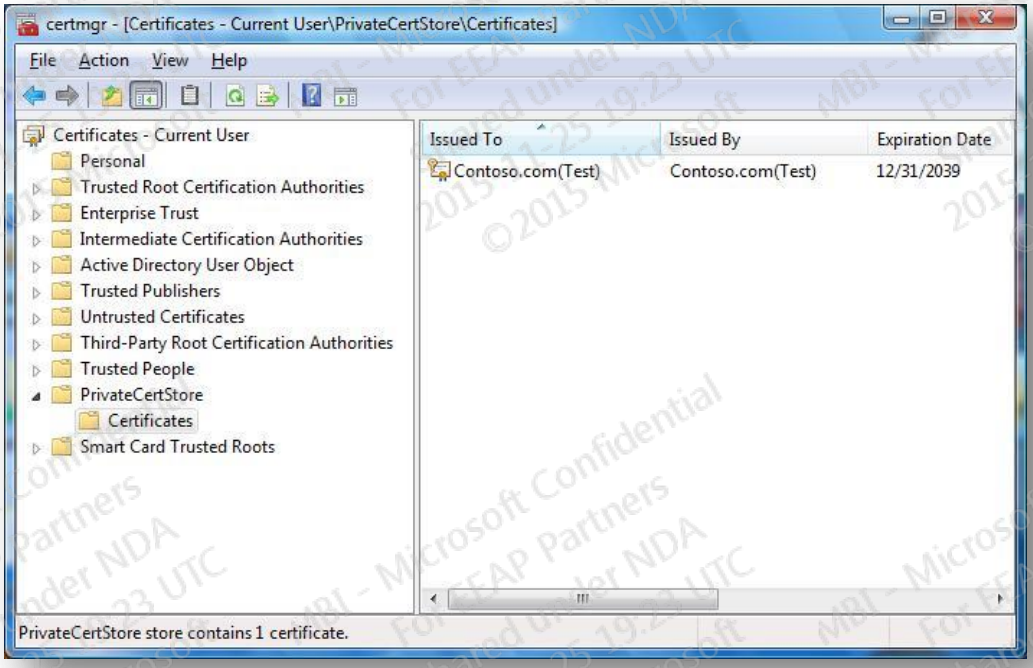


Figure 6: Certificate store that shows the test certificate

To view the certificate details, double-click the certificate in the right pane.

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

Figure 7: Viewing the test certificate

23.3.3 STEP 3: INSTALL THE TEST CERTIFICATE IN THE TRUSTED ROOT CERTIFICATION AUTHORITIES CERTIFICATE STORE

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

The CA certificate must be added to the Trusted Root Certification Authorities certificate store only once. It can then be used for all test-signature verification steps on that system.

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

For example, the following command uses Certmgr.exe to put a copy of the Contoso.com(test) certificate in the `localMachine` Trusted Root Certification Authorities certificate store:

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

Parameter	Description
/add ***CertificateName***	Adds the certificate to the specified certificate store. The file name containing the certificate in the example is ContosoTest.cer.
/s	Specifies that the certificate is to be added to a system store.
/r ***CertStoreLocation***	Specifies the location of the Trusted Root Certification Authorities certificate store for the local PC as "localMachine root". This means that it is stored under `HKEY_LOCAL_MACHINE`.

To view the Contoso.com(test) certificate in the Trusted Certification Authorities certificate store, start the Certificates MMC snap-in, as discussed in Step 2.

23.3.4 STEP 4: TEST-SIGN AN APP BINARY FILE WITH AN EMBEDDED SIGNATURE

Embedded-signing refers to adding a digital signature to the app's binary file itself.

23.4 HOW TO EMBEDDED-SIGN AN APP BINARY

SignTool is used to embedded-sign binary files, including test-signing binary files by using a test certificate. This example uses SignTool to test-sign the sample binary file, app.exe.

The following command-line example signs app.exe, by using the test certificate that was created in Step 2, Contoso.com(Test). It also adds a timestamp to the digital signature:

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

Parameter	Description
sign	Signs the specified binary file.
/v	Specifies the verbose option, which displays successful execution and warning messages.
/ph	Specifies that page hashing will be set. This is only available on Windows 8 and higher.
/s CertificationStoreName	Specifies the certificate store, `PrivateCertStore`, that contains the test certificate.
/n TestCertificateName	Specifies the test certificate with the subject name Contoso.com(Test).
/t URL	Specifies a digital signature, time-stamped by the TSA that the URL indicates.
FileName	Specifies the name of the binary file to be embedded-signed, c:\app.exe. For best results, copy the file locally rather than signing it on a network share.

SignTool can also be used to verify the embedded signature. The certificate for the CA that issued the test certificate must be installed in the Trusted Root Certification Authorities certificate store before verification. For more information on installing the test CA certificate, see Step 3 of this walkthrough.

The following example uses SignTool to verify the test-signature on **app.exe**:

```
Signtool verify /pa /v c:\app.exe
```

Parameter	Description
verify	Verifies the signature in the specified file.
/pa	Specifies to use the Authenticode verification policy when verifying the signature.
/v	Specifies the verbose option, which displays successful execution and warning messages.
FileName	Specifies the name of the binary file that contains the signature to be verified, c:\app.exe.

23.5 HOW TO INSTALL AND VERIFY A TEST-SIGNED APP

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

> While following the walkthrough to become familiar with the tools and process, continue testing the app on the same system that was used for test-signing.

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

1. Install the test certificates.
2. Install the test-signed app.

23.5.1 STEP 1: INSTALL THE TEST CERTIFICATES

To prepare the test system, configure it with the certificates that were used to test-sign the app.

Two certificates must be installed on the test system:

- The test certificate that was used to sign the app binary.
- The certificate of the CA that issued the test certificate.

> If the test PC is the same one used to test-sign the app, the certificates are already installed. Skip this step and proceed to Step 2.

The test certificate, Contoso.com(Test), was created in Step 2: create a test certificate by using MakeCert to sign the app.exe app binary. The certificate is packaged in a file that is named ContosoTest.cer.

To install the test certificates using the Windows SDK

1. Copy ContosoTest.cer from the system that was used to generate the test certificate to the test system.
2. Open an elevated command SDK window.
3. Install Contoso.com(Test)in the localMachine Trusted Root Certification Authorities certificate store by running the following command:

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

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

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

Parameter	Description
/add CertificateName	Adds the certificate in the specified certificate file to the certificate store.
/s	Specifies that the certificate store is a system store.
/r RegistryLocation	Specifies that the registry location of the system store is under HKEY_LOCAL_MACHINE.
CertificateStore	Specifies the certificate store, trusted publisher.

The second step is to put the certificate of the CA that issued the test certificate in the Trusted Root Certification Authorities certificate store. The issuing CA for Contoso.com(Test) is Contoso.com(Test).

The following command uses Certmgr.exe to put a copy of the Contoso.com(Test) certificate in the Trusted Root Certification Authorities certificate store, also under HKEY_LOCAL_MACHINE:

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

Parameter	Description
/add CertificateName	Adds the certificate to the specified certificate store. The file name

	containing the certificate in the example is ContosoTest.cer.
/s	Specifies that the certificate is to be added to a system store.
/r CertStoreLocation	Specifies the location of the Trusted Root Certification Authorities certificate store for the local PC as “localMachine root". This means that it is stored under `HKEY_LOCAL_MACHINE`.

To install the test certificates from Windows Explorer

1. Copy ContosoTest.cer from the system that was used to generate the test certificate to the test system.
2. Open Windows Explorer and browse to the directory where ContosoTest.cer is located.
3. Right click **ContosoTest.cer**, and then click **Install Certificate**.
4. When the Certificate Import wizard opens, click **Next**.
5. Select **Place All Certificates in the Following Store**, and then click **Browse**.
6. Select **Trusted Root Certification Authorities**, and then click **OK**.
7. Click **Next** and then click **Finish**.
8. Right click **ContosoTest.cer**, and then click **Install Certificate**.
9. When the Certificate Import wizard opens, click **Next**.
10. Select **Place All Certificates in the Following Store**, and then click **Browse**.
11. Select **Trusted Publishers,** and then click **OK**.
12. Click **Next** and then click **Finish**.

To view the certificate stores on the test system, use the MMC Certificates snap-in. For details, see "Step 3: Install the Test Certificate in the Trusted Root Certification Authorities Certification Store" earlier in this appendix.

23.5.2 STEP 2: ENABLE THE KERNEL-MODE TEST-SIGNING BOOT CONFIGURATION OPTION

Test-signed apps cannot be verified by Windows Security Center by default. The PC must be put into test mode. This will enforce the signing of antimalware apps for Windows Security Center operability, but will allow the test certificates to chain to any root CA.

To use the BCDEdit tool to enable the boot configuration test-signing option

1. Open an elevated command window by right-clicking the icon and clicking **Run as Administrator.**
2. Use the following command to enable test-signing:

```
bcdedit.exe /set TESTSIGNING ON
```

Parameter	Description
/set	Sets a boot entry option value.
TESTSIGNING ON	Sets the TESTSIGNING option to ON.

BCDEdit is the new boot configuration editor and is included with Windows 8 and later versions of Windows. For more information on BCDEdit, see the white paper titled “Boot Configuration Data in Windows 8.”

When the `BCDEdit` option for test-signing is enabled, Windows 8 and higher do the following:

- Displays a watermark with the text “Test Mode” in all four corners of the desktop, to remind users the system has test-signing enabled.

- The certificate validation is not required to chain up to a trusted root certification authority. However, each antimalware app file must have a digital signature to be interoperable with Windows Security Center.

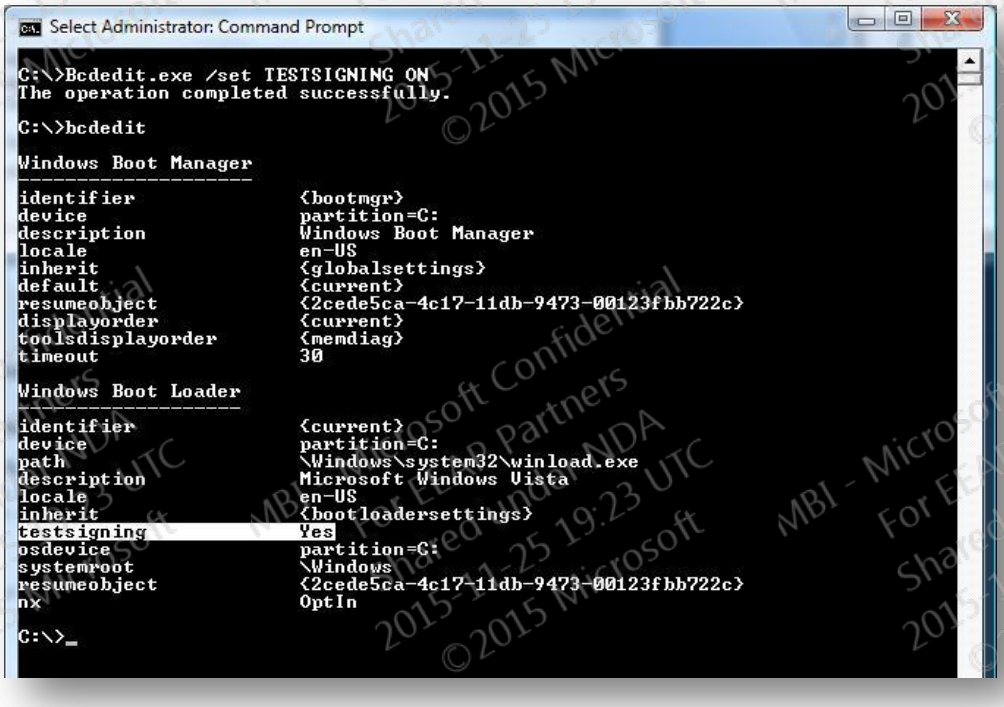


Figure 8: Using BCDEdit to enabled test-signing

23.5.3 STEP 3: ENABLE CODE INTEGRITY EVENT LOGGING AND SYSTEM AUDITING

Code Integrity is the kernel-mode component that implements signature verification. It generates system events that are related to image verification and logs the information in the Code Integrity log:

- The Code Integrity operational log view shows only image verification error events.
- The Code Integrity verbose log view shows the events for successful signature verifications.

The following procedure shows how to enable Code Integrity verbose event logging to view all successful signature verification events.

To enable Code Integrity verbose event logging

1. Start Event Viewer.

   To do this, click the **Start** button, right-click **Computer,** and then click **Manage**. This starts the Computer Management Control Panel item.

   Event Viewer can also be started from an elevated Command Prompt window by running the `eventvwr.exe` command.

2. Expand the **Event Viewer's App and Services Log** by clicking the associated triangle in the left pane. Further expand the folders under **Microsoft\Windows\Code Integrity**.
3. Click the **Code Integrity** node to give it focus.

4. Right-click **Code Integrity,** click **View,** and then click **Show Analytic and Debug Logs** on the shortcut menu.

   This adds an additional node called **Verbose**.

5. Right-click the **Verbose** node and then click **Properties**.
6. On the **General** tab of the **Properties** dialog box, check the **Enable Logging** option.

System event records can also be enabled, which include Code Integrity image verification failure events.

To enable the audit policy to generate audit events in the system category for failed operations, enter the following command in an elevated Command Prompt window:

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

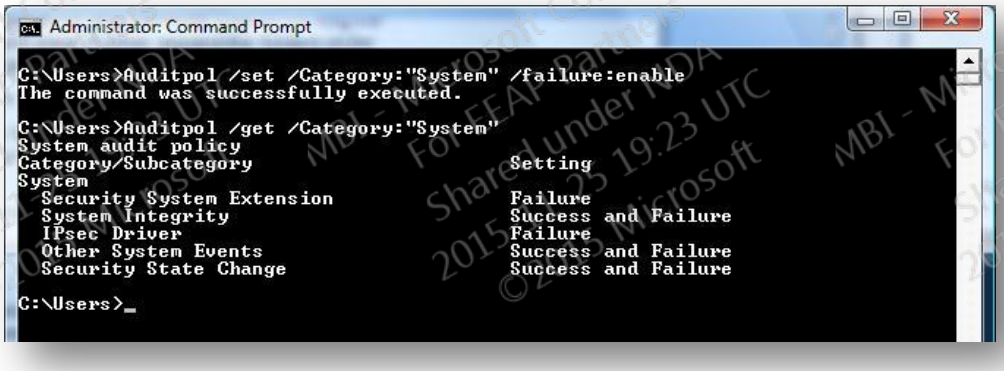


Figure 9: Enabling security audit policy

23.5.4 STEP 4: REBOOT THE TEST PC

Reboot the test PC to allow the kernel-mode boot configuration options to take effect.

23.5.5 STEP 5: COPY AND INSTALL THE TEST-SIGNED APP TO THE TEST PC

After the system has rebooted, the test-signed app can be installed and loaded.

23.5.6 STEP 6: VERIFY THAT THE TEST-SIGNED APP IS OPERATING CORRECTLY

Once the test-signed antimalware app is installed and registered with Windows, its status should be reflected in Windows Security Center. If it is not, check the Code Integrity log for signature verification errors.

23.6 HOW TO RELEASE-SIGN AN APP BINARY

Release-signing identifies the publisher of an app binary that runs on Windows 8 and higher. Binaries are release-signed by using a software publisher certificate (SPC) and a related cross-certificate. This section provides a walkthrough of the release-signing process. The actual process and infrastructure for release-signing will vary from company to company. For more information see Microsoft's Code Signing Best Practices guide.

To release-sign app binaries for Windows 8 and higher

1. Prepare a PC for release-signing.
2. Obtain a SPC for signing app binaries.
3. Download the related cross-certificate.
4. Release-sign the driver image file.

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

23.6.1 STEP 1: PREPARE THE PC FOR RELEASE-SIGNING

> This step is identical to Step 1 in the section titled "How to Test-Sign a Binary App." It is repeated here for convenience.

To prepare a PC for release-signing, install the signing tools.

The examples in the walkthrough assume that the SDK is installed and uses an elevated SDK command window. The window's PATH environment variable includes the directory that contains the code-signing tools. An elevated command window is one that runs with administrative privileges. An elevated command window is not strictly necessary once capicom.dll is registered, but access to the keys is required.

To open an elevated command window

1. Click the **Start** button, point to **All Programs** and click **Microsoft Windows SDK**.
2. Right-click **CMD Shell** and select **Run as Administrator** from the shortcut menu.

23.6.2 STEP 2: OBTAIN AN SPC

Release-signing requires a code-signing certificate, also referred to as a Software Publisher Certificate (SPC) from a commercial CA. Follow the CA's instructions for how to acquire the code-signing certificate and install the private key on the signing PC. For a list of cross-certificate SPC CAs, see Resources at the end of this appendix.

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

SignTool can be used to sign binaries with code-signing certificates issued from commercial CAs.

Some certification authorities store the digital certificate's private key in a *.pvk file and store the public key in an *.spc or *.cer file. For example, Verisign Class-3 certificates are currently packaged as a pair of *.pvk and *.spc files. Convert the *.pvk and *.spc files into the *.pfx format for secure portability of the certificate and private key. The Pvk2pfx code-signing tool produces a *.pfx file from *.pvk and *.spc files.

The following example converts a *.pvk file that is named abc.pvk and a .*spc that is named abc.spc into a *.pfx file that is named abc.pfx:

```
Pvk2pfx -pvk abc.pvk -pi pvkpassword -spc abc.spc -pfx abc.pfx -po
pfxpassword -f
```

Parameter	Description
-pvk PvkFileName	The input .pvk file, abc.pvk.
-pi PassWord	The .pvk file's password.
-spc SpcFileName	The input SPC file, abc.spc.
-pfx	The output PFX file, abc.pfx. If the **-pfx** argument is omitted Pvk2pfx opens an Export Wizard and ignores the **-po** and **-f** parameters.
-po OptionalPassword	The .pfx file's password. If no .pfx password is specified, the .pfx file is assigned the same password that is associated with the .pvk file, as specified by the **-pi** argument.
-f	An overwrite of an existing .pfx file.

For signing apps, the certificates and key stored in the *.pfx file must be imported into the local Personal certificate store. SignTool does not support using *.pfx files. The restriction is due to a conflict in adding cross-certificates in the signature while using a certificate from a *.pfx file.

To import the *.pfx file into the local Personal certificate store

1. Start **Windows Explorer** and then double-click the *.pfx file to open the Certificate Import Wizard.
2. Complete the Certificate Import Wizard to import the code-signing certificate into the Personal certificate store.
3. The certificate and private key are now available for SignTool to use.

An alternate way to import the *.pfx file into the local Personal certificate store is with the CertUtil command-line utility.

To import the *.pfx file by using CertUtil

1. Open a Command Prompt window and change the directory to the folder that contains the *.pfx file.
2. Run the following command:

```
certutil –user –p password –importPFX PFXFile
```

Parameter	Description
-p password	The .pvk file's password.
-importPFX PFXfile	The .pfx file to import.
-user	Places the certificate in the “Current User” Personal store.

The **certutil** command-line has been corrected to include the **-user** parameter.

To view SPC properties, use the MMC **Certificates** snap-in (Certmgr.msc) to view the certificates in the Personal certificate store.

To use the MMC Certificates snap-in

1. Click the **Start** button, and then click **Run**.
2. To start the **Certificates** snap-in, type **certmgr.msc** and click **OK**.
3. In the snap-in's left pane, select the **Personal certificate store folder**.
4. Click the Certificates folder and double-click the certificate that is to be used for release-signing.
5. On the **Details** tab of the **Certificate** dialog box, select **Subject** from the list of fields to highlight the certificate's subject name. This is the name that is used with SignTool's `/n` argument in the examples in this section.

After finishing the signing process, remove the certificate and private key from the system currently in use. Certmgr.exe can be used from the command-line to view, export, or delete certificates from the Personal certificate store.

Follow the steps below to delete a certificate.

1. Open the **Certificates** MMC snap-in (described above) and browse to the certificate to delete.
2. Right click the certificate, point to **All Tasks**, and then click **Export**.
3. On the wizard Welcome page, click **Next**.
4. Select **Delete the private key if export is successful** and then click **Next**.
5. Complete the remainder of the wizard.
6. Delete the certificate that was just exported.

23.6.3 STEP 3: OBTAIN A CROSS-CERTIFICATE

A cross-certificate and an SPC are required for release-signing.

To determine which cross-certificate is needed for antimalware app code signing

1. Click the **Start** button, and then click **Run**.
2. To start the MMC **Certificates** snap-in, type **Certmgr.msc** and click **OK**.
3. Locate the signing certificate in the certificate store. The certificate should be listed in one of the following locations, depending on how it was installed:
   Current user, Personal certificate store
   Local machine certificate store
4. To open the **Certificate** dialog box, double click the certificate.
5. On the **Certification Path** tab, select the top-most certificate in the certification path.
6. Click **View Certificate**.
7. On the **Details** tab, find the **Issuer Name** and **Thumbprint** for the CA that issued the signing certificate.
8. Locate the corresponding cross-certificate in the root authority cross-certificate list Web page.
9. Download a copy of the appropriate cross-certificate to the correct directory.

23.6.4 STEP 4: RELEASE-SIGN A BINARY APP FILE BY USING AN EMBEDDED SIGNATURE

Embedded-signing refers to adding a digital signature to the app's binary file itself.

SignTool is used to embedded-sign binary files, including release-signing binary app files by using an SPC. The following example signs the binary file app.exe by using the SPC for Verisign and the related cross-certificate. It also adds a timestamp to the digital signature, which is recommended to prevent the signature from expiring when the certificate expires.

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

Parameter	Description
Sign	Signs the specified binary file.
/v	Specifies the verbose option, which displays successful execution and warning messages.
/ph	Specifies that page hashing will be set. This is only available on Windows 8 and higher.
/ac CrossCertName	Adds the MSCV-VSClass3.cer cross-certificate.
/s CertificationStoreName	Specifies the certificate store that is named my, which contains the SPC.
/n CertName	Specifies an SPC that is named Contoso.com. For a description of how to obtain an SPC's subject name, see the procedure titled "View SPC Properties," "Step 2: Obtain an SPC," earlier in this appendix.
/t URL	Specifies a digital signature, time stamped by the TSA that the URL indicated.
FileName	The name of the binary file to be embedded-signed, app.exe.

After the file is embedded-signed, use SignTool to verify the signature. Check the results to verify that the root of SPC's certificate chain for kernel policy is "Microsoft Code Verification Root." The following command-line verifies the signature on app.exe:

```
Signtool verify /kp /v c:\app.exe
```

Parameter	Description
Verify	Verifies the signature in the specified file.
/kp	Verifies that the kernel policy has been met.
/v	Specifies the verbose option, which displays successful execution and warning messages.
FileName	Specifies the name of the binary file that contains the signature to be verified, c:\app.exe.

23.7 HOW TO INSTALL AND VERIFY A RELEASE-SIGNED APP

23.7.1 STEP 1: DISABLE THE KERNEL-MODE TEST-SIGNING BOOT CONFIGURATION OPTION

If the kernel-mode test-signing boot configuration option was enabled to install a test-signed app, disable the option.

To use the BCDEdit tool to disable the boot configuration test-signing option

1. Open an elevated command window by right-clicking the icon and clicking **Run as administrator**.
2. Use the following command to enable test-signing:

```
bcdedit.exe /set TESTSIGNING OFF
```

Parameter	Description
/set	Sets a boot entry option value.

TESTSIGNING OFF	Sets the TESTSIGNING option to OFF.

By default, the test-signing option is not defined in Windows 8 and higher. This is equivalent to setting the test-signing boot configuration option to OFF.

23.7.2 STEP 2: ENABLE CODE INTEGRITY EVENT LOGGING AND SYSTEM AUDITING

> This step is identical to step 3 in the section titled "How to Install and Verify a Test-Signed App." It is repeated here for convenience.

Code Integrity is the kernel-mode component that implements signature verification. It generates system events that are related to image verification and logs the information in the Code Integrity log:

- The Code Integrity operational log view shows only image verification error events.
- The Code Integrity verbose log view shows the events for successful signature verifications.

The following procedure shows how to enable Code Integrity verbose event logging to view all successful signature verification events.

To enable Code Integrity verbose event logging

1. Start Event Viewer. To do this, click the **Start** button, right-click **Computer,** and then click **Manage**. This starts the Computer Management Control Panel item. Event Viewer can also be started from an elevated Command Prompt window by running **eventvwr.exe** at the command-line.
2. Expand the **Event Viewer's App and Services Log** by clicking the associated triangle in the left pane. Further expand the folders under **Microsoft\Windows\Code Integrity**.
3. Click the **Code Integrity** node to give it focus.
4. Right-click **Code Integrity,** click **View,** and then click **Show Analytic and Debug Logs** on the shortcut menu.
5. This adds an additional node called **Verbose**.
6. Right-click the **Verbose** node and select **Properties** from the shortcut menu.
7. On the **General** tab of the **Properties** dialog box, check the **Enable Logging** option.

System event records can also be enabled, which include Code Integrity image verification failure events.

To enable the audit policy to generate audit events in the system category for failed operations, enter the following command from an elevated command window:

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

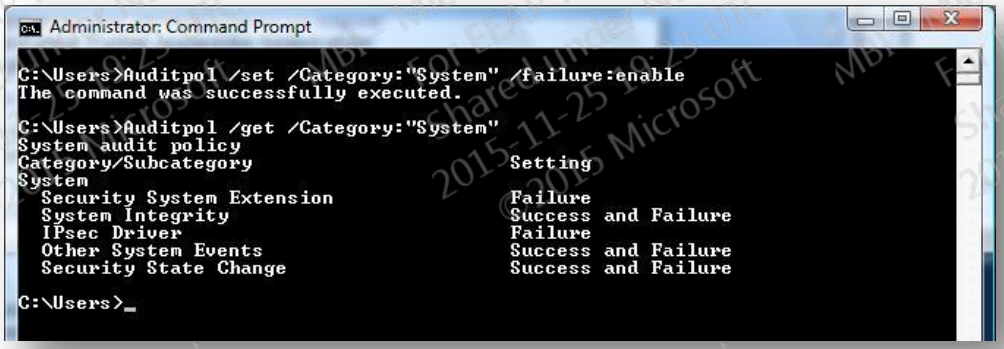


Figure 10: Enabling the system category audit policy.

23.7.3 STEP 3: REBOOT THE TEST PC

Reboot the test PC to allow the kernel-mode boot configuration options to take effect.

23.7.4 STEP 4: COPY AND INSTALL THE RELEASE-SIGNED APP TO THE TEST PC

After the system has rebooted, the release-signed app can be installed and loaded.

23.7.5 STEP 5: VERIFY THAT THE RELEASE-SIGNED APP IS OPERATING CORRECTLY

Once the release-signed antimalware app is installed and registered with Windows, its status should be reflected in Windows Security Center. If it is not, check the Code Integrity log for signature verification errors.

23.8 RESOURCES

For questions about digital signatures for kernel-mode drivers, send email to: signsup@microsoft.com.

The following links provide more information about driver signing and related topics.

General:

- Windows SDK download site: http://www.microsoft.com/downloads/details.aspx?FamilyID=c2b1e300-f358-4523-b479-f53d234cdccf&DisplayLang=en
- Microsoft .NET Framework SDK download site: http://www.microsoft.com/downloads/details.aspx?FamilyID=fe6f2099-b7b4-4f47-a244-c96d69c35dec&DisplayLang=en
- Boot Configuration Data in Windows 8 and Windows 8.1: http://www.microsoft.com/whdc/system/platform/firmware/bcd.mspx

Code Signing:

- Code-Signing Best Practices: http://www.microsoft.com/whdc/winlogo/drvsign/best_practices.mspx
- Creating strong passwords: http://www.microsoft.com/resources/documentation/windows/xp/all/proddocs/en-us/windows_password_tips.mspx?mfr=true

- Deploying Authenticode with Cryptographic Hardware for Secure Software Publishing: http://www.microsoft.com/technet/security/topics/cryptographyetc/authenticodets.mspx
- Microsoft Cross-Certificates for Windows Vista Kernel Mode Code Signing: http://www.microsoft.com/whdc/winlogo/drvsign/crosscert.mspx

This Web page includes:

- A list of Root Authority cross-certificates
- A list of CAs that provide SPCs for kernel-mode code signing

Tools:

- Certificate Creation Tool (Makecert.exe): http://go.microsoft.com/fwlink/?LinkId=95774
- Certificate Manager Tool (Certmgr.exe): http://go.microsoft.com/fwlink/?LinkId=95775
- Debugging Tools for Windows: http://www.microsoft.com/whdc/DevTools/Debugging/default.mspx
- SignTool: http://go.microsoft.com/fwlink/?LinkId=95786
- Using MakeCat: http://go.microsoft.com/fwlink/?LinkId=95790
- Using PVK2PFX: https://msdn2.microsoft.com/en-us/library/aa906332.aspx

24 APPENDIX D: HOW WSC WORKS IN WINDOWS 10

24.1 AUTO-DETECTION WITH WINDOWS SECURITY CENTER APIS

Antimalware apps use the Windows Security Center APIs to report their health state to Windows Security Center. This approach allows antimalware apps to provide notifications via WSC to users without having to create their own notification mechanism.

Action Center provides a centralized interface to determine whether certain app is installed on the PC to help protect users from Internet security threats. Best practices suggest a PC that has the following to help protect it from threats:

- Automatic Updating is enabled.
- An antimalware app is installed and scans all files on-access with up-to-date signatures.
- A firewall is installed and turned on.
- Other security settings are in the user's recommended statuses of Internet protection and User Account Control:
    - Internet security settings status
    - User account control notifications
    - Windows Smart Screen settings
    - Network access protection settings
    - Windows Activation Status

To allow Windows Security Center to reliably detect the installation and status of antivirus, antispyware, and firewall apps, the apps must notify Windows Security Center with status on the installation and current protection status of antivirus, antispyware, and firewall apps by using the appropriate Windows Security Center APIs, as described in this document.

24.1.1 HOW DOES WINDOWS SECURITY CENTER DETECT ANTIVIRUS, ANTISPYWARE, AND FIREWALL?

Windows Security Center detects antivirus, antispyware, and firewall apps and provides the current protection status of the app.

App developers must call functions in the new Windows Security Center COM API to indicate the app is installed and the current state/status of the app. A sample library is provided with this whitepaper to enable integration with minimal development.

This can be accomplished by:

1. App developer can develop a service that will run independently of their existing antivirus, antispyware, or firewall code and will monitor the status of the PC. The benefit of this approach is that binary updates are not necessary and therefore a single code base could be developed that can detect all app versions. The Windows Security Center team will provide such a sample service that calls into the Windows Security Center library to update current status. The downside to such an approach is that it requires extra

resources to continually monitor the PC (extra service, threads, memory) and it can be tricky to implement correctly.

- OR –

2. Optimally, the calling of functions in the new Windows Security Center COM API to update the state can be integrated directly into the app. This approach requires fewer resources than running an independent service and it should be easier to maintain the state correctly.

24.1.2 FIREWALL REQUIREMENTS

This section provides an overview of the Windows Security Center functional requirements for antivirus apps. The overall requirements for firewall are, at a high level, to inform users that they are protected from:

- Malicious code on the Internet, which can use vulnerable services on a PC to infect it with a worm or other payload.
- External parties accessing private data.

Firewall apps send status updates to Windows Security Center by using the Windows Security Center IWscFWStatus API. Windows Security Center uses these status updates to determine whether the firewall app is protecting the PC or is out-of-date.

24.1.3 ANTIVIRUS REQUIREMENTS

This section provides an overview of the Windows Security Center functional requirements for antivirus apps.

24.1.3.1 ANTIVIRUS STATES

At a high level, there are six main antivirus states that Windows Security Center detects. These four states assume that the antivirus app has been updated to use the Windows Security Center APIs.

WSC state	Definition	WSC notification
Not installed	Antivirus app is not installed on the PC.	"Find a Program" button.
On and up-to-date	Antivirus app is installed and protecting the user. The app subscription is up-to-date and real-time scanning is turned on. The user has the same level of protection as would be available on a newly installed/purchased/activated antivirus program. The antivirus is reporting ON to WSC.	User is in a protected state, no action required.
On and out-of-date	Antivirus app is installed and not protecting the user. The app subscription or signatures may be out-of-date.	"Update now" button
Expired	Antivirus app is installed and the user's subscription has expired. This means that the app is not providing the user and the operating system with the level of protection that would be performed with an installed and activated app including real-time protection and regular updating of signatures. Note: antivirus apps are required to report as expired if: The app no longer offers real-time protection.	Day 1-15: "Update now" button Day < 15: User choices for renew, new purchase or install Windows Defender

	The user is no longer updating signatures (app deliberately stops users from updating signatures). The subscription between the user and the app developer is not initiated or no longer active.	
Snoozed	Antivirus app is installed, snoozed and not actively protecting the user.	"Turn on now" button
Off	Antivirus app is installed and is reporting OFF to WSC. It is not protecting the user, performing any scanning (real-time or background), and all services or components of the antivirus are turned off. Note: antivirus apps that report off must stop messaging to the user.	"Turn on now" button

24.1.4 HOW WINDOWS SECURITY CENTER DETERMINES THE LEVEL OF ANTIVIRUS PROTECTION

To determine if an antivirus app is protecting a user, Windows Security Center looks at four parameters:

- Is antivirus app installed and performing real-time/on-access scanning? (yes/no)
- If yes, are the signatures more than 15 days old? (yes/no)
- Is antivirus app in a snooze state? (yes/no)
- Has the antivirus app subscription expired? (yes/no)

24.1.4.1 IS AN ANTIVIRUS APP INSTALLED AND PERFORMING REAL-TIME/ON-ACCESS SCANNING?

Antivirus apps report status updates to Windows Security Center by using the Windows Security Center `IWscAVStatus` API. If more than one antivirus app reports status updates to Windows Security Center, each antivirus app on the system must follow the steps outlined above. In the most common of these scenarios, a user may have two antivirus apps installed, but only one with real-time/on-access scanning enabled. In this case, the user will get feedback only on the enabled antivirus app. In the highly unlikely event that more than one antivirus app is enabled with real-time/on-access scanning, Windows Security Center will simply inform users that they are protected by more than one app.

Real-time/on-access scanning is one of the key functions of any antivirus app, especially for consumer users. Real-time/on-access scanning data provides users with a perspective on their current functioning level of security. Without real-time scanning, a user's PC may be vulnerable to certain forms of malware.

24.1.4.2 IS THE SIGNATURE SET UP-TO-DATE?

Antivirus apps report status updates to Windows Security Center by using the Windows Security Center `IWscAVStatus` API. The process and messaging within Windows Security Center will refer to the third-parties as "reporting" their status. For example, Contoso antivirus reports that it is out-of-date when the signatures are 15 days old. The 7-day mark for stale signatures will signal to Windows that the signatures are out-of-date. The app developer is responsible for reporting that its signatures are up-to-date using the `UpdateStatus` API by setting

the `productUptoDate` attribute to true or false. Windows 10 use only the information reported to it via WSC and appropriately informs the user of the state of their antivirus app.

24.1.4.3 IS THE APP IN AN OFF OR SNOOZED STATE?

When an antivirus app is in an off or snoozed state, the PC is in an unprotected state and Windows Security Center does not consider the antivirus app actively protecting the PC.

Important: antimalware app can manage the notification experience when a PC wakes up from hibernation, sleep, or from being shut down for a period of time. As the PC wakes up from a shutdown state, there is a short delay that may last up to 6 minutes before Action Center reports on security status. The app can manage this experience by determining how to report the security status to Windows Security Center.

24.1.4.4 IS ANTIVIRUS APP EXPIRED?

When an antivirus app reports `WSC_SECURITY_PRODUCT_STATE_EXPIRED`, Windows Security Center does not consider the antivirus app actively protecting the PC and will show the user a red warning (a tile in Action Center, and a bubble notification on the desktop) that instructs the user to renew their subscription. This warning will continue to show until an antivirus sends a status update to Windows Security Center and reports `WSC_SECURITY_PRODUCT_STATE_ON` and sets the `productUptoDate` attribute to true.

If after 15 days the user takes no action to renew their subscription, Windows Security Center will show a new warning which provides two additional options (purchase an antimalware online or enable Windows Defender). This warning will also continue to show until the user decides to choose a protection option and an antivirus app reports either `WSC_SECURITY_PRODUCT_STATE_ON, WSC_SECURITY_PRODUCT_STATE_OFF` or `WSC_SECURITY_PRODUCT_STATE_SNOOZED`.

24.1.5 ANTISPYWARE REQUIREMENTS

This section provides an overview of the Windows Security Center functional requirements for antispyware apps. The overall requirements for antispyware apps are to inform users that they are protected from:

- The installation of potentially unwanted software where the software's main intent is to collect personal information without knowledge or permission or change system configurations without appropriate consent or control.
- App that changes system configurations without appropriate consent or control from the user.

Windows Security Center periodically scans to check if antispyware app is enabled to determine the state of antispyware app protection.

Antispyware apps report status updates to Windows Security Center by using the `IWscASStatus` API. Windows Security Center uses these status updates to determine whether the antispyware app is protecting the PC or is out-of-date.

25 APPENDIX E: APP REMEDIATION CODE SAMPLES

It's important to note here that the code examples provided are simply examples. These are not intended to be compiled and used in an app. For the purposes of clarity and readability, these samples do not included standard coding practices such as error handling and memory management.

25.1 DETERMINE THE WINDOWS STORE APP DIRECTORY & DETERMINE AFFECTED APP

```
IPackageManager *pkgMgr;
ActivateInstance(RuntimeClass_Windows_Management_Deployment_PackageManager
, &pkgMgr);
Collections::IIterable<Package*> *pkgList;
pkgMgr->FindPackages(&pkgList);
Collections::IIterator<Windows::ApplicationModel::Package*> *pkgIter;
pkgList->First(&pkgIter);
boolean hasCurrent;
pkgIter->get_HasCurrent(&hasCurrent);
while (hasCurrent)
                Windows::ApplicationModel::IPackage *pkgInfo;
                pkgIter->get_Current(&pkgInfo);

                Windows::Internal::String strPath;
                Windows::Storage::IStorageFolder *pkgFolder;
Windows::Storage::IStorageItem *pkgFolderAsItem
pkgInfo->get_InstalledLocation(&pkgFolder);
pkgFolder->QueryInterface(__uuidof(Windows::Storage::IStorageItem),
&pkgFolderAsItem);
pkgFolderAsItem->get_Path(&strPath);

if (MalwareIsInThisFolder(strPath))
                Windows::Internal::String strMoniker;
                IPackageId *pkgId;
                pkgInfo->get_Id(&pkgId);
                pkgId->get_FullName(&strMoniker);

                ReportThisPackageAsMalware(strMoniker);

pkgIter->MoveNext(&hasCurrent);
```

25.2 DETERMINE IF THE WINDOWS STORE APP IS AFFECTED

```
bool IsMalwarePackage(String pathToPackage)
If (IsMalware(pathToPackage + "AppxBlockmap.xml") ||
IsMalware(pathToPackage + "AppxSignature.p7x"))
                return true;
IStream *blockmapStream;
SHCreateStreamOnFileEx(pathToPackage + "AppxBlockmap.xml",
```

```
STGM_READ | STGM_SHARE_DENY_WRITE | STGM_FAILIFTHERE,
FILE_ATTRIBUTE_NORMAL, FALSE,  NULL,
&blockmapStream);
IAppxFactory *appxFac;
CoCreateInstance(__uuidof(AppxFactory), NULL, CLSCTX_INPROC_SERVER,
__uuidof(IAppxFactory), &appxFac);
IAppxBlockMapReader *blockmapReader;
appxFac->CreateValidatedBlockMapReader(
                blockmapStream,
                pathToPackage + "AppxSignature.p7x",
                &blockmapReader);
IAppxBlockMapFilesEnumerator *fileIter;
blockmapReader->GetFiles(&fileIter);
boolean hasCurrent;
fileIter->GetHasCurrent(&hasCurrent);
while (hasCurrent)
IAppxBlockMapFile *file;
fileIter->GetCurrent(&file);
LPWSTR filename;
file->GetName(&filename);
if (IsMalware(pathToPackage + filename))
return true;
                                                fileIter-
>MoveNext(&hasCurrent);
                                return false;
```

25.3 CALLING SETPACKAGESTATE() TO DISABLE THE APP AND SET THE CUSTOMER UP FOR REMEDIATION

```
Windows::Foundation::Initialize(RO_INIT_MULTITHREADED);

Windows::Management::Deployment::IPackageManager* packageManager;
Windows::Foundation::ActivateInstance("Windows.Management.Deployment.Packa
geManager", &packageManager);
packageManager-
>SetPackageState("Microsoft.SDKSamples.AssociationLaunching.JS_1.0.0.0_neu
tral__8wekyb3d8bbwe",
Windows::Management::Deployment::PackageState_Tampered);
packageManager->Release();

Windows::Foundation::Uninitialize();
```

26 APPENDIX F: MEASURING AND DEBUGGING WEB PERFORMANCE

26.1 MEASURING AND DEBUGGING WEB PERFORMANCE

The Windows Performance Analysis Toolkit (WPT) is a powerful set of tools from Microsoft that allow profiling of all aspects of a Windows computer by using Event Tracing for Windows (ETW). Using the Windows Performance Tools you can measure not only the overall elapsed time of operations, but also look at the time spent in individual browser and operating system components.

We would strongly recommend that developers building security software take the time to learn the WPT toolset. Internet Explorer versions 8, 9, 10, and 11 provide an extensive collection of ETW events. Using the WTP toolset developers can measuring CPU and GPU activity, identify working set patterns, viewing network activity, and generally understand runtime patterns. The WPT toolset will be vital for anyone optimizing the performance of their software.

26.2 PERFORMANCE METRICS THAT MATTER

Microsoft's goal with Internet Explorer is to provide the world's fastest browser. Web performance is a complex and nuanced problem, so it's important that we focus on the real world web performance metrics that matter to customers. We're focused on the following five metrics:

1. Display Time: Perform user actions faster than any modern browser
2. Elapsed Time: Execute Web site code faster than any modern browser
3. CPU Time: Effectively scale computation better than any modern browser
4. Resource Utilization: Require less overall system resources than any modern browser
5. Power Consumption: Require less power than any modern browser

We believe being the world's fastest browser means being best in breed at all five of these objectives across the real world scenarios that matter to customers.

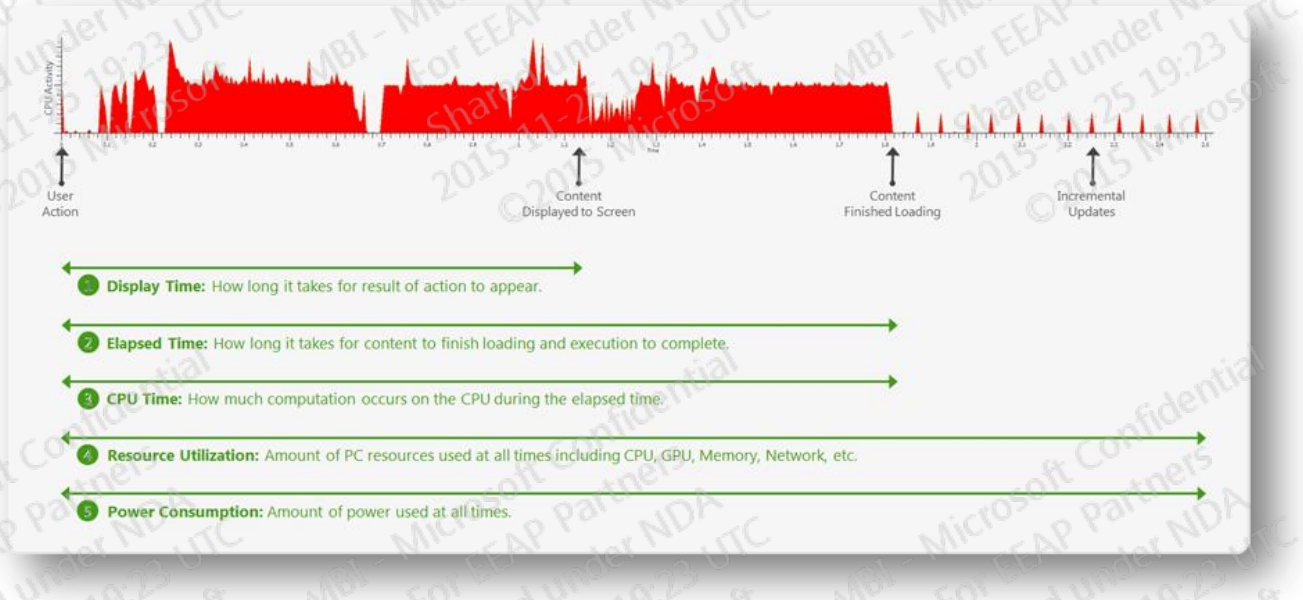


Figure 11: Internet Explorer Performance Objectives

Let's take a closer look at what each of these five objectives means through the scenario of loading a large sports site over a DSL connection for the first time. The diagram in Figure 11 shows what happens on the CPU while the sports site is loading, important points along the way, and how these map to our objectives. We use these same metrics for complex AJAX/Web2.0 JavaScript applications as well.

26.2.1 PERFORMANCE GOAL #1: DISPLAY TIME

The most important objective is what we refer to as Display Time. This has many names across the industry including time to glass and primary paint. Display Time measures the time from when the user performs an action until the user sees the result of that action on the screen. In the case of the sports site, this is the duration from when the user navigates to the site until when the site is visually complete loading. Our objective is to display the results of these user actions faster than any browser, providing customers with real-time responsiveness.

26.2.2 PERFORMANCE GOAL #2: ELAPSED TIME

Most sites continue to perform work in response to the user action after the content has been displayed to the screen. This might include downloading user data (e.g., email messaging) or sending analytics back to a provider. From the users' perspective the site might appear loaded; however, significant work is often occurring in the background which can impact responsiveness. Our objective is to execute the Web site code (HTML, CSS, and script) more efficiently than any browser—making sites load faster, Web experiences more responsive, and enabling Web developers to create richer experiences.

26.2.3 PERFORMANCE GOAL #3: CPU TIME

Web browsers are almost exclusively limited by the CPU. What work a browser performs on the CPU and how efficiently that work occurs will make the single largest impact to performance. That's one of the reasons

offloading work to the GPU has made such a significant impact to IE's performance. The amount of CPU time required to perform the action and the CPU efficiency are critical. Our objective is to more effectively use the CPU and leverage multiprocessor architectures better than any browser.

26.2.4 PERFORMANCE GOAL #4: RESOURCE UTILIZATION

Building a fast browser means ensuring resources across the entire PC work well together. This includes network utilization, memory usage patterns, GPU processing, graphics, memory, and hundreds of other dimensions. Since customers run several applications at the same time on their PC, it's important for browsers to responsibly share these resources with other applications. Our objective is to effectively use system resources like the GPU while requiring less system resources (including working set, CPU load, and GPU load) than any modern HTML5 browser.

26.2.5 PERFORMANCE GOAL #5: POWER CONSUMPTION

When utilizing the underlying PC hardware, it's important to take power consumption into consideration. The more efficiently a browser uses power, the longer the battery life in mobile scenarios, the lower the electricity costs for operating the device, and the smaller the environmental impact. Power and performance are complimentary goals and our objective is to require less power than any other browser without compromising performance.

26.3 PATTERNS USED BY SECURITY SOFTWARE

Most security products use some of unsupported and undocumented techniques to hook into Internet Explorer in order to secure the user. These techniques have a significant impact on web browsing performance, reliability, and system battery life. These techniques should be avoided.

These techniques can be categorized into the following four patterns: network and disk interference, JavaScript interference, visual interference, and general interference.

26.3.1.1 NETWORK AND DISK INTERFERENCE FROM SECURITY SOFTWARE

Modern browsers are designed to issue network requests as quickly as possible, read data from the network in real time, and parallelize network activity. Many security products intercept and monitor network traffic - whether at the TCP, WinInet, or Internet Explorer level. This has a direct impact on how quickly the browser can load.

26.3.1.2 OUTBOUND NETWORK TRAFFIC DELAYS

The Internet Explorer HTML parser is designed to sprint through an HTML document, identify resources that need to be downloaded, and issue the network requests for these resources as quickly as possible. Within the first few milliseconds Internet Explorer will have issued all network requests associated with a website. Monitoring outbound network requests may directly impact performance and should be minimized. This is especially true for security products which spend time doing an action on the outbound URI (comparing against a list, recording the request, etc.).

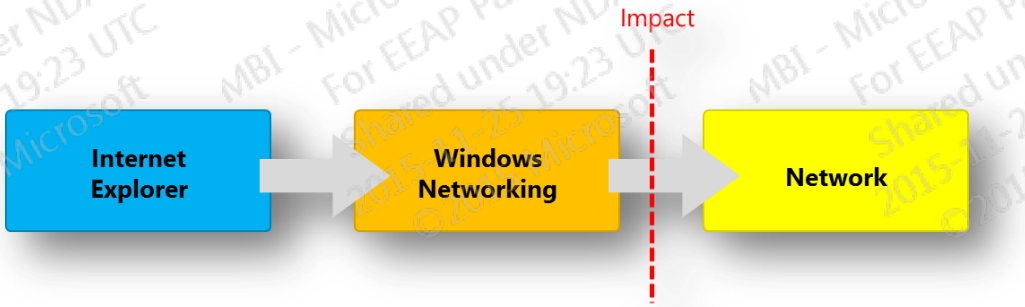


Figure 12: Security products can delay how quickly IE can issue the network request

26.3.2 DISK ACCESS INTERFERENCE

Files downloaded from the web may be cached locally for future efficiencies. On average about ~40% of the resources requested can be pulled from this cache which avoids unnecessary network activity. Many security products have general purpose software that tracks disk reads and writes. This process can lead to performance issues at the point in time the file is read, and more importantly it can have a cascading effect on the website load because both network and local files flood the browser at the same time. There is the false perception that "caches" in the security software eliminate this problem - that has not proven to be the case in real world scenarios.

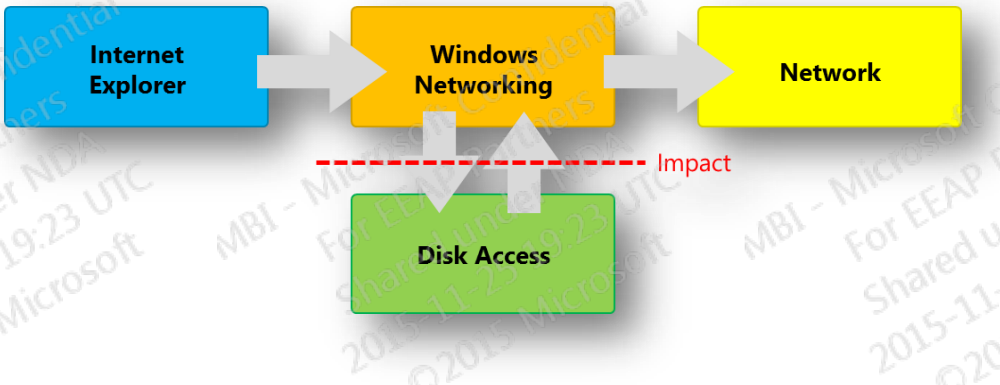


Figure 13: Security products can reduce the benefit of caches

26.3.3 REDUCED PARALLELISM

The HTML5 web platform is inherently single threaded. There are only a handful of activities that can be parallelized across CPU cores today; however they're important CPU intensive activities including image decoding and JavaScript compilation. These two activities occur in parallel during early stages of loading the website when more CPU time is available. When security software delays these activities from occurring, for example if the security software were to scan the JavaScript as it's loaded from disk, this parallelism is often lost and the

computation needs to occur on the primary thread. It's important that security software not impact the parallelism of the web platform.

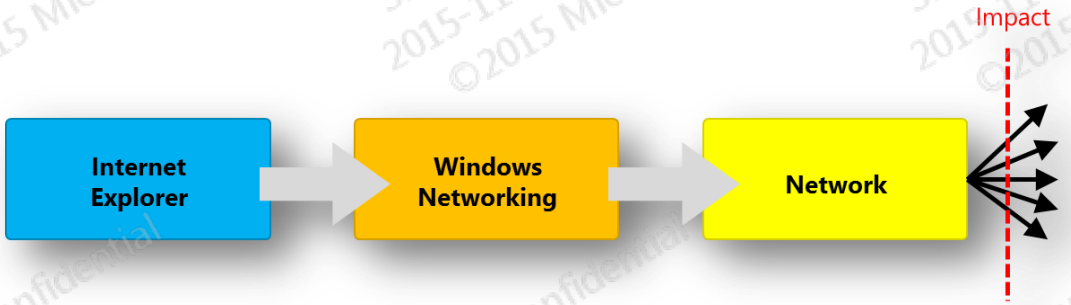


Figure 14: Security products can delay activities reducing parallelism in critical paths

26.4 JAVASCRIPT INTERFERENCE FROM SECURITY SOFTWARE

Internet Explorer contains a new JavaScript engine, codenamed Chakra, which compiles JavaScript into native machine code and supports modern HTML5 and ECMAScript 5 standards. Over the last three years JavaScript engines have improved performance by 100x in many scenarios and are approaching the performance of traditional native compilers. Many security products attempt to integrate with Chakra which has a direct impact on performance.

26.4.1 REVERTING TO LEGACY JAVASCRIPT ENGINE

For compatibility reasons Internet Explorer 9, 10, and 11 continue to support the legacy JScript interpreter which was designed during the Internet Explorer 6 timeframe and for web patterns common at the turn of the century. This interpreter is only used in Internet Explorer compatibility modes and only when required to maintain compatibility with earlier versions. Some security software products force Internet Explorer to always use the legacy interpreter. This has significant performance implications and also impacts functionality since the legacy interpreter does not support HTML5/ES5.

Software products should not call the **IActiveScript** interface, http://msdn.microsoft.com/en-us/library/ky29ffxd(v=VS.94).aspx. This interface forces Internet Explorer to use the legacy JavaScript interpreter.

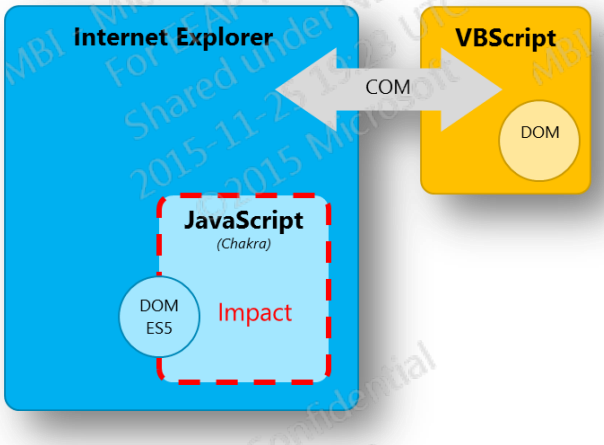


Figure 15: Security products can force IE to always use the legacy interpreter

26.4.2 REDIRECTING JAVASCRIPT CALLS

Versions of Internet Explorer prior to Internet Explorer 9 used COM to communicate between the Internet Explorer browser and a shared JavaScript engine that was used across the Windows operating system. The amount of time spent in COM marshaling was extremely expensive and impacted real word performance. The new Chakra JavaScript engine is natively integrated into Internet Explorer and no longer uses COM. Some security software products inject into Internet Explorer using techniques such as API detouring, ClassID hijacking, or DLL replacement. When this occurs hundreds of thousands of calls that would previously be intra-process communication become out of process communication which comes with a very high CPU cost.

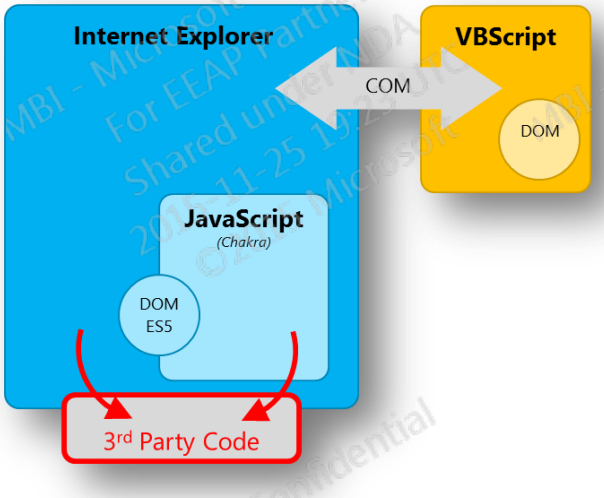


Figure 16: Security products can cause hundreds of thousands of intra-process calls to become out of process calls

26.4.3 PARTICIPATING IN DYNAMIC JAVASCRIPT EXECUTION

The JavaScript language is extremely flexibly which makes static code analysis challenging. Some security products watch how the JavaScript type system and code evolve over time. For example products often attempt to monitor the JavaScript eval function and closely watch XML HTTP Request (XHR) activity. In many cases security products that attempt to participate in script execution are causing the dual execution Chakra JavaScript engine to use the new Chakra interpreter rather than executing native machine code.

26.5 VISUAL INTERFERENCE FROM SECURITY SOFTWARE

The Internet Explorer browser provides many supported mechanisms, including toolbars and browser helper objects (BHO's), which allow the ecosystem to surface features through the user interface. Many security products use these supported API's to provide additional information to the user. When these API's are used inefficiently they will degrade the overall performance of the browser.

26.5.1 REDUNDANT LAYOUT WORK

Modern websites are complex applications that happen to be written in HTML, CSS and JavaScript and execute in the browser. Many security products manipulate websites. For example, a security product may provide URL reputation services and display badges next to every link on the site to indicate whether the URL is safe or unsafe. When a change occurs to the sites object model (DOM) the browser needs to layout the page. Popular websites go through the layout process 3-5 times while being loaded. Most security products that make changes to the website do so incrementally. This forces the browser to layout the page dozens if not hundreds of additional times and can result extensive delays and duplicative work. Changing the visual content of a website is one of the most expensive operations security software can do and should be avoided.

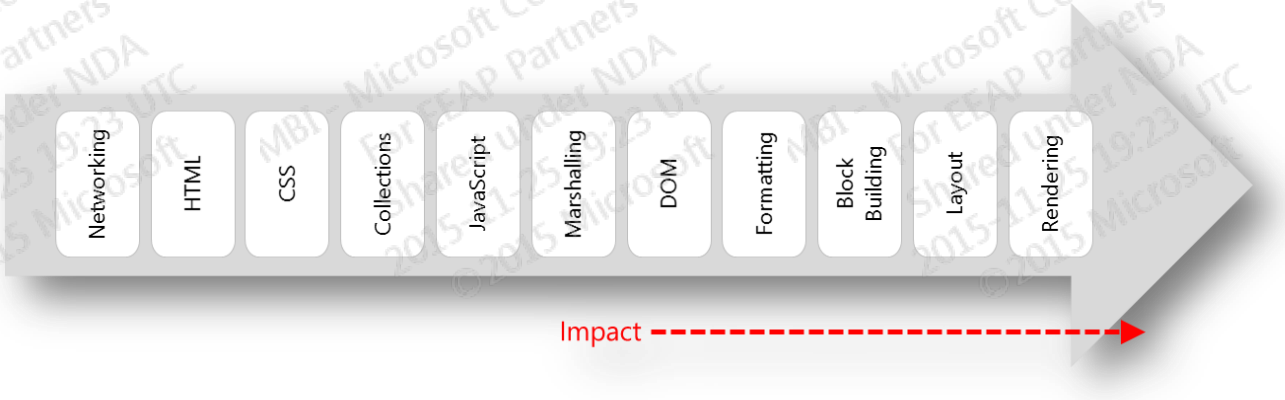


Figure 17: Security products can force the browser to go through the layout process many times

A common example of this type of visual interference is when security products insert badges next to links to indicate whether they are safe to visit.

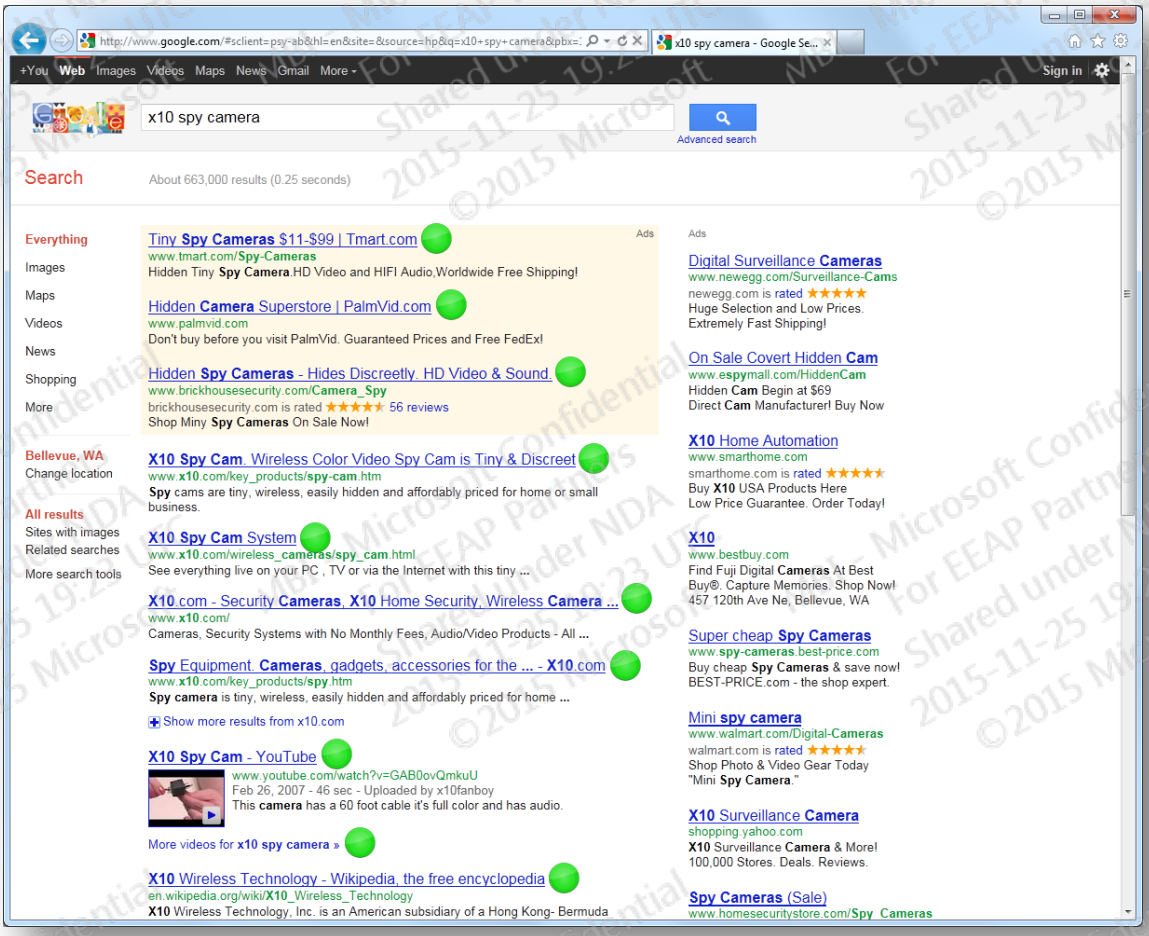


Figure 18: Inserting badges can force the browser to go through layout process multiple times

26.5.2 COMPETING WITH WEBSITE CODE

Over the last decade websites have moved from static content constructed on the server to dynamic personalized experiences constructed on the client. When changes occur to modern websites events fly through the system and most websites have code or frameworks that respond to these events. When security products make changes to the website code-for example adding badges, disabling links, or adding JavaScript code–this will cause modern websites to respond and perform more work than they otherwise would.

26.6 GENERAL INTERFERENCE FROM SECURITY SOFTWARE

In addition to the web platform specific considerations above there are a few general performance bottlenecks often seen when profiling security software in the context of Internet Explorer. These best practices apply to all Windows applications and are important to track. At the end of the day performance is about good engineering!

26.6.1 SYNCHRONOUS SERVER COMMUNICATION

The web evolves rapidly and many security products communicate with the cloud to download information. It is not uncommon to see this communicate occur synchronously. In some cases this communication may synchronously block webpages from loading while the security product is attempting to make a decision about how to proceed. Synchronous communication with the cloud will directly impact performance and should be avoided.

26.6.2 ALTERING INTERNET EXPLORER ASSUMPTIONS AND RUNTIME PATTERNS

Internet Explorer is composed of eleven subsystems highly optimized to efficiently work together on the Windows platform. Many security products alter the runtime patterns of either Windows or the browser. For example, security software may adjust thread priorities or memory locations. These changes may defeat HTML5 performance optimizations around how the browser subsystems work together. Altering the underlying assumptions of Internet Explorer will directly impact performance and should be minimized.

26.7 RESOURCES

Internet Explorer Resources

- Internet Explorer Blog: http://blogs.msdn.com/b/ie

Inside Internet Explorer Performance & Architecture: http://channel9.msdn.com/Events/PDC/PDC10/CD09

Windows Performance Measurement and Debugging

- Windows Performance Analysis Toolkit: http://msdn.microsoft.com/en-us/performance/cc709422
- Measuring Browser Performance with the Windows Performance Tools: http://blogs.msdn.com/b/ie/archive/2010/06/21/measuring-browser-performance-with-the-windows-performance-tools.aspx

Windows 8 and Windows 8.1 Performance Assessment:

- http://www.windows8update.com/2011/09/14/build-conference-session-video-windows-8-introduction-to-the-boot-performance-assessment-3/

Internet Explorer Performance Lab: reliably measuring browser performance

- http://blogs.msdn.com/b/b8/archive/2012/02/16/internet-explorer-performance-lab-reliably-measuring-browser-performance.aspx

Web Performance Best Practices

- BUILD 2012: Performance Tricks to Make Apps and Sites Faster: http://channel9.msdn.com/Events/Build/2012/3-132
- BUILD 2011: 50 Performance Tricks for Windows Store Apps: http://channel9.msdn.com/events/BUILD/BUILD2011/PLAT-386T

27 APPENDIX G: HOW TO USE THE IEXTENSIONVALIDATION INTERFACE

27.1 REGISTRATION

To register to use this interface, security software need to first register a DLL component as an in-process server that implements the IExtensionValidation interface. To register to use this interface, you will need to write the following registry key as a part of your application installation:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Internet Explorer\Extension Validation\{CLSID}
```

Under this component, you will need to define an inProcServer32 registry key. Here the {CLSID} is the CLSID for your security product found under `HKEY_CLASSES_ROOT\CLSID\`.

As a part of your uninstallation process, you will need to make sure that you remove this registry key.

27.2 IEXTENSIONVALIDATION INTERFACE

Internet Explorer calls the IExtensionValidation interface with context on the ActiveX control and web page content to check with the registered security product whether it is safe or not to load an ActiveX control. Internet Explorer will only ask one security product to make this determination.

For Internet Explorer to call a security product using the IExtensionValidation interface, the security product must implement all IExtensionValidation methods. Otherwise, Internet Explorer will not call the security product.

The IExtensionValidation interface reference can also found on MSDN at http://msdn.microsoft.com/en-us/library/dn301826(v=vs.85).aspx.

27.2.1 MEMBERS

The IExtensionValidation interface inherits from the IUnknown interface. IExtensionValidation also has these types of members: structures and methods.

27.2.2 STRUCTURES

Enumeration	Description
ExtensionValidationResults	This structure contains the valid output values that security products can use when making a determination whether an ActiveX control should be loaded.
ExtensionValidationContexts	This structure contains the valid contexts in which the ActiveX control is created.

27.2.2.1 EXTENSIONVALIDATIONRESULTS

This structure contains the valid output values that security products must use when making a determination whether or not it is safe to load an ActiveX control. Syntax and members are listed in this section.

```
typedef enum ExtensionValidationResults
{
    ExtensionValidationResultNone = 0x00,
    ExtensionValidationResultDoNotInstantiate = 0x01,
    ExtensionValidationResultArrestPageLoad = 0x02,
} ExtensionValidationResults;
```

`ExtensionValidationResultNone`

Security products should return this result if the security product determines that it is safe to load the ActiveX control.

`ExtensionValidationResultDoNotInstantiate`

Security products should return this result if the security product determines that it is unsafe to load the ActiveX control, but the web page is otherwise safe.

`ExtensionValidationResultArrestPageLoad`

Security products should return this result if the security product determines that the web page is entirely unsafe.

27.2.2.2 EXTENSIONVALIDATIONCONTEXTS

This structure contains the information that Internet Explorer will share with security products on the context in which the ActiveX control is created. Security products may use this information to help make a determination on whether an Active control is safe to load. Syntax and members are listed in this section.

```
typedef enum ExtensionValidationContexts
{
   ExtensionValidationContextNone = 0x00,
   ExtensionValidationContextDynamic = 0x01,
   ExtensionValidationContextParsed = 0x02,
} ExtensionValidationContexts;
```

27.2.2.3 MEMBERS

`ExtensionValidationContextNone`

This value should never be provided.

`ExtensionValidationContextDynamic`

This value is provided when JavaScript or VBScript is attempting to create the ActiveX control.

`ExtensionValidationContextParsed`

This value is provided when the control is specified in the web page markup. This includes the case where script updates the web page to include an ActiveX control in the markup.

27.2.3 METHODS

Method	Description
DisplayName	Internet Explorer will call this method to determine the name of the security software to display on the blocking dialog. Security products will need to implement this method, otherwise, the IExtensionValidation::Validate method will not be called for the security product.
Validate	Internet Explorer will call this method prior to instantiating an ActiveX control. Internet Explorer will call this method individually for every ActiveX control on a web page. Security products will need to implement this method to let Internet Explorer know whether to load the control, block the control, or block the entire page.

27.2.3.1 IEXTENSIONVALIDATION::DISPLAYNAME

The IExtensionValidation::DisplayName method must be implemented by security products to return the full security product name. Internet Explorer will use this method to display the security product name in the blocking dialog. If this method is not implemented, Internet Explorer will not call the IExtensionValidation::Validate method for the security product.

Syntax:

```
HRESULT DisplayName(
  [out, string] LPWSTR* displayName
);
```

Parameters:

```
displayName
```

The LPWSTR string that contains the name of the security product. The string must be non-zero length and less than 256 characters long. The string should contain the full product name and major version. For example, a Contoso security product may return the string “Contoso Internet Security 2013”.

This returns one of the following values.

Return code	Description
S_OK	A string containing the full product name is given.
S_FALSE	A string containing the full product name has not been given. The IExtensionValidation::Validate method will not be called for this security product.

27.2.3.2 IEXTENSIONVALIDATION::VALIDATE

Internet Explorer will call the IExtensionValidation::Validate method for the registered security product every time it is about to instantiate an ActiveX control on a web page. Security products will need to implement this method to make the determination whether it is safe or not for Internet Explorer to load an ActiveX control.

Syntax:

```
HRESULT Validate(
    [in] REFGUID extensionGuid,
    [in, string] LPWSTR extensionModulePath,
    [in] DWORD extensionFileVersionMS,
    [in] DWORD extensionFileVersionLS,
    [in] IHTMLDocument2* htmlDocumentTop,
    [in] IHTMLDocument2* htmlDocumentSubframe,
    [in] IHTMLElement* htmlElement,
    [in] ExtensionValidationContexts contexts,
    [out] ExtensionValidationResults* results
);
```

Parameters:

`extensionGuid`

The REFGUID contains the GUID of the ActiveX control. This parameter is never null.

`extensionModulePath`

The LPWSTR string that contains the path to the DLL that implements the CoCreate of the ActiveX control. This parameter is never null.

`extensionFileVersionMS`

The DWORD that contains the most significant integer of the ActiveX control version. This parameter is never null.

`extensionFileVersionLS`

The DWORD that contains the least significant integer of the ActiveX control version. This parameter is never null.

`htmlDocumentTop`

The IHTMLDocument2 pointer to the top level document of the web page. This parameter is never null.

`htmlDocumentSubframe`

The IHTMLDocument2 pointer to the subdocument which contains the ActiveX control markup or the context in which the script that creates the ActiveX control exists. This parameter can be null if the ActiveX control doesn't exist in a subdocument.

`htmlElement`

The IHTMLElement pointer to the HTML element that contains the ActiveX control markup. This parameter is null if script creates the ActiveX control.

`contexts`

The ExtensionValidationContexts structure indicating whether the ActiveX control is present in the markup or has been created directly in script.

```
results
```

The ExtensionValidationResults structure that contains the security products security determination on whether to load the ActiveX control, block the ActiveX control, or block the entire page.

This returns one of the following values.

Return code	Description
S_OK	The security product is able to make a security determination.
S_FALSE	The security product failed to make a security determination.

In the following example, a user visits a web page that loads a Silverlight control in the top level document through script. In this example, Internet Explorer will provide the following parameter values in its IExtensionValidation::Validate method call.

```
> extensionGuid
{dfeaf541-f3e1-4c24-acac-99c30715084a}

> extensionModulePath
c:\Program Files (x86)\Microsoft Silverlight\5.1.20125.0\npctrl.dll

> extensionFileVersionMS
0x50001

> extensionFileVersionLS
0x4e9d0000

> htmlDocumentTop
0x14e0eca0

> htmlDocumentSubframe
0x00000000

> htmlElement
0x00000000

> contexts
ExtensionValidationContextDynamic (0n1)
```

28 APPENDIX H: CREATE AN AUTOMATIC MAINTENANCE TASK

The following section details how developers can create a task using a task definition in XML or C language. Task developers should keep in mind that their maintenance activity should not launch any user-interface that requires interaction with the user as Automatic Maintenance is completely silent and runs when the user is not present.

28.1 USING XML

Task Scheduler has a built-in command-line tool *schtasks.exe* that can be used to import a task definition in XML format. The schema for task definition is documented at http://msdn.microsoft.com/en-us/library/aa383609(v=VS.85).aspx.

The following is an example of an automatic maintenance task defined in XML.

```
<Task xmlns="http://schemas.microsoft.com/windows/2004/02/mit/task">
      <RegistrationInfo>

<SecurityDescriptor>D:(A;OICI;FA;;;BA)(A;OICI;FA;;;SY)(A;OICI;GR;;;AU)(A;;
FRFX;;;LS)</SecurityDescriptor>
        <URI>\Microsoft\Windows\Power Efficiency
Diagnostics\AnalyzeSystem</URI>
        <Source>$(@%systemRoot%\system32\energytask.dll,-601)</Source>
        <Author>$(@%systemRoot%\system32\energytask.dll,-600)</Author>
        <Description>$(@%systemRoot%\system32\energytask.dll,-
602)</Description>
        <Version>1.0</Version>
      </RegistrationInfo>
      <Settings>
        <Enabled>true</Enabled>
        <AllowStartOnDemand>true</AllowStartOnDemand>
        <MultipleInstancesPolicy>IgnoreNew</MultipleInstancesPolicy>
        <DisallowStartIfOnBatteries>false</DisallowStartIfOnBatteries>
        <StopIfGoingOnBatteries>false</StopIfGoingOnBatteries>
        <StartWhenAvailable>true</StartWhenAvailable>
        <Hidden>false</Hidden>
        <WakeToRun>false</WakeToRun>
        <RunOnlyIfIdle>false</RunOnlyIfIdle>
        <UseUnifiedSchedulingEngine>true</UseUnifiedSchedulingEngine>
        <ExecutionTimeLimit>PT5M</ExecutionTimeLimit>
        <MaintenanceSettings>
          <Period>P7D</Period>
          <Deadline>P10D</Deadline>
```

```
      </MaintenanceSettings>
    </Settings>
    <Principals>
      <Principal id="LocalSystemAccount">
        <UserId>S-1-5-18</UserId>
      </Principal>
    </Principals>
    <Actions Context="LocalSystemAccount">
      <ComHandler>
        <ClassId>{927ea2af-1c54-43d5-825e-0074ce028eee}</ClassId>
      </ComHandler>
    </Actions>
</Task>
```

To create the task on a Windows PC, save the above xml as a text file and use the following command-line:

```
Schtasks.exe /create /tn [task name] /xml [text file name]
```

28.1.1 USING C LANGUAGE

An automatic maintenance task can also be created using C code.

The following is sample code that can be used to configure a task's automatic maintenance settings:

```
#
#define _WIN32_DCOM

#include <windows.h>
#include <iostream>
#include <stdio.h>
#include <comdef.h>
#include <wincred.h>
//  Include the task header file.
#include <taskschd.h>
# pragma comment(lib, "taskschd.lib")
# pragma comment(lib, "comsupp.lib")
# pragma comment(lib, "credui.lib")

using namespace std;


int __cdecl wmain()
```

```
{

    //  ------------------------------------------------------
    //  Initialize COM.
    HRESULT hr = CoInitializeEx(NULL, COINIT_MULTITHREADED);

    if( FAILED(hr) )
    {
        printf("\nCoInitializeEx failed: %x", hr );
        return 1;
    }

    //  Set general COM security levels.
    hr = CoInitializeSecurity(
        NULL,
        -1,
        NULL,
        NULL,
        RPC_C_AUTHN_LEVEL_PKT_PRIVACY,
        RPC_C_IMP_LEVEL_IMPERSONATE,
        NULL,
        0,
        NULL);

    if( FAILED(hr) )
    {
        printf("\nCoInitializeSecurity failed: %x", hr );
        CoUninitialize();
        return 1;
    }


    ITaskService *pService = NULL;
    ITaskFolder *pRootFolder = NULL;
    ITaskDefinition *pTask = NULL;
    IRegistrationInfo *pRegInfo= NULL;
    IPrincipal *pPrincipal = NULL;
    ITaskSettings *pSettings = NULL;
    ITaskSettings3 *pSettings3 = NULL;
    IMaintenanceSettings *pMaintenanceSettings = NULL;
    IActionCollection *pActionCollection = NULL;
    IAction *pAction = NULL;
    IExecAction *pExecAction = NULL;
    IRegisteredTask *pRegisteredTask = NULL;

    //  ------------------------------------------------------
    //  Create a name for the task.
    LPCWSTR wszTaskName = L"Time TriggerMaintenance Test Task";
```

```
    //  Get the windows directory and set the path to notepad.exe.
    wstring wstrExecutablePath = _wgetenv( L"WINDIR");
    wstrExecutablePath += L"\\SYSTEM32\\NOTEPAD.EXE";


    //  ------------------------------------------------------
    //  Create an instance of the Task Service.
    hr = CoCreateInstance( CLSID_TaskScheduler,
                           NULL,
                           CLSCTX_INPROC_SERVER,
                           IID_ITaskService,
                           (void**)&pService );
    if (FAILED(hr))
    {
        printf("Failed to create an instance of ITaskService: %x", hr);
        goto CleanUp;
    }

    //  Connect to the task service.
    hr = pService->Connect(_variant_t(), _variant_t(),
        _variant_t(), _variant_t());
    if( FAILED(hr) )
    {
        printf("ITaskService::Connect failed: %x", hr );
        goto CleanUp;
    }

    //  ------------------------------------------------------
    //  Get the pointer to the root task folder.  This folder will hold
the
    //  new task that is registered.
    hr = pService->GetFolder( _bstr_t( L"\\") , &pRootFolder );
    if( FAILED(hr) )
    {
        printf("Cannot get Root folder pointer: %x", hr );
        goto CleanUp;
    }

    //  If the same task exists, remove it.
    pRootFolder->DeleteTask( _bstr_t( wszTaskName), 0  );

    //  Create the task definition object to create the task.
    hr = pService->NewTask( 0, &pTask );

    if (FAILED(hr))
    {
        printf("Failed to CoCreate an instance of the TaskService class:
%x", hr);
        goto CleanUp;
```

```
    }

    //  ---------------------------------------------------
    //  Get the registration info for setting the identification.
    hr = pTask->get_RegistrationInfo( &pRegInfo );
    if( FAILED(hr) )
    {
        printf("\nCannot get identification pointer: %x", hr );
        goto CleanUp;
    }

    hr = pRegInfo->put_Author( L"Author Name" );
    if( FAILED(hr) )
    {
        printf("\nCannot put identification info: %x", hr );
        goto CleanUp;
    }

    //  ---------------------------------------------------
    //  Create the principal for the task - these credentials
    //  are overwritten with the credentials passed to
RegisterTaskDefinition
    hr = pTask->get_Principal( &pPrincipal );
    if( FAILED(hr) )
    {
        printf("\nCannot get principal pointer: %x", hr );
        goto CleanUp;
    }

    //  Set up principal logon type to interactive logon
    hr = pPrincipal->put_LogonType( TASK_LOGON_INTERACTIVE_TOKEN );
    if( FAILED(hr) )
    {
        printf("\nCannot put principal info: %x", hr );
        goto CleanUp;
    }

    //  ---------------------------------------------------
    //  Create the settings for the task
    hr = pTask->get_Settings( &pSettings );
    if( FAILED(hr) )
    {
        printf("\nCannot get settings pointer: %x", hr );
        goto CleanUp;
    }


  hr = pSettings->QueryInterface(__uuidof(ITaskSettings3),
(void**)&pSettings3 );
```

```
    if( FAILED(hr) )
    {
        printf("\nCannot put idle setting information: %x", hr );
        goto CleanUp;
    }

    hr = pSettings3->CreateMaintenanceSettings( &pMaintenanceSettings );
    if( FAILED(hr) )
    {
        wprintf(L"\nCannot CreateMaintenanceSettings: %x", hr );
        goto CleanUp;
    }

    hr = pMaintenanceSettings->put_Period ( _bstr_t(L"P7D") );
    if( FAILED(hr) )
    {
        wprintf(L"\nCannot put_Period: %x", hr );
        goto CleanUp;
    }

    hr = pMaintenanceSettings->put_Deadline ( _bstr_t(L"P10D") );
    if( FAILED(hr) )
    {
        wprintf(L"\nCannot put_Period: %x", hr );
        goto CleanUp;
    }


    //  ------------------------------------------------------
    //  Add an action to the task. This task will execute notepad.exe.

    //  Get the task action collection pointer.
    hr = pTask->get_Actions( &pActionCollection );
    if( FAILED(hr) )
    {
        printf("\nCannot get Task collection pointer: %x", hr );
        goto CleanUp;
    }

    //  Create the action, specifying that it is an executable action.
    hr = pActionCollection->Create( TASK_ACTION_EXEC, &pAction );
    pActionCollection->Release();
    if( FAILED(hr) )
    {
        printf("\nCannot create the action: %x", hr );
        goto CleanUp;
    }

    //  QI for the executable task pointer.
```

```
    hr = pAction->QueryInterface(
        IID_IExecAction, (void**) &pExecAction );
    pAction->Release();
    if( FAILED(hr) )
    {
        printf("\nQueryInterface call failed for IExecAction: %x", hr );
        goto CleanUp;
    }

    //  Set the path of the executable to notepad.exe.
    hr = pExecAction->put_Path( _bstr_t( wstrExecutablePath.c_str() ) );
    pExecAction->Release();
    if( FAILED(hr) )
    {
        printf("\nCannot put action path: %x", hr );
        goto CleanUp;
    }

    //  ------------------------------------------------------
    //  Save the task in the root folder.
    hr = pRootFolder->RegisterTaskDefinition(
            _bstr_t( wszTaskName ),
            pTask,
            TASK_CREATE_OR_UPDATE,
            _variant_t(),
            _variant_t(),
            TASK_LOGON_INTERACTIVE_TOKEN,
            _variant_t(L""),
            &pRegisteredTask);
    if( FAILED(hr) )
    {
        printf("\nError saving the Task : %x", hr );
        goto CleanUp;
    }

    printf("\n Success! Task successfully registered. " );

CleanUp:

    //  Clean up.
    if ( pRegisteredTask ) pRegisteredTask->Release();
    if ( pExecAction ) pExecAction->Release();
    if ( pAction ) pAction->Release();
    if ( pActionCollection ) pActionCollection->Release();
    if ( pMaintenanceSettings) pMaintenanceSettings->Release();
    if ( pSettings3 ) pSettings3->Release();
    if ( pSettings ) pSettings->Release();
    if ( pRootFolder ) pRootFolder->Release();
    if ( pTask ) pTask->Release();
```

```
    if ( pRegisteredTask ) pRegisteredTask->Release();

    CoUninitialize();
    return 0;
}
```

28.2 VALIDATING TASKS

Task author should validate that the task has been created and runs successfully as part of maintenance.

28.2.1 VALIDATING TASK CREATION

Export the task definition to a file, using the following command-line and ensure that the task definition is as expected.

```
Schtasks.exe /Query /tn [task name] /xml [text file name]
```

28.2.2 VALIDATING TASK EXECUTION

Launch the task by running the following command-line and validate that Task Scheduler UI (taskschd.msc) shows the task as run.

```
SCHTASKS /Run /tn "\Microsoft\Windows\TaskScheduler\Manual Maintenance"
```

29 APPENDIX I: HOW TO TEST AND VALIDATE PERFORMANCE REQUIREMENTS FOR ANTIMALWARE APPS

29.1 OVERVIEW

The high level process for testing and validating that a third party antimalware app meets the Performance Requirements for Antimalware Apps is:

1. Select and prepare a Device Under Test (DUT) with Windows 10 following the machine prep guidelines
2. Measure and define the performance baseline on the DUT, by executing the designated assessments with Windows Defender enabled.
3. Install and prepare the antimalware app to be validated.
4. Measure the performance of the antimalware app on the DUT by executing the designated assessments with the antimalware app enabled
5. Review and analyze results against the performance criteria, which is generally a %tolerance of a specific metric as measured against the baseline.

29.2 HARDWARE AND SOFTWARE CONFIGURATION

The Device Under Test (DUT) selected should be representative of a shipping device that includes a similar hardware and software configuration as an OEM would ship for Windows 10.

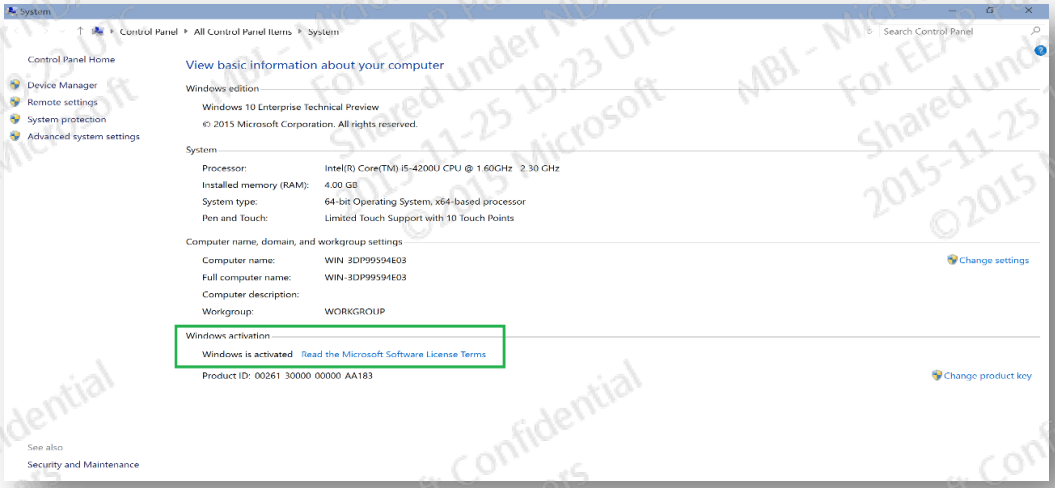
29.3 TEST ENVIRONMENT

Component	Setting	Notes
Display setting	Brightness = optional	
Power policy	Balanced Display dimming timeout disabled Display off timeout disabled Adaptive brightness disabled S3 timeout disabled S4 timeout disabled	
Radios	Wi-Fi on **and** connected All other radios (e.g., Bluetooth) ON but not connected	Connect to a consumer class wireless router that has Internet access
Power source	AC	Only use the AC adapter provided by the OEM
Other networking	Ethernet disconnected	No Ethernet cable connected to the RJ-45 port
OS Build	Windows 10 Client	
Domain	Not joined	
Drivers	Signed drivers that an OEM ships with the device	
Network sharing	Disabled	

External devices/accessories	Not connected unless shipped/bundled with the device (e.g., keyboard dock)	

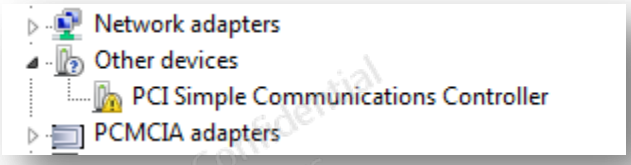
29.4 MACHINE PREP

1. If the device has already gone through the First Boot Experience, recover the device to its original factory image.
2. Once the original factory image is restored, complete the First Boot Experience.
3. During the First Boot Experience, set up a local admin account without a password instead of a Microsoft Account.
4. Allow First Boot Experience to complete.
5. After completing First Boot Experience, verify that the device is activated. OS activation requires a valid OS product key that is either injected in the firmware or manually entered and internet connectivity. Refer below to confirm OS activation.



Example 1 - Windows Activation

6. Perform the tasks described below.
    a. Run Windows Update and allow it to install all software components identified.
    b. Use Device Manager to verify that the DUT has no problems with drivers or devices.
        i. Investigate and fix all devices marked with a "!" as shown below.



Example 2 - Driver not Found

c. Download and install the latest virus definition files and perform a scan using the default antimalware app (Windows Defender or third party).
d. Run Maintenance from Control Panel -> All Control Panel Items → Security and Maintenance.
e. Enable autologon.
    i. `HKLM\Software\Microsoft\Windows NT\CurrentVersion\WinLogon`.
f. The following values should be configured.

Value name	Value type	Data
DefaultUserName	REG_SZ	*<TestUserName>*
DefaultPassword	REG_SZ	*<LOCALPASSWORD>*
AutoAdminLogon	REG_SZ	1

g. If the system has access to a timer server, make sure the clock has been synchronized with the server time. If the clock time changes during assessment execution, it might cause failures.
h. Run the machine prep script. This can take a long time to run. The script will reboot the system at the end.

7. Prior to running the video playback battery rundown, performed the additional tasks described below.
    a. Fully charge the battery.

29.5 VALIDATION

Validating compliance with the performance is done using publicly available tools (Windows Assessment and Deployment Kit) or manually. Refer below for validation steps. For the requirements that use the Windows Assessment and Deployment kit, download the packaged assessments first.

29.5.1 TOTAL BOOT TIME (EXCLUDING BIOS) AND SHUTDOWN TIME (FAST STARTUP)

This metric captures the time from the start of the shutdown phase, to the end of writing the hiberfile to disk and the transition to a lower power state (S4).

29.5.1.1 HOW TO RUN THE TEST

1. If the packaged assessment is not already on the DUT's local disk, copy the packaged assessment to the DUT's local disk.
2. Run the scripts per the instructions that were included with the packaged assessments[23].

29.5.2 INTERNET EXPLORER STARTUP PERFORMANCE

[23] The Packaged Assessments for Windows 10 Antimalware Performance are not available with the first draft of this document and will be made available at a later date.

The Internet Explorer® Startup Performance assessment measures the time to fully render a new Internet Explorer window on the desktop, with a single tab and simple content. This measurement includes the load time of the IExplore.exe process and the frame-creation and tab-creation intervals. The assessment also measures the performance of extensions that are loaded and initialized during startup. It doesn't measure network or browsing performance.

29.5.2.1 HOW TO RUN THE TEST

1. If the packaged assessment is not already on the DUT's local disk, copy the packaged assessment to the DUT's local disk.
2. Run the scripts per the instructions that were included with the packaged assessments.[24]

29.5.3 INTERNET EXPLORER SECURITY SOFTWARE IMPACT

The Internet Explorer security software impact assessment measures aspects of Internet Explorer that are typically impacted by antimalware and other browser add-ins. The assessment measures the impact of security software on the display time, CPU time, and resource utilization of Internet Explorer.

29.5.3.1 HOW TO RUN THE TEST

1. If the packaged assessment is not already on the DUT's local disk, copy the packaged assessment to the DUT's local disk.
2. Run the scripts per the instructions that were included with the packaged assessments.

29.5.4 WINDOWS STORE APP PERFORMANCE

The Windows Store app performance assessment can help you optimize your app for a better customer experience. The assessment measures how quickly the app opens and suspends, and the amount of resources it uses on the PC. You can use this assessment to help you improve an individual app, or to help you optimize a Windows image by picking fast and fluid apps that run well on your PC.

29.5.4.1 HOW TO RUN THE TEST

1. If the packaged assessment is not already on the DUT's local disk, copy the packaged assessment to the DUT's local disk.
2. Run the scripts per the instructions that were included with the packaged assessments.

29.5.4.2 HOW TO LOAD RESULTS WITH THE WINDOWS ASSESSMENT CONSOLE

The Windows Assessment Console (WAC.exe) is a GUI tool that simplifies analyzing and comparing sets of perf data, collected by running ADK perf assessments. You can use it to load and compare two sets of results.xml files. Results.xml files are the results output of the performance assessments. To evaluate an antimalware apps

performance against the Windows Defender baseline, you need to load and analyze both sets of results using the WAC as described following.

1. Launch the Windows Assessment Console (WAC.exe) which is located in the "Assessment and Deployment Kit\Windows Assessment Toolkit\amd64" folder, below the install location of your ADK for Windows 10.

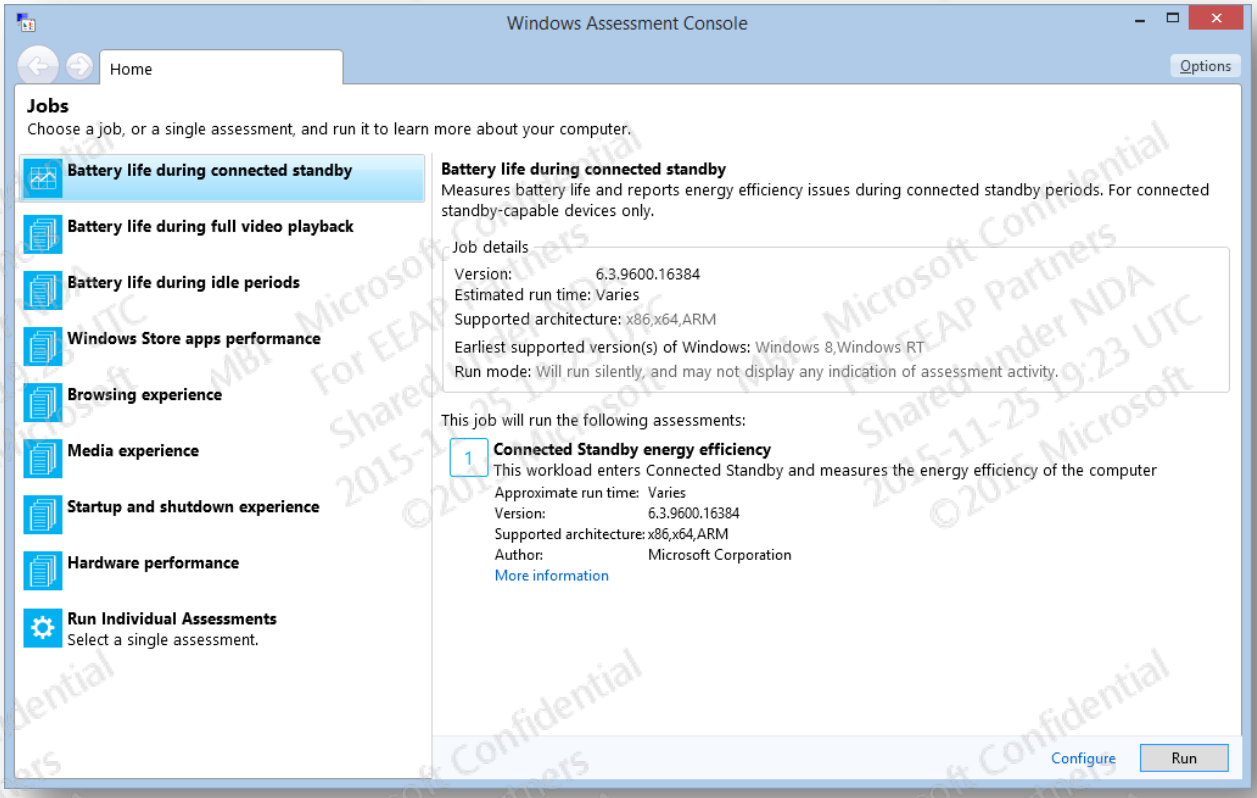


Figure 19: WAC on Launch

2. Load your first set of results data Options > Open Results…

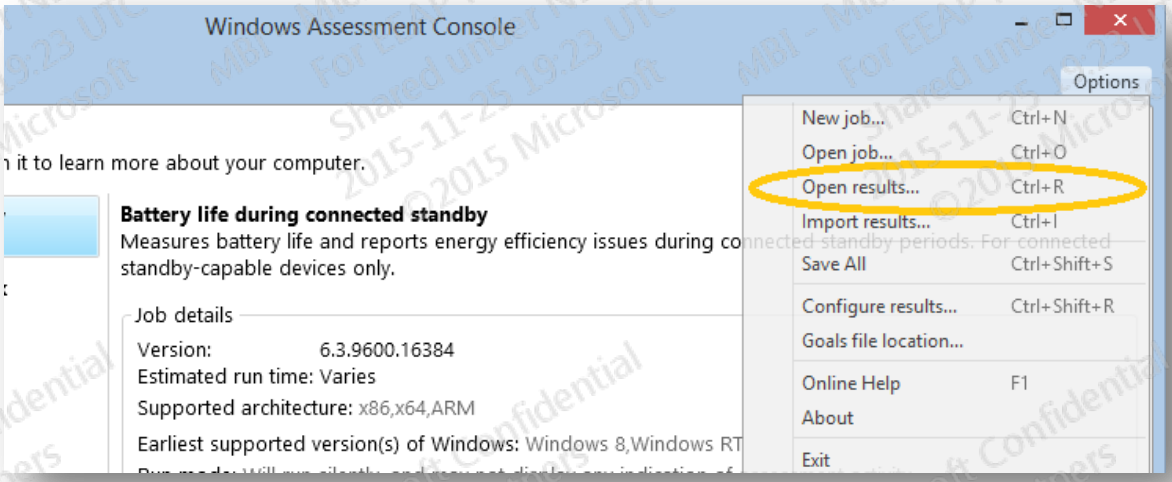


Figure 20: Opening Results using WAC

3. Browse to and select your first results.xml file.
4. After loading WAC will switch to the results view. You can load your second set of results by pressing the "Add results" button, and loading your second results.xml by following the file dialogs as per the first ime.

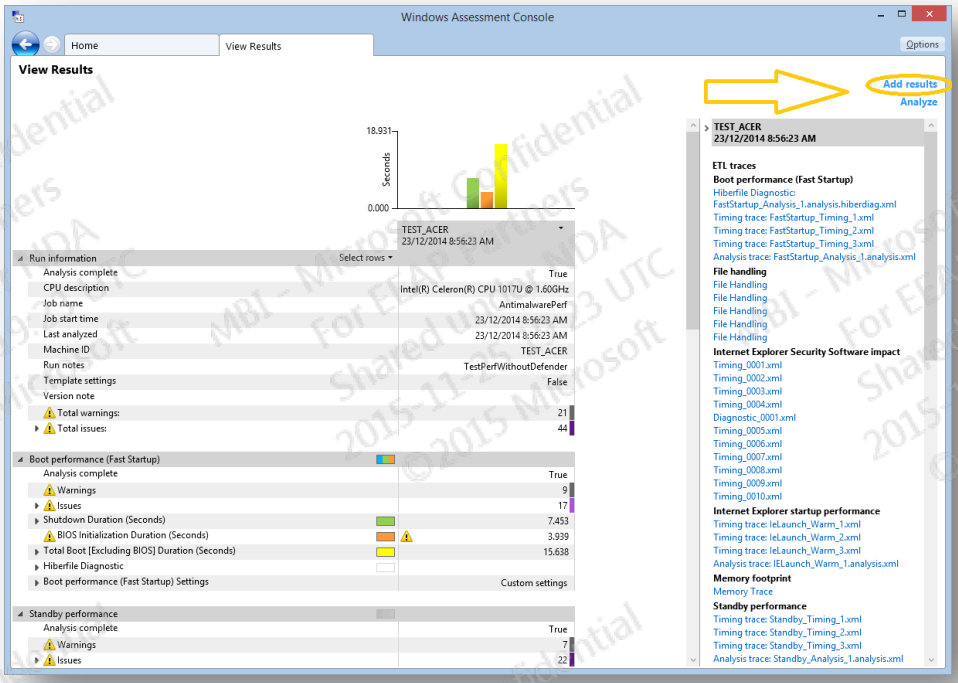


Figure 21: WAC with one set of results loaded, adding the second for comparison

5. Once the second set of results has been loaded you will see them side by side in WAC as shown:

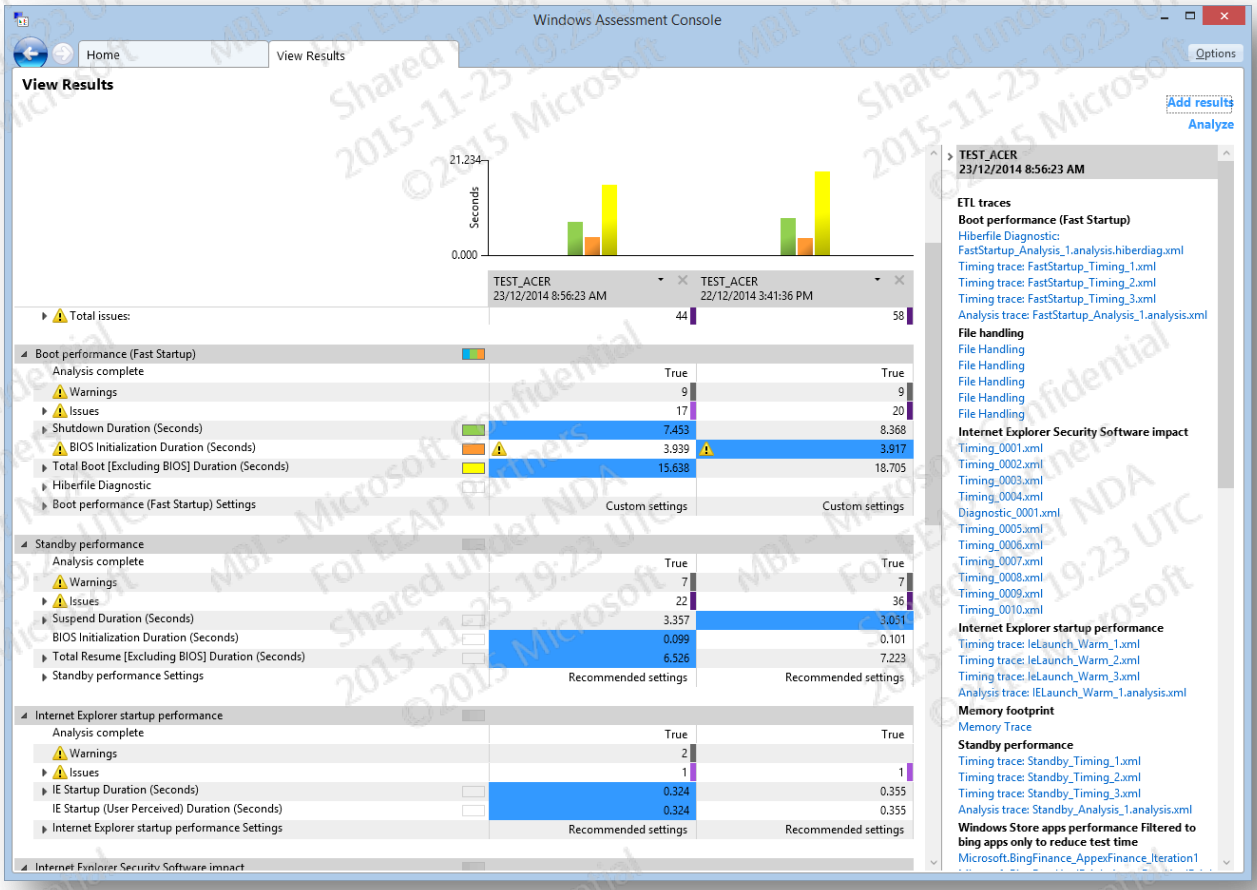
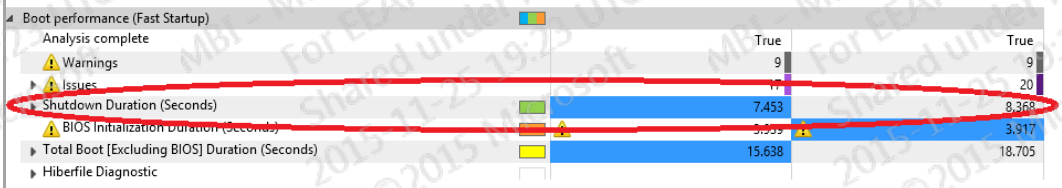


Figure 22: WAC with two sets of results loaded for comparison

6. The relevant results and metrics which map to the Windows 10 Antimalware Performance requirements are shown following:

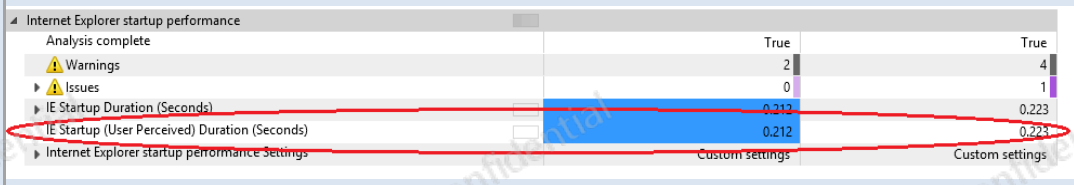
Requirement	Assessment / Metric in WAC
Total boot time	Boot performance (Fast Startup) — Analysis complete: True / True; Warnings: 9 / 9; Issues: 17 / 20; Shutdown Duration (Seconds): 7.453 / 8.368; BIOS Initialization Duration (Seconds): 3.917; Total Boot [Excluding BIOS] Duration (Seconds): 15.638 / 18.705; Hiberfile Diagnostic

Shutdown time



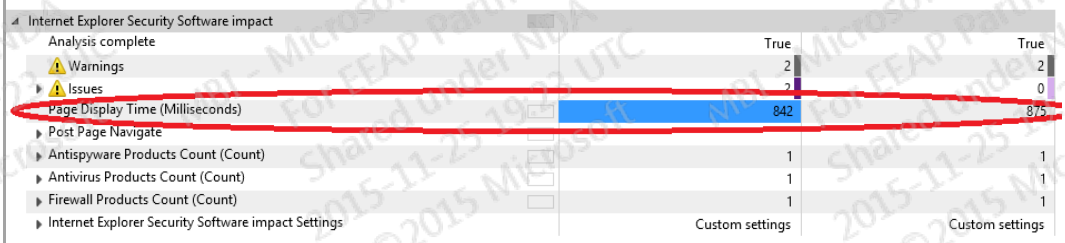
Boot performance (Fast Startup)		
Analysis complete	True	True
Warnings	9	9
Issues	17	20
Shutdown Duration (Seconds)	7.453	8.368
BIOS Initialization Duration (Seconds)	3.939	3.917
Total Boot [Excluding BIOS] Duration (Seconds)	15.638	18.705
Hiberfile Diagnostic		

Internet Explorer Startup Performance



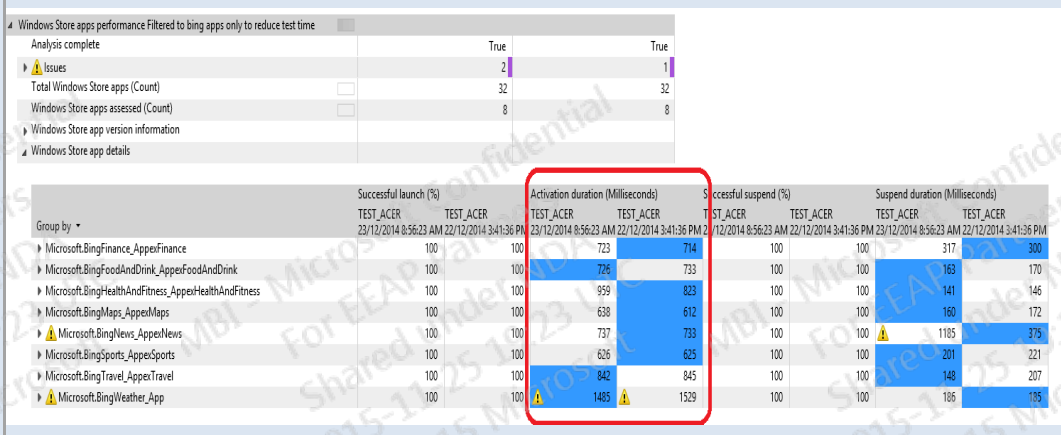
Internet Explorer startup performance		
Analysis complete	True	True
Warnings	2	4
Issues	0	1
IE Startup Duration (Seconds)	0.212	0.223
IE Startup (User Perceived) Duration (Seconds)	0.212	0.223
Internet Explorer startup performance Settings	Custom settings	Custom settings

Internet Explorer Security Software Impact



Internet Explorer Security Software impact		
Analysis complete	True	True
Warnings	2	2
Issues	2	0
Page Display Time (Milliseconds)	842	875
Post Page Navigate		
Antispyware Products Count (Count)	1	1
Antivirus Products Count (Count)	1	1
Firewall Products Count (Count)	1	1
Internet Explorer Security Software impact Settings	Custom settings	Custom settings

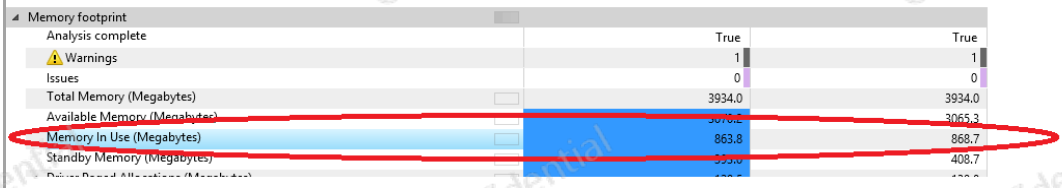
Windows Store App Performance



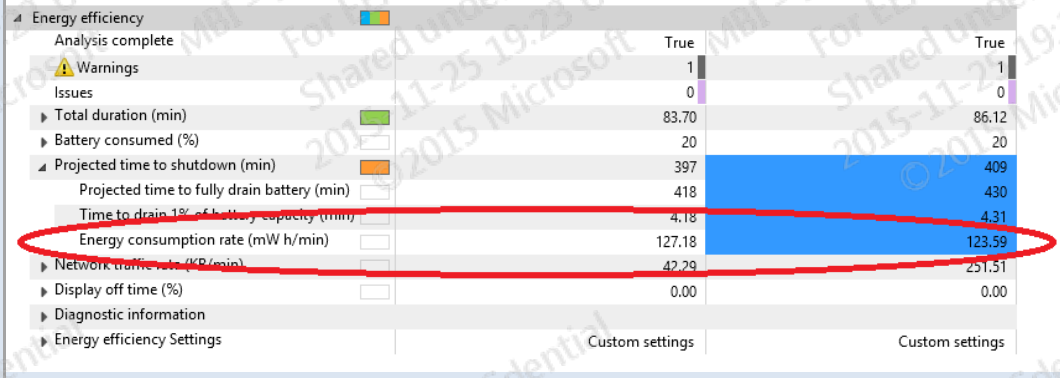
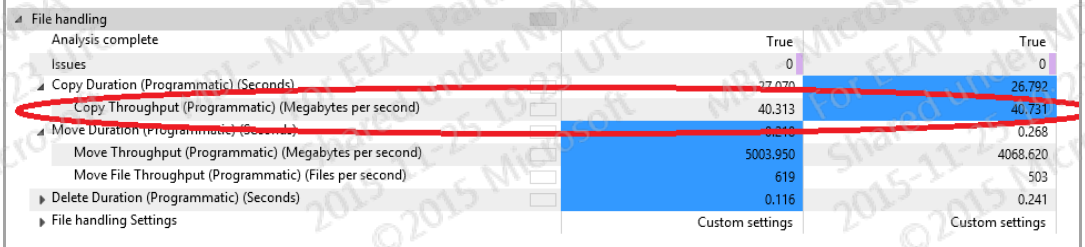
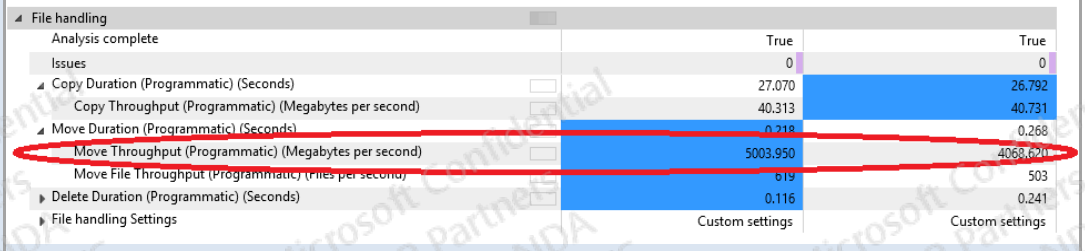
Windows Store apps performance Filtered to bing apps only to reduce test time		
Analysis complete	True	True
Issues	2	1
Total Windows Store apps (Count)	32	32
Windows Store apps assessed (Count)	8	8
Windows Store app version information		
Windows Store app details		

Group by	Successful launch (%)		Activation duration (Milliseconds)		Successful suspend (%)		Suspend duration (Milliseconds)	
	TEST_ACER 23/12/2014 8:56:23 AM	TEST_ACER 22/12/2014 3:41:36 PM	TEST_ACER 23/12/2014 8:56:23 AM	TEST_ACER 22/12/2014 3:41:36 PM	TEST_ACER 2/12/2014 8:56:23 AM	TEST_ACER 22/12/2014 3:41:36 PM	TEST_ACER 23/12/2014 8:56:23 AM	TEST_ACER 22/12/2014 3:41:36 PM
Microsoft.BingFinance_AppexFinance	100	100	723	714	100	100	317	300
Microsoft.BingFoodAndDrink_AppexFoodAndDrink	100	100	726	733	100	100	163	170
Microsoft.BingHealthAndFitness_AppexHealthAndFitness	100	100	959	823	100	100	141	146
Microsoft.BingMaps_AppexMaps	100	100	638	612	100	100	160	172
Microsoft.BingNews_AppexNews	100	100	737	733	100	100	1185	375
Microsoft.BingSports_AppexSports	100	100	626	625	100	100	201	221
Microsoft.BingTravel_AppexTravel	100	100	842	845	100	100	148	207
Microsoft.BingWeather_App	100	100	1485	1529	100	100	186	185

Memory footprint



Memory footprint		
Analysis complete	True	True
Warnings	1	1
Issues	0	0
Total Memory (Megabytes)	3934.0	3934.0
Available Memory (Megabytes)	3070.2	3065.3
Memory In Use (Megabytes)	863.8	868.7
Standby Memory (Megabytes)	533.0	408.7

Video playback battery rundown	
File handling copy	
File handling move	
OOBE Duration	This assessment will be updated in future.

29.6 RESOURCES

- Windows Assessment and Deployment Kit
- The 'packaged assessments' referred to are included as a resource accompanying this document.[25]

[25] The Packaged Assessments for Windows 10 Antimalware Performance are not available with the first draft of this document and will be made available at a later date.

30 APPENDIX J: RECOMMENDED PERFORMANCE GUIDELINES FOR ANTIMALWARE APPS

30.1 MAINTAINING CACHES FOR PREVIOUSLY SCANNED FILES

Antimalware apps should maintain a non-volatile cache which is retained across reboots. This cache should not be affected by signature updates. It should only contain files that have reputation data (e.g., signed by a trusted authority). If the files in the cache are modified they should be invalidated.

Antimalware apps should maintain a volatile cache which tracks files scanned in the current boot session. This cache should be valid for the current boot session. If the files in the cache are not modified they should not be scanned again until next reboot. Portions of the cache can be invalidated when signatures are updated. If the files in the cache are modified they should be invalidated.

30.2 OPTIMIZED SIGNATURE MAINTENANCE AND LOADING

Loading static signatures **on demand** during boot can significantly reduce overall memory and disk read footprint. Our guidance is to maintain a **data structure in memory to represent static signatures** and only demand-load specific portions of the signature file for which there is a match in the data structure. Note that signature cache files for hot signatures that are required by most scans can still be loaded and kept in memory. The recommendation is to use Xpress compression APIs to achieve an improved compression ratio and better decompression time for signature cache files.

30.3 USING LOW CPU, DISK AND MEMORY PRIORITIES

All activity that is not blocking explicit application requests should use **low CPU, memory and disk priorities** so these do not interfere with other foreground activity. Use **low memory priority for all real-time scans**, so that portions of the file that are not used will be put on the standby list at low page priority, thus not repurposing more valuable pages later.

Memory priority for a thread/process determines how long its pages will remain in memory, since un-used pages in the page cache are ordered such that those pages with the lowest priority leave memory first. Note that only files being scanned should be accessed at low memory priority, not using private data structures (since repurposing these can cause a significant amount of hard faults in a resume from hibernate scenario).

Every process and thread in Windows has a memory priority assigned to it. The process memory priority determines the default memory priority of new threads. The thread memory priority determines the minimum priority of the physical pages that are added to the process working set by that thread. Unused pages in the page cache are ordered such that those pages with the lowest priority leave memory first. As a result, memory priority assigned to a thread/process determines how long its pages are going to remain in memory.

30.3.1 SETPRIORITYCLASS AND SETTHREADPRIORITY

- Use `SetPriorityClass` with background mode (`PROCESS_MODE_BACKGROUND_BEGIN` and `PROCESS_MODE_BACKGROUND_END`) to reduce CPU, memory and I/O priorities for a specific process.

- Use `SetThreadPriority` to ensure that only specific threads will run at low priority.
- Note that these functions will set **ALL** resource priorities (CPU, Disk and Memory) on a process or thread.

Additional MSDN documentation is available for SetPriorityClass and SetThreadPriority.

30.3.2 SETTHREADINFORMATION AND SETPROCESSINFORMATION

There were APIs introduced in Windows 8 to set **only memory priority** to low, and in some cases, app developers may only want to lower memory priority, since lowering the I/O priority may slow down an app's operations to an unacceptable level.

Additional MSDN documentation is available for SetThreadInformation and SetProcessInformation.

30.4 IMPROVING IO PATTERNS

In cases where a thread is sequentially accessing more than a certain number of files, antimalware app developers can ensure that future scans of files opened by that thread will first read the file in its entirety before proceeding to do the scan. To prevent additional disk seeks caused by scanning different parts of a file during sequential file operations (like file copy), use the new PreFetch API in Windows 8 to perform 1MB read sizes when performing sequential access on large files.

30.4.1 PREFETCH API

The sequential access detection mechanism in memory manager ramps up read sizes in powers of two up to 1 MB so it takes 6 extra IOs per file to get up to 1MB read size. The PreFetch API introduced in Windows 8 will allow antimalware app developers to call prefetch on the file and get 1MB read sizes without the extra IOs.

Additional MSDN documentation is available for PrefetchVirtualMemory.

30.4.2 AVOID INTERFERING WITH THE PREFETCHER IO

Prefetcher is critical for overall Windows performance. It is tightly integrated with the cache manager and memory manager to make disk accesses more efficient, thus improving performance. Prefetching should be used for important scenarios, such as boot and application launch. This improves performance by changing IO patterns from random to semi-sequential. Prefetched data is placed in the system page cache. No user application can access prefetched data until it passes the necessary security checks to open, read, map or execute the file, just as it cannot access other data in the system page cache. Interfering with the prefetcher can have a negative impact on performance.

30.4.2.1 PROBLEMATIC BEHAVIOR

Here's a description of the impact that intercepting the Prefetcher I/Os could have on performance.

1. The Prefetcher needs to open all files first and issue all I/Os at the same time to be effective.
2. The Antimalware filter drivers scan files at create time by issuing synchronous I/Os to them, which forces the Prefetcher to block on random I/Os across the disk.

3. This prevents the Prefetcher from building a deep queue and taking advantage of semi-sequential read throughput.

30.4.2.2 RECOMMENDED BEHAVIOR

It is possible to avoid this problem by identifying and ignoring I/Os initated by the Prefetcher.

1. Antimalware filter driver identifies prefetcher-issued Create requests and does not perform scans at that time.
2. This is secure since prefetched data is in the system page cache and applications have to still open a file handle to access prefetched data, which will trigger scans at that time.
3. Scan times will be faster as fewer disk I/Os are necessary since prefetching has already occurred.

30.4.2.3 IDENTIFY PREFETCHER ACTIVITY AND I/OS

1. All create requests issued by the prefetcher will contain an ECP defined as GUID_ECP_PREFETCH_OPEN.
    a. It is important to check that it was issued from kernel mode.
2. All further prefetcher activity on a given file involves the file object created with the above ECP.
3. If operations other than create need to be identified, a filter should maintain their own state.
    a. Can be accomplished by using Filter Manager's stream handle contexts.

30.5 OTHER DESIGN BEST PRACTICES

The antimalware app user interface should not be launched unless explicitly requested by the user (i.e. no pop-ups immediately after boot or resume). Files and directories accessed by specific inbox processes should be added to the exclusion list for scans (e.g., MUI files). Antimalware apps should not scan files on close, especially during file copy operations. Files will still be scanned synchronously during file open before allowing the open to succeed. Antimalware apps should not register for pre-shutdown notifications unless absolutely necessary and antimalware app developers should ensure that the proper service timeouts are available if the app must register for them. Background scans should use buffered IO (to avoid too many disk reads) at low memory priority.

30.6 THE DAM KERNEL-MODE DRIVER

DAM is a kernel-mode driver that is loaded and initialized at system boot if the system supports Connected Standby. This is determined by evaluating if the AOAC field in the `SYSTEM_POWER_CAPABILITIES` structure returned by `CallNtPowerInformation` is set to `TRUE`.

When the DAM is enabled and the desktop app process is created, the DAM adds the app process to a system-managed job object:

- If the process was created in session 0, DAM adds the process to a job object, subject to **throttling**.
- If the process was created in an interactive session (session 1 or higher), DAM adds the process to a job object, subject to **suspension**.

For Windows 10, job objects can be nested. This means that the DAM's usage of job objects does not interfere with an app's existing usage of job objects.

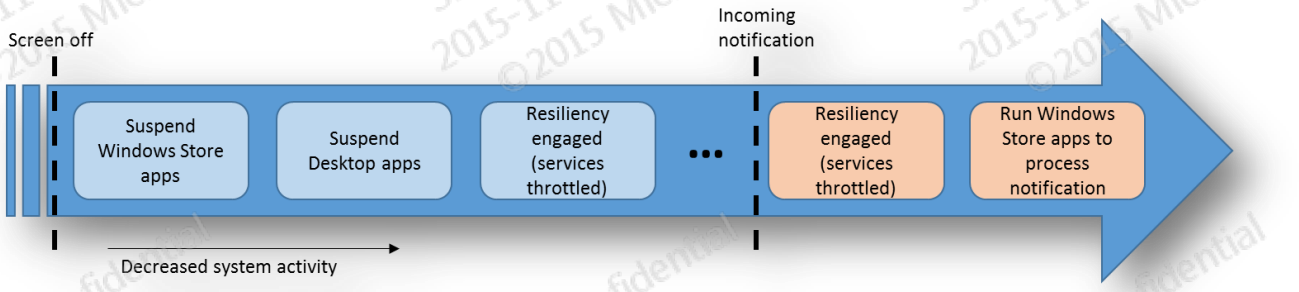


Figure 23: Overview of Connected Standby transition

When the screen is on, the DAM is disengaged and does not impact any processes on the system. The DAM will be engaged after all Windows Store apps and desktop apps are suspended. Also, all inbox services have had a chance to listen for Connected Standby entry notification. Similarly, on an incoming notification for a Windows Store app (e.g. a VoIP call) DAM disengages service throttling before indicating the packet to the Windows Store app. Once the push notification is processed the system reengages DAM throttling.

When the system is in Connected Standby, depending on activity on the system, DAM might throttle or suspend processes:

- Processes that are subject to suspension have all their threads suspended (not allowed to run under any circumstances); app state (process memory) is maintained,
- Processes that are subject to throttling cycle between suspended and unsuspended (a large majority of time is spent in the suspended state),
    - Windows might also detect that critical activities are occurring and might un-suspend throttled services for longer periods of time during this activity.
    - While in Connected Standby, sensors and networks might not be available, so throttled processes should be designed to be resilient to poor network conditions (for most processes, this doesn't require any change).

When DAM suspension is engaged or disengaged, DAM triggers delivery of a `WM_POWERBROADCAST` message to those processes subject to suspension that have opted-in to message delivery (via an API call or compatibility shim). After a few seconds delay, DAM suspends the process. There are no user notifications provided when DAM throttling is engaged or disengaged. Processes should not need modification; processes continue to function, albeit at a slower rate.

30.6.1 MANIFESTATION

Processes are often suspended or throttled during the Connected Standby state. To the majority of suspended apps, this should be very similar to an S3 suspend / resume or S4 hibernate / resume transition. Manifestations

might include but are not limited to inconsistencies in uptime / runtime vs. wall clock time, inconsistencies in timer behavior, or dramatic changes in operating system state before and after the suspend completes.

Suspension and throttling happens as a unit (all processes are suspended and unsuspended in unison, and all processes that can be throttled are throttled and un-throttled in unison), so communication between two suspended processes or two throttled processes does not pose a problem.

Apps that rely on cross-process communication in the following manner might need special consideration:

- **Communication between processes in session 0 (throttled) and session 1+ (suspended)** – examples include tray icons or UI components that represent current service state, e.g., antimalware or Firewall.
- **Communication between components in user mode (session 0 or 1) and drivers (which are neither throttled nor suspended)** – examples include services that do work on behalf of a driver. In these cases, if cross-process communication is not handled correctly, apps can appear hung or unresponsive (though the user might not see this impact directly, as the screen will be off when in Connected Standby). In most cases, however, services and drivers should already be developed to be robust against cross-process communication issues.

Antimalware app developers who create apps for, or dependent on, the web should consider how process suspension affects connection lifetimes and handshakes. Additionally, because network connectivity might not be available when in Connected Standby mode, developers of processes created in session 0 should be especially aware of how intermittent network connections affect the process.

30.6.2 DAM SOLUTIONS

Windows Store apps are not impacted by the DAM. If a non-Windows Store app is impacted by the DAM, the app can request notifications before suspension is engaged (for example, to save state or close network connections) using one of the following methods:

- If the app has a window (HWND) and notifications will be handled through the window procedure, the antimalware app developer can call `RegisterSuspendResumeNotification` to register for these messages (or `UnregisterSuspendResumeNotification` to unregister). Antimalware app developers can use `DEVICE_NOTIFY_WINDOW_HANDLE` in the Flags parameter, and pass the window's HWND in as the Recipient parameter. The message received is the `WM_POWERBROADCAST` message.
- If the app does not have a window (HWND) or a direct callback is needed, the antimalware app developer can call `PowerRegisterSuspendResumeNotification` to register for these messages (or `PowerUnregisterSuspendResumeNotification` to unregister). Antimalware app developers must use `DEVICE_NOTIFY_CALLBACK` in the Flags parameter and pass a value of type `PDEVICE_NOTIFY_SUBSCRIBE_PARAMETERS` in the Recipient parameter.
- If the app cannot be recompiled, the app can opt in to receive these `WM_POWERBROADCAST` messages via the AppCompat toolkit (using the `PromoteDAM` shim).
- Driver timeouts of less than 5 minutes should be avoided when communicating with user mode components. This will prevent increased IO delays experienced by the driver.

The suspend message is `WM_POWERBROADCAST` with `wParam=PBT_APMSUSPEND` (the same); this message is broadcast concurrently to all opted in processes on the system. The app must perform any work on the suspend path quickly and efficiently. The timeout after the broadcast notification is global, not per process, so the process might be contending for system resources if the work required in this path is extensive.

The resume message is `WM_POWERBROADCAST` with `wParam=PBT_APMRESUME`; this message is broadcast concurrently to all opted in processes after a resume. Relative time of delivery to the system exit from Connected Standby is not guaranteed.

31 APPENDIX K: REMOTE DEVICE HEALTH ASSESSMENT

MSRT periodically communicates machine state to the cloud and this is used to produce device health information. Online services can use this device health information to protect services, assets and users. In Windows 8.1, new APIs have been added to uniquely identify a Windows computer, based on TPM AIK. The following sections document the APIs that can be used to access this information. Here are some of the basic concepts.

31.1 WINDOWS ID API

This enables a way to uniquely identify a Windows PC, even across an OS re-install. This ID is available for Windows 8.1 computers on all SKUs and for desktop apps only.

31.2 DRIVER LOAD NOTIFICATION API

This is used to pre-read secured boot and measured boot in Windows 8.1. This provides an asynchronous call back when a run-time driver is loaded, as well as auditing, telemetry and asynchronous scanning of run-time drivers. This cannot be used to block drivers.

31.3 REMOTE CLOUD ANALYSIS

This is used to remotely determine device health. This analysis uses MSRT thread reports and the Windows Defender heartbeat to produce a 'device claim'. The different types of evaluation available cover the following instances:

- PC had malware that stole user passwords
- PC was infected with malware that was not remediated
- Windows Defender was terminated on the PC
- TCG/TPM logs evaluated
- State of machine security

31.4 API DETAILS

31.4.1 TBSI_SHAHASH

This calculates SHA hash or HMAC using BCrypt.

Syntax:

```
NTSTATUS Tbsi_ShaHash(
    _In_ LPCWSTR pszAlgId,
    _In_reads_opt_(cbKey) PBYTE pbKey,
    _In_ UINT32 cbKey,
    _In_reads_(cbData) PBYTE pbData,
    _In_ UINT32 cbData,
    _Out_writes_to_opt_(cbResult, *pcbResult) PBYTE pbResult,
    _In_ UINT32 cbResult,
    _Out_ PUINT32 pcbResult
);
```

Parameters:

Parameter	Description
PCWSTR pszAlgId	Hash algorithm, must be 'BCRYPT_SHA1_ALGORITHM', 'BCRYPT_SHA256_ALGORITHM', or 'BCRYPT_SHA384_ALGORITHM'
PBYTE pbKey, UINT32 cbKey	Optional HMAC key and length. If present, HMAC will be calculated. Otherwise, hash will be calculated.
PBYTE pbData, UINT32 cbData	Data to hash and length
PBYTE pbResult, UINT32 cbResult	Pointer to result buffer and length of buffer
PUINT32 pcbResult	Returns length of result

Return value: NTSTATUS

Return value	Description
STATUS_SUCCESS	Success
STATUS_INVALID_PARAMETER	Invalid hash algorithm specified
STATUS_BUFFER_TOO_SMALL	Output buffer too small
STATUS_NO_MEMORY	Memory allocation failure

Additional details

Requirements	
Minimum supported client	Windows 8.1 (Desktop apps only)
Minimum supported server	Windows 8.1
Header	Tbs_internal.h
Library	Tbs.lib
DLL	Tbs.dll

31.4.2 TBSI_FILTERLOG

This is a filter event log that contains only entries for PCRs that are specified in PCR mask.

Syntax:

```
NTSTATUS Tbsi_FilterLog(
    _In_reads_(cbEventLog) PBYTE pbEventLog,
    _In_ UINT32 cbEventLog,
    _In_ UINT32 pcrMask,
    _Out_writes_to_opt_(cbOutput, *pcbResult) PBYTE pbOutput,
    _In_ UINT32 cbOutput,
    _Out_ PUINT32 pcbResult
);
```

Parameters:

Parameter	Description
PBYTE pbEventLog, UINT32 cbEventLog	TPM measured boot event log (TCG log) to filter and length of log
UINT32 pcrMask	Mask of PCRs for which to retain events. Lsb represents PCR 0.
PBYTE pbOutput, UINT32 cbOutput	Pointer to result buffer and length of buffer.
PUINT32 pcbResult	Returns length of result.

Return value: NTSTATUS

Return value	Description
STATUS_SUCCESS	Success
STATUS_INVALID_PARAMETER	Parameter error
STATUS_BUFFER_TOO_SMALL	Output buffer too small
STATUS_UNSUCCESSFUL	Internal consistency error

Additional details

Requirements	
Minimum supported client	Windows 8.1 (Desktop apps only)
Minimum supported server	Windows 8.1
Header	Tbs_internal.h
Library	Tbs.lib
DLL	Tbs.dll

31.4.3 GETDEVICEID

This reads the TPM-derived device ID as a sequence of bytes. Optionally, this can also retrieve the TPM backing state flag. If the PC does not have a TPM, a 32 byte device identifier is generated and persisted.

On PCs equipped with TPM 1.2, a 2048 bit RSA key is generated and the 256 bit hash of the public key is used as the device identifier. In case the key is deleted or the TPM is cleared, a new key is generated.

On PCs equipped with TPM 2.0, a 2048 bit RSA key is generated and the 256 bit hash of the public key is used as the device identifier. In case the key is deleted, the same key and device identifier are regenerated. If the TPM is cleared, a new key and device identifier will be generated.

Syntax:

```
HRESULT GetDeviceID(
    _Out_writes_bytes_to_opt_(cbWindowsAIK, *pcbResult) PBYTE pbWindowsAIK,
    _In_ UINT32 cbWindowsAIK,
    _Out_ PUINT32 pcbResult,
    _Out_opt_ BOOL *pfProtectedByTPM
);
```

Parameters:

Parameter	Description
PBYTE pbWindowsAIK, UINT32 cbWindowsAIK	Pointer to result buffer and length of buffer. Call 'GetDeviceID' passing it NULL for 'pbWindowsAIK', 0 for 'cbWindowsAIK', and NULL for 'pfProtectedByTPM', to receive in '*pcbResult' the size in characters of the buffer to allocate to read the DeviceID. Allocate this size buffer, and call 'GetDeviceID' again passing it the buffer pointer for 'pbWindowsAIK' and the buffer size for 'cbWindowsAIK'. The DeviceID will be returned in the buffer.
PUINT32 pcbResult	Returns length of result.
BOOL *pfProtectedByTPM	Optional pointer to a BOOL value to receive the TPM backing state flag

Return value: HRESULT

Return value	Description
S_OK	Success
0x8007007a	Output buffer too small
0x80070002, 0x80070003	Device ID not yet available

Additional details

Requirements	
Minimum supported client	Windows 8.1 (Desktop apps only)
Minimum supported server	Windows 8.1
Header	Tbs_internal.h
Library	Tbs.lib

Requirements	
DLL	Tbs.dll

31.4.4 GETDEVICEIDSTRING

This reads the TPM-derived device ID as a Base64-encoded string. Optionally this can be used to retrieve the TPM backing state flag.

Syntax:

```
HRESULT GetDeviceIDString(
    _Out_writes_to_opt_(cchWindowsAIK, *pcchResult) PWSTR pszWindowsAIK,
    _In_ UINT32 cchWindowsAIK,
    _Out_opt_ PUINT32 pcchResult,
    _Out_opt_ BOOL *pfProtectedByTPM
);
```

Parameters:

Parameter	Description
PBYTE pbWindowsAIK, UINT32 cbWindowsAIK	Pointer to result buffer and length of buffer. Call 'GetDeviceID' passing it NULL for 'pbWindowsAIK', 0 for 'cbWindowsAIK', and NULL for 'pfProtectedByTPM', to receive in '*pcbResult' the size in characters of the buffer to allocate to read the DeviceID. Allocate this size buffer, and call 'GetDeviceID' again passing it the buffer pointer for 'pbWindowsAIK' and the buffer size for 'cbWindowsAIK'. The DeviceID will be returned in the buffer.
PUINT32 pcbResult	Returns length of result.
BOOL *pfProtectedByTPM	Optional pointer to a BOOL value to receive the TPM backing state flag

Return value: HRESULT

Return value	Description
S_OK	Success
0x8007007a	Output buffer too small
0x80070002, 0x80070003	Device ID not yet available

Additional details

Requirements	
Minimum supported client	Windows 8.1 (Desktop apps only)

Requirements	
Minimum supported server	Windows 8.1
Header	Tbs_internal.h
Library	Tbs.lib
DLL	Tbs.dll

32 APPENDIX L: POWERSHELL'S USE OF THE AMSI API

32.1 OVERVIEW

One of the issues in detecting script-based malware based on file monitoring is the ability for malware authors to obfuscate their code through dynamic techniques such as eval() (in VBScript) and Invoke-Expression (in PowerShell).

Additionally, PowerShell supports both interactive use and input automation, creating an avenue for malware that isn't even based on files.

32.2 POWERSHELL AMSI API INVOCATION DETAILS

The current AMSI API takes the following as input:

- amsiContext: The global handle to save AMSI from re-initializing its data structures all the time. This is initialized with:
    - **appName**: The name of the host application performing the scan
    - **content**: The content to scan
    - **contentName**: Source metadata about the content to scan
    - amsiSession: A handle to a session representing a related cluster of AMSI calls

The three in bold have some discretion in how the calling application supplies them.

In PowerShell, here is how they are handled:

- appName: "PowerShell_{0}_{1}"
    - {0}: The full path to the application hosting PowerShell. For example, c:\windows\system32\windowspowershell\v1.0\powershell.exe (for the console host). Or c:\windows\system32\windowspowershell\v1.0\powershell_ise.exe (for the graphical host). Or c:\program files\foo\app.exe (for a custom host).
    - {1}: The ProductVersion taken from the application hosting PowerShell. For example, PowerShell v4 (which does not support this API) would send in:
        - PS> Get-Process -Name PowerShell | % MainModule | % ProductVersion
        - 6.3.9600.17396
- content: The text of the script about to be invoked.
- contentName: The file path that was the source of the content being scanned, if it is available. If the content was entered at the command line, this is the empty string.
- amsiSession: A new session is created for each PowerShell "pipeline" that is run. For example, a script that calls a script that calls a script is still a single pipeline. Or the command "script1.ps1; script2.ps1" is still a single pipeline.

33 APPENDIX M: JSCRIPT/VBSCRIPT'S USE OF THE AMSI API

33.1 OVERVIEW

One of the security risks with running JScript and VBScript through Windows Script Host (WScript and CScript) rises from the obfuscation opportunity afforded by dynamic code execution. In order to improve protection capability for script-based malware, with Windows 10 any dynamic JScript/VBScript code running through Window Script Host can be scanned by AMSI before execution. All scripts rejected by AMSI will not be executed and an error will be returned as a result. The specific places where dynamic code can be run are:

- JScript
    a. Eval
    b. Function constructor
- VBScript
    a. Eval
    b. Execute/ExecuteGlobal

33.2 JSCRIPT/VBSCRIPT AMSI API INVOCATION DETAILS

The current AMSI API takes the following as input:

- amsiContext: The global handle to save AMSI from re-initializing its data structures all the time. This is initialized with:
    - **appName**: The name of the host application performing the scan
    - **content**: The content to scan
    - **contentName**: Source metadata about the content to scan
    - amsiSession: A handle to a session representing a related cluster of AMSI calls

The three in bold have some discretion in how the calling application supplies them.

In PowerShell, here is how they are handled:

- appName: The language name (Jscript or VBScript)
- content: The text of the dynamic code about to be executed (for eval, new Function, etc.)
- contentName: The current scriptFilename (if succeeded querying it from the host, otherwise it is null).

amsiSession: this is not used.

Used APIs:

- AmsiUninitialize
- AmsiInitialize
- AmsiScanString

The amsi library is dynamically loaded by script engine.

34 APPENDIX N: UAC/CONSENT.EXE'S USE OF THE AMSI API AND THE UX FLOW

34.1 OVERVIEW

There are **four** different scenarios where a UAC elevation request would occur resulting in consent.exe being launched. Consent.exe is always in all scenarios including Auto-Elevation, the only exception being if UAC is totally disabled.

The different scenarios are

1. EXE Elevation
2. COM Elevation
3. MSI Elevation
4. ActiveX Installation Elevation

Each of these types have some unique data associated with them, below are the data structures that represent these data which are passed into the antimalware provider.

34.2 METADATA AVAILABLE TO AMSI CONSUMERS

34.2.1 EXE ELEVATION

EXE elevation request context is simple, it contains the **Application Name** and its **Commandline Parameters**. Additionally binaries which operate only on a DLL input from their commandline would have a valid lpwszDLLParameter pointing to the DLL. As of today UAC parses commandline only for **mmc.exe** elevations

```
typedef struct AMSI_UAC_REQUEST_EXE_INFO
{
    ULONG           ulLength;
    [string] LPWSTR  lpwszApplicationName;
    [string] LPWSTR  lpwszCommandLine;

    // Points to an extension dll to be loaded by the EXE (for eg. mmc.exe),
which requires
    // a separate reputation check.
    [string] LPWSTR  lpwszDLLParameter;
} AMSI_UAC_REQUEST_EXE_INFO, *LPAMSI_UAC_REQUEST_EXE_INFO;
```

34.2.2 COM ELEVATION

COM elevation request context contains the **CLSID** being elevated and its corresponding binary which hosts the COM class. Additionally the image path of the process requesting COM elevation is passed in through `lpwszRequestor`.

```
typedef struct AMSI_UAC_REQUEST_COM_INFO
{
    ULONG            ulLength;
    [string] LPWSTR  lpwszServerBinary;
    [string] LPWSTR  lpwszRequestor;
    GUID             Clsid;
} AMSI_UAC_REQUEST_COM_INFO, *LPAMSI_UAC_REQUEST_COM_INFO;
```

34.2.3 MSI ELEVATION

MSI elevation request context is the most complex request of all. The MSI action could be because of Update, Install, Uninstall, etc. as defined in `AMSI_UAC_MSI_ACTION`. It contains basic information like the ProductName, ProductVersion, Language, and ManufactureName.

The MSI file path is passed in through `lpwszPackagePath`. `lpwszPackageSource` points to the path of the original source, through which the MSI was initially installed onto the system.

An MSI update could be by means of MSI-patches, these are passed in through `ppwszUpdates`. `ppwszUpdateSources` points to the original source path if any.

```
typedef [v1_enum] enum AMSI_UAC_MSI_ACTION
{
    // MSI actions that can be elevated.

    AMSI_UAC_MSI_ACTION_INSTALL             = 0,
    AMSI_UAC_MSI_ACTION_UNINSTALL           = 1,
    AMSI_UAC_MSI_ACTION_UPDATE              = 2,
    AMSI_UAC_MSI_ACTION_MAINTENANCE         = 3,
    AMSI_UAC_MSI_ACTION_MAX                 = 4
} AMSI_UAC_MSI_ACTION;
```

```
typedef struct AMSI_UAC_REQUEST_MSI_INFO
{
    ULONG                                   ulLength;
    AMSI_UAC_MSI_ACTION                     MsiAction;

    [string] LPWSTR                         lpwszProductName;
    [string] LPWSTR                         lpwszVersion;
    [string] LPWSTR                         lpwszLanguage;
    [string] LPWSTR                         lpwszManufacturer;
    [string] LPWSTR                         lpwszPackagePath;
    [string] LPWSTR                         lpwszPackageSource;
```

```
    ULONG                                   ulUpdates;
    [string, size_is(ulUpdates, )] LPWSTR*  ppwszUpdates;
    [string, size_is(ulUpdates, )] LPWSTR*  ppwszUpdateSources;
} AMSI_UAC_REQUEST_MSI_INFO, *LPAMSI_UAC_REQUEST_MSI_INFO;
```

34.2.4 ACTIVEX INSTALLATION ELEVATION

ActiveX elevation is triggered via the browser when a website tries to install an ActiveX that requires elevation. The elevation request context is fairly simple consisting of the URL from where it was downloaded and the local file path where the Active has been downloaded to.

```
typedef struct AMSI_UAC_REQUEST_AX_INFO
{
    ULONG            ulLength;
    [string] LPWSTR  lpwszLocalInstallPath;
    [string] LPWSTR  lpwszSourceURL;
} AMSI_UAC_REQUEST_AX_INFO, *LPAMSI_UAC_REQUEST_AX_INFO;
```

In addition to the above, all elevation requests have some common information as defined below.

`bAutoElevateRequest` is TRUE if the elevation request would be auto-elevated without a consent prompt. This would happen if the binary is signed by **"Windows Publisher"** and resides in an OS directory in case of EXE elevations. However in case of COM elevations both the COM Server Binary and the Requester Binary have to be signed by the **"Windows Publisher"** and reside in an OS directory. The other two elevation types do not support AutoElevate.

```
typedef [v1_enum] enum AMSI_UAC_REQUEST_TYPE
{
    // Request to launch an EXE elevated.
    AMSI_UAC_REQUEST_TYPE_EXE = 0,

    // Request to launch an COM server Outofproc elevated.
    AMSI_UAC_REQUEST_TYPE_COM = 1,

    // Request to launch an MSI elevated.
    AMSI_UAC_REQUEST_TYPE_MSI = 2,

    // Request to launch an ActiveX installation elevated.
    AMSI_UAC_REQUEST_TYPE_AX  = 3,

    AMSI_UAC_REQUEST_TYPE_MAX = 4

} AMSI_UAC_REQUEST_TYPE;
```

```
typedef [v1_enum] enum AMSI_UAC_TRUST_STATE
{
    // UAC determined the binary to be from a trusted publisher.
```

```
    AMSI_UAC_TRUST_STATE_TRUSTED   = 0,

    // UAC determined the binary to be from an untrusted publisher.
    AMSI_UAC_TRUST_STATE_UNTRUSTED = 1,

    // UAC determined the binary to be from a publisher blacklisted by the
admin.
    AMSI_UAC_TRUST_STATE_BLOCKED   = 2,

    AMSI_UAC_TRUST_STATE_MAX       = 3

} AMSI_UAC_TRUST_STATE;
```

```
typedef struct AMSI_UAC_REQUEST_CONTEXT
{
    ULONG                 ulLength;
    // Contains a ProcessID for AMSI_UAC_REQUEST_TYPE_EXE elevation requests,
    // contains 0 otherwise.
    ULONG                 ulRequestorProcessId;

    AMSI_UAC_TRUST_STATE  UACTrustState;

    AMSI_UAC_REQUEST_TYPE  Type;

    [switch_is(Type)]
        union {
            [case(AMSI_UAC_REQUEST_TYPE_EXE)]
AMSI_UAC_REQUEST_EXE_INFO  ExeInfo;
            [case(AMSI_UAC_REQUEST_TYPE_COM)]
AMSI_UAC_REQUEST_COM_INFO  ComInfo;
            [case(AMSI_UAC_REQUEST_TYPE_MSI)]
AMSI_UAC_REQUEST_MSI_INFO  MsiInfo;
            [case(AMSI_UAC_REQUEST_TYPE_AX)]  AMSI_UAC_REQUEST_AX_INFO   Acti
veXInfo;
        }RequestType;

    // Contains a non-zero value if this is a Auto-Elevate request.
    BOOL   bAutoElevateRequest;
} AMSI_UAC_REQUEST_CONTEXT, *LPAMSI_UAC_REQUEST_CONTEXT;
```

AMSI_UAC_TRUST_STATE defines what the UAC logic has deduced based on the signature information and the kind of dialog about to be displayed.

AMSI_UAC_TRUST_STATE_TRUSTED means the Signature was validated and chained to an installed root certificate. UAC would display a dialog with a "**Blue Band**" and display **Verified Publisher**.

AMSI_UAC_TRUST_STATE_UNTRUSTED means either the signature was invalid or the binary did not contain any signature at all. UAC would display a dialog with an "**Orange Band**" and display **Unknown Publisher**.

AMSI_UAC_TRUST_STATE_BLOCKED means the certificate has been explicitly blocked by the administrator. UAC would display a dialog with a "**Red Band**" and the elevation request is aborted.

```
typedef struct AMSI_UAC_REQUEST_CONTEXT
{
    ULONG                   ulLength;
    AMSI_UAC_TRUST_STATE    UACTrustState;
    BOOL                    bAutoElevateRequest;

    AMSI_UAC_REQUEST_TYPE   Type;

    [switch_is(Type)]
        union {
            [case(AMSI_UAC_REQUEST_TYPE_EXE)] AMSI_UAC_REQUEST_EXE_INFO
ExeInfo;
            [case(AMSI_UAC_REQUEST_TYPE_COM)] AMSI_UAC_REQUEST_COM_INFO
ComInfo;
            [case(AMSI_UAC_REQUEST_TYPE_MSI)] AMSI_UAC_REQUEST_MSI_INFO
MsiInfo;
            [case(AMSI_UAC_REQUEST_TYPE_AX)]  AMSI_UAC_REQUEST_AX_INFO
ActiveXInfo;
        }RequestType;

    BYTE    byReserved[32];

} AMSI_UAC_REQUEST_CONTEXT, *LPAMSI_UAC_REQUEST_CONTEXT;
```

AMSI_UAC_TRUST_STATE defines what the UAC logic has deduced based on the signature information and the kind of dialog about to be displayed.
AMSI_UAC_TRUST_STATE_TRUSTED means the Signature was validated and chained to an installed root certificate. UAC would display a dialog with a "**Blue Band**" and display **Verified Publisher**.
AMSI_UAC_TRUST_STATE_UNTRUSTED means either the signature was invalid or the binary did not contain any signature at all. UAC would display a dialog with an "**Orange Band**" and display **Unknown Publisher**.
AMSI_UAC_TRUST_STATE_BLOCKED means the certificate has been explicitly blocked by the administrator. UAC would display a dialog with a "**Red Band**" and the elevation request is aborted.

34.3 REQUIREMENTS FOR AV WHEN CONSUMING AND RESPONDING TO AMSI-UAC.

Since UAC is a very sensitive code path, providers need to be loaded in an out-of-proc COM server.

The provider can spend up to 1 second making a decision, UAC call will timeout after 1 second, and fall back to default UAC path.

The AMSI controller is also responsible for making sure the Out-of-proc that is instantiated is a legitimate binary and is at least running as an antimalware PPL (protected process light).

34.4 SCREENSHOTS

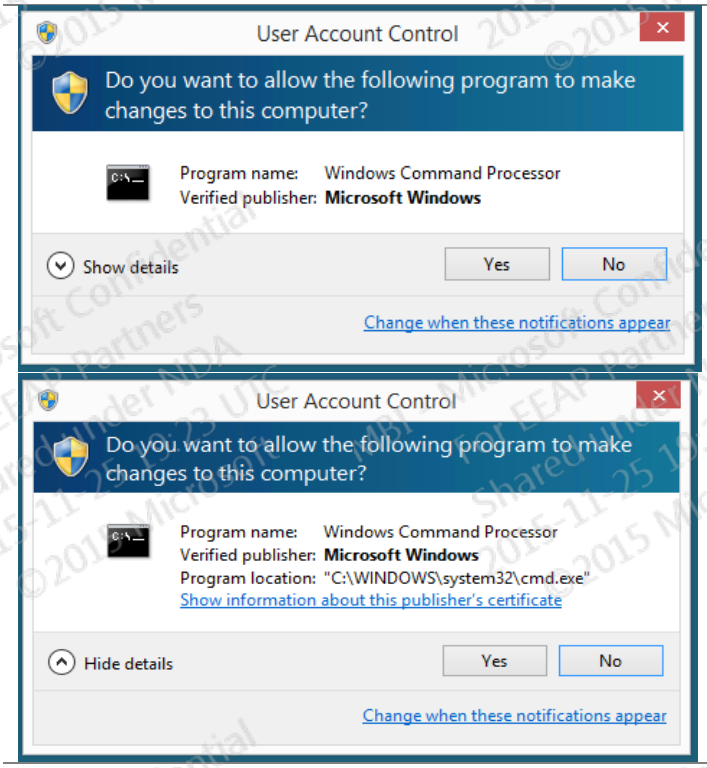
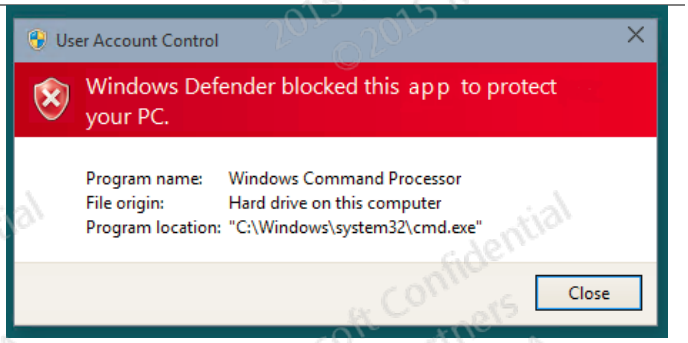
Old UAC	Improved UAC
	

Figure 24: UAC

35 APPENDIX O: USER NOT PROTECTED SCENARIOS FOR WIN8 AND WIN8.1

This section applied to Windows 8 and Windows 8.1 but was never implemented. It has been removed and will not appear in future versions of this document.