Federal Office
for Information Security

Work Package 2: Analysis of Windows 10

Version: 1.0

Federal Office for Information Security
Post Box 20 03 63
D-53133 Bonn
Phone: +49 22899 95820-0
E-Mail: bsi@bsi.bund.de
Internet: https://www.bsi.bund.de

Table of Contents

Figures

Tables

1 Introduction

1.1 Zusammenfassung

Dieses Kapitel stellt das Ergebnis von Arbeitspaket 2 des Projekts “SiSyPHuS Win10: Studie zu Systemaufbau, Protokollierung, Härtung und Sicherheitsfunktionen in Windows 10“ dar. Das Projekt wird durch die Firma ERNW GmbH im Auftrag des Bundesamts für Sicherheit in der Informationstechnik (BSI) durchgeführt.

Ziel dieses Arbeitspakets ist die Analyse der Architektur des Microsoft Windows 10 Betriebssystems (Build 1607, 64-bit, long-term servicing branch (LTSB), Deutsch), sowie Logging-Funktionalitäten und verschiedener wohldefinierter Komponenten. Die wesentlichen Inhalte dieser Arbeit sind:

- Erstellung einer konzeptionellen und auch feingranularen Übersicht der Architektur des Windows 10 Betriebssystems, welche die wesentlichen Bausteine des Systems identifiziert und deren Funktionalitäten und Zusammenspiel erläutert. Hierbei liegt ein besonderer Fokus auf den in Windows 10 erstmalig eingeführten grundlegenden Neuerungen in der Architektur durch Nutzung von Virtualisierung auf Betriebssystemebene.
- Für jede der folgenden im Vorfeld festgelegten Komponenten wird eine kompakte Übersicht der Architektur und ihrer Bestandteile und deren Zusammenhänge gegeben:
  - Power Shell und Windows Script Host
  - Telemetrie
  - Device Guard
  - Virtual Secure Mode (VSM)
  - Trusted Platform Module (TPM)
  - Unified Extensible Firmware Interface (UEFI) “Secure Boot”
  - Universal Windows Platform (UWP)

  Eine detaillierte Analyse der einzelnen Komponenten erfolgt in weiteren, dediziert hierfür vorgesehenen Arbeitspaketen im Projektverlauf.
- Eine Übersicht über Logging-Funktionalitäten in Windows 10, insbesondere EventLog und Event Tracing for Windows (ETW). Konfigurationsmöglichkeiten für Benutzer werden auch diskutiert, wobei auch hier weitere Details in einem dedizierten Arbeitspaket folgen.

Die wesentlichen Ergebnisse der drei Beiträge des Kapitels werden im Folgenden zusammengefasst, wobei für weitere Details auf die ausführlichen Erläuterungen in den jeweiligen englischsprachigen Abschnitten verwiesen wird.

Die Veröffentlichung von Windows 10 markiert eine wesentliche Neuerung in der Architektur von Microsoft-Windows Betriebssystemen: Die Einführung des VSM, welcher sich des Konzepts der Virtualisierung bedient, um kritische Kernelfunktionen zu schützen. VSM ist eine optionale Funktion deren Aktivierung zu relevanten architeturellen Unterschieden zwischen Windows 10 und dessen Vorgängerversionen führt. In Abschnitt 2.1 wird eine Übersicht über beide Modi (traditionell ohne VSM sowie mit VSM) gegeben.

Die traditionelle Architektur besteht aus den folgenden Bestandteilen, die sich in Userland und Kernelland untergliedern lassen:

Das Userland von Windows 10 beinhaltet die folgenden Systembestandteile:

- **System support processes**: Dies sind Prozesse, die wesentliche Systemfunktionen des Kernels im Userland darstellen.

- **Services**: Services sind 'klassische' Windows-Dienste, die vom Betriebssystem verwaltet werden.
- **Applications**: 32- oder 64-bit-Benutzeranwendungen.
- **Windows Subsystem**: Das Windows Subsystem besteht aus den folgenden Teilen:
  - Der Subsystem-Prozess `csrss.exe`, der grundlegende Funktionen wie Prozesserstelleung und -terminierung bereitstellt.
  - Der Treiber `win32k.sys`, der insbesondere Systemausgaben, Fensterdarstellung und Eingabe von Peripheriegeräten kontrolliert.
  - Der Konsolen-Hostprozess `conhost.exe`, der Unterstützung und Funktionalität für Konsolenanwendungen bereitstellt.
  - Verschiedene Subsystem-DLLs (z.B. `kernel32.dll`, `advapi32.dll`, `user32.dll`, und `gdi32.dll`), die Funktionen für Userland-Prozesse bereitstellen.
- **ntdll.dll**: Dies ist eine Bibliothek, welche die Windows Funktionen für direkte Interaktion mit Kernelfunktionen bereitstellt.

Im Kernelland von Windows 10 finden sich die folgenden Komponenten:

- **Kernel**: Der eigentliche Betriebssystem-Kern, der in der ausführbaren Datei `ntoskrnl.exe` implementiert ist, stellt die elementaren Funktionen zur Verwaltung der Systemressourcen zur Verfügung und übernimmt beispielsweise das Scheduling von Prozessen.
- **Drivers**: Treiber sind Kernel-Module (typischerweise mit der Dateiendung `.sys`), welche geladen werden können und eine Schnittstelle zwischen Kernel und Hardware darstellen. Tatsächlich kommunizieren jedoch auch Treiber nicht direkt mit der Hardware, sondern durch den HAL.
- **Hardware abstraction layer (HAL)**: HAL ist ein Kernel-Modul, welches eine direkte Schnittstelle zur Hardware darstellt und in der Datei `hal.dll` implementiert ist. Die Schnittstelle abstrahiert architekturspezifische Details der Hardware und ermöglicht somit die Hardwareunabhängigkeit des restlichen Windows-Systems.

Die traditionelle Architektur des Windows Betriebssystems stellt keine Isolationsmechanismen zwischen Kernel-Funktionalitäten bereit. Eine solche Isolation würde jedoch eine sichere Ausführung kritischer Kernel-Funktionalitäten wie die Verifikation von Code-Integrität ermöglichen, weswegen in Windows 10 die VSM-Technologie eingeführt wurde.

Ein Windows 10 System mit aktiviertem VSM besteht aus einem Hyper-V Hypervisor, auf dem zwei Kernel-Modi ausgeführt werden: der non-secure mode und der secure kernel mode (SKM). Der Hypervisor verknüpft einen “virtuellen Vertrauens-Level” (virtual trust level, VTL) mit einer virtuellen Maschine, in der diese Kernel-Modi ausgeführt werden. Diese erlauben es, auf hardwarenaher Ebene eine Isolation umzusetzen, wie beispielsweise die Verwaltung von separaten Seitentabellen (pagetables) zur Speicherverwaltung. Der Hypervisor assoziiert einen VTL von 0 mit einer virtuellen Machine, welche der non-secure kernel Modus zugeordnet ist, und ein VTL von 1 mit einer virtuellen Machine, welche dem SKM zugeordnet ist.

Der non-secure kernel mode stellt im Wesentlichen eine typische nicht-sichere Windows Benutzerumgebung nach traditioneller Architektur bereit.

Der SKM ist ein minimaler Kernel, der eine isolierte, sichere Windows Benutzerumgebung bereitstellt. Diese Umgebung wird als sicher betrachtet, da der Hypervisor die Ausführung des SKM vom non-secure kernel isoliert. Zusätzlich wurden strikte Entwicklungsvorgaben (wie beispielsweise verschlüsselte Inter-Prozess-Kommunikation und verifizierbare Code-Integrität) für die Prozesse in dieser Umgebung definiert.

Auf Basis dieser Architekturbeschreibung wird in Abschnitt 2.3 zur Klärung der Terminologie anschließend noch eine technische Definition des Begriffs “Komponente” im Kontext dieses Dokuments vorgenommen.

Abschnitt 3 beinhaltet die Übersicht über die Architektur der einzelnen bereits genannten Komponenten, welche bereits in der Projektausschreibung zur nähere Untersuchung vorgesehen wurden:

- PowerShell und Windows Script Host

  PowerShell ist eine für administrative Zwecke vorgesehene Benutzerschnittstelle, die auf einer Scripting-Umgebung auf Basis des .NET-Frameworks basiert. Diese Schnittstelle erlaubt eine enge Interaktion mit dem zugrunde liegenden Windows-Betriebssystem .z.B. für Management- und Konfigurationsaufgaben. Die PowerShell unterstützt die Interpretation von Code, der in klassischer PowerShell-Syntax übergeben wurde wie auch die Ausführung von Scripts zur Automatisierung von komplexen Aufgaben. Dies sind typischerweise Dateien mit der Dateiendung `.ps1`.

  Die PowerShell Schnittstelle wird in den ausführbaren Dateien `powershell.exe` und `powershell_ise.exe` implementiert, wobei erstere eine Kommandozeilenschnittstelle bereitstellt und letztere eine grafische Schnittstelle zur Entwicklung von PowerShell Skripten.

  Die eigentliche PowerShell Engine, an die ausgeführte PowerShell-Befehle von diesen Schnittstellen weitergereicht werden, ist in der Bibliothek `System.Management.Automation.ni.dll` implementiert.

  Power Shell ist modular aufgebaut und kann durch sogenannte Module in seiner Funktion erweitert werden.

  Der Windows Script Host stellt eine ganze Bandbreite an Scripting-Möglichkeiten in unterschiedlichen Sprachen zur Verfügung, wie z.B. JavaScript und VBScript.

  WSH besteht aus einer zentralen Scripting Engine in den Dateien `cscript.exe` und `wscript.exe`, welche wiederum eine Konsolen- und eine grafische Schnittstelle bieten. Diese Engine selbst ist sprachunabhängig und die Funktionalität zur Interpretation verschiedener Sprachen wird in einzelnen Bibliotheken (wie z.B. `vbscript.dll` und `jscript.dll`) bereitgestellt und kann so um weitere Sprachen erweitert werden.

- Telemetrie

  Microsoft Telemetrie ist eine Komponente, welche u.A. Daten über Systemabstürze und Systembenutzung sammelt.

  Der Kern der Telemetrie-Komponente ist der *Diagnostics tracking service* der in der Bibliothek `diagtrack.dll` implementiert ist. Dieser Dienst wird automatisch beim Systemstart innerhalb eines `svchost.exe` Service-Host-Containers unter dem Namen `DiagTrack` gestartet.

  Telemetriedaten werden von unterschiedlichen Komponenten des Windows Systems (z.B. System Support Prozesse und Benutzeranwendungen) in Form von ETW Logs an den Diagnostics tracking service übergeben. Dieser sammelt die Telemetriedaten mit seinen *ETW trace session Listenern: Diagtrack-Listener* und *Autologger-Diagtrack-Listener*.

  Der Ordner `%ProgramData%\Windows\Diagnosis\`, ein versteckter Ordner der nur für privilegierte Benutzer zugreifbar ist stellt den Hauptablageort für den Diagnostics tracking service und seine Listener dar. Zusätzlich lädt der Diagnostics tracking service Einstellungen von durch Microsoft betriebenen Servern und speichert diese im Verzeichnis `%ProgramData%\Windows\Diagnosis\DownloadedSettings`.

- Virtual Secure Mode

  Der Hyper-V Hypervisor unterscheidet zwischen sogenannten Partitionen, wobei jede Partition eine virtuelle Maschine darstellt. Die virtuelle Maschine, in welcher der Benutzer VSM aktiviert, enthält den *virtualization management stack*, eine Sammlung von virtualisierungsbezogenen Diensten und Prozessen. Wenn VSM aktiviert ist, wird beim Systemstart zunächst die virtuelle Maschine, welche den SKM ausführt gestartet und der Hypervisor weist dieser ein VTL von 1 zu, der sie effektiv von anderen virtuellen Maschinen isoliert. Danach erstellt der Hypervisor die virtuelle Maschine, die den

virtualization management stack beinhaltet und weist dieser ein VTL von 0 zu, in dem der Benutzer agiert, nachdem Windows 10 gebootet ist. Das Konzept von VTLs erlaubt eine feingranulare Isolation zwischen virtuellen Maschinen, wie beispielsweise Speicherisolation, virtuelle Prozessor Status und Overlay Pages.

- Device Guard

  Device Guard ist eine Windows 10 Komponente zur Verifikation von Code-Integrität, um die Ausführung von nicht vertrauenswürdigem Code (Code der entweder manipuliert wurde oder aus einer nicht vertrauenswürdigen Quelle stammt) zu verhindern. Device Guard implementiert zwei Hauptfunktionen: User-mode code integrity (UMCI) und kernel-mode code integrity (KMCI), welche die Verifikation der Code-Integrität von Usermode bzw. Kernelmode Objekten (wie beispielsweise Anwendungen und Scripts bzw. Kernel-Treibern) erzwingt. Device Guard verwendet Whitelist-Regeln, die auf kryptografischen Informationen (digitale Signaturen oder kryptographische Hashes) zur Verifikation von Code basieren. Diese Regeln können durch administrative Benutzer editiert werden und werden zunächst in Form eines Extensible Markup Language(XML)-Dokuments gespeichert und dann in ein Binärformat umgewandelt. Die Device Guard Funktionalitäten sind als Kernel Routinen implementiert. Falls VSM nicht aktiviert ist, wird die gesamte Funktionalität in der Bibliothek `ci.dll` implementiert. Andernfalls werden die Funktionalitäten aufgeteilt und die sicherheitskritischen Funktionalitäten mit Hilfe von `skci.dll` in dem SKM ausgeführt.

- Trusted Platform Module und UEFI “Secure Boot”

  Secure Boot ist ein Standard zum sicheren Booten von Computern, so dass der Computer nur Software lädt, welche der Computerhersteller und/oder der Betriebssystemhersteller als vertrauenswürdig erachtet. Das TPM ist ein Standard für einen sicheren Krypto-Koprozessor, der primär für die sichere Speicherung von kryptographischen Schlüsseln und zur Messung der Integrität von kritischer Systemsoftware genutzt wird. Obgleich Secure Boot nicht zwingend ein TPM erfordert, kann das TPM als Teil einer Secure Boot-basierten Bootprozedur genutzt werden, um das Sicherheitsniveau weiter zu erhöhen. Hierbei werden Vertrauensbeziehungen zwischen dem Computerhersteller und zu ladender Software technisch durch die Benutzung von X.509 Zertifikaten und kryptografischen Hashes realisiert.

  Secure Boot erfordert einen Computer, welcher die UEFI-Spezifikation erfüllt. UEFI speichert System-Bootvariablen im nicht-flüchtigen RAM des Computers (NVRAM). Unter diesen Variablen finden sich die genannten X.509 Zertifikate und Hashes. UEFI verwaltet so eine ganze Datenbank an Zertifikaten und Hashes, welche auch als Secure Database bezeichnet wird.

- Universal Windows Platform

  UWP ist eine Plattformarchitektur (d.h. eine Umgebung zur Ausführung von Anwendungen), die von Microsoft entwickelt und erstmalig in Windows 10 eingeführt wurde. UWP bietet eine Ausführungsumgebung für Benutzeranwendungen, welche als Universal Windows Apps (UWAs) bezeichnet werden. UWP stellt hierbei eine Umgebung für die Ausführung einer Anwendung auf heterogenen Windows-Plattformen, wie beispielsweise Computer, Smartphones, X-Box-Konsolen und weitere dar. Zusätzlich unterstützt UWP die Ausführung von in unterschiedlichen Sprachen wie C++, C# und VB.NET entwickelten UWAs.

  Der Kern von UWP ist die Windows Runtime *(RT)* API , auf der UWAs entwickelt und ausgeführt werden. Windows RT API wiederum baut auf der Windows API auf, die in den Subsystem DLLs und der `ntdll.dll` Bibliothek implementiert wird.

  Hervorzuheben ist, dass UWAs aus Sicherheitsgründen in Bezug auf die Nutzung von durch die Windows RT und der Windows API bereitgestellten Windows-Funktionalitäten eingeschränkt sind und in einer eingeschränkten Container-/Sandbox-artigen Umgebung ausgeführt werden. Diese Einschränkungen werden durch zwei separate Sicherheitsmechanismen erzwungen:

  - Windows mandatory integrity control (MIC): MIC ist ein Sicherheitsmechanismus von Windows, welcher den Zugriff von Prozessen auf Objekte (wie zum Beispiel Dateien, Verzeichnisse und Hauptspeicher), basierend auf einem Vertrauenswürdigkeitslevel, einschränkt. MIC unterscheidet

hierbei zwischen fünf solcher Level: untrusted, low, medium, high, und system. Windows ordnet UWAs das Level low zu, was diese in ihren Zugriffen stark einschränkt.

- UWA capabilities: Zusätzlich zu den Einschränkungen durch MIC erzwingt UWP weitere Einschränkungen durch die Nutzung sogenannter Capabilities. Capabilities beziehen sich auf Windows-Funktionalitäten, die für eine UWA nicht zur Verfügung stehen, so lange nicht explizit eine entsprechende Capability mit der UWA verknüpft wird. Die Capabilities, die mit einer konkreten UWA verknüpft sind, werden in einem XML-artigen Format als Teil des Manifests der Anwendung spezifiziert.

Abschnitt 3.7 identifiziert und diskutiert die Funktionalität weiterer potentiell relevanter Komponenten, die bislang nicht betrachtet wurden, um eine Basis für die Auswahl von Komponenten für weitere Untersuchungen zu unterstützen. Die Komponenten werden hinsichtlich ihrer Sicherheitsrelevanz beurteilt und übersichtlich aufgelistet. Für die vollständige Liste verweisen wir auf den betreffenden Abschnitt, da sich diese nicht sinnvoll weiter zusammenfassen lässt.

Abschnitt 4 gibt schließlich einen Überblick über Logging-Funktionalitäten in Windows 10: Abschnitt 4.1 behandelt die Logging-Mechanismen EventLog und ETW und die Abschnitte 4.2 und 4.3 diskutieren Konfigurations-Möglichkeiten selbiger.

1.2 Executive Summary

This chapter implements the work plan outlined in Work Package 2 of the project "SiSyPHuS Win10: Studie zu Systemaufbau, Protokollierung, Härtung und Sicherheitsfunktionen in Windows 10" (orig., ger.), contracted by the German Federal Office for Information Security (orig., ger., Bundesamt für Sicherheit in der Informationstechnik (BSI). The work planned as part of Work Package 2 has been conducted by ERNW GmbH in the time period between May and July 2017, in accordance with the time plan agreed upon by ERNW GmbH and the German Federal Office for Information Security.

The objective of this work package is the analysis of the architecture and logging capabilities of the Windows 10 operating system, as well as of individual system functionalities (e.g., telemetry, code integrity verification). We refer to these as components. As required by the German Federal Office for Information Security, the exact release of the Windows 10 system in focus is build 1607, 64-bit, long-term servicing branch (LTSB), German language. The core contributions of this work are:

- An analysis of the architecture of Windows 10: We provide both, a high level (i.e., conceptual), as well as a fine-granular overview of the architecture of the Windows 10 operating system. We identify and structure the building blocks of this system, discuss their main functionalities, and analyze their dependencies and mutual interactions. As part of this analysis, we take into account virtualization as a factor driving a major change in the architecture of Windows systems, first introduced in Windows 10. In addition, we provide a technical definition of the term component and identify components, which have not been designated for analysis as part of the SiSyPHuS Win10 project. These, however, are relevant targets for a detailed analysis in terms of their criticality;
- An analysis of the architecture of Windows 10 components: We provide a compact overview of the architecture of each component designated for analysis as part of the SiSyPHuS Win10 project: PowerShell, Windows Script Host, Telemetry, Device Guard, Virtual Secure Mode (VSM), Trusted Platform Module (TPM), Unified Extensible Firmware Interface (UEFI) "Secure Boot", and Universal Windows Platform (UWP). We identify and analyze the integral parts of each component down to file level (e.g., we identify relevant library and executable files) and discuss their interactions. We emphasize that detailed analyses of the architectures and inner working mechanisms of these components is conducted in designated work packages planned as further parts of the SiSyPHuS Win10 project;
- An overview of logging capabilities: We analyze in detail the architecture of the core logging facilities of Windows 10 – EventLog and Event Tracing for Windows (ETW). In addition, we discuss the ways in which Windows 10 users may configure these facilities.

This work is structured as follows:

- Section 2 provides an overview of the architecture of Windows 10: Section 2.1 reviews the traditional architecture of Windows systems, which also applies to Windows 10; Section 2.2 analyzes the novel architectural concept of Windows systems with virtualization as a key enabling technology, first introduced in Windows 10; Section 2.3 provides a technical definition of the term component and defines criteria for characterizing components that may be considered within the scope of the SiSyPHuS Win10 project;
- Section 3 provides an overview of the architecture of the components designated for analysis as part of the SiSyPHuS Win10 project: Section 3.1 – Section 3.6 provide an overview of the architecture of the components PowerShell and Windows Script Host, Telemetry, Virtual Secure Mode, Device Guard, Trusted Platform Module and UEFI "Secure Boot", and Universal Windows Platform, respectively; Section 3.7, based on the criteria defined in Section 2, identifies and discusses the functionalities of additional components, which have not been designated for analysis as part of the SiSyPHuS Win10 project yet, however, are relevant potential targets for a detailed analysis;

- Section 4 provides an overview of logging capabilities: Section 4.1 discusses the core logging facilities of Windows 10 – EventLog and ETW; Section 4.2 and Section 4.3 discuss the ways in which EventLog and ETW may be configured by Windows 10 users;
- In the Appendix: the 'Tools' section lists used tools and where they can be found; the 'List of Services' and 'List of Drivers' sections list system services and drivers deployed in Windows 10, relevant to the discussions in Section 3.7; the 'Audit Policy Categories and Event IDs' section provides additional information on EventLog, relevant to the discussions in Section 4.

2 Architecture Overview

In this section, an overview of the architecture of the Windows 10 operating system (i.e., of the system's core parts, their implementations and dependencies) is provided. The release of Windows 10 marks a major shift in the architecture of the Windows operating systems; that is, Microsoft introduced the concept of VSM, which utilizes virtualization technology for securing critical system functionalities through isolation, access control, and establishment of trust boundaries. VSM is an optional system feature such that when disabled, the Windows 10 system has an architecture identical to that of its predecessors (referred to as the traditional architecture). In this section, we provide an overview of both the traditional architecture of Windows 10 (Section 2.1) and the architecture of Windows 10 when VSM is enabled (i.e., VSM architecture, Section 2.2).

2.1 Traditional Architecture

Figure 1 presents a compact, whereas Figure 2 a more detailed, overview of the architecture of the Windows 10 operating system. This architecture consists of the following parts: system support processes, services, applications, the Windows subsystem, `ntdll.dll`, drivers, kernel, and the hardware abstraction layer (HAL) ([Yosif 2017], Chapter 2). In this section, we provide a compact overview of the functionalities and implementation of each of these parts.

Windows features two basic execution modes (also referred to as contexts): user mode (referred to as *user land* in Figure 1) and kernel mode (referred to as *kernel land* in Figure 1). The major distinction between these two modes is that threads running in user land, in contrast to those running in kernel land, can exist as separate processes with virtual address spaces allocated to them ([Yosif 2017], Chapter 1). This, among other things, enables Windows to establish and enforce control over access to the memory and execution context of each thread running in user land (see, for example, the discussion on 'protected process light' in Section 3.7.4).

The user land of Windows 10 is populated with the following parts of the system:

System support processes: We refer to as system support processes core Windows processes, whose operation is required for the Windows system to properly function. The system support processes are the following:

- *Session manager* (executable: `%SystemRoot%\System32\smss.exe`): The main responsibility of the session manager is the initialization and establishment of user sessions, which includes tasks such as registry initialization and mapping of core system library files ([Yosif 2017], Chapter 2). Through the establishment of user sessions, the session manager enables the concurrent use of the Windows system by multiple users as follows: Windows creates a single (i.e., a master) instance of the session manager at system start-up. This instance creates instances of itself (i.e., slave instances) at each subsequent logon (session) for performance reasons ([Yosif 2017], Chapter 2). After a slave instance finishes the initialization of the session for which it was created, Windows terminates it, leaving the session manager master instance active.

The master session manager process is created at system startup by the process named System (*System* in Figure 2). This process hosts a kernel-mode thread, which, among other things, is responsible for kernel initialization. This is when it creates the master session manager process.

- *Local session manager* (executable: `%SystemRoot%\System32\lsm.exe`): The local session manager manages established user sessions. It communicates with other system support processes through the remote procedure call (RPC) and advanced local procedure call (ALPC) protocols for receiving or relaying relevant notifications. For example, the Winlogon system support process informs the local session manager when a user has logged on. We emphasize that in Windows 10, in contrast to previous versions of Windows, the local session manager is implemented as a service.

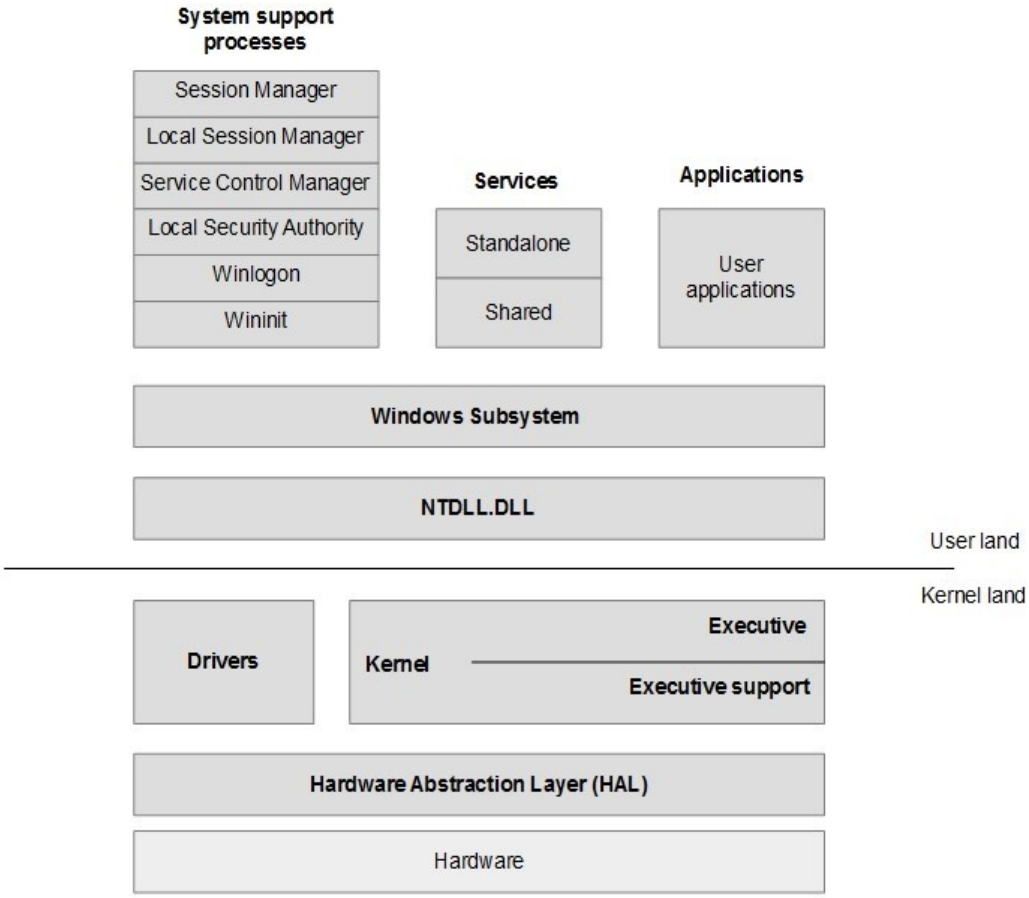


Figure 1: A compact overview of the architecture of Windows 10

- *Winlogon* (executable: `%SystemRoot%\System32\winlogon.exe`): Winlogon manages user logon and logoffs. Winlogon collaborates closely with the local security authority system support process. At each user logon, Winlogon captures the credentials provided by the user and sends them to the local security authority. It then validates the credentials and if they are valid, it notifies Winlogon of their validity, after which a user session is initialized. Winlogon is activated at any point over the run-time of a given Windows system, when the secure authentication sequence is triggered by pressing the key combination «Ctrl»+«Alt»+«Delete».
- *Local security authority* (executable: `%SystemRoot%\System32\lsass.exe`): The local security authority is responsible for the establishment and application of local security policies. This includes user and group privileges, password policies, logging of security-relevant events, validation of user credentials, and so on. Most of the functionalities of the local security authority is implemented in the library file `%SystemRoot%\System32\lsarv.dll`.
- *Service control manager* (executable: `%SystemRoot%\System32\services.exe`): The service control manager is responsible for starting, stopping, and interacting with services. We discuss on services in more detail later in this section.
- *Wininit* (executable: `%SystemRoot%\System32\wininit.exe`): Wininit initializes the Windows system and the initial user session established at system start-up. Wininit performs the following tasks as part of a larger sequence of activities: initialization of the Windows scheduling infrastructure, creation of the service control manager and the local security authority support processes (see Figure 2), and initialization of the local session manager, after which it waits for system shutdown.

Services: Services are processes, which perform a variety of tasks without user interaction. An example service is *AppXSvc* (see Figure 2), which provides support for deploying Windows Store applications (see

Section 3.6). The operation of services is managed by the system support process *service control manager*. Services can be standalone (i.e., services that operate as separate processes) or shared (i.e., services that operates as part of a service host process named `svchost.exe` for performance reasons).

Applications: User applications, which can be 32-bit or 64-bit. Later in this section, we discuss more on how processes, and therefore applications, interact with the core of the Windows operating system.

Windows subsystem: The Windows subsystem provides an execution environment to processes. The Windows subsystem is composed of the following parts:

- the subsystem process (referred to as the client/server run-time subsystem process, executable: `%SystemRoot%\System32\csrss.exe`, see Figure 2): this process performs crucial functions, such as creating and deleting processes and threads.
- the driver `win32k.sys`: this driver manages system output, controls window displays, captures input from peripheral devices, and so on.
- the console host process (executable: `%SystemRoot%\System32\conhost.exe`): this process provides support for console applications.
- *subsystem DLLs* (e.g., `kernel32.dll`, `advapi32.dll`, `user32.dll`, and `gdi32.dll`): these DLL library files implement functions used by user-land processes to invoke Windows functionalities implemented as part of the kernel. For example, the `kernel32.dll`, `advapi32.dll`, `user32.dll`, and `gdi32.dll` implement functionalities needed by almost all user processes, such as memory management and drawing functionalities ([Yosif 2017], Chapter 2).

Although the Windows subsystem spans multiple layers of the architecture of Windows 10 as presented in Figure 1 and Figure 2, in these figures, the subsystem is depicted as a single layer for the sake of simplicity.

ntdll.dll: the `ntdll.dll` library file is a core Windows library file that implements functions for interaction with kernel functions (referred to as system service functions). These can be invoked from processes running in user land. In addition, this file implements functions supporting the operation of the functions implemented in the subsystem DLLs, e.g., functions implementing inter-process communication (IPC).

The kernel land of Windows 10 is populated with the following parts of the system:

Kernel: The kernel, implemented in the executable `%SystemRoot%\System32\ntoskrnl.exe`, performs fundamental tasks, such as thread scheduling and synchronization. It consists of two parts: *executive* and *executive support*. The executive implements the core kernel functionalities, such as implementation and managing of the system registry and the plug and play mechanism (i.e., identification and loading of the drivers required for the operation of a plugged in device). The functionalities of executive are implemented in functions (i.e., the system service functions), some of which can be invoked from user land. The executive support consists of a set of functions supporting the functionalities implemented in the executive, such as memory allocation functions.

Drivers: Drivers are loadable kernel modules (typically with the extension `.sys`), which represent an interface between the kernel executive functionalities managing input/output (I/O), and hardware. Depending on the type of I/O they interface, drivers may be file-system drivers (i.e., drivers that interface file I/O operations), protocol drivers (i.e., drivers that implement network protocols, such as TCP/IP), and so on. Drivers do not communicate directly with hardware, but through the HAL, which we discuss next.

HAL: HAL is a loadable kernel module representing a direct interface to the hardware on which a given instance of the Windows system is running. The HAL is implemented in the `%SystemRoot%\System32\hal.dll` library file. This interface is used by all parts of the Windows system running in kernel land for communication with hardware components. HAL abstracts hardware-dependent, architecture-specific details (e.g., interrupt controllers) and therefore, it enables the portability of the rest of the Windows system.

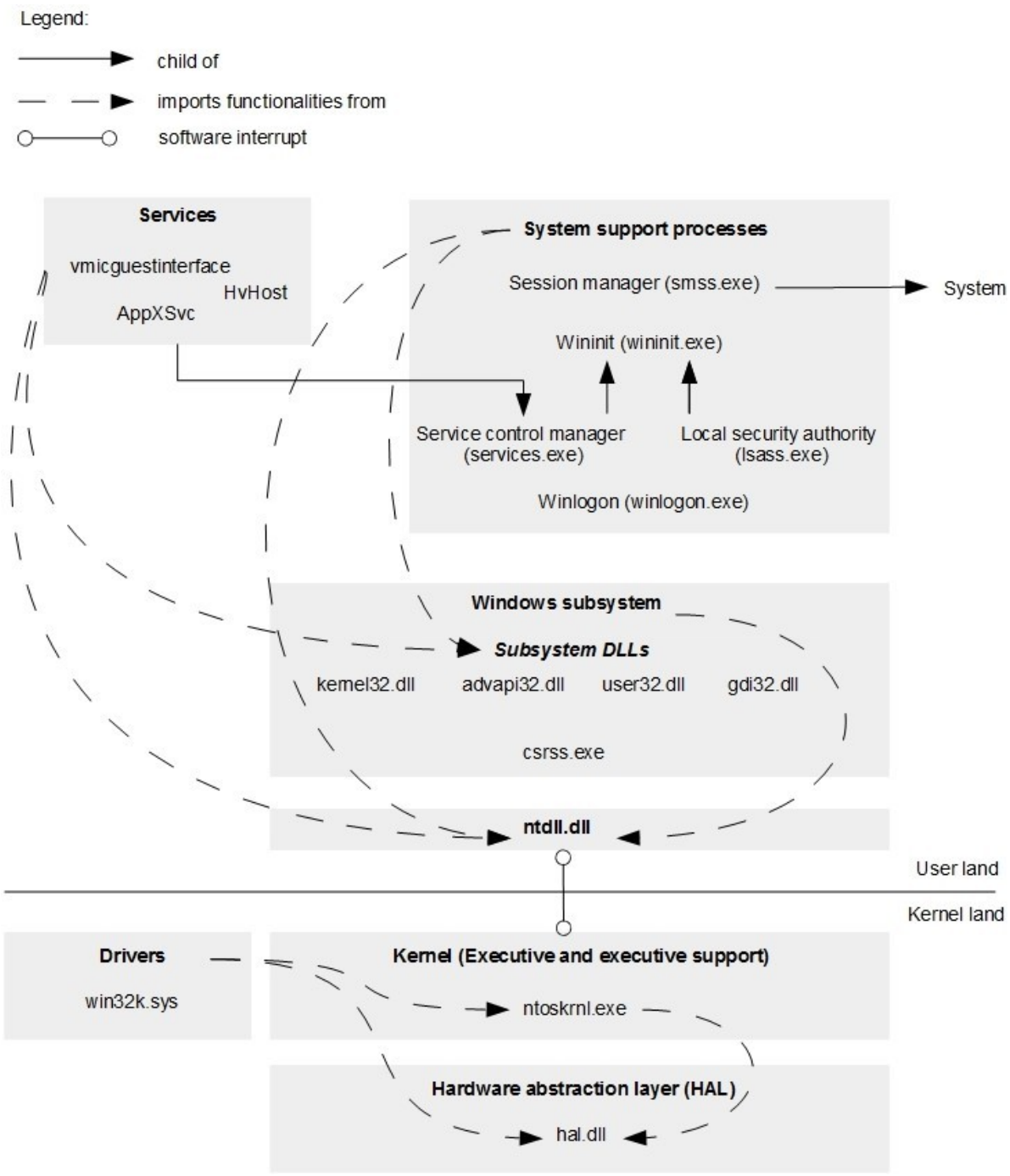


Figure 2: An overview of the architecture of Windows 10

We now discuss how user processes use functionalities of the Windows system (i.e., functionalities of the Windows subsystem and kernel), which are implemented and exposed as programming interfaces. This helps to better understand the interactions and dependencies between the different parts of the Windows 10 system in general. Figure 2 presents an overview of these interactions and dependencies (see the dashed lines).

User processes typically use functionalities of the Window system by invoking functions defined as part of the Windows application programming interface (API). This is implemented as part of the Windows subsystem (see the dashed line between *Services*, *System support processes*, and *Subsystem DLLs* in Figure 2). The Windows API is an interface of the Windows operating system supporting both 32-bit and 64-bit computer architectures. The Windows API functions are documented as part of the documentation of the

Windows Software Development Kit (SDK)[1] and in the Microsoft Developer Network (MSDN) (see, for example, the documentation on the `CreateProcess` Windows API function).[2]

In order to use kernel functionalities, the functions implemented as part of the Windows API invoke functions (i.e., system calls) implemented in the `ntdll.dll` file (see the dashed line between *Windows subsystem* and `ntdll.dll` in Figure 2). Alternatively, the functions implemented in the `ntdll.dll` file can also be invoked from user processes. In this case, their API implementation is commonly referred to as the native system service API (see the dashed line between *Services*, *System support processes*, and `ntdll.dll` in Figure 2).

In case the system calls implemented in `ntdll.dll` require kernel functionalities, they issue a software interrupt in order to change the execution context to kernel mode. This is where the needed functionalities are implemented and can be executed (see the interrupt symbol in Figure 2). The kernel also implements functionalities made available only to drivers (see the dashed line between *Drivers* and *Kernel* in Figure 2). These are implemented as functions commonly known as routines. For example, `ExAllocatePoolWithTag` is a routine, which drivers call to allocate heap memory. Figure 3 is a depiction of the implementation of this routine as part of the kernel executable (as shown in the `windbg` debugger).

In case kernel functionalities or drivers need to interact with hardware components, they invoke functions implemented as part of the HAL (see the dashes lines between *Drivers*, *Kernel* and *HAL* in Figure 2).

```
dumpbin /exports c:\windows\system32\ntoskrnl.exe
Dump of file c:\windows\system32\ntoskrnl.exe
[...]
150   92 002236E8 ExAllocatePoolWithQuota = ExAllocatePoolWithQuota
151   93 000117E0 ExAllocatePoolWithQuotaTag = ExAllocatePoolWithQuotaTag
152   94 00248A40 ExAllocatePoolWithTag = ExAllocatePoolWithTag
153   95 0008D5D0 ExAllocatePoolWithTagPriority = ExAllocatePoolWithTagPriority
154   96 00124FE8 ExAllocateTimer = ExAllocateTimer
[...]
```

Figure 3: Implementation of the ExAllocatePoolWithTag routine in the Windows kernel

We now demonstrate the use of Windows functionalities by user processes, which we discussed above, by taking the example of a user process creating a child process. Figure 4 depicts this example as a set of screenshots of the output of the `windbg` debugger. Figure 4 depicts the function call chain resulting in a child process being created. The process creating the child process (i.e., the parent process) calls the `CreateProcessW` function implemented as part of the Windows API. This, in turn, invokes the `NtCreateUserProcess` function implemented in `ntdll.dll`. The `CreateProcessW` function is exported by the `kernel32.dll` file (see Figure 2) as alias to the `CreateProcessWStub` function. This function invokes the actual implementation of `CreateProcessW` in the `kernelbase.dll` library file.

The `NtCreateUserProcess` function issues a software interrupt in order to change the execution context to kernel mode. This function also specifies the *system service index* (see Figure 4). The index is an integer specifying the handler of the interrupt implemented as part of the `KiServiceTable` kernel structure (executable: `ntoskrnl.exe`). When the kernel completes its operation, it returns relevant results and switches the execution context back to user mode. After this, the parent process continues its execution in this mode.

1 https://developer.microsoft.com/en-us/windows/desktop/develop [Retrieved: 7/5/2017]

2 https://msdn.microsoft.com/en-us/library/windows/desktop/ms682425(v=vs.85).aspx [Retrieved: 7/5/2017]

Figure 4: A function call chain in Windows 10

2.2 Virtual Secure Mode Architecture

In this section, we provide a brief overview of the architecture of Windows 10 when its VSM feature is enabled, which is depicted in Figure 5. For a detailed analysis of the architecture and functionalities of Windows 10 when VSM is enabled, we refer to Work Package 6 of this project.

VSM is a design principle and feature set of the Windows 10 operating system. The traditional architecture of the Windows operating systems (see Section 2.1) does not establish isolation boundaries between kernel functionalities by design. Such isolation boundaries would allow the secure execution of critical kernel functionalities, such as verifying code integrity. The lack of boundaries may lead to severe security breaches. In order to address the previously mentioned issue, Microsoft introduced the VSM feature of Windows 10. This feature enhances the security of the system by establishing kernel-level isolation boundaries through the use of virtualization technology.

From a high-level perspective, the architecture of a VSM-enabled Windows 10 consists of an underlying Hyper-V hypervisor, on top of which operates a single virtual machine (also referred to as a partition)

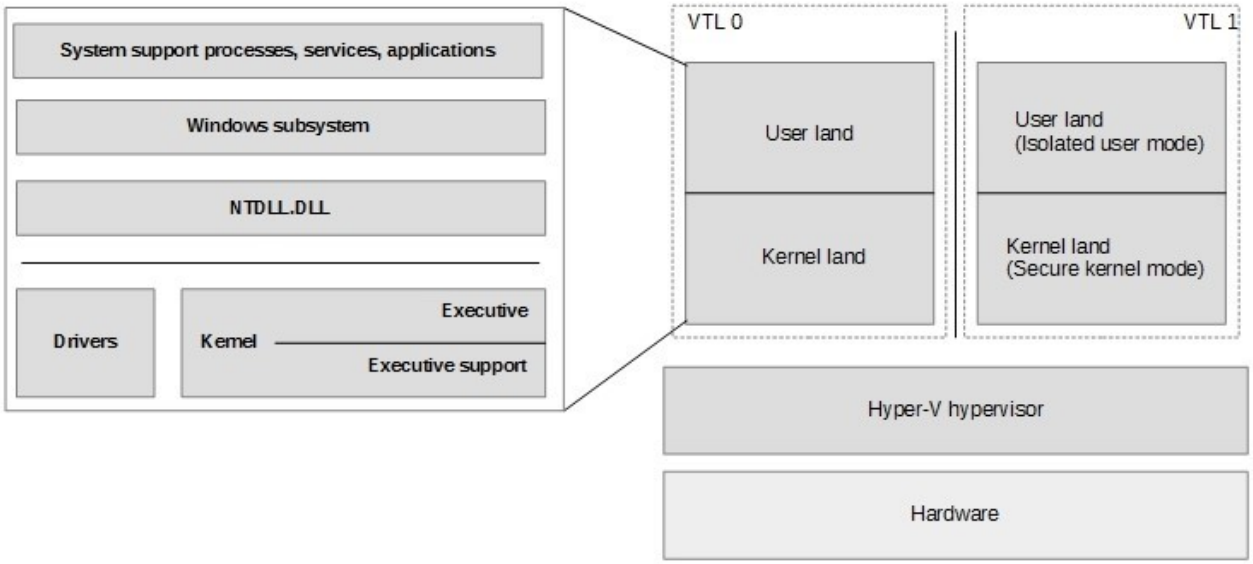


Figure 5: An overview of the architecture of Windows 10 (VSM enabled)

executing the non-secure and the *secure kernel mode* (SKM, see Figure 5). The hypervisor associates a *virtual trust level (VTL)* with these modes, which allows it to establish low-level isolation boundaries (e.g., management of separate page tables and memory allocations). The hypervisor associates a VTL of 0 with the virtual processor(s) assigned to the non-secure kernel mode, and a VTL of 1 with the virtual processor(s) assigned to the SKM. The non-secure kernel mode hosts a typical, non-secure Windows user environment having the traditional architecture discussed in Section 2.1, with minor modifications as discussed later in this section. The SKM is a minimal kernel hosting an isolated, secure Windows user environment. This environment is referred to as secure, since the hypervisor and the SKM isolate its operation from the non-secure kernel and Windows user environment as well as from external entities. In addition, the hypervisor and the SKM enforce strict security requirements (e.g., encrypted IPC and verifiable code integrity) for the processes running in this environment. This is also known as *isolated user mode (IUM)* and the processes running in it as trustlets.

Because of the strict security requirements established in the IUM, this mode is the execution context of security-critical processes. An example is the *local security authority* support process, which, among other things, manages user authentication (see Section 2.1). For example, when VSM is enabled, the local security authority process (`lsass.exe`) running in the non-secure Windows environment does not perform the actual credential verification. This task is delegated to a secure part of this process (executable: `LsaIso.exe`), which runs in the IUM. This separation of the functionalities of the local security authority prevents any abuse of the local security authority for the purpose of accessing user credentials in an unauthorized manner, including abuses from a user with administrator privileges. Figure 6a and Figure 6b depict the deployment of the local security authority in a Windows 10 system with VSM disabled and enabled, respectively.

a) b)

Figure 6: The deployment of the local security authority in a Windows 10 system with a) VSM disabled; b) VSM enabled

2.3 Terminology and Scope

In Section 2, we provided an overview of the architecture of the Windows 10 operating system and introduced its parts comprising this architecture. This allows for defining the scope of the rest of the work planned as part of this work package (see Section 3 and Section 4). To this end, we first define the term component as used within the scope of this work package.

Under *component*, we understand:

a the implementation of a single functionality of the Windows operating system (e.g., telemetry, virtual secure mode),

b which consists of single or multiple parts of this system, whereas parts of the Windows operating systems are: system support processes, services, applications, the Windows subsystem, `ntdll.dll`, drivers, the Windows kernel, and the HAL (see Section 2.1).

Given the above definition of the term component, we now define the criteria for specifying the components that are in the focus of this work package and project:

- Criterion C1) components that have already been designated for analysis as part of this project (i.e., PowerShell, Windows Script Host, Telemetry, Device Guard, Virtual Secure Mode, Trusted Platform Module, UEFI "Secure Boot", and Universal Windows Platform); or
- Criterion C2) components that have not been designated for analysis as part of this project, however, which are Windows 10-specific (i.e., that is present only in Windows 10, see Section 3.7 for a better understanding of what a Windows 10-specific component is) and security-critical, where a security-critical component is a:

  a component that performs security functions (e.g., file-system monitoring); or

  b component whose functionalities could be abused, *not necessarily* by exploiting an implementation vulnerability, to cause serious security and/or privacy issues (e.g., components that perform location tracking).

We now establish the goals (i.e., we present the contributions) of this work package:

a an overview of the architecture of the Windows 10 operating system (Section 2);

b an overview of the architecture of each component in the scope of this work package (Section 3) ;

c an overview of the logging capabilities of the Windows 10 operating system as well as of each component in the scope of this work package (Section 4).

In addition to the above, we analyze the architecture of Windows 10 in order to identify components that have not been designated for analysis as part of this project, however, which fulfill criterion C2) and therefore may be considered in the scope of this project. We provide an overview of the architecture, and the logging capabilities of each of these components in other work packages of this project.

3 Component Architecture

In this section, a compact overview of the architecture of each component that is in the scope of this work package (see Section 2.3) is provided. For a detailed overview of the architectures and functionalities of these components, we refer to the respective work packages of this project.

With respect to the criterion C1) defined in Section 2.3, the components PowerShell, Windows Script Host, Telemetry, Device Guard, Virtual Secure Mode, Trusted Platform Module, UEFI “Secure Boot”, and Universal Windows Platform are within the scope of this work package and project. Their architectures are discussed in Section 3.1 - 3.6. Components that fulfill the criterion C2) defined in Section 2.3, and therefore are considered within the scope of this project, are discussed in Section 3.7.

3.1 PowerShell and Windows Script Host

In Section 2.2.1.1 and 2.2.1.2, we discuss the architecture of the PowerShell and Windows Script Host, respectively.

3.1.1 PowerShell

PowerShell is a user interface of the Windows 10 operating system, which offers a rich set of functionalities to users, with a focus on system administration functionalities. PowerShell is based on an extensive scripting environment built on top of the powerful .NET framework. This allows for the close interaction of users with the underlying Windows operating system for management and configuration purposes.

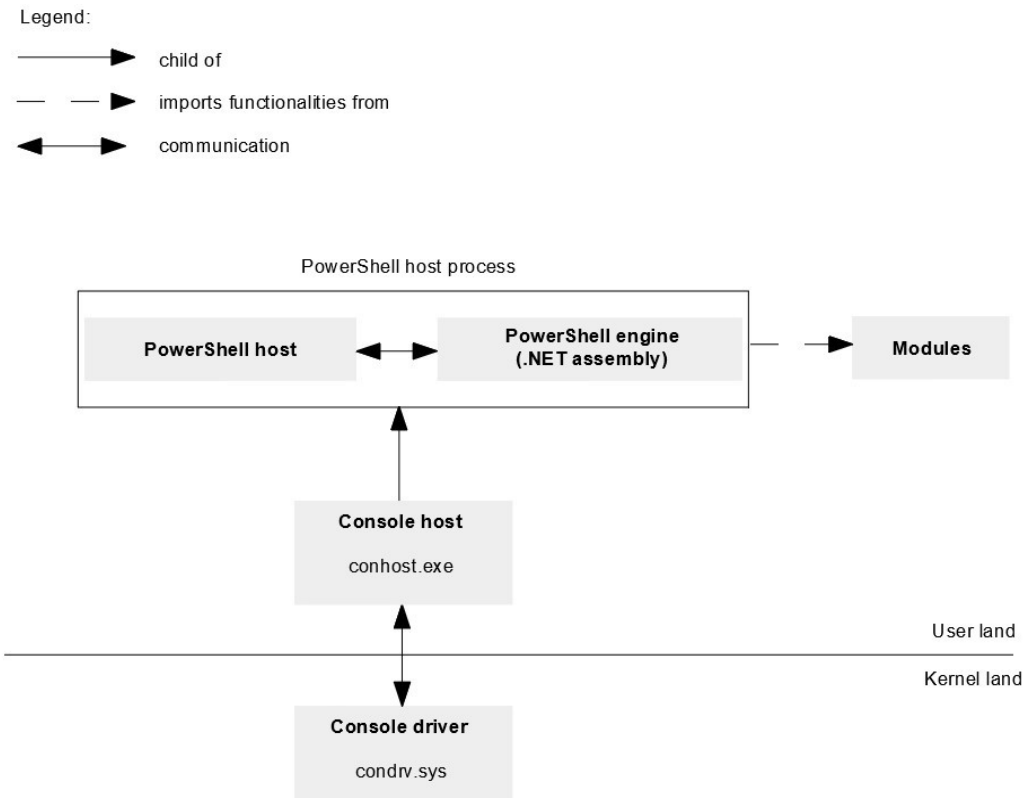


Figure 7: The architecture of PowerShell

The PowerShell scripting environment allows for the interpretation of code written in PowerShell syntax (i.e., PowerShell code) as well as the automation of complex tasks through the use of PowerShell scripts (i.e., files that encapsulate PowerShell code, typically with the `.ps1` extension).

PowerShell may take as input user commands that are implemented as .NET objects, referred to as commandlets (cmdlets). The results of these commands are also .NET objects, whose values are presented to users in a textual format.

Figure 7 provides a compact overview of the architecture of PowerShell as deployed at the Windows 10 system analyzed as part of this work package (see Section 1); that is, PowerShell of version 5.1.14393.0.

The interface that PowerShell presents to users is implemented in the `%SystemRoot%\System32\WindowsPowerShell\v1.0\powershell.exe` and `%SystemRoot%\System32\WindowsPowerShell\v1.0\powershell_ise.exe` executables (*PowerShell host* in Figure 7). `powershell.exe` implements a command line interface (CLI), whereas `powershell_ise.exe` implements the PowerShell integrated scripting environment (ISE). This is a graphical environment for interacting with PowerShell and developing PowerShell scripts. The interface that PowerShell presents to users is hosted and presented to users in the form of a Windows user application, referred to as the PowerShell host process in this work (*PowerShell host process* in Figure 7).

The PowerShell host takes user commands as input and passes them to the PowerShell scripting engine (*Powershell engine* in Figure 7) for processing; that is, the PowerShell host is effectively the front-end of the PowerShell engine, which performs PowerShell functionalities. When the PowerShell engine has processed a given user command, it passes the results back to the PowerShell host, which displays them to users. The PowerShell engine is implemented as a library (i.e., DLL) file, named `System.Management.Automation.ni.dll` and stored in the `%SystemRoot%\assembly` folder. The PowerShell engine is hosted by the PowerShell host process (*PowerShell host process* in Figure 7).

PowerShell is designed to be modular; that is, the functionalities of PowerShell can be extended or shrunk by adding or removing PowerShell modules (*Modules* in Figure 7). In its essence, a PowerShell module is a collection of functionalities, grouped together into a single unit. They can be imported by the PowerShell host process, which hosts the PowerShell host and the PowerShell engine. By default, PowerShell loads a given set of core modules (i.e., modules that provide core functionalities, such as directory listing), which are depicted in Figure 8. PowerShell is distributed with a set of modules, however, in addition to these modules, users can develop custom modules.

```
PS C:\Windows\system32> Get-Module | Select-Object Name, Version, ModuleType, Path | Format-List

Name       : Microsoft.PowerShell.Management
Version    : 3.1.0.0
ModuleType : Manifest
Path       : C:\Windows\system32\WindowsPowerShell\v1.0\Modules\Microsoft.PowerShell.Management\Microsoft.PowerShell.Management.psd1

Name       : Microsoft.PowerShell.Utility
Version    : 3.1.0.0
ModuleType : Manifest
Path       : C:\Windows\system32\WindowsPowerShell\v1.0\Modules\Microsoft.PowerShell.Utility\Microsoft.PowerShell.Utility.psd1

Name       : PSReadline
Version    : 1.2
ModuleType : Script
Path       : C:\Program Files\WindowsPowerShell\Modules\PSReadline\1.2\PSReadLine.psm1

Name       : SecureBoot
Version    : 2.0.0.0
ModuleType : Binary
Path       : C:\Windows\Microsoft.Net\assembly\GAC_64\Microsoft.SecureBoot.Commands\v4.0_10.0.0.0__31bf3856ad364e35\Microsoft.SecureBoot.Commands.dll
```

Figure 8: PowerShell core modules

With respect to how PowerShell modules are implemented, there are different types of PowerShell modules:

- Script modules: Script modules are implemented as PowerShell code. These modules are typically implemented in files with the `.psm1` extension.

- Binary modules: Binary modules are implemented as library (i.e., DLL) files, which contain compiled code.
- Dynamic Modules: Dynamic modules are modules that are not loaded from static sources, for example, files that contain PowerShell code or library files. Instead, they are generated dynamically by using the `New-Module` cmdlet.

Additional relevant information (e.g., names of authors or copyright information) can be associated with a given script or binary module through the use of module manifests. Module manifests are implemented as PowerShell data files, which typically have the `.psd1` extension.

The Windows 10 operating system hosts the core PowerShell executables, `powershell.exe` and `powershell_ise.exe`, as follows: when a PowerShell core executable is invoked by a user, Windows creates an instance of the *console host* process (executable: `conhost.exe`, see Section 2.1). This process manages input/ouput requests from console-based applications and relays these requests to the c*onsole driver* (executable: `condrv.sys`). The driver communicates the requests to the Windows kernel for processing ([Yosif 2017], Chapter 2).

3.1.2 Windows Script Host

The Windows Script Host (WSH) is a Windows technology that provides scripting abilities with a wide range of supported features. WSH is a platform, which, among other things, interprets and executes scripts written in a variety of languages (e.g., JScript and VBScript). WSH is typically used for performing tasks that require automation, such as system administration tasks. Next, we discuss the architecture of the WSH deployed in Windows 10 (see Section 1). Figure 9 provides a compact overview of this architecture.

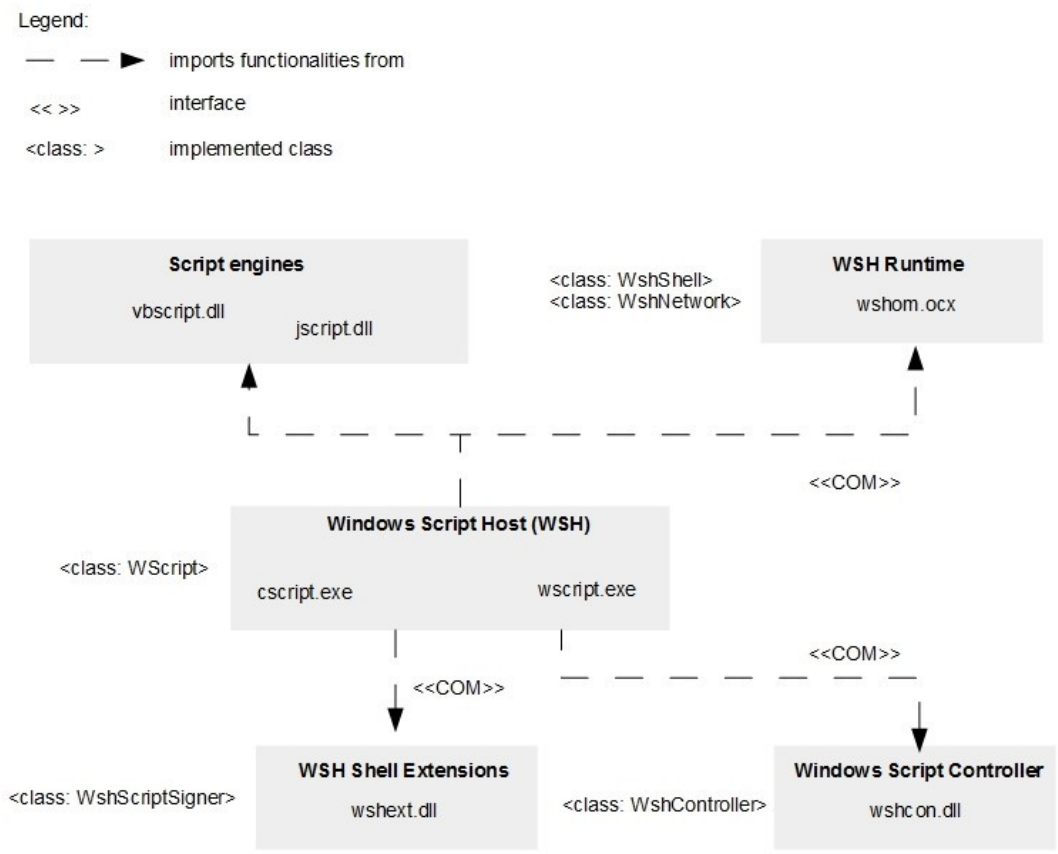


Figure 9: The architecture of WSH

WSH features a central script engine (*WSH* in Figure 9) implemented as the executables`%SystemRoot/System32/cscript.exe` and `%SystemRoot/System32/wscript.exe`. `cscript.exe` allows for executing scripts in console mode, whereas `wscript.exe` for executing scripts in graphical mode.

The WSH central script engine itself is language-independent; that is, it does not have the ability to interpret specific scripting languages, but imports language interpretation functionalities from external script engines (*Script engines* in Figure 9). These script engines are implemented as library files (i.e., DLL files) and are typically located in the `%SystemRoot%` directory. Examples are `vbscript.dll` (i.e., a VBScript script engine) and `jscript.dll` (a JScript script engine). Different scripting engines may be installed so that the scripting in other languages (e.g., PerlScript) is supported by WSH. The use of external script engines makes WSH a highly modular and extensible scripting platform.

WSH implements an object model, that is, it provides a set of objects (instances of classes) for manipulation of script execution as well as helper functions for a variety of tasks (e.g., printing to screen, mapping network drives, and so on). The WSH object model is an implementation of the core functionality of WSH (i.e., script execution). The object model of WSH is exposed to the Windows operating system and other applications through component object model (COM) interfaces (*<<COM>>* in Figure 9). Among other things, WSH uses these COM interfaces to interact with other applications (e.g., to query structured query language -SQL-databases).

The object model of WSH consists of 14 objects, where the root object is the WScript object. For a detailed description of the functionalities of each of the objects, we refer to ([Lissoir 2013], Chapter 1). The root WScript object is implemented in the central script engine (*<class: Wscript>* in Figure 9). The other objects are implemented in the following locations, which are parts of the architecture of WSH:

- *WSH Runtime*: The WSH Runtime, implemented as an ActiveX control (file `%SystemRoot%\wshom.ocx`, see Figure 9), implements the directly instantiable objects WshShell and WshNetwork (*<class: WshShell>* and *<class: WshNetwork>* in Figure 9) and their child objects. Under directly instantiable objects, we understand objects that are registered as COM objects at the Windows system. Child objects of directly instantiable objects are not registered as COM objects and can be accessed only through their parents. Figure 10 presents the directly instantiable object WshNetwork registered as a COM object. We verified the existence of the WshNetwork COM object using the `OleView` utility, which we compiled with `Microsoft Visual Studio`.

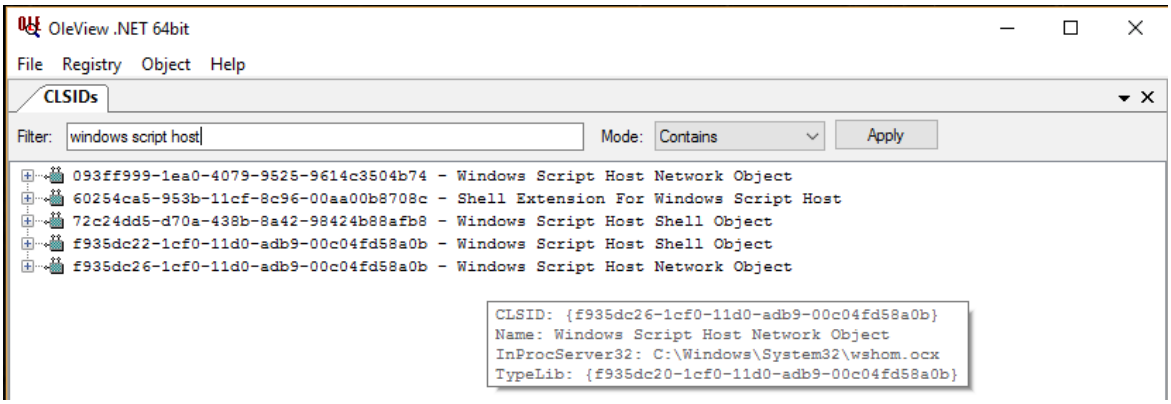


Figure 10: The COM interface of WSH: The WSHNetwork COM object

Through the implementation of the WshShell and WshNetwork objects, the WSH Runtime provides a runtime environment for scripts and allows for performing crucial tasks, such as creating registry keys, establishing network connections, and so on.

- *Windows Script Controller*: The Windows Script controller, implemented in the `%SystemRoot%\wshcon.dll` library file, implements the directly instantiable object WshController (*<class: WshController>* in Figure 9) . The Windows Script controller enables the remote execution of scripts.

- *WSH Shell Extensions*: The WSH Shell Extensions, implemented in the `%SystemRoot%\wshext.dll` library file, implements the directly instantiable object WshScriptSigner (*<class: WshScriptSigner>* in Figure 9). The WSH Shell Extensions enables the digital signing of scripts.

3.2 Telemetry

Microsoft Telemetry (referred to as Telemetry) is a Windows 10 component collecting system crash and usage data (i.e., telemetry data) and uploading this data to remote locations administered by Microsoft for diagnostic purposes. The Telemetry component assembles technical information about the device on which the Windows operating system is installed, and how this system and installed software is performing. Next, we discuss the architecture of the Telemetry component deployed in Windows 10 (see Section 1). Figure 11 provides a compact overview of this architecture.

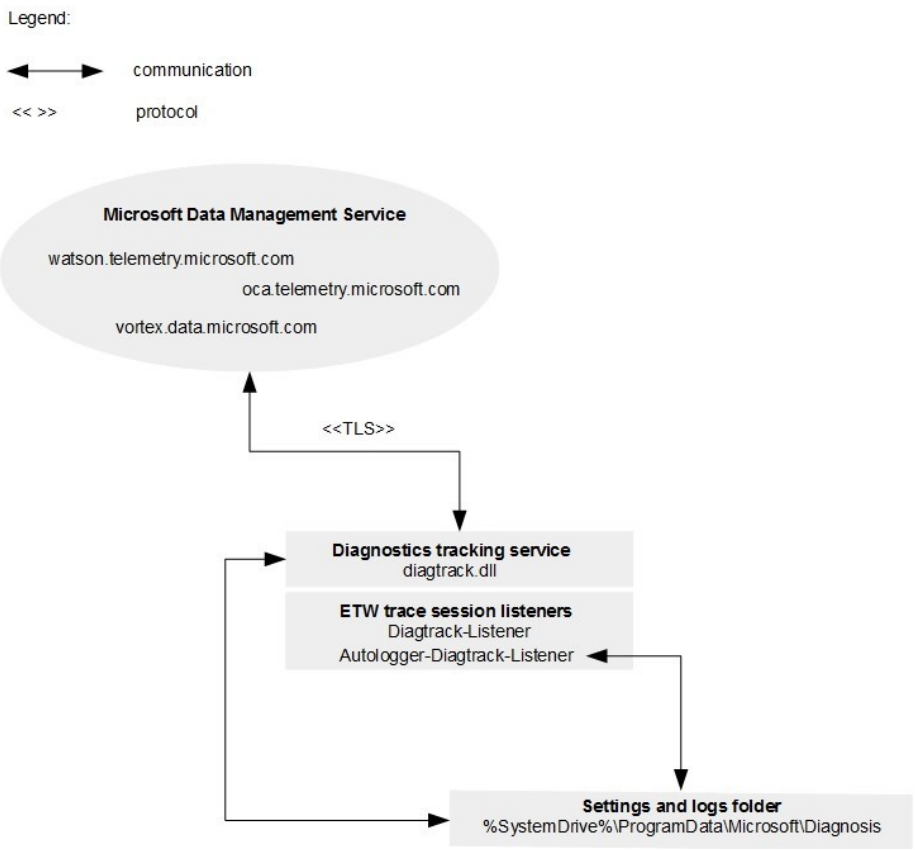


Figure 11: The architecture of Microsoft Telemetry

The core of the Telemetry component is the *Diagnostics tracking service* implemented in the `%SystemRoot%\System32\diagtrack.dll` file. This service is started automatically at system start-up within an `svchost.exe` service host container, registered under the name `DiagTrack` (see Figure 12 for a snippet of the output of the `Process Explorer` tool). Among other things, the Diagnostics tracking service collects relevant crash and usage data and transmits this data to remote servers administered by Microsoft (e.g., to servers with the hostnames *vortex.data.microsoft.com* or *oca.telemetry.microsoft.com*). These servers are part of the Microsoft's back-end infrastructure for processing and storage of telemetry data, named *Microsoft Data Management Service*.[3] Figure 13 depicts a capture of the network traffic, created using the `Microsoft Network Monitor` tool, between the Windows 10 system that is subject of analysis (with the IP address

3 https://docs.microsoft.com/en-us/windows/configuration/configure-windows-telemetry-in-your-organization [Retrieved: 7/5/2017]

`172.18.1.184`) and the remote server (with the hostname *db5.vortex.data.microsoft.com.akadns.net*). The Diagnostrics tracking service transmits telemetry data in an encrypted form, using the transport layer security (*TLS*) protocol.

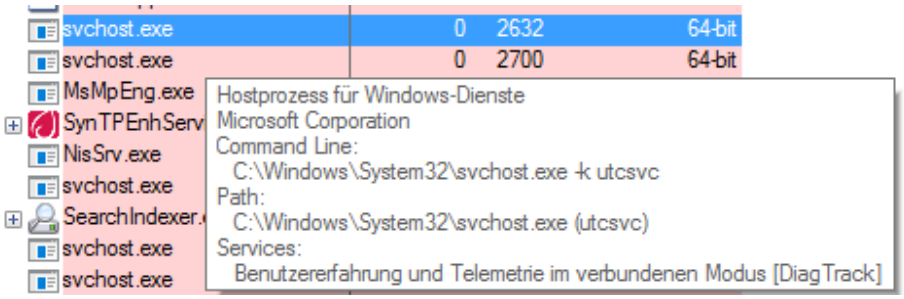


Figure 12: Deployment of the DiagTrack service

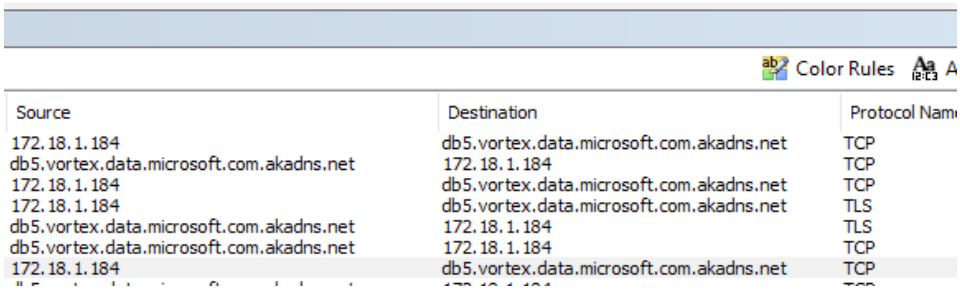


Figure 13: Network traffic between Windows 10 and the Microsoft Data Management Service

Telemetry data is provided by various parts of the Windows system (e.g., system support processes and user applications, see Section 2.1) to the Diagnostics tracking service mainly in the form of ETW logs. The Diagnostics tracking service collects this data using its *ETW trace session listeners: Diagtrack-Listener* and *Autologger-Diagtrack-Listener*. The ETW trace session listeners are configured to receive trace telemetry data from a given set of *trace providers* (parts of the Windows system, such as applications, the kernel, and so on). These implement ETW trace capturing and provisioning to session listeners. Figure 14 depicts some of the providers configured for the *Diagtrack-Listener* session listener, which we list using the `Performance Monitor` Windows utility.

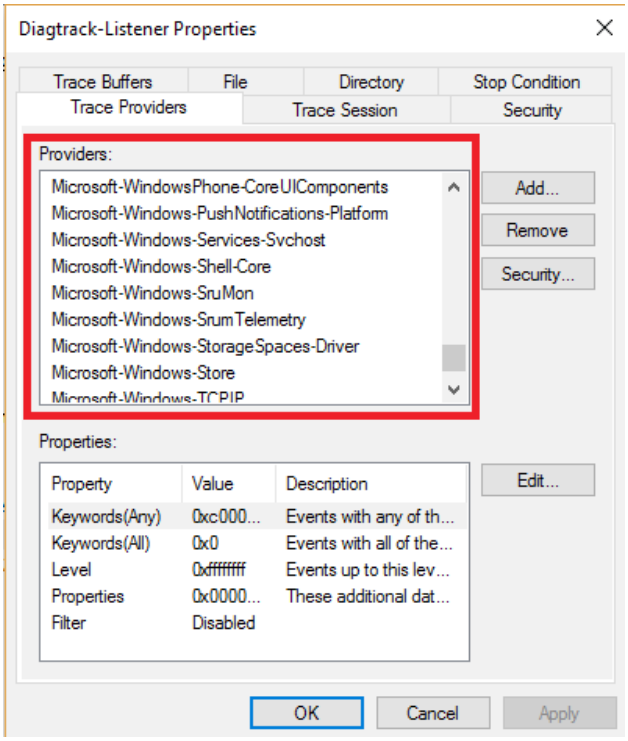


Figure 14: Trace providers configured for Diagtrack-Listener

The *Autologger-DiagTrack-Listener* is an ETW trace listener of type autologger indicating that it receives trace data during system booting. *Autologger-DiagTrack-Listener* is configured to store trace data in the *`%ProgramData%\Windows\Diagnosis`*`\ETLLogs\AutoLogger\AutoLogger-Diagtrack-Listener.etl` file.

The *DiagTrack-Listener* is an ETW trace listener providing trace data at the Diagnostics tracking service during operation. This listener is not configured to store trace data in persistent file storage, but to deliver trace data to the Diagnostics tracking service at run-time. The configurations of *Autologger-DiagTrack-Listener* and *DiagTrack-Listener* can be viewed and modified using the `Performance Monitor` Windows utility.

In addition to the telemetry data provided by the ETW trace listeners, the Diagnostics tracking service implements capabilities for an in-depth inspection of running processes, which is an additional source of relevant telemetry data. This is indicated by the libraries loaded from the Diagnostics tracking service, for example, the `dbghelp.dll` library file, which implements debugging functions. We verified that the Diagnostics tracking service loads this library file with the `Dependency Walker` utility. In addition, the Diagnostics tracking service collects system, platform information (e.g., driver version numbers) by querying relevant sources, such as the system registry and Windows setup logs. We verified this by observing, using the `Process Monitor` utility, the Diagnostics tracking service issuing the `RegOpenKey` and `RegQueryKey` functions in order to read system information. The Diagnostics tracking service, its listeners, and the trace providers can be configured to log and process telemetry data of various types and granularities. We refer to the official Microsoft documentation[3] for more details.

The folder `%ProgramData%\Windows\Diagnosis\` (*Settings and log folder* in Figure 11), a hidden folder available only to privileged users, is the main disk storage location of the Diagnostics tracking service and its listeners. For example, the *Autologger-DiagTrack-Listener* stores in the Diagnosis folder the log file it produces. In addition, the Diagnostics tracking service downloads relevant settings from servers administered by Microsoft and stores these settings in the `%ProgramData%\Windows\Diagnosis\DownloadedSettings` folder.

3.3 Virtual Secure Mode

Section 2.2 provides a compact overview of the architecture of Windows 10 when VSM is enabled and introduces terms such as non-secure kernel mode and SKM, trustlets, and IUM. In this section, we describe in greater detail the functionalities of the Hyper-V hypervisor as the software layer that provides the actual isolation boundaries within the Windows operating system (see Section 2.2). We also provide a more detailed overview of the architecture of Windows 10 when VSM is enabled, with a trustlet running as part of the VSM-enabled platform.

The Hyper-V hypervisor distinguishes between *partitions*, where each partition is a virtual machine (VM). The partition in which the user enables VSM contains the *virtualization management stack*. The stack is a collection of virtualization-related services and processes (e.g., the Hyper-V services, such as `vmicguestinterface` and `vmicheartbeat`, see the Appendix). When VSM is enabled, the root partition is created. Then, the SKM is started and Hyper-V assigns a VTL of 1 to it (*VTL 1* in Figure 15), isolating it from the non-secure kernel mode as discussed later. After the SKM is started, Hyper-V starts the non-secure kernel mode that hosts the virtualization management stack and assigns a VTL of 0 to it (*VTL drop* and *VTL 0* in Figure 15), in which the user operates after Windows 10 is booted [Ionescu 2015].

The concept of VTLs enables a fine-granular isolation ([Mic 2017], Chapter 15):

- memory access isolations: each VTL has a set of memory access protections associated with it. This prevents memory associated with a given VTL from being accessed by an entity operating in another VTL;
- virtual processor states: each virtual processor maintains a per-VTL state, where each VTL has a set of private virtual processor registers assigned to it;
- interrupts: each VTL has a separate interrupt system for preventing interference in interrupt delivery and procession from entities operating in other VTLs.

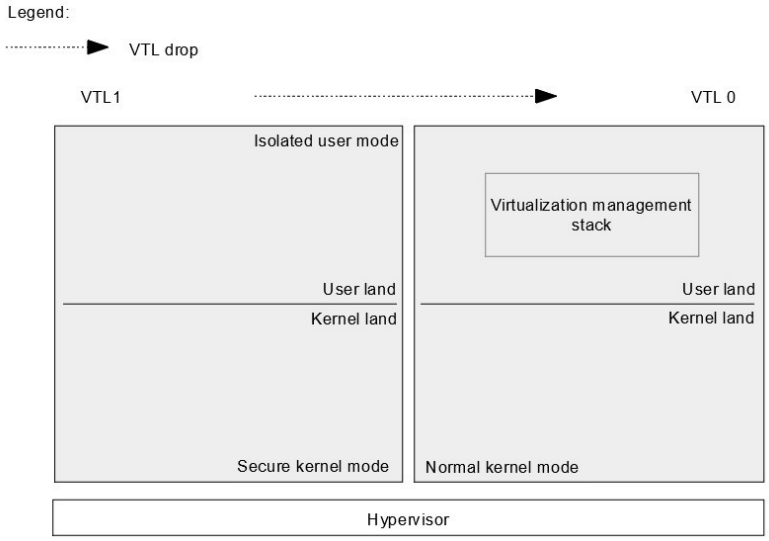


Figure 15: VTLs

- overlay pages: some overlay pages are maintained per-VTL. Overlay pages are special-purpose pages that accompany the memory address space assigned to a partition. These pages store, for example, hypervisor routine call addresses.

VTL 1 is more privileged than VTL 0. Among other things, this includes configuring VTL 0, such as locking its translation lookaside buffer (TLB) in order to prevent TLB invalidations. For more details on VTLs, see the Microsoft's Hypervisor Top Level Functional Specification ([Mic 2017], Chapter 15).

We emphasize that second level address translation (SLAT) is a requirement for activating Hyper-V and therefore VSM ([Finn 2013], Chapter 1).[4] SLAT is a hardware-assisted virtualization technology enabling fast translations between guest-virtual addresses (addresses that provide the partition's view of its virtual memory address space), guest-physical addresses (addresses that provide the partition's view of its physical address space), and system-physical addresses (addresses that define the actual physical address space).

Figure 16 depicts a more detailed architecture of VSM than that presented in Figure 5 (Section 2.2). The normal and the secure kernel modes are implemented in the `%SystemRoot\System32\ntoskrnl.exe` and `%SystemRoot\System32\securekernel.exe` executables, respectively. We presented the functionalities of the non-secure kernel mode in Section 2.1. Based on the libraries we observed to be loaded by trustlets using the `Dependency Walker` utility (i.e., the secured partial implementation of the *local security authority*, `LsaIso.exe` – see Section 2.2), the SKM implements two core functionalities, cryptography and code integrity. These are implemented in the kernel modules `cng.sys` and `skci.dll` (see Section 3.4). A typical *IUM application* (i.e., a trustlet) loads the core IUM library implemented in the `iumbase.dll` library file, which, in turn, loads the `iumdll.dll` file. The latter implements the native IUM system call API interacting directly with the secure kernel, which is analogous to the `ntdll.dll` library file (see Section 2.1). Identical to the normal system calls implemented in `ntdll.dll` (see Section 2.1), the system calls implemented in `iumdll.dll` library file issue software interrupts to invoke kernel functionalities. This changes the execution context to SKM, where the needed functionalities are implemented (see the interrupt symbol in Figure 16) [Ionescu 2015].

4 https://channel9.msdn.com/Blogs/Seth-Juarez/Windows-10-Virtual-Secure-Mode-with-David-Hepkin [Retrieved: 7/5/2017]

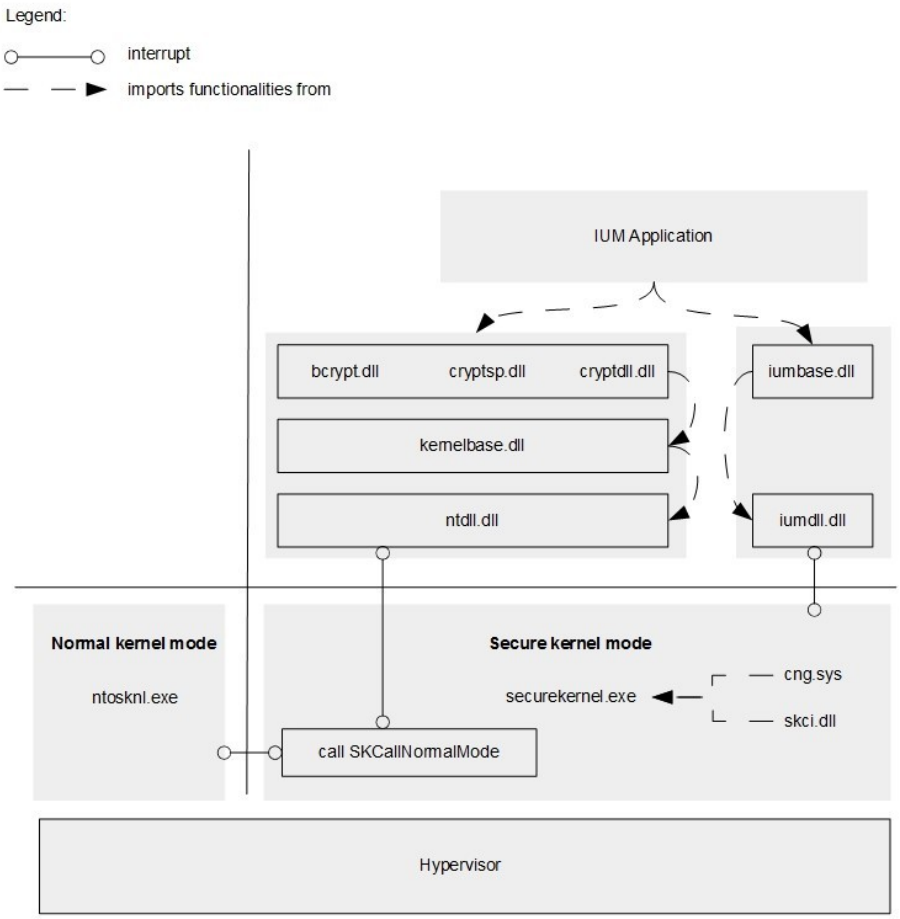


Figure 16: The architecture and function execution paths of VSM

An IUM application may also load standard, traditional Windows libraries to use functionalities of the Windows system by invoking functions defined as part of these libraries. For example, some of the Windows libraries loaded by the `LsaIso.exe` trustlet are the Windows cryptography libraries implemented in the library files `bcrypt.dll`, `cryptsp.dll`, and `cryptdll.dll`. These load the native system service API implemented in the `kernelbase.dll` and `ntdll.dll` library files (see Section 2.1). We emphasize that standard Windows functionalities are performed by the non-secure kernel mode implemented in the `ntoskrnl.exe` executable. Therefore, the SKM does not perform these functionalities itself, but relays (proxies) the IUM's invocations of the library functions implementing these functionalities to the non-secure kernel mode by invoking the `SkCallNormalMode` routine (`call SkCallNormalMode` in Figure 16) [Ionescu 2015]. This routine invokes a hypercall to switch the execution context to the non-secure kernel mode, where the functionality is performed. Figure 17 depicts the invocation of `SkCallNormalMode` by the `securekernel.exe` executable as observed with the `windbg` debugger.

```
securekernel!SkSyscall+0x350:
00000001`4002e3e0 488364246000    and     qword ptr [rsp+60h],0
00000001`4002e3e6 488d4c2460      lea     rcx,[rsp+60h]
00000001`4002e3eb 8a442450        mov     al,byte ptr [rsp+50h]
00000001`4002e3ef 88442461        mov     byte ptr [rsp+61h],al
00000001`4002e3f3 664489742462    mov     word ptr [rsp+62h],r14w
00000001`4002e3f9 e812260100      call    securekernel!SkCallNormalMode
```

Figure 17: An invocation of the SkCallNormalMode routine

3.4 Device Guard

The Device Guard component of Windows 10 is a set of features, which includes preventing the execution of untrusted code. Untrusted code is program code whose integrity and authenticity cannot be verified. Device Guard implements a feature referred to as configurable code integrity. Configurable code integrity takes user-defined criteria into account in order to verify images, that is, to allow only specific images – executable files – to execute. These criteria may involve cryptographic information (e.g., hash values) or non-cryptographic information (e.g., file names).

The configurable code integrity feature can be structured into two categories: user-mode code integrity (UMCI) and kernel-mode code integrity (KMCI) ([Yosif 2017], Chapter 7). UMCI is for entities that operate in user-mode, such as user applications and services. KMCI is for entities that operate in kernel-mode. This includes the kernel and its extensions, such as drivers (see Section 2.1). The UMCI and KMCI implementations of the configurable code integrity feature are also known as Windows Defender Application Control (WDAC). The protection of both user- and kernel-land code implemented by the UMCI and KMCI functionalities of WDAC prevents a wide spectrum of threats, such as malware modifying the execution of executables through the injection of malicious code.

Next, we discuss the architecture of DeviceGuard. Figure 18 provides a compact overview of this architecture.

Device Guard implements rules defined in a policy file editable by users with administrator privileges. This file is originally written in an XML-like format and is converted into a binary format for deployment. The policy file can be digitally signed in order to prevent modifications after it is deployed. Figure 19 depicts an example policy file, which we created by executing the `New-CIPolicy` PowerShell cmdlet (see Section 3.1.1). This cmdlet allows for the automatic creation of comprehensive Device Guard policies assuming that the code (e.g., scripts, executables) deployed in the system, at which the command is executed, is trusted.[5]

A typical Device Guard policy file consists of rules grouped into two sections:[6]

- *Rule options*: The rules defined as part of this section (see the `<Rule>` tags in Figure 19) are used for configuring the overall functionality of Device Guard. An example is the 'Enabled: UMCI' rule, which enables the UMCI functionality of Device Guard.
- *File rules*: The rules defined as part of this section (see the `<File rules>` and `<Allow>` tag in Figure 19) configure verification for images. This configuration is done based on associating a specific level with file rules. Such levels specify at what level a given image is trusted. This work refers to these levels as policy levels. Example levels that may be associated with a given file are 'Hash' (verification based on a hash derived from the file's contents), 'FileName' (verification based on the name of the file), and 'Publisher' (verification based on the certificate used for signing the file).[7] The code integrity levels make DeviceGuard highly configurable and allow for administrators to decide on a trade-off between convenience (i.e., practical usefulness) and strictness of verification. For example, in contrast to 'FileName', the code integrity level 'Hash' prevents any modification of a given file's contents. However, the policy file in which this level is specified would have to be updated at every file modification. This makes 'Hash' a code integrity level that is operationally challenging for verifying frequently modified files.

5 This is known as generation of policies on 'golden' systems, documented at: https://docs.microsoft.com/en-us/windows/device-security/device-guard/deploy-code-integrity-policies-steps [Retrieved: 7/5/2017]

6 A comprehensive documentation on the rules that may be defined as part of a Device Guard policy is available at: https://docs.microsoft.com/en-us/windows/device-security/device-guard/deploy-code-integrity-policies-policy-rules-and-file-rules [Retrieved: 7/5/2017]

7 A comprehensive documentation on the code integrity levels is available at: https://docs.microsoft.com/en-us/windows/device-security/device-guard/deploy-code-integrity-policies-policy-rules-and-file-rules#code-integrity-file-rule-levels [Retrieved: 7/5/2017]

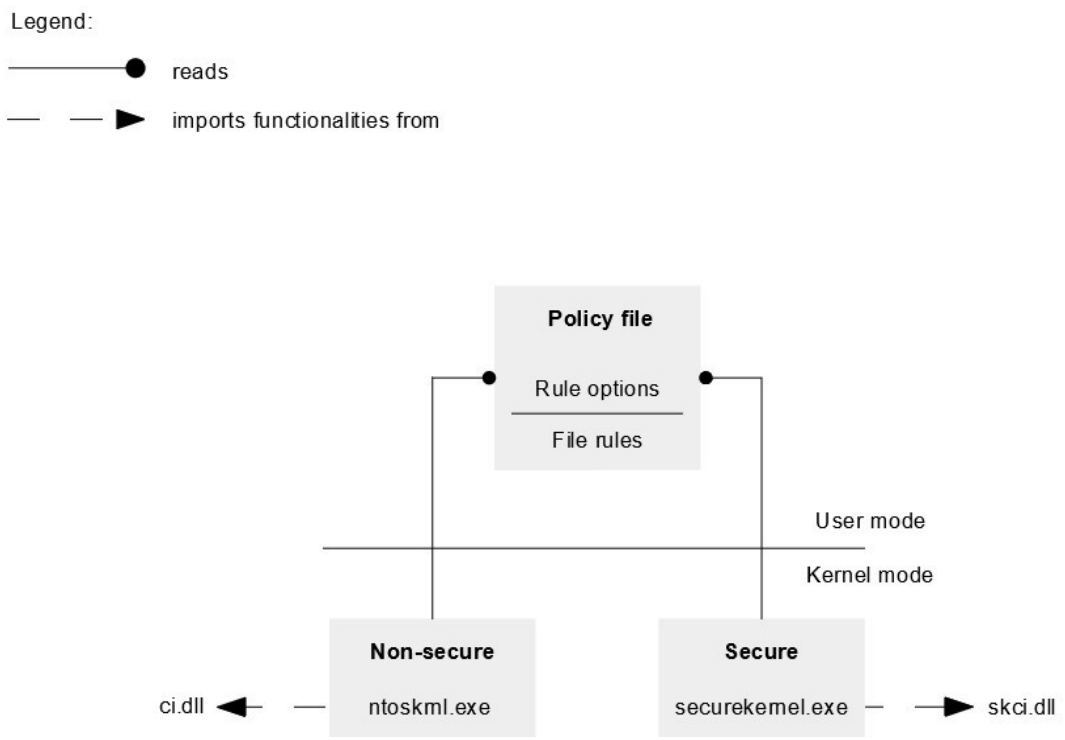


Figure 18: The architecture of Device Guard

The Device Guard image verification functionalities are implemented as kernel routines. The way in which these functionalities are deployed as an integral part of the Windows system depends on whether VSM (see Section 2.2) is enabled. If VSM is disabled, the complete set of Device Guard image verification functionalities are implemented as part of the `ci.dll` library file, which is loaded by the executable `ntoskrnl.exe` implementing the *non-secure* kernel mode (see Section 2.1).

If VSM is enabled, the Device Guard image verification functionalities are split between the non-secure kernel mode and the SKM; that is, the security-critical functionalities performing image verification are executed in the SKM isolated by the Hyper-V hypervisor. This enhances the security of Device Guard itself by preventing attackers that have gained access to the non-secure kernel mode to tamper with the image verification functionalities of Device Guard (i.e., to implement and deploy a mechanism for bypassing Device Guard). The way in which the Device Guard functionalities are executed when VSM is enabled is in line with the principle of leveraging VSM to harden the security of Windows components. This is done by executing the security-critical functionalities of these components, traditionally executed in the non-secure kernel mode, in the SKM. An example is the execution of the *local security authority* support process (see Section 2.2). The image verification functionalities of Device Guard executed in the SKM are also known as hypervisor code integrity verification (HVCI) ([Yosif 2017], Chapter 7). If VSM is enabled, the image verification functionalities of Device Guard are implemented as part of the `skci.dll` library file. This file is loaded by the executable `securekernel.exe` implementing the SKM (see Section 2.2). The secure counterparts of the routines performing image verification, which traditionally are executed in the non-secure kernel mode, have the prefix of `Skci`.

```
<?xml version="1.0" encoding="utf-8"?>
<SiPolicy xmlns="urn:schemas-microsoft-com:sipolicy">
  <VersionEx>10.0.0.0</VersionEx>
  <PolicyTypeID>{A244370E-44C9-4C06-B551-F6016E563076}</PolicyTypeID>
  <PlatformID>{2E07F7E4-194C-4D20-B7C9-6F44A6C5A234}</PlatformID>
  <Rules>
    <Rule>
      <Option>Enabled:Unsigned System Integrity Policy</Option>
    </Rule>
    <Rule>
      <Option>Enabled:Audit Mode</Option>
    </Rule>
    <Rule>
      <Option>Enabled:Advanced Boot Options Menu</Option>
    </Rule>
    <Rule>
      <Option>Required:Enforce Store Applications</Option>
    </Rule>
    <Rule>
      <Option>Enabled:UMCI</Option>
    </Rule>
  </Rules>
  <!--EKUS-->
  <EKUs />
  <!--File Rules-->
  <FileRules>
    <Allow  ID="ID_ALLOW_A_D"
            FriendlyName="C:\Windows\diagnostics\system\WindowsUpdate\DiagPackage.dll Hash Sha1"
            Hash="02CFCFD19CD678538B8F0A12E51EC994A875396B" />
    [...]
```

Figure 19: A Device Guard policy file

3.5 Trusted Platform Module and Unified Extensible Firmware Interface “Secure Boot”

Secure Boot is a standard for the secure booting of computers such that a computer boots using only software that is trusted by the computer's manufacturer. This standard can be combined with implementations of operating system vendors, such as the Measured Boot and Trusted Boot features of the Windows system. The TPM is a standard for a secure cryptoprocessor primarily used for the secure storage of encryption keys and measuring the integrity of critical system software. At the time of writing, two versions of this standard exist, 1.2 and 2.0, both of which are supported by Windows 10.

Although Secure Boot does not require the TPM, it can be used as part of a Secure Boot-based booting procedure of a given computer for further securing the booting of the computer. Given that they ensure that only trusted, non-tampered software components take part in the booting of operating systems, Secure Boot and TPM are effective mechanisms for protecting against attacks and malware modifying these components. Next, we discuss the architecture and operating principles of the Secure Boot and the TPM components deployed in the Windows 10 operating system that is subject of analysis (see Section 1). Figure 20 provides a compact overview of this architecture.

Secure Boot controls the booting of the operating system installed on a computer based on established trust relationships and software integrity guarantees between the computer's manufacturer, i.e., the original equipment manufacturer (OEM), and the vendor of the operating system. These trust relationships and

guarantees are technically realized through the use of X.509 certificates and encryption hashes, as discussed next.

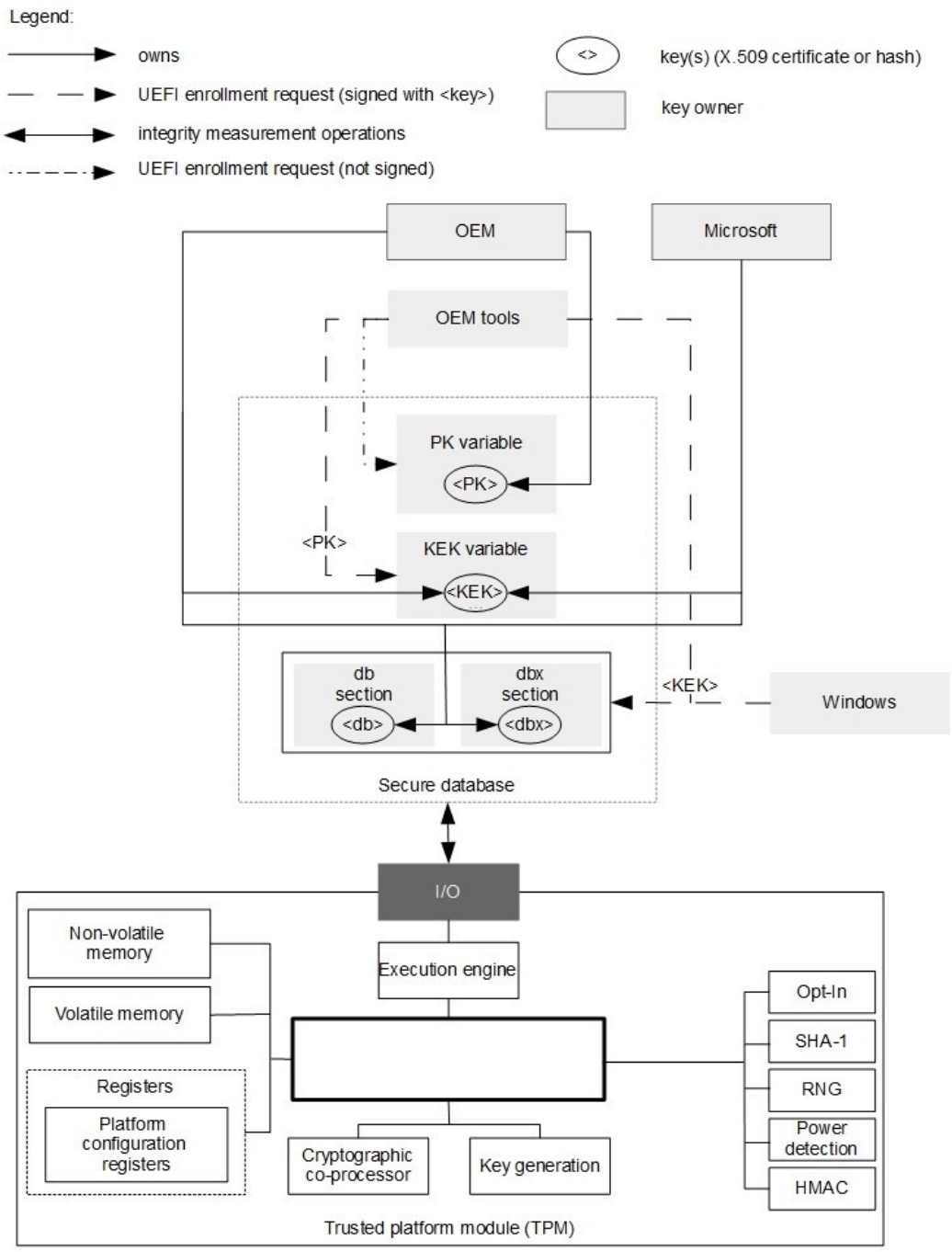


Figure 20: The architecture of TPM and UEFI "Secure Boot"

Secure Boot requires a computer that meets the UEFI specification of version 2.3.1, Errata C or higher.[8] UEFI, a successor to the basic input/output system (BIOS) firmware interface, is a specification that defines the

8 https://docs.microsoft.com/en-us/windows-hardware/manufacture/desktop/secure-boot-overview [Retrieved: 7/5/2017]; the UEFI specifications are available at: http://www.uefi.org/specifications [Retrieved: 7/5/2017]

software interface between the operating system and the firmware installed on a given computer. Among other things, UEFI defines a system boot manager responsible for enforcing system boot settings and starting the operating system loader, which initializes the operating system installed on the computer.

UEFI stores relevant system booting variables in the computer's non-volatile RAM (NVRAM). Among these variables are X.509 certificates and encryption hashes that establish the trust relationship between the computer's manufacturer and the operating system vendor, which is the essence of the Secure Boot standard. UEFI maintains a database of certificates and encryption hashes, referred to as *secure database*. This database is structured into the following sections implemented as UEFI *variables* ([UEFIF 2017], Section 31):

- platform key (*PK*): The PK section of the secure database stores the platform key, a X.509 certificate owned by the computer's manufacturer. This key is enrolled (i.e., stored in the UEFI secure database) at manufacture time (*OEM tools* in Figure 20) and is used to attest to the integrity of the computer's hardware components. Figure 21 depicts the PK we extracted using the `chipsec` toolset from the secure database of the platform that is subject of analysis (see Section 1), where *Lenovo Ltd.* is the OEM.

Figure 21: A PK owned by Lenovo Ltd.

- key enrollment key (*KEK*): The KEK section of the secure database stores multiple key enrollment keys (KEKs). Typically, at least two keys are stored in this section: i) a X.509 certificate owned by the OEM; and ii) a X.509 certificate owned by the vendor of the operating system installed on the computer (e.g., Microsoft). KEKs may be enrolled at manufacture time and/or when firmware update is performed. Certificates can be stored in the KEK section of the secure database only through UEFI key enrollment requests signed with the PK (<*PK*> in Figure 20); that is, only the OEM can populate this section. Through the OEM enrolling the KEK owned by the vendor of the installed operating system, a trust relationship between the OEM and this vendor is established. Figure 22 depicts the KEKs we extracted using the `chipsec` toolset from the secure database of the platform that is subject of analysis (see Section 1), where *Lenovo Ltd.* is the OEM and *Microsoft Corporation* is the operating system vendor.
- allowed database (*db*): The db section of the secure database stores multiple X.509 certificate and/or encryption hashes used to attest to the trustworthiness and/or integrity of operating system booting software and UEFI software (e.g., UEFI drivers), such as the Windows operating system loader (e.g., the `%SystemRoot\System32\winload.exe` and `%SystemRoot\System32\winload.efi` executables). Certificates or hashes may be enrolled in the db section of the secure database at manufacture time, when firmware update is performed, or by the Windows operating system, for example, when being installed. Certificates can be stored in this section only through UEFI key enrollment requests signed with a valid KEK (<*KEK*> in Figure 20) – a KEK stored in the KEK section of the secure database; that is, typically, only the OEM and vendor of the installed operating system can populate this section. Through the OEM and the operating system vendor enrolling certificate and hashes guaranteeing the trustworthiness and integrity of operating system booting and UEFI software, a trust relationship between the OEM, the operating system vendor, and this software is established.

  According to the UEFI specification ([UEFIF 2017], Section 31), the operating system won't boot if invalid or no certificates and/or encryption hashes are stored in the db section of the secure database. This is

because such a scenario indicates the use of not trusted, or corrupted, operating system booting and/or UEFI software. This is the essence of the Secure Boot standard.

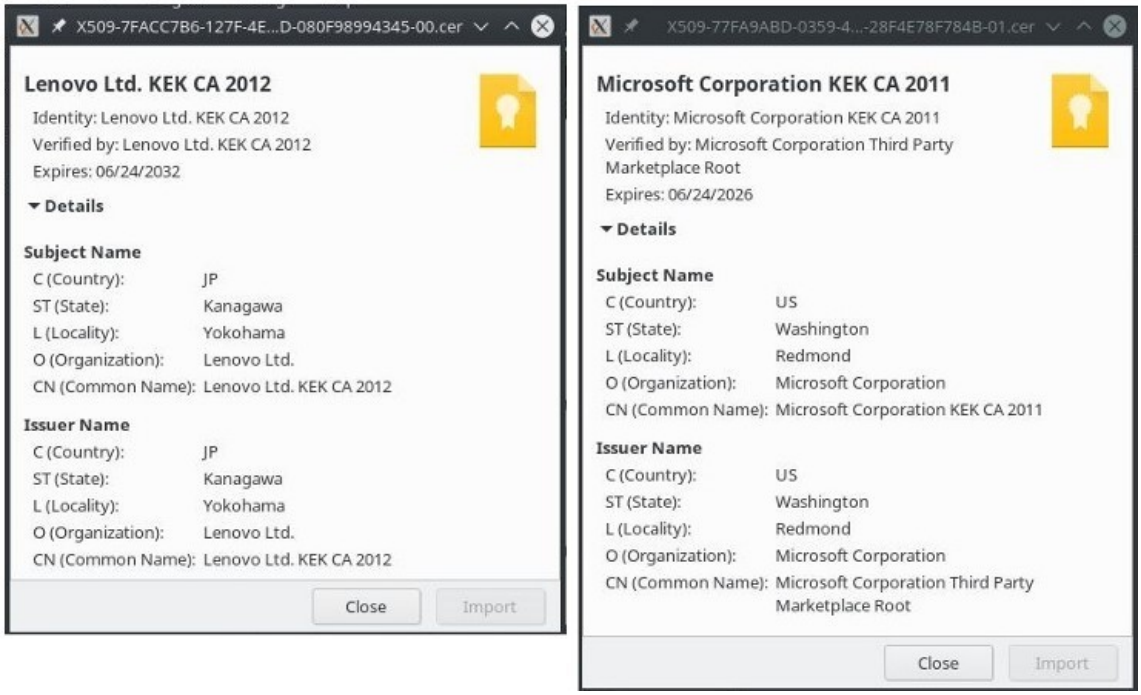
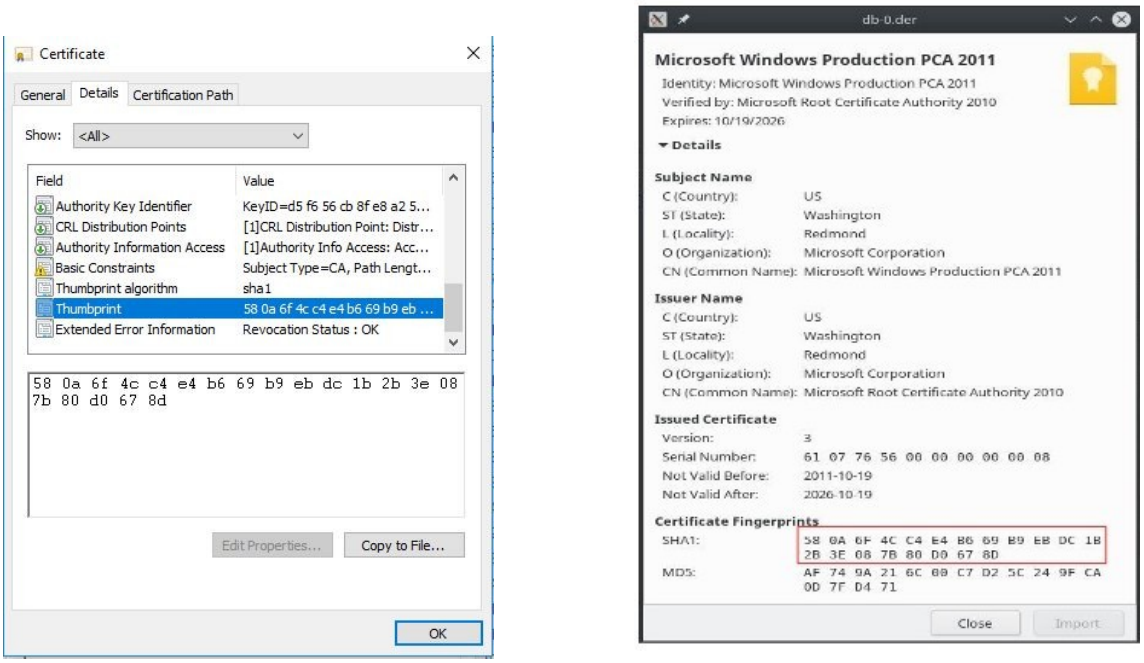


Figure 22: KEKs owned by Lenovo Ltd. and Microsoft Corporation

Figure 23a depicts the certificate with which the Windows system loader (executable: `%SystemRoot\System32\winload.exe`) is digitally signed. Figure 23b depicts a certificate owned by the Windows vendor (i.e., *Microsoft Corporation*), which we extracted using the `chipsec` toolset from the secure database of the platform that is subject of analysis (see Section 1). In Figure 23a and Figure 23b can be observed that the SHA-1 fingerprints of both certificates are equal, which, according to the Secure Boot standard ([UEFIF 2017], Section 31), is a condition for the Windows system loader to be started by the platform firmware. This is an example of the implementation of the trust relationships and integrity guarantees that have to be present for the Windows operating system to be initialized in accordance with the Secure Boot standard.

- disallowed database (*dbx*): The dbx section of the secure database, a counterpart to the db section of this database, may store multiple X.509 certificates and/or encryption hashes that are not longer valid (i.e., that have been revoked). According to the Secure Boot standard ([UEFIF 2017], Section 31), operating system booting software and/or UEFI software (e.g., UEFI drivers) that are signed with certificates, or their contents match with hashes, stored in the dbx section of the secure database, cannot be executed. This prevents the execution of booting software and/or UEFI software that is no longer trusted or is considered compromised.



a) *b)*

Figure 23: X.509 certificate: a) of the Windows system loader; b) stored in the db section of the secure database

Figure 24 depicts the sections of the secure database discussed above, which we extracted from the firmware of the platform that is subject of analysis (see Section 1) using the `UEFITool` toolset.

```
SioPolicy                EVSA entry    Name
DtsProtocolSetupVar      EVSA entry    Name
IccString                EVSA entry    Name
db                       EVSA entry    Name
dbx                      EVSA entry    Name
KEK                      EVSA entry    Name
PK                       EVSA entry    Name
```

Figure 24: The PK, KEK, db, and dbx sections of the secure database

As mentioned earlier, the TPM is a standard, managed by the Trusted Computing Group (TCG)[9] for a secure cryptoprocessor primarily used for secure storage of encryption keys and measuring the integrity of critical system software. This includes measurement of the integrity of critical UEFI variables, such as the sections of the secure database discussed above. Through enabling the measurement of the integrity of these variables, which, in turn, enables the detection of unauthorized modifications of the secure database, the TPM may be used as a part of a mechanism for securing the Secure Boot procedure itself. As defined in the TPM specification, version 1.2, revision 116 ([TCGM 2011], Section 4), which is the latest TPM specification at the time of writing, a TPM consists of the following major components:

- *I/O:*The I/O component manages the flow of data (e.g., commands) over the communication bus attached to the TPM hardware (e.g., it redirects data to other TPM components).
- *non-volatile memory*: The non-volatile memory is a memory region with a limited size, primarily used for storing long-term keys ([Tomlinson 2008], Chapter 7). An example is the storage root key (SRK), which is the root key of the hierarchy of keys used for secure (i.e., encrypted) data storage.

9 https://trustedcomputinggroup.org/ [Retrieved: 7/5/2017]

- *volatile memory*: The volatile memory is a memory region used as the primary working memory of the TPM. For example, the `physicalPresenceV` flag is stored in this region. This flag implements access control over critical TPM operations requiring the direct interaction with the TPM owner (i.e., a person with TPM administrative rights) for these operations to be performed ([TCGM 2011], Section 4).
- *execution engine*: The execution engine responds to commands directed to the TPM and executes the appropriate code stored in the TPM. Since it is permanently stored at the TPM and is tamper-proof, this code, commonly known as the core root of trust for measurement (CRTM), is considered trustworthy.
- *registers*: As typical with processors, the memory available to it (i.e., the previously mentioned volatile and non-volatile memory) is structured into multiple registers. Examples are the platform configuration registers (PCRs), which, among other things, are used as part of the previously discussed Secure Boot procedure.

  According to the TPM EFI platform specification, version 1.22, revision 15 ([TCGE 2014], Section 6.4) (this is the latest TPM EFI specification at the time of writing), PCR 7 (i.e., the 7-th PCR register, out of 16) is used for storing integrity *measurement* data (i.e., hash values) of the UEFI variables PK and KEK. Therefore, the values stored in PCR 7 can be used for detecting unauthorized modifications of the PK and KEK sections of the secure database (see Figure 20).
- *cryptographic co-processor*: The cryptographic co-processor specializes in performing encryption-related operations, such as those performed by its core engines:
  - *RNG*: This engine generates random numbers needed for encryption operations (e.g., key generation);
  - *SHA-1* and *HMAC*: These engines calculate Secure Hash Algorithm (SHA)-1 and keyed-hash message authentication code (HMAC) hashes, respectively, for example, when integrity measurement is performed;
  - *key generation*: This engine creates Rivest, Shamir and Adleman (RSA) key pairs.
- *Power detection*: The core functionality of the power detection TPM component is to maintain the power state of the TPM based on notifications about the power state of the platform.
- *Opt-in*: The opt-in TPM component implements mechanisms for securing critical TPM operations, such as turning the TPM off. Among other things, it maintains the state of the relevant flags stored in the TPM memory, such as the previously mentioned `physicalPresenceV` flag.

We refer to the official TPM specification [TCGM 2011] for more details on the TPM.

3.6 Universal Windows Platform

UWP is a platform architecture (i.e., an application execution environment) developed by Microsoft and first introduced in Windows 10. UWP provides an execution environment for user applications (see Section 2.1), named Universal Windows Apps (UWAs), specifically designed to operate as part of the UWP. The UWP architecture provides an environment for the execution of UWAs on heterogeneous Windows platforms, such as computers, smartphones, X-Box devices, tablets, and so on. In addition, UWP supports the execution of UWAs developed in a variety of languages, such as C++, C#, and VB.NET.

UWAs are distributed to users primarily through the Windows Store, anR on-line library of UWAs maintained by Microsoft. For an UWA to be available for download from the Windows Store, it has to be submitted (i.e., uploaded) to the Store, after which it is subjected to a certification process. This process, which includes testing the security and the reliability of the application's code,[10] is for the identification of potential security, reliability, and performance issues before the UWA is made available to Windows users.

Next, we discuss the architecture of the UWP deployed in the Windows 10 operating system that is subject of analysis (see Section 1). Figure 25 provides a compact overview of this architecture.

10 https://docs.microsoft.com/en-us/windows/uwp/publish/the-app-certification-process [Retrieved: 7/5/2017]

The core of the UWP is the *Windows Runtime (RT) API* , based on which UWAs are developed and operate. The Windows RT API is based on COM, which makes the functionalities of this API language-agnostic ([Richter 2013], Chapter 1); that is, Windows RT API-based applications, and therefore UWAs, can be developed in a variety of languages. The Windows RT API is built on top of the Windows API (*Win32 API* in Figure 25) implemented in the subsystem DLLs and the `ntdll.dll` library file (see Section 2.1). Functions declared as part of the RT API use and extend functionalities of the Windows system by invoking functions declared as part of the Windows API. UWAs may also invoke functions declared as part of the Windows API, for example, for performance gains.

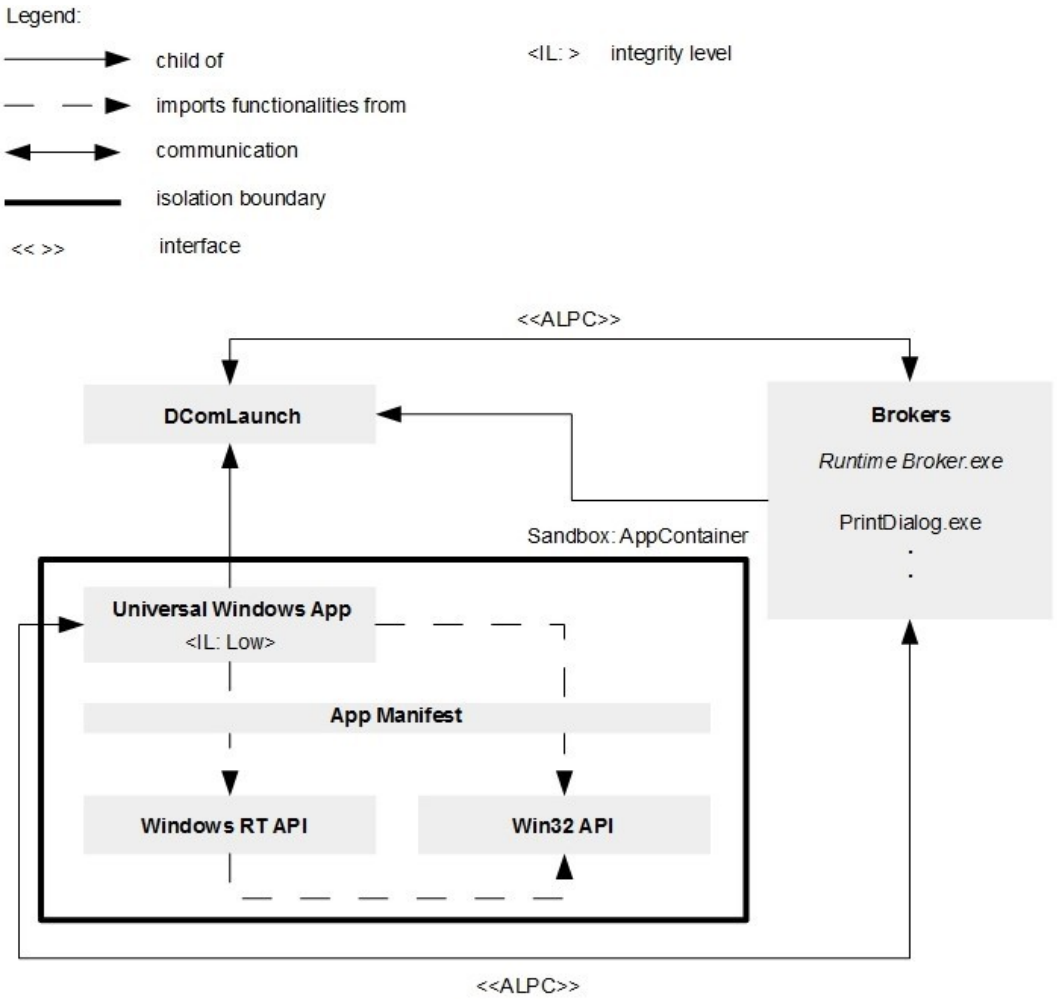


Figure 25: The architecture of UWP

We emphasize that UWAs are restricted in terms of what Windows functionalities, implemented as part of the Windows RT and Windows APIs, are available to them. UWAs run within a restricted, containerized, sandbox environment, named AppContainer (*Sandbox: AppContainer* in Figure 25), for security reasons. The restrictions, to which UWAs are subjected, are enforced by two separate yet complimentary security mechanisms:

- Windows mandatory integrity control (MIC): MIC is a Windows security mechanism for controlling access to objects (e.g., files, folders, and process memory spaces) by processes based on a level of trustworthiness associated to processes. MIC distinguishes between five integrity levels – untrusted, low, medium, high, and system - with values of 0x[0-4]0, respectively. A lower value indicates a lower level of trustworthiness.[11] Windows assigns the integrity level of low (*<IL: Low>* in Figure 25) to UWAs. This

11 https://msdn.microsoft.com/en-us/library/bb625963.aspx [Retrieved: 7/5/2017]

significantly restricts the functionalities (e.g., reading, or writing to, specific files, access to network interfaces and certain devices) available to them.

- UWA capabilities: On top of the restrictions enforced by MIC, UWP enforces additional object access restrictions through the use of capabilities. Capabilities refer to Windows functionalities that are subjects of restrictions, that is, which are not available to a given UWA, unless a capability representing these functionalities is explicitly associated with the UWA. Example capabilities are 'removableStorage', a capability enabling UWAs to access files stored on removable storage devices, and 'internetClient', a capability enabling UWAs to receive data from the Internet.[12] Some capabilities, such as those we previously mentioned, are defined by Microsoft. However, UWA developers may define their own capabilities based on the existing ones. These capabilities are subject of analysis as part of the certification process once an UWA is submitted to Windows Store ([Yosif 2017], Chapter 7).

  The capabilities associated with a given UWA are specified in an XML-like format as part of the application's manifest file (*App Manifest* in Figure 25). Figure 26 depicts an excerpt of the manifest file of the Photos UWA, which is distributed with Windows 10, where the capabilities associated to it are defined. The manifest file of a given UWA is distributed, along with the application's executable and other auxiliary files, as part of the UWA's application package. UWAs application packages are stand-alone container files (typically with the `.appx` extension) representing the distribution format of UWAs. For example, UWAs are uploaded to the Windows Store in the form of application packages.

```
<Capabilities>
  <rescap:Capability Name="enterpriseDataPolicy" />
  <wincap:Capability Name="storeAppInstall" />
  <uap:Capability Name="removableStorage" />
  <uap:Capability Name="videosLibrary" />
  <uap:Capability Name="picturesLibrary" />
  <uap:Capability Name="musicLibrary" />
  <uap:Capability Name="contacts" />
  <rescap:Capability Name="contactsSystem" />
  <Capability Name="internetClientServer" />
  <Capability Name="privateNetworkClientServer" />
  <DeviceCapability Name="webcam" />
  <DeviceCapability Name="microphone" />
</Capabilities>
```

Figure 26: An excerpt of the manifest file of the Photos UWA

When an UWA is started, the UWP reads the capabilities defined in the UWA's manifest file and represents the capabilities in the form of security identifiers (SIDs). The SIDs are then placed in the access token assigned to the UWA. The core Windows access control mechanism functions based on access tokens and SIDs. Based on the UWA's access token and the SIDs placed within, the Windows system allows or denies access to the UWA to protected Windows functionalities (e.g., writing to files, and access to network interfaces). In addition to the capabilities represented in the form of SIDs, UWAs access tokens contain the integrity levels assigned to UWAs (i.e., low), which we previously discussed.

Figure 27a depicts the access token of the Photos UWA. This token is represented as a structure in the Windows kernel, which we observed using the `windbg` debugger. The variable `IntegrityLevelIndex` stores the value of the integrity level of low (i.e., `0x10`). The variable `CapabilitiesHash` is an array storing the capability-derived SIDs associated with the Photos UWA. Figure 27b depicts the capabilities (i.e., SIDs) and the integrity level associated with the Photos UWA, which we listed using the `Process Explorer` utility (part of the `Sysinternals` suite).

Since the AppContainer sandbox environment imposes strict restrictions on UWAs, some common useful functionalities are not available to UWAs. Examples are printing and file-system browsing. As a solution to this issue, Windows 10 allows for extending the capabilities of UWAs with specific extensions

12 A comprehensive list of UWA capabilities is available at: https://msdn.microsoft.com/en-us/library/windows/apps/hh464936.aspx [Retrieved: 7/5/2017]

defined in an application's manifest file.[13] These extensions are implemented by *brokers,* with which UWAs communicate through IPC channels.

```
1  lkd> dt nt!_TOKEN ffffb706d2797060
2  [...]
3     +0x098 UserAndGroups     : 0xffffb706`d27974e8 _SID_AND_ATTRIBUTES
4     +0x0a0 RestrictedSids    : (null)
5     +0x0a8 PrimaryGroup      : 0xffffb706`ca0a6f50 Void
6     +0x0b0 DynamicPart       : 0xffffb706`ca0a6f50  -> 0x501
7     +0x0b8 DefaultDacl       : 0xffffb706`ca0a6f6c _ACL
8     +0x0c0 TokenType         : 1 ( TokenPrimary )
9     +0x0c4 ImpersonationLevel : 0 ( SecurityAnonymous )
10    +0x0c8 TokenFlags        : 0x4a00
11    +0x0cc TokenInUse        : 0x1 ''
12    +0x0d0 IntegrityLevelIndex : 0x10
13    +0x0d4 MandatoryPolicy   : 1
14    +0x0d8 LogonSession      : 0xffffb706`c5cdb870 _SEP_LOGON_SESSION_REFERENCES
15    +0x0e0 OriginatingLogonSession : _LUID
16    +0x0e8 SidHash           : _SID_AND_ATTRIBUTES_HASH
17    +0x1f8 RestrictedSidHash : _SID_AND_ATTRIBUTES_HASH
18    +0x308 pSecurityAttributes : 0xffffb706`c91b5140 _AUTHZBASEP_SECURITY_ATTRIBUTES_INFORMATION
19    +0x310 Package           : 0xffffb706`c9e519d0 Void
20    +0x318 Capabilities      : 0xffffb706`caaea3a0 _SID_AND_ATTRIBUTES
21    +0x320 CapabilityCount   : 0xe
22    +0x328 CapabilitiesHash  : _SID_AND_ATTRIBUTES_HASH
23    +0x438 LowboxNumberEntry : 0xffffb706`c5f22830 _SEP_LOWBOX_NUMBER_ENTRY
24    +0x440 LowboxHandlesEntry : 0xffffb706`ca47c0e0 _SEP_LOWBOX_HANDLES_ENTRY
25    +0x448 pClaimAttributes : (null)
26    +0x450 TrustLevelSid     : (null)
27    +0x458 TrustLinkedToken  : (null)
28    +0x460 IntegrityLevelSidValue : (null)
29 [...]
```

a)

S-1-15-2-2226957697-3030467180-2301525-4248967783-2024719031-2325529081-2915787518	AppContainer
APPLICATION PACKAGE AUTHORITY\Removable storage	Capability
APPLICATION PACKAGE AUTHORITY\Your Contacts	Capability
APPLICATION PACKAGE AUTHORITY\Your home or work networks	Capability
APPLICATION PACKAGE AUTHORITY\Your Internet connection, including incoming connections from the Internet	Capability
APPLICATION PACKAGE AUTHORITY\Your music library	Capability
APPLICATION PACKAGE AUTHORITY\Your pictures library	Capability
APPLICATION PACKAGE AUTHORITY\Your videos library	Capability
S-1-15-3-1024-2897291008-3029319760-3330334796-465641623-3782203132-742823505-3649274736-3650177846	Capability
S-1-15-3-1024-373139346-748750918-1948434659-2643498477-4072104851-1007166015-1979446734-3878125657	Capability
S-1-15-3-1024-4267310653-3012624349-32869343-335676702-674013981-1531007892-2777328540-762217067	Capability
S-1-15-3-2105443330-1210154068-4021178019-2481794518	Capability
S-1-15-3-2226957697-3030467180-2301525-4248967783-2024719031-2325529081-2915787518	Capability
S-1-15-3-3845273463-1331427702-1186551195-1148109977	Capability
S-1-15-3-787448254-1207972858-3558633622-1059886964	Capability
BUILTIN\Administrators	Deny
NT AUTHORITY\Local account and member of Administrators group	Deny
Mandatory Label\Low Mandatory Level	Integrity

b)

Figure 27: The Photos UWA: a) the access token structure; b) associated capabilities and integrity level

Brokers are processes that perform specific tasks and that have a higher integrity level assigned to them (i.e., medium) than that assigned to UWAs (i.e., low). For example, the PrintDialog broker, implemented in the `PrintDialog.exe` executable file, is responsible for printing tasks. The operation of the individual brokers is managed by the Runtime Broker (executable: `%SystemRoot\System32\RuntimeBroker.exe`). For example, the Runtime Broker triggers the creation and shutting down of the

13 For more detail, see https://msdn.microsoft.com/en-us/library/windows/apps/hh464906 [Retrieved: 7/5/2017]

brokers when requested by UWAs through ALPC-based IPC channels (<<*ALPC*>> in Figure 25) ([Yosif 2017], Chapter 7).

When a user starts a given UWA, the *DcomLaunch* Windows service creates the process hosting the UWA. It also creates the process hosting the Runtime Broker for handling potential requests for broker functionalities from the UWA. When such a request is received, the Runtime Broker relays the request to the *DcomLaunch* service through an ALPC-based communication channel ([Yosif 2017], Chapter 7), which then creates the process hosting the broker. Figure 28 depicts how the Photos UWA, the Runtime Broker, and the PrintDialog run in Windows 10, which we viewed using the `Process Explorer` utility (part of the `Sysinternals` suite).

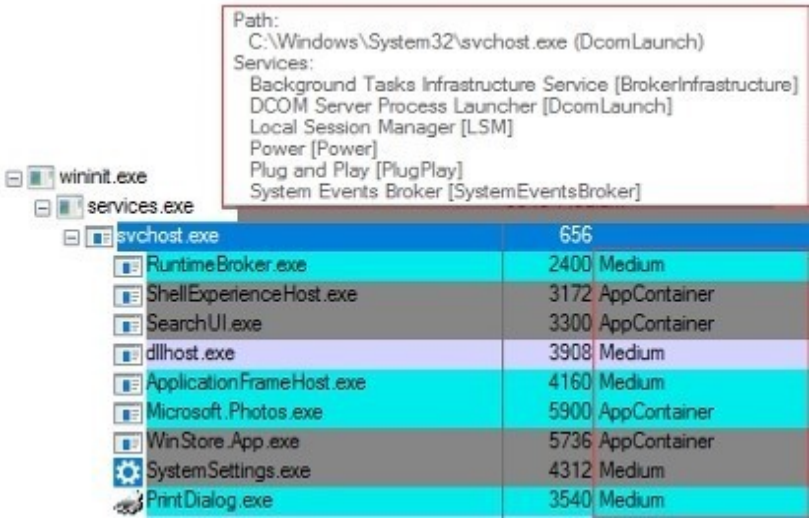


Figure 28: Deployment of the Photos UWA, the Runtime Broker, and the PrintDialog broker

3.7 Other Components

The focus of this section is the identification of Windows 10 components that have not been designated for analysis as part of this project, however, that fulfill the criterion C2) defined in Section 2.3. Next, we present the technical approach for identifying these components.

In order to identify components that have not been designated for analysis as part of this project and fulfill the criterion C2) (see Section 2.3), we take the following approach: i) we consult relevant available documentation (e.g., the Windows Internals book [Yosif 2017] and official documentation provided by Microsoft), and ii) we compare the Windows 10 system with its predecessors and observe major relevant differences between the two. We compare the Windows 10 and the Windows 7 operating systems in order to identify security-critical *Windows 10-specific components (i.e.,* components present only in Windows 10 and not in Windows 7, or components present in Windows 7 and in Windows 10, however, majorly modified). We compare Windows 10 with Windows 7 because Windows 10 represents the first major revision of the Windows operating systems since the previous generation of Windows systems, of which Windows 7 is a representative. The Windows 8 system, which was released between the Windows 7 and Windows 10, is considered an intermediary system having features and functionalities similar to those of both Windows 7 and Windows 10. The compared operating systems were fully updated and back-up snapshots were taken before the comparison took place.

In Table 1, detailed technical information on the compared operating systems is presented, that is:

- 'Release': the name, release, and version of the operating system;
- 'Update': the date and time at which we fully updated the system;
- 'Snapshot': the date and time at which we backed-up the system; and

- 'SHA-1 hash': a SHA-1 hash of the system installation files, which is relevant for verifying the system's authenticity.

The exact release of the Windows 10 operating system used for the comparison presented in this section is a requirement set by the German Federal Office for Information Security (see Section 1). We use the Enterprise edition of the Windows 7 system for this comparison since this edition is shipped with a full set of features. The Windows 7 and Windows 10 operating systems presented in Table 1 were downloaded from their official download locations at https://msdn.microsoft.com [Retrieved: 7/5/2017]. We performed default installations, that is, we used the options 'Express settings' when installing the Windows 10 system, and 'Apply recommended settings' when installing the Windows 7 system.

	Windows 10	**Windows 7**
Release	Windows 10, build 1607, 64bit, LTSB, German language	Windows 7, 64bit, Enterprise edition with Service Pack 1, German language
Update	12.05.2017, 13:21 CET	12.05.2017, 17:09 CET
Snapshot	12.05.2017, 14:47 CET	12.05.2017, 18:00 CET
SHA-1 Hash	013D675AE09148F42A9FFE3D3BC53FDD861DB7A5	08B453546E193A65D13D436E3C37CED7256E1988

Table 1: Comparison between Windows 7 and Windows 10: Technical Information

All parts of the Windows operating system discussed in Section 2.1 were considered as part of this comparison. We applied a top-down approach comparing separately each of the different parts of the Windows system (i.e., system support processes, services, the Windows subsystem, `ntdll.dll`, drivers, and HAL) in order to identify relevant (i.e., security-critical, Windows 10-specific) components (see criterion C2, Section 2.3). We emphasize that the final decision on what components are considered security-critical is based on the expertise of the authors.

Next, the results of the comparison are discussed. The discussion is structured with respect to the previously mentioned parts of the Windows operating system.

3.7.1 System Support Processes

The system support processes running at the operating systems in focus (see Table 1) were compared using the `Process Explorer` utility (part of the `Sysinternals` suite). We identified the following major differences:

- The Local Session Manager (see Section 2.1) is implemented as a system support process at Windows 7 (executable: `lsm.exe`). At Windows 10, it is implemented as a service running inside a service hosting process for performance reasons ([Yosif 2017], Chapter 2).
- The console host (executable: `conhost.exe`) is a process running as a child process of the core executable of the Windows subsystem (executable: `csrss.exe`) at Windows 7. At Windows 10, it runs as a child process of instances of the command line executable (executable: `cmd.exe`) for performance reasons ([Yosif 2017], Chapter 2).
- The System process in Windows 10 features new child processes, that is, 'Memory Compression' (a mechanism for optimizing memory usage) and 'Interrupts' (interrupt routine handlers and scheduling) ([Yosif 2017], Chapter 2).

None of the above differences between Windows 10 and Windows 7 indicates components that fulfill the criterion C2 defined in Section 2.3 and that warrant detailed further analysis.

We emphasize that when VSM is enabled, the Windows 10 operating system features a new process hosting a kernel thread, named 'Secure System', created by the secure kernel, and an additional, secure part of the local security authority (executable: `LsaIso.exe, see Section 2.2`). These novelties are already designated for analysis as part of Work Package 6 of this project.

3.7.2 Services

The services running at the operating systems in focus (see Table 1) were compared based on identifying new services present only in the Windows 10 operating system as well as on comparing the functionalities of the services that are present in both Windows 7 and Windows 10. We performed the former by identifying service names present only in Windows 10 using a PowerShell script developed by ERNW GmbH, with which we extracted the names of the services present in the compared operating systems. We performed the latter by analyzing the imported libraries from the services that are present both in Windows 7 and Windows 10 using the `dumpbin` utility in an effort to identify major and relevant differences in functionalities.

In the Appendix, section 'List of Services', the names of all services (Service Name) present in Windows 10 are listed. The services that are present only in Windows 10 (i.e., that are Windows 10-specific) are marked in bold.

In Table 2, we present the names of the services that we identified as part of components that fulfill the criterion C2) defined in Section 2.3 and that warrant a detailed analysis. In Table 2, we also present the names of these components as well as a brief description of their core functionalities. The component names presented in Table 2 are either official names as given by Microsoft, or names as the components in focus are commonly referred to. An exception is the name 'Unknown services', which refers to a set of unknown and undocumented services (as of 6th July, 2017) whose functionalities are not yet clear.

Service Name	Component	Description
AppVClient	App-V	A virtualization environment allowing for delivering Windows applications to users as virtualized applications.
CDPSvc AJRouter embeddedmode	Windows IoT	A framework for the handling and management of internet-of-things (IoT), peripheral devices interacting with a Windows system.
DcpSvc	Data Collection and Publishing	A service allowing for applications to upload data to the Cloud.
diagnosticshub. standardcollector.service	DiagnosticsHub	A functionality of the Windows system collecting and handling log files, which may be uploaded to remote sites.
DmEnrollmentSvc	Mobile Device Management	An environment for administering mobile devices, such as smartphones, tablet computers, laptops and desktop computers.
DsmSvc	Driver management	A service detecting, downloading, and installing device-related software.
fhsvc	Windows File History	A file back-up mechanism.
lfsvc	Geolocation	A service monitoring the current location of the system and managing geofences.
NgcCtnrSvc	Microsoft Passport	An authentication mechanism.

Service Name	Component	Description
NgcSvc		
PhoneSvc	Microsoft Telephony	An environment allowing for the integration of Windows systems with communication devices and networks (e.g.,multimedia conferencing).
PimIndexMaintenanceSvc_409eb MessagingService_409eb UnistoreSvc_409eb UserManager WpnUserService_409eb DsSvc UserDataSvc_409eb workfolderssvc CDPUserSvc_409eb OneSyncSvc_409eb	Unknown services	Unknown, undocumented services (as of 6th July, 2017) introduced in the Anniversary Update of the Windows 10 operating system.
Sense WdNisSvc	Windows Defender	A Windows security mechanism monitoring a system's activities in order to detect viruses, malwares, and malicious activities.
SensorDataService	Windows Sensor	A framework for the delivery and handling of data from sensors (e.g., cameras, peripheral devices).
shpamsvc	Windows SharedPC	A mode of the Windows 10 operating system allowing for multiple user accesses.
UsoSvc DoSvc	Windows Update	A service delivering and installing updates to Windows systems.
WalletService	Microsoft Wallet	A mechanism for on-line payment.
wlidsvc	Microsoft Account	A mechanism for logging into Windows systems.

Table 2: Identified relevant services and components

3.7.3 Drivers

The drivers deployed at the operating systems in focus (see Table 1) were compared based on identifying new drivers present only in the Windows 10 operating system. We identified new drivers by comparing the outputs of the Windows built-in `driverquery` utility, which we ran at the Windows 7 and Windows 10 systems. This utility provides a comprehensive list of deployed drivers.

In the Appendix, section 'List of Drivers', the names of all drivers deployed in Windows 10 are listed. The drivers that are present only in Windows 10 (i.e., that are Windows 10-specific) are marked in bold.

We identified the component 'Microsoft Windows Application Compatibility Infrastructure', partially implemented through the driver named 'ahcache', as a component that may be considered within the scope of this work package and project. This component provides a legacy execution environment for backward compatibility. To this end, the Application Compatibility Infrastructure provides an unique program execution workflow in whose security significant efforts have been invested,[14] which makes it a worthy target for a detailed investigation. The Application Compatibility Infrastructure is not a Windows 10-specific

14 https://technet.microsoft.com/en-us/library/dd837644(v=ws.10).aspx [Retrieved: 7/5/2017]

component, however, its functionalities and architecture have been significantly extended and modified, as demonstrated by the presence of the Windows 10-specific driver 'ahcache'.

3.7.4 Windows Subsystem, ntdll.dll, Windows Kernel, and HAL

In this section, we discuss components identified by analyzing the parts of the Windows system, Windows subsystem, `ntdll.dll`, the Windows kernel, and HAL, due to the similar functional properties and integrated workflows of these system parts. We compared the functionalities of these parts of the Windows 7 and Windows 10 systems in focus (see Table 1) using the tools Windows debugger (`windbg`) and `Dependency Walker`, as well as tools from the `Sysinternals` suite. We identified the following components as components that that fulfill the criterion C2) defined in Section 2.3 and that may be considered within the scope of this work package and project:

PatchGuard: PatchGuard is a crucial kernel security mechanism, which prevents kernel manipulations ([Yosif 2017], Chapter 7). PatchGuard exists since the release of Windows XP, however, it has been heavily modified with addded functionalities (e.g., active kernel monitoring), which makes it a worthy target for a detailed analysis as part of this project.

PatchGuard is implemented in the Windows kernel and its code is obfuscated, that is, the Windows kernel does not export functions and/or variables with descriptive names indicating PatchGuard functionalities. Therefore, the identification of the presence of PatchGuard by analyzing the Windows kernel (executable: `ntoskrnl.exe`) is very challenging. However, PatchGuard triggers a system crash associated with a specific system error code (i.e., `CRITICAL_STRUCTURE_CORRUPTION`, numerical value: 109) when an activity violates its policies. This enabled us to verify the presence of PatchGuard by debugging the Windows 10 kernel using the `windbg` debugger and displaying reference information on the implemented kernel bug check (i.e., a routine handling system crashes) for this code. In Figure 29, the output of the `windbg` debugger is presented.

```
kd> !analyze -show 109
CRITICAL_STRUCTURE_CORRUPTION (109)
This bugcheck is generated when the kernel detects that critical kernel code or
data have been corrupted. There are generally three causes for a corruption:
1) A driver has inadvertently or deliberately modified critical kernel code
 or data. See http://www.microsoft.com/whdc/driver/kernel/64bitPatching.mspx
2) A developer attempted to set a normal kernel breakpoint using a kernel
 debugger that was not attached when the system was booted. Normal breakpoints,
 "bp", can only be set if the debugger is attached at boot time. Hardware
 breakpoints, "ba", can be set at any time.
3) A hardware corruption occurred, e.g. failing RAM holding kernel code or data.
Arguments:
Arg1: 0000000000000000, Reserved
Arg2: 0000000000000000, Reserved
Arg3: 0000000000000000, Failure type dependent information
Arg4: 0000000000000000, Type of corrupted region, can be
      0   : A generic data region
      1   : Modification of a function or .pdata
      2   : A processor IDT
      3   : A processor GDT
      4   : Type 1 process list corruption
      5   : Type 2 process list corruption
      6   : Debug routine modification
      7   : Critical MSR modification
      8   : Object type
      9   : A processor IVT
      a   : Modification of a system service function
      b   : A generic session data region
      c   : Modification of a session function or .pdata
      d   : Modification of an import table
```

Figure 29: PatchGuard bug check code reference

HyperGuard: HyperGuard is a novel kernel security mechanism introduced in the Anniversary Update of Windows 10 ([Yosif 2017], Chapter 7). Its functionalities are simiar to, and extend, those of PatchGuard, with the main difference that HyperGuard is deployed in VTL 1 (i.e., in the SKM, see Section 2.2) making it isolated from potential malicious users operating at VTL 0. Being a novel and crucial system security mechanism, we consider HyperGuard a worthy target for a detailed analysis as part of this project.

```
kd> !analyze -show 18c
HYPERGUARD_VIOLATION (18c)
This bugcheck is generated when the kernel detects that critical kernel code or
data have been corrupted. There are generally three causes for a corruption:
1) A driver has inadvertently or deliberately modified critical kernel code
 or data. See http://www.microsoft.com/whdc/driver/kernel/64bitPatching.mspx
2) A developer attempted to set a normal kernel breakpoint using a kernel
 debugger that was not attached when the system was booted. Normal breakpoints,
 "bp", can only be set if the debugger is attached at boot time. Hardware
 breakpoints, "ba", can be set at any time.
3) A hardware corruption occurred, e.g. failing RAM holding kernel code or data.
Arguments:
Arg1: 0000000000000000, Type of corrupted region, can be
      1001 : A generic data region
      1002 : A page hash mismatch
      1004 : A processor IDT
      1005 : A processor GDT
      1007 : Debug routine modification
      1008 : A dynamic code region
      1009 : A generic shareable data region
      100a : A hypervisor overlay region
      100b : A processor mode misconfiguration
      100c : An extended processor control register
      100d : A secure memory region
      1102 : IDTR modification
      1103 : GDTR modification
Arg2: 0000000000000000, Failure type dependent information
Arg3: 0000000000000000, Reserved
Arg4: 0000000000000000, Reserved
```

Figure 30: HyperGuard bug check code reference

The presence of the HyperGuard component can be verified by debugging the Windows kernel and displaying the reference information on the implemented kernel bug check for the error code `HYPERGUARD_VIOLATION` (numerical value: 18c). HyperGuard triggers a system crash associated with this error code when an activity violates its policies. We verified the presence of HyperGuard by debugging the Windows 10 kernel using the `windbg` debugger and displaying reference information on the implemented kernel bug check for the code `HYPERGUARD_VIOLATION`. In Figure 30, the output of the `windbg` debugger is presented.

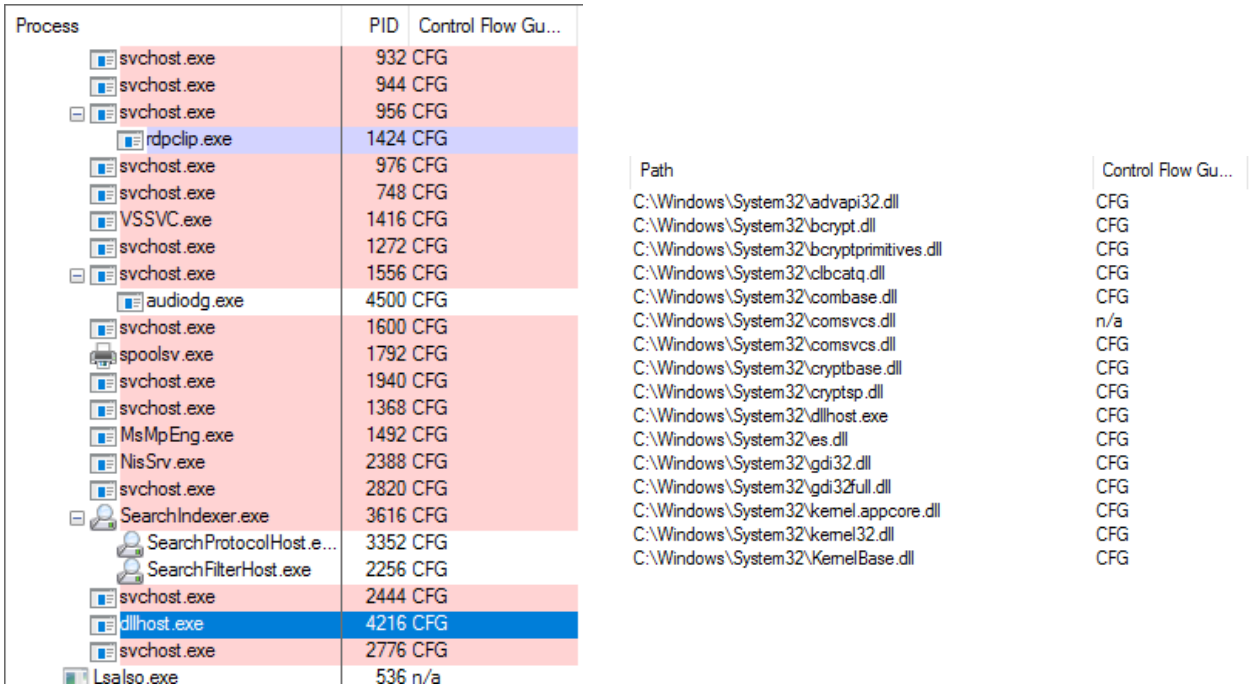
ControlFlowGuard:ControlFlowGuard (CFG) is an important Windows 10-specific security mechanism (i.e., an exploit prevention mechanism) enabling control over what functions can be invoked by a given executable or a function implemented in a given library (i.e., a DLL) file. CFG complements existing exploit prevention mechanisms, such as address space layout randomization (ASLR), making it crucial for the security of a given Windows system. Therefore, we consider CFG a worthy target for a detailed analysis as part of this project.

CFG is largely implemented as a set of functions in the `ntdll.dll` library file, which is an important part of the Windows system (see Section 2.1). Its presence can be verified by identifying the CFG-related functions exposed by this file. Figure 31 presents the output of the `dumpbin` utility, that is, the function 'RtlGuardCheckLongJumpTarget' exported by the `ntdll.dll` file (named '`ntdllwin1603.dll`' in Figure 31). This function is known as one of the core CFG functions, which is not present in the `ntdll.dll` file of the Windows 7 system.[15]

```
C:\Program Files (x86)\Microsoft Visual Studio 14.0>dumpbin /exports c:\ernw\ntdll.dll | find "Guard"
       1041  408 0004D250 RtlGuardCheckLongJumpTarget = RtlGuardCheckLongJumpTarget
```

Figure 31: The RtlGuardCheckLongJumpTarget CFG function

15 http://blog.trendmicro.com/trendlabs-security-intelligence/control-flow-guard-improvements-windows-10-anniversary-update/ [Retrieved: 7/5/2017]



Process	PID	Control Flow Gu...
svchost.exe	932	CFG
svchost.exe	944	CFG
svchost.exe	956	CFG
rdpclip.exe	1424	CFG
svchost.exe	976	CFG
svchost.exe	748	CFG
VSSVC.exe	1416	CFG
svchost.exe	1272	CFG
svchost.exe	1556	CFG
audiodg.exe	4500	CFG
svchost.exe	1600	CFG
spoolsv.exe	1792	CFG
svchost.exe	1940	CFG
svchost.exe	1368	CFG
MsMpEng.exe	1492	CFG
NisSrv.exe	2388	CFG
svchost.exe	2820	CFG
SearchIndexer.exe	3616	CFG
SearchProtocolHost.e...	3352	CFG
SearchFilterHost.exe	2256	CFG
svchost.exe	2444	CFG
dllhost.exe	4216	CFG
svchost.exe	2776	CFG
Lsalso.exe	536	n/a

Path	Control Flow Gu...
C:\Windows\System32\advapi32.dll	CFG
C:\Windows\System32\bcrypt.dll	CFG
C:\Windows\System32\bcryptprimitives.dll	CFG
C:\Windows\System32\clbcatq.dll	CFG
C:\Windows\System32\combase.dll	CFG
C:\Windows\System32\comsvcs.dll	n/a
C:\Windows\System32\comsvcs.dll	CFG
C:\Windows\System32\cryptbase.dll	CFG
C:\Windows\System32\cryptsp.dll	CFG
C:\Windows\System32\dllhost.exe	CFG
C:\Windows\System32\es.dll	CFG
C:\Windows\System32\gdi32.dll	CFG
C:\Windows\System32\gdi32full.dll	CFG
C:\Windows\System32\kernel.appcore.dll	CFG
C:\Windows\System32\kernel32.dll	CFG
C:\Windows\System32\KernelBase.dll	CFG

a) *b)*

Figure 32: The CFG flag associated with: a) executables; b) loaded library files

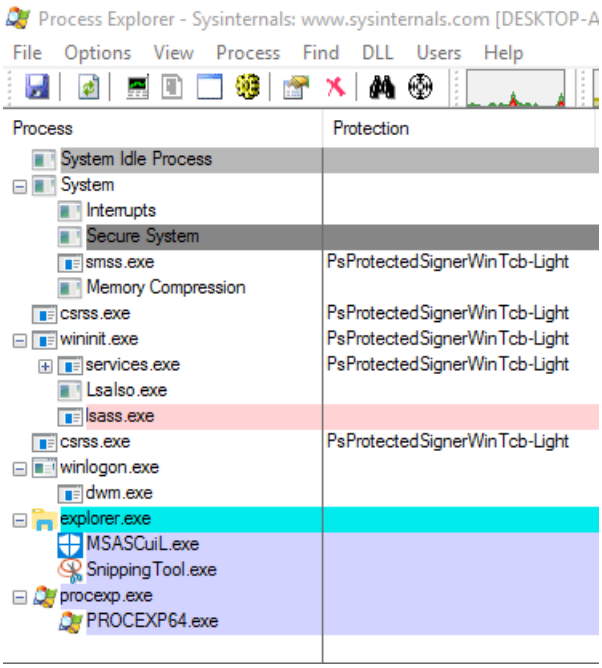
The presence of CFG can also be verified in user land by observing the CFG flag associated with applications and library files that are under CFG protection. The `Process Explorer` utility (part of the `Sysinternals` suite) displays the status of this flag for all running processes and loaded library files. Figure 32a and Figure 32b depict the output of the `Process Explorer` utility, where entries with the field 'CFG' indicate that CFG protection is enabled for the respective processes (Figure 32a) and loaded library files (Figure 32b). We emphasize that protection of loaded library files is an important feature of CFG. It mitigates a variety of attacks bypassing other exploitation prevention mechanisms, for example, ASLR.

Protected Process Light: Protected Process Light (PPL) is a significantly re-designed version of the Protected Process security mechanism implemented as part of multiple older versions of the Windows operating system.[16] PPL is an important security mechanism protecting the memory space of processes marked as protected from accesses by other untrusted processes. Therefore, we consider PPL a worthy target for a detailed analysis as part of this project.

PPL is implemented in the Windows kernel and its presence can be verified by examining relevant kernel objects. For example, the `EPROCESS` kernel structure, whose instances store information on each running process, features the field '`Protection`'. This field indicates whether a process is marked as protected as well as what rights does the process have for accessing other processes' memory spaces. We identified the presence of this field as part of the kernel of the Windows 10 system in focus (see Table 1) by debugging this kernel with the `windbg` debugger and printing out the declaration of the `EPROCESS` structure (see Figure 33a). The presence of PPL can also be verified in user land by observing the PPL flag associated with processes that are protected by PPL. The `Process Explorer` utility (part of the `Sysinternals` suite) displays the status of this flag for all running processes (see 'Protection', Figure 33b).

16 http://www.alex-ionescu.com/?p=97 [Retrieved: 7/5/2017]

```
kd> dt nt!_EPROCESS
   [...]
   +0x620 VadRoot          : _RTL_AVL_TREE
   +0x628 VadHint          : Ptr64 Void
   +0x630 VadCount         : Uint8B
   +0x638 VadPhysicalPages : Uint8B
   +0x640 VadPhysicalPagesLimit : Uint8B
   +0x648 AlpcContext      : _ALPC_PROCESS_CONTEXT
   +0x668 TimerResolutionLink : _LIST_ENTRY
   +0x678 TimerResolutionStackRecord : Ptr64 _PO_DIAG_STACK_RECORD
   +0x680 RequestedTimerResolution : Uint4B
   +0x684 SmallestTimerResolution : Uint4B
   +0x688 ExitTime         : _LARGE_INTEGER
   +0x690 InvertedFunctionTable : Ptr64 _INVERTED_FUNCTION_TABLE
   +0x698 InvertedFunctionTableLock : _EX_PUSH_LOCK
   +0x6a0 ActiveThreadsHighWatermark : Uint4B
   +0x6a4 LargePrivateVadCount : Uint4B
   +0x6a8 ThreadListLock   : _EX_PUSH_LOCK
   +0x6b0 WnfContext       : Ptr64 Void
   +0x6b8 Spare0           : Uint8B
   +0x6c0 SignatureLevel   : UChar
   +0x6c1 SectionSignatureLevel : UChar
   +0x6c2 Protection       : _PS_PROTECTION
   +0x6c3 HangCount        : UChar
   +0x6c4 Flags3           : Uint4B
   +0x6c4 Minimal          : Pos 0, 1 Bit
   +0x6c4 ReplacingPageRoot : Pos 1, 1 Bit
   +0x6c4 DisableNonSystemFonts : Pos 2, 1 Bit
   +0x6c4 AuditNonSystemFontLoading : Pos 3, 1 Bit
   [...]
```



a) b)

Figure 33: a) the Protection field of the EPROCESS *structure; b) the PPL process flag*

3.7.5 Summary

In this section, a summarizing overview of the components identified as potential targets for a detailed analysis as part of this work package and project (see Section 3.7.1 - 3.7.4) is provided. Table 3 lists the names and brief descriptions of these components. We emphasize that the components listed in Table 3 have been selected based on the criteria for what makes a component a worthy target for analysis as part of this work package and project, which are defined in Section 2.3. In addition, Table 3 associates with each component:

- a *threat*, which is directly related to the functionalities of the component described in the 'Description' column of Table 3; we emphasize that this threat is not necessarily the only one that can be associated with the component and that the realization of the threat can lead to other treats. We present only the threat directly related to the functionalities of the component for the sake of brevity and clarity.

  We now present brief descriptions of the considered threats. We emphasize that the threat categories we define are not comprehensive and that they are defined for the convenience of discussion.

  - *Unauthorized system access*: Unauthorized access to the system of any accessible electronic device (e.g., a workstation, printer, mobile device) from a remote site.
  - *Unauthorized system activities*: Unauthorized process or user activities at a given system (e.g., writing to processes' memory space, manipulation of kernel objects, reading or writing protected files).
  - *Malicious code deployment*: Deployment of malicious code on a given system from a remote site or from within the system.
  - *Information disclosure*: Disclosure of information to unauthorized parties. The following threats are derived from this threat: *personal information disclosure*: disclosure of personal information (e.g., age, location, and address); and *application information disclosure*: disclosure of application, technical information (e.g., registry values, memory utilization statistics, and application usage statistics).

- an *attack scenario*, which is a possible scenario of practically realizing the threat associated with the category (i.e., we developed attack scenarios for demonstration purpose only).
- *properties*, which are component properties that indicate the security criticality of components, and they are: i) 'mandatory' ('*M*' in Table 3), indicating that the component is a mandatory system component (i.e., it cannot be turned off or disabled, for example, as part of system hardening procedures); and ii) 'full system compromise' ('*F*' in Table 3), indicating that a successful compromise of this component surely leads to a complete attacker's control over the system (i.e., to the most severe consequences). The components PatchGuard and HyperGuard (marked in bold in Table 3) fulfill both criteria, and therefore may be considered as the most security-critical components out of those listed in Table 3.

Component	**Description**	**Properties**	
		M	F
Windows Sensor	A framework for the delivery and handling of data from sensors (e.g., cameras, peripheral devices).		
	Threat: Personal information disclosure *Attack scenario:* An attacker extracts personal user information (e.g., surroundings, location) from the storage mechanisms of Windows Sensor.		
Windows SharedPC	A mode of the Windows 10 operating system allowing for multiple user accesses.		
	Threat: Unauthorized system access *Attack scenario:* An attacker abuses misconfigured SharedPC deployment to gain access to Windows 10 systems.		
Microsoft Telephony	An environment allowing for the integration of Windows systems with communication devices and networks (e.g.,PSTN, VoIP, multimedia conferencing).		
	Threat: Personal information disclosure *Attack scenario*: An attacker can tap into private conversations (e.g., VoIP conferences).		
Microsoft Account	A mechanism for logging into Windows systems.		
	Threat: Unauthorized system access *Attack scenario*: An attacker abuses a design vulnerability in the remote authentication protocol in order to gain unauthorized system access or credentials.		
Mobile Device Management	An environment for administering mobile devices, such as smartphones, tablet computers, laptops and desktop computers.		

Component	Description	Properties	
		M	F
	Threat: Unauthorized system access *Attack scenario*: An attacker abuses misconfigured Mobile Device Management deployment or a design vulnerability in the Mobile Device Management enrollment protocols to gain access to enterprise resources.		
App-V	A virtualization environment allowing for delivering Windows applications to users as virtualized applications.		
	Threat: Unauthorized system activities *Attack scenario*: An attacker breaks out of the App-V sandbox environment.		
Windows Update	A service delivering and installing updates to Windows systems.		
	Threat: Malicious code deployment *Attack scenario*: An attacker configures or manipulates Windows Update to download malicious updates from an attacker-owned location.		
Geolocation	A service monitoring the current location of the system and managing geofences.		
	Threat: Personal information disclosure *Attack scenario*: An attacker reverses the storage mechanisms of Geolocation to obtain location data of system users.		
Driver management	A service detecting, downloading, and installing device-related software.		
	Threat: Malicious code deployment *Attack scenario*: An attacker configures or manipulates Driver management to download malicious driver updates from an attacker-owned location.		
Windows Defender	A Windows security mechanism monitoring a system's activities in order to detect viruses, malwares, and malicious activities.		

Component	Description	Properties	
		M	F
	Threat: Malicious code deployment *Attack scenario*: An attacker exploits a vulnerability in the scanning features of Windows Defender in order to bypass its protections.		
Microsoft Wallet	A mechanism for on-line payment.		
	Threat: Personal information disclosure *Attack scenario*: An attacker abuses a vulnerability in the Microsoft Wallet payment protocol in order to obtain confidential information (e.g., shipment address).		
Unknown services	Unknown, undocumented services (as of 6th July, 2017) introduced in the Anniversary Update of the Windows 10 operating system.		
	Threat: n/a; the functions and capabilities of these services should be investigated in order to be decided whether these services pose security risks. *Attack scenario*: n/a		
Windows IoT	A framework for the handling and management of IoT, peripheral devices interacting with a Windows system.		
	Threat: Unauthorized system access *Attack scenario*: An attacker abuses a vulnerability in the Connected Devices Platform Protocol in order to gain unauthorized resource access.		
Microsoft Passport	An authentication mechanism.		
	Threat: Unauthorized system access *Attack scenario*: An attacker abuses an authentication vulnerability in Microsoft Passport to gain unauthorized access to resources.		
Windows File History	A file back-up mechanism.		

Component	Description	Properties	
		M	F
	Threat: Information disclosure *Attack scenario*: An attacker obtains confidential data by triggering an unauthorized backup or gaining access to backed-up data by, for example, abusing misconfigured Windows File History deployment.		
Data Collection and Publishing	A service allowing for applications to upload data to the Cloud.		
	Threat: Information disclosure *Attack scenario*: An attacker is able to read locally cached application information (to be sent to an-online backend) due to insecure deployment of Data Collection and Publishing.		
DiagnosticsHub	A functionality of the Windows system collecting and handling log files, which may be uploaded to remote sites.		
	Threat: Application information disclosure *Attack scenario*: An attacker abuses DiagnosticsHub's log processing capabilities in order to obtain application-related information.		
Windows Application Compatibility Infrastructure	A legacy program execution environment for backward compatibility.	X	
	Threat: Unauthorized system activities *Attack scenario*: An attacker abuses the Windows Application Compatibility Infrastructure execution workflow in order to gain unauthorized system access.		
PatchGuard	A Windows security mechanism, which prevents kernel manipulations.	X	X
	Threat: Unauthorized system activities *Attack scenario*: An attacker bypasses PatchGuard protections to manipulate the Windows kernel, which is effectively a full system compromise.		

Component	Description	Properties	
		M	F
HyperGuard	A Windows security mechanism, which prevents kernel manipulations and operates as a mechanism isolated from system users though the use of virtualization.	X	X
	Threat: Unauthorized system activities *Attack scenario*: An attacker bypasses HyperGuard protections to manipulate the Windows kernel, which is effectively a full system compromise.		
ControlFlowGuard	A Windows security mechanism, which provides access control over function invokations within program code, therefore, being effectively an exploitation prevention mechanism.	X	
	Threat: Unauthorized system activities *Attack scenario*: An attacker bypasses ControlFlowGuard protections to successfully exploit a vulnerability.		
Protected Process Light	A Windows security mechanism, which protects the memory space of certain processes, marked as protected, from accesses by other untrusted processes.	X	
	Threat: Unauthorized system activities *Attack scenario*: An attacker bypasses Protected Process Light protections to compromise a process.		

Table 3: A summarizing overview of relevant components ['M' stands for 'mandatory'; 'F' stands for 'full system compromise'; 'X' marks the fulfillment of the additional criteria 'M' and/or 'F']

4 Logging Capabilities

In this section, we provide an overview of the logging capabilities of the Windows 10 operating system and of each component in the scope of this work package (see Section 2.3). We refer to the overall logging configuration space of a component (i.e., of how the logging capabilities of a component can be configured) as logging domain.

Section 4.1 presents the architectures and inner working mechanisms of the core logging facilities of Windows 10, ETW and EventLog, and discusses their integration. Section 4.2 presents the logging configuration space of the EventLog facility/component.[17] Section 4.3 presents the logging configuration space of each component that has been designated for analysis as part of this project (i.e., PowerShell, Windows Script Host, Telemetry, Device Guard, Virtual Secure Mode, Trusted Platform Module, UEFI "Secure Boot", and Universal Windows Platform). We emphasize that we focus our discussions in the previously mentioned sections on the logging settings of components that are officially documented by Microsoft. Settings that have not been officially documented and might be relevant to the component analysis done as part of the work packages of this project, are discussed in the respective work packages.

4.1 Windows 10: Logging Capabilities

In this section, we discuss the core logging facilities of Windows 10, ETW and EventLog. It should be emphasized that since the release of Windows Vista, EventLog has been integrated into ETW. Before their integration, EventLog and ETW existed as separate components. The former was primarily used for logging of Windows system events (e.g., kernel events), whereas the latter for fine-granular logging of application events ([Tulloch 2009], Chapter 21). In this section, we first discuss the architecture of ETW and then elaborate on how EventLog operates as part of ETW.

ETW is a Windows facility enabling the customizable logging of events through instrumenting code with logging functions defined as part of the ETW API.[18] The release of Windows 10 marks the introduction of the TraceLogging API, a new, simplified ETW-based API, which may also be used for instrumenting code for logging purposes.[19] Figure 34 presents the architecture of ETW ([Soulami 2012], Chapter 12), which we discuss next.

ETW has been designed with performance in mind; ETW does not perform expensive logging operations, such as synchronous file flushing using system calls. This results in frequent context switches. Instead, ETW delegates logging to the Windows kernel, which writes relevant events to designated memory buffers (*in-memory buffers* in Figure 34) and asynchronously flushes the events to files.

Having the kernel to manage logging is not only efficient, but also addresses issues present when it comes to asynchronous flushing events to files. An example is lost logging information due to unexpected crashes of the program logging the events. Multiple buffers are managed by a single kernel entity (environment) named *session* (*Session 1-n* in Figure 34), which is also known as logger.

Events are provided to sessions for storing in buffers (e*vents* in Figure 34) by associated *ETW providers*. These are conceptual entities implemented, using the ETW API, as part of a logging user application or kernel-mode entity (e.g., a driver). An ETW provider is used for logging a given category of events, for example, the TCP/IP driver implements its own provider.

17 In contrast to EventLog, the ETW logging facility itself cannot be extensively configured by users of the Windows system (e.g., users cannot define logging policies). This is because ETW is fully implemented as an API, with only a few configuration interfaces exposed to operating system users (e.g., enable and disable configuration options).

18 https://docs.microsoft.com/en-us/windows-hardware/drivers/devtest/event-tracing-for-windows--etw- [Retrieved: 7/5/2017]

19 https://docs.microsoft.com/en-us/windows-hardware/drivers/devtest/tracelogging-api [Retrieved: 7/5/2017]

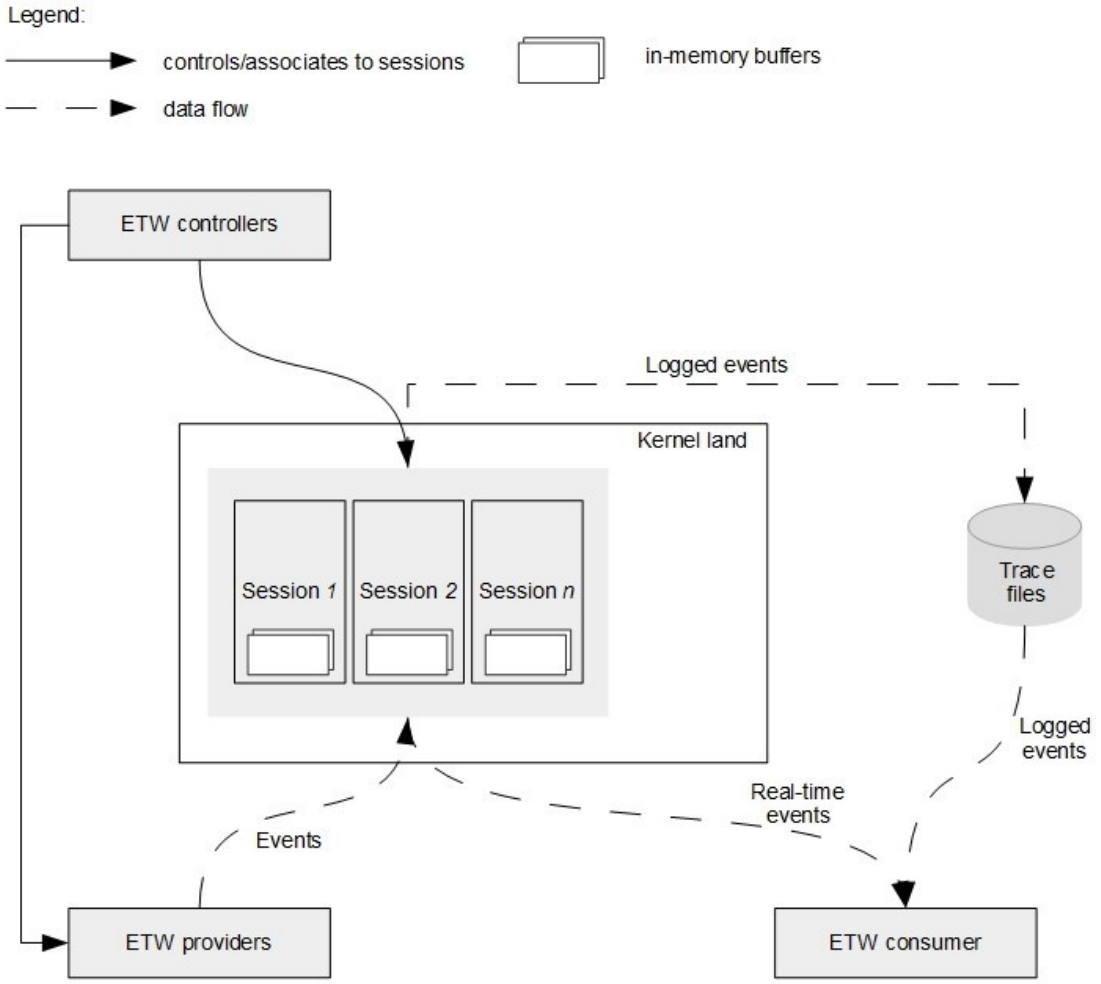


Figure 34: The architecture of ETW

The operation of sessions (e.g., activation and deactivation) and association of providers to sessions is done by special-purpose applications referred to as *ETW controllers*. The session management functionalities are implemented as part of the *subsystem DLLs* (see Section 2.1). For example, the `StartTrace` function used for activating a session is implemented in the `advapi32.dll` library file. An example ETW controller is the `xperf` utility, distributed as part of the `Windows Performance Toolkit`.[20]

The events managed by sessions are consumed by *ETW consumers*. ETW consumers are applications viewing and parsing events, such as the `xperf` or `Windows Event Viewer` utilities. Events may be distributed to ETW consumers in real-time (*real-time events* in Figure 34) or when flushed to files (*logged events* and *trace files* in Figure 34).

We now discuss on how EventLog is integrated into ETW. In contrast to ETW, EventLog does not exist as a distinguishable component of the Windows operating system. The EventLog logging functionalities are implemented as ETW logging functionalities. The Windows service EventLog acts as an ETW consumer consuming events provided by designated sessions delivering events in real-time. This service converts received events in `.evtx` format and stores them in log files, such that each event bears a unique identification number, referred to as Event ID. The `.evtx` format is the format of EventLog files, which are parsed and presented to users by the Windows utility `Event Viewer`.

20 https://docs.microsoft.com/en-us/windows-hardware/test/wpt/ [Retrieved: 7/5/2017]

Figure 35: The relation between ETW and EventLog

Figure 35 depicts the relation between ETW and EventLog as implemented in the Windows kernel. We depicted the kernel implementation presented in Figure 35 using the `windbg` debugger. We aim to verify that the EventLog service consumes events from ETW sessions. To this end, we first isolated the EventLog service in its own service host process (i.e., `svchost.exe`, see Section 2.1) using the command `sc config EventLog type=own`., which makes tracking its operation and deployment easier. The EventLog service was running as part of a `svchost.exe` process with a PID of 1764.

We than started tracking the deployment and use of an ETW session delivering events to the EventLog service. The `EventLog-Security` session (*[1]* in Figure 35) is an ETW session managing security-relevant events, such as user logons, privilege escalations, and so on. Investigating the implementation of the `EventLog-Security` session as a kernel structure (*[2]* in Figure 35) revealed the `Consumer` field (*[3]* in Figure 35). This field is an implementation of the ETW consumer attached to the session.

Analyzing the contents of the `Consumer` field (*[4]* in Figure 35), we identified the `ProcessObject` structure (*[5]* in Figure 35), which points to a Windows kernel structure of type `EPROCESS` containing process information (e.g., PID); that is, the `ProcessObject` structure points to the process representing the ETW consumer. Investigating the contents of the `ProcessObject` structure (*[6]* in Figure 35), we identify the PID of the process acting as an ETW consumer (i.e., the value of the `Cid` field: 06e4 in hexadecimal format, or 1764 in decimal format). This PID is same as the PID of the EventLog service, which proves that the EventLog service consumes events from the `EventLog-Security` ETW session. These events are then converted and stored in EventLog files, which can be verified by viewing the open file handles of the EventLog service using the `Process Explorer` utility (part of the `Sysinternals` suite).

4.2 Logging Domain: EventLog

The EventLog logging facility of Windows 10, that is, the EventLog service (see Section 4.1) is configured by modifying registry entries located at `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\EventLog`. For example, by setting `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\EventLog\Security\File` to a given file location, the EventLog service will store relevant security-related logged events at that location. In addition, the execution behavior of the EventLog service (e.g., the way in which the service is started) can be configured by using the `Services` Windows utility.

In terms of what events EventLog logs, Windows users can configure EventLog by editing a policy referred to as the audit policy. The audit policy structures different types of events (e.g., security-relevant events and system occurrences) in different pre-defined categories, referred to as audit categories. Based on these categories, a user can enable or disable the logging of the events belonging to a given category. These categories are further structured into sub-categories for fine-granular configuration. The audit policy is configured (i.e., the logging of events belonging to certain categories is enabled or disabled) by using the `Group Policy Editor`, that is, by navigating to `Computer Configuration → Windows Settings → Security Settings → Advanced Audit Policy Configuration → System Audit Policies` in the left panel of the window. In addition, this can be done by modifying the value of the registry key `HKEY_LOCAL_MACHINE\SECURITY\Policy\PolAdtEv`.

Next, we describe each of the existing audit categories, which includes a brief description of the type of events belonging to the category, sub-categories, and Event IDs that may be generated if the category is enabled. In the Appendix, section 'Audit Policy Categories and Event IDs', structured with respect to the different audit categories, we provide tables that present sub-categories (column 'Sub-category'), associated Event IDs and their descriptions (columns 'Event IDs' and 'Event message', respectively), and a link with additional information on the sub-category (field 'Detailed description'). All information presented in these tables is as provided by Microsoft. The statement 'Information on Event IDs not disclosed by Microsoft' indicates that information on a given audit category is not made publicly available by Microsoft.

The audit categories are the following:

- Account Logon: To this audit category belong authentication-related events. This category includes the following subcategories: Audit Credential Validation, Audit Kerberos Authentication Service, Audit Kerberos Service Ticket Operations, and Audit Other Logon/Logoff Events. For more information, see the Appendix, section 'Account Logon'.
- Account Management: To this audit category belong events changing user and computer accounts, and groups. This category includes the following subcategories:Audit Application Group Management, Audit Computer Account Management, Audit Distribution Group Management, Audit Other Account Management Events, Audit Security Group Management, and Audit User Account Management. For more information, see the Appendix, section 'Account Management'.
- Detailed Tracking: To this audit category belong activities of individual applications and users on that computer. This category includes the following subcategories: Audit DPAPI Activity, Audit PNP Activity,

Audit Process Creation, Audit Process Termination, and Audit RPC Events. For more information, see the Appendix, section 'Detailed Tracking'.

- Domain Service (DS) Access: To this audit category belong attempts to access and modify objects in Active Directory Domain Services. This category includes the following subcategories: Audit Detailed Directory Service Replication, Audit Directory Service Access, Audit Directory Service Changes, and Audit Directory Service Replication. For more information, see the Appendix, section 'DS Access'.
- Logon/Logoff: To this audit category belong attempts to log on to a computer interactively or over a network. This category includes the following subcategories: Audit Account Lockout, Audit User/Device Claims, Audit IPsec Extended Mode, Audit Group Membership, Audit IPsec Main Mode, Audit IPsec Quick Mode, Audit Logoff, Audit Logon, Audit Network Policy Server, Audit Other Logon/Logoff Events, and Audit Special Logon. For more information, see the Appendix, section 'Logon/Logoff'.
- Object Access: To this audit category belong attempts to access specific objects or types of objects on a network or computer (e.g., files, directories). This category includes the following subcategories: Audit Application Generated, Audit Certification Services, Audit Detailed File Share, Audit File Share, Audit File System, Audit Filtering Platform Connection, Audit Filtering Platform Packet Drop, Audit Handle Manipulation, Audit Kernel Object, Audit Other Object Access Events, Audit Registry, Audit Removable Storage, Audit SAM, and Audit Central Access Policy Staging. For more information, see the Appendix, section 'Object Access'.
- Policy Change: To this audit category belong changes to important security policies on a local system or network. This category includes the following subcategories: Audit Audit Policy Change, Audit Authentication Policy Change, Audit Authorization Policy Change, Audit Filtering Platform Policy Change, Audit MPSSVC Rule-Level Policy Change, and Audit Other Policy Change Events. For more information, see the Appendix, section 'Policy Change'.
- Privilege Use: To this audit category belong events indicating the use of certain permissions on one or more systems. This category includes the following subcategories: Audit Non-Sensitive Privilege Use, Audit Sensitive Privilege Use, and Audit Other Privilege Use Events. For more information, see the Appendix, section 'Privilege Use'.
- System: To this audit category belong events indicating changes to a computer that are not included in other categories and that have potential security implications. This category includes the following subcategories: Audit IPsec Driver, Audit Other System Events, Audit Security State Change, Audit Security System Extension, and Audit System Integrity. For more information, see the Appendix, section 'System'.
- Global Object Access Auditing: To this audit category belong events indicating attempts to file-system or registry objects access. This category includes the following subcategories: File System and Registry. For more information, see the Appendix, section 'Global Object Access Auditing'.

4.3 Logging Domain: Components

Besides being configured by developers through the ETW API (see Section 4.1), the logging capabilities of Windows components (e.g., PowerShell, UWP) that log information in EventLog format can be configured by users using the `Event Viewer` utility. This is done based on configuring event logging channels enabled by the components. These channels are properties of the ETW providers implemented as part of the components (see Section 4.1), and represent log sinks used for structuring the event log output of an ETW provider and storing this output (e.g., in files). There are four types of event logging channels:[21]

- admin: this channel receives events that target primarily administrators and support personnel. An example is an event that occurs when an application fails to connect to a printer. The events processed by the admin channel typically have a message associated with them that gives the reader direct instructions of what must be done to handle a reported issue.

21 https://msdn.microsoft.com/en-us/library/windows/desktop/aa382741(v=vs.85).aspx [Retrieved: 7/5/2017]

- operational: this channel receives events that are typically used for reporting a system event or analyzing and diagnosing a problem. An example is an event that occurs when a printer is added or removed from a system.
- analytic: this channel receives events that describe program operation and indicate internal application issues that cannot be directly handled by users.
- debug: this channel receives events intended for developers to diagnose a problem for debugging.

Event logging channels are configured using the `Event Viewer` utility by first selecting the channel to be configured and then right clicking on `Properties`. Channels can be located using the tree-structured view in the leftmost panel. Event logging channels, which are associated with components, are typically located under `Application and services logs` → `Microsoft` → `Windows` → `<component>`, where `<component>` is the name of the component, such as `DeviceGuard` or `PowerShell`. For example, the operational channel of the Device Guard component (see Section 3.4) is configured by modifying the properties of the channel located under `Application and services logs` → `Microsoft` → `Windows` → `DeviceGuard` → `Operational`. Figure 36 depicts the properties window. This window is the same for any other event logging channel and presents to users the following configuration options:[22]

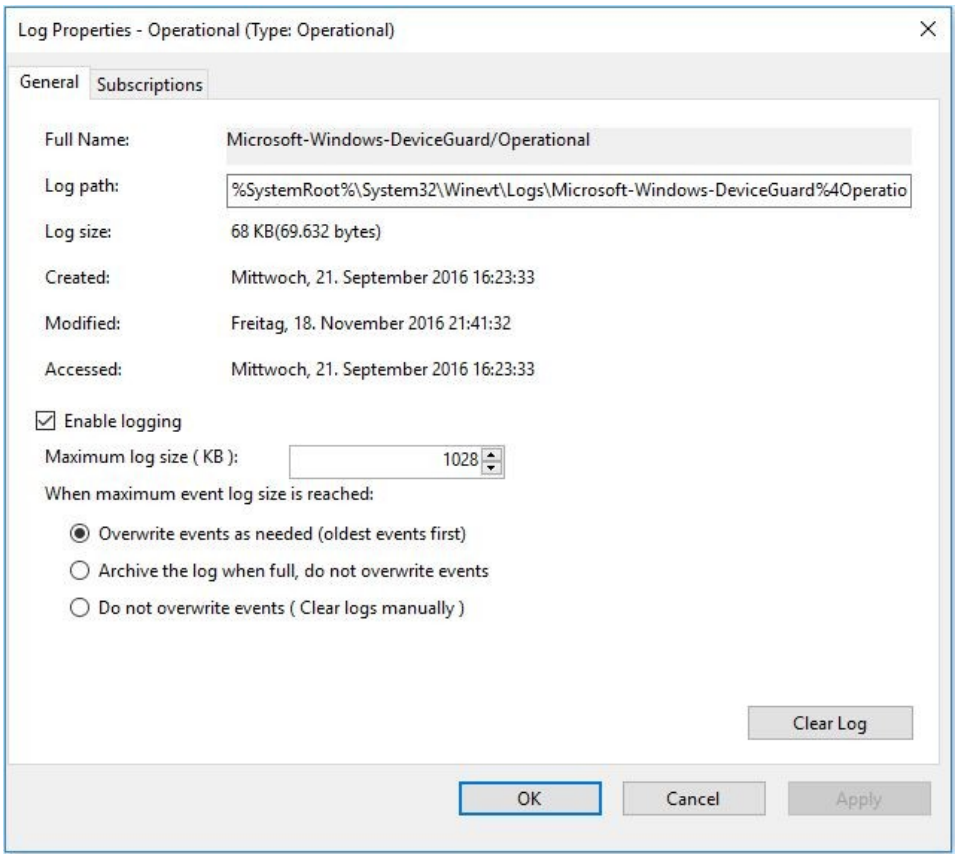


Figure 36: The properties window of the operational channel of Device Guard

- *Log path*: This configuration option is used for specifying the path at which the `.evtx` log file (see Section 4.1), in which the channel stores events, is located.
- *Enable logging*: This configuration option is used for enabling or disabling the channel. We emphasize that the analytic and debug channels are disabled to default. To enable these channels for all Windows components, select `Application and services logs` → `Microsoft` → `Windows` and right click on `View` → `Show Analytic and Debug Logs`.

22 https://technet.microsoft.com/en-us/library/cc766178(v=ws.11).aspx [Retrieved: 7/5/2017]

- *Maximum log size (KB)*: This configuration option is used for setting the maximum size (in KBs) of the `.evtx` log file located at the path specifying using the `Log path` configuration option.
- *Overwrite events as needed (oldest events first)*: This configuration option is used for configuring the channel to continue storing events when the log file has reached its maximum size. Each new incoming event replaces the oldest event stored in the log file.
- *Archive the log when full, do not overwrite events*: This configuration option is used for configuring the channel to continue storing events when the log file has reached its maximum size, such that the log file is automatically archived in a separate file and no events are overwritten.
- *Do not overwrite events (Clear logs manually)*: This configuration option is used for configuring the channel to require manual clearing of the log file when it has reached its maximum size. If the log file is not manually cleared, the event logging channel will discard (i.e., not log) incoming events. The log file in which a given event logging channel stores events can be cleared by selecting the channel and clicking on 'Clear Log' under the 'Action' menu, or by clicking the 'Clear Log button depicted in Figure 36.
- The Event IDs (see Section 4.1) under which a given component can log events is viewed using the `wevtutil` utility. Such an overview is useful, for example, for developing a tool analyzing logged events by interpreting single or correlating multiple Event IDs. When executed as `wevtutil gp <provider> /getmessage /getevents`, where `<provider>` is an ETW provider implemented as part of a given component, the `wevtutil` utility displays all Event IDs under which the component can log events. All ETW providers can be viewed by issuing `wevtutil el`. Figure 37 depicts an excerpt of the output of the command `wevtutil gp Microsoft-Windows-DeviceGuard /getmessage /getevents`, where Microsoft-Windows-DeviceGuard is an ETW provider implemented as part of the Device Guard component. In Figure 37, the value of the `value` field is an Event ID and that of the `message` field is the respective Event ID description.

```
event:
    value: 7001
    version: 0
    opcode: 0
    channel: 16
    [...]
    message: Device Guard failed to process the Group Policy to enable Virtualization Based
    Security (Status = %1): %2
event:
    value: 7002
    version: 0
    opcode: 0
    channel: 16
    [...]
    message: Device Guard failed to process the Group Policy to disable Virtualization Based
    Security (Status = %1): %2
event:
    value: 7010
    version: 0
    opcode: 0
    channel: 16
    [...]
    message: Device Guard successfully processed the Group Policy: Configurable Code
    Integrity Policy = %1,  Policy file path = %2, Reboot required = %3, Status = %4.
```

Figure 37: Output of the wevtutil utility

Appendix

Tools

Tool	Availability and Description
Chipsec	*Availability*: https://github.com/chipsec [Retrieved: 7/5/2017] *Description*: A framework for analyzing hardware, system firmware (BIOS/UEFI), and platform components.
Dependency Walker	*Availability:* http://www.dependencywalker.com/ [Retrieved: 7/5/2017] *Description*: A tool that builds a hierarchical tree diagram of loaded modules by a given executable.
driverquery	*Availability:* Distributed with Windows 10 *Description*: A tool that queries installed drivers and displays relevant driver information.
dumpbin	*Availability:* Compiled using Microsoft Visual Studio *Description*: A tool that parses headers of Windows executable files and displays relevant information (e.g., loaded modules).
IDA	*Availability:* https://www.hex-rays.com/products/ida/index.shtml [Retrieved: 7/5/2017] *Description*: A disassembly and debugging framework.
Microsoft Network Monitor	*Availability:* https://www.microsoft.com/en-us/download/details.aspx?id=4865 [Retrieved: 7/5/2017] *Description*: A network traffic sniffer.
OleView	*Availability:* Compiled using Microsoft Visual Studio *Description*: A tool for viewing the COM interface.
Performance Monitor	*Availability:* Distributed with Windows 10 *Description*: A utility displaying information on how processes affect the computer's performance (e.g., network and CPU utilization).
Sysinternals Suite (includes Process Monitor, Process Explorer, and Strings)	*Availability:* https://technet.microsoft.com/de-de/sysinternals/bb545021.aspx [Retrieved: 7/5/2017] *Description*: A suite of tools for analyzing the Windows system (e.g., analyzing operation of processes, services, and enumeration of loaded libraries by processess).
UEFITool	*Availability:* https://github.com/LongSoft/UEFITool/ [Retrieved:

	7/5/2017] *Description*: A tool for analyzing the UEFI firmware.
wevtutil	*Availability:* Distributed with Windows 10 *Description*: A tool for querying running logging mechanisms.
Windows Debugger (windbg)	*Availability:* https://developer.microsoft.com/en-us/windows/hardware/download-windbg [Retrieved: 7/5/2017] *Description*: A debugger for the Windows system.

List of Services

AJRouter
ALG
AppIDSvc
Appinfo
AppMgmt
AppReadiness
AppVClient
AppXSvc
AudioEndpointBuilder
Audiosrv
AxInstSV
BcmBtRSupport
BDESVC
BFE
BITS
BrokerInfrastructure
Browser
BthHFSrv
bthserv
CDPSvc
CDPUserSvc_409eb
CertPropSvc
ClipSVC
COMSysApp
CoreMessagingRegistrar
cphs
CryptSvc
CscService
DcomLaunch
DcpSvc
defragsvc
DeviceAssociationService
DeviceInstall
DevQueryBroker
Dhcp
diagnosticshub.standardcollector.service
DiagTrack
DmEnrollmentSvc
dmwappushservice
Dnscache
DoSvc
dot3svc
DPS
DsmSvc
DsSvc
EapHost
EFS
embeddedmode
EntAppSvc
EventLog
EventSystem
Fax
fdPHost
FDResPub
fhsvc
FontCache
FrameServer
gpsvc
hidserv
HomeGroupListener
HomeGroupProvider
HvHost

IBMPMSVC
icssvc
igfxCUIService1.0.0.0
IKEEXT
iphlpsvc
irmon
KeyIso
KtmRm
LanmanServer
LanmanWorkstation
lfsvc
LicenseManager
lltdsvc
lmhosts
LPlatSvc
LSM
MapsBroker
MessagingService_409eb
MpsSvc
MSDTC
MSiSCSI
msiserver
NcaSvc
NcbService
NcdAutoSetup
Netlogon
Netman
netprofm
NetSetupSvc
NetTcpPortSharing
NgcCtnrSvc
NgcSvc
NlaSvc
nsi
OneSyncSvc_409eb
p2pimsvc
p2psvc
PcaSvc
PeerDistSvc
PerfHost
PhoneSvc
PimIndexMaintenanceSvc_409eb
pla
PlugPlay
PNRPAutoReg
PNRPsvc
PolicyAgent
Power
PrintNotify
ProfSvc
QWAVE
RasAuto
RasMan
RemoteAccess
RemoteRegistry
RetailDemo
RmSvc
RpcEptMapper
RpcLocator
RpcSs
SamSs
SCardSvr
ScDeviceEnum
Schedule
SCPolicySvc
SDRSVC
seclogon
SENS
Sense
SensorDataService
SensorService
SensrSvc
SessionEnv
SharedAccess
ShellHWDetection
shpamsvc
smphost
SmsRouter
SNMPTRAP
Spooler
sppsvc
SSDPSRV
SstpSvc
StateRepository
stisvc
StorSvc
svsvc
swprv
SynTPEnhService
SysMain
SystemEventsBroker
TabletInputService

TapiSrv
TermService
Themes
TieringEngineService
tiledatamodelsvc
TimeBrokerSvc
TrkWks
TrustedInstaller
tzautoupdate
UevAgentService
UI0Detect
UmRdpService
UnistoreSvc_409eb
upnphost
UserDataSvc_409eb
UserManager
UsoSvc
VaultSvc
vds
vmicguestinterface
vmicheartbeat
vmickvpexchange
vmicrdv
vmicshutdown
vmictimesync
vmicvmsession
vmicvss
VSS
W32Time
WalletService
wbengine
WbioSrvc
Wcmsvc
wcncsvc
WdiServiceHost
WdiSystemHost
WdNisSvc
WebClient
Wecsvc
WEPHOSTSVC
wercplsupport
WerSvc
WiaRpc
WinDefend
WinHttpAutoProxySvc
Winmgmt
WinRM
wisvc
WlanSvc
wlidsvc
wmiApSrv
WMPNetworkSvc
workfolderssvc
WPDBusEnum
WpnService
WpnUserService_409eb
wscsvc
WSearch
wuauserv
wudfsvc
WwanSvc
XblAuthManager
XblGameSave
XboxNetApiSvc

List of Drivers

1394ohci
3ware
ACPI
AcpiDev
acpiex
acpipagr
AcpiPmi
acpitime
ADP80XX
AFD
ahcache
AmdK8
AmdPPM
amdsata
amdsbs
amdxata
AppID
applockerflt
AppvStrm
AppvVemgr
AppvVfs
arcsas

AsyncMac
atapi
b06bdrv
BasicDisplay
BasicRender
bcbtums
bcmfn
bcmfn2
Beep
bowser
BthAvrcpTg
BthEnum
BthHFEnum
bthhfhid
BthLEEnum
BTHMODEM
BthPan
BTHPORT
BTHUSB
btwampfl
buttonconver
CapImg
cdfs
cdrom
cht4iscsi
cht4vbd
circlass
CLFS
clreg
CmBatt
CNG
cnghwassist
CompositeBus
condrv
CSC
dam
Dfsc
disk
dmvsc
drmkaud
DXGKrnl
e1iexpress
ebdrv
EhStorClass
EhStorTcgDrv
ErrDev
exfat
fastfat
fdc
FileCrypt
FileInfo
Filetrace
flpydisk
FltMgr
FsDepends
fvevol
gencounter
genericusbfn
GPIOClx0101
GpuEnergyDrv
HdAudAddServ
HDAudBus
HidBatt
HidBth
hidi2c
hidinterrupt
HidIr
HidUsb
HpSAMD
HTTP
hvservice
hwpolicy
hyperkbd
i8042prt
iagpio
iai2c
iaLPSS2i_GPI
iaLPSS2i_I2C
iaLPSSi_GPIO
iaLPSSi_I2C
iaStorAV
iaStorV
ibbus
IBMPMDRV
igfx
IndirectKmd
intaud_WaveE
IntcAzAudAdd
IntcDAud
intelide
intelpep
intelppm

iorate
IpFilterDriv
IPMIDRV
IPNAT
irda
IRENUM
isapnp
iScsiPrt
iwdbus
kbdclass
kbdhid
kdnic
KSecDD
KSecPkg
ksthunk
lltdio
LSI_SAS
LSI_SAS2i
LSI_SAS3i
LSI_SSS
luafv
lunparser
MbmUsbSerial
megasas
megasas2i
megasr
MEIx64
MkBusFilter
mlx4_bus
MMCSS
Modem
monitor
mouclass
mouhid
mountmgr
mpsdrv
MRxDAV
mrxsmb
mrxsmb10
mrxsmb20
MsBridge
Msfs
msgpiowin32
mshidkmdf
mshidumdf
msisadrv
MSKSSRV
MsLldp
MSPCLOCK
MSPQM
MsRPC
MsSecFlt
mssmbios
MSTEE
MTConfig
Mup
mvumis
NativeWifiP
ndfltr
NDIS
NdisCap
NdisImPlatfo
NdisTapi
Ndisuio
NdisVirtualB
NdisWan
ndiswanlegac
ndproxy
Ndu
NetAdapterCx
NetBIOS
NetBT
NETwNe64
Npfs
npsvctrig
nsiproxy
NTFS
Null
nvraid
nvstor
Parport
partmgr
passthrupars
pci
pciide
pcip
pcmcia
pcw
pdc
PEAUTH
percsas2i
percsas3i

PptpMiniport
Processor
Psched
pvhdparser
QWAVEdrv
ramparser
RasAcd
RasAgileVpn
Rasl2tp
RasPppoe
RasSstp
rdbss
rdpbus
RDPDR
RdpVideoMini
rdyboost
ReFSv1
RFCOMM
rspndr
s3cap
sbp2port
scfilter
scmbus
scmdisk0101
sdbus
sdstor
SerCx
SerCx2
Serenum
Serial
sermouse
sfloppy
SiSRaid2
SiSRaid4
spaceport
SpbCx
srv
srv2
srvnet
stexstor
storahci
storflt
stornvme
storqosflt
storufs
storvsc
storvsp
swenum
Synth3dVsc
Synth3dVsp
SynTP
Tcpip
Tcpip6
tcpipreg
tdx
terminpt
TPM
tsusbflt
TsUsbGD
tsusbhub
tunnel
UASPStor
UcmCx0101
UcmTcpciCx01
UcmUcsi
Ucx01000
UdeCx
udfs
UEFI
UevAgentDriv
Ufx01000
UfxChipidea
ufxsynopsys
umbus
UmPass
UrsChipidea
UrsCx01000
UrsSynopsys
usbccgp
usbcir
usbehci
usbhub
USBHUB3
usbohci
usbprint
usbser
USBSTOR
usbuhci
usbvideo
USBXHCI
vdrvroot
VerifierExt

VfpExt
vhdmp
vhdparser
vhf
Vid
vmbus
VMBusHID
vmbusr
vmgid
VMSP
vmsproxy
VMSVSF
VMSVSP
volmgr
volmgrx
volsnap
volume
vpci
vpcivsp
vsmraid
VSTXRAID
vwifibus
vwififlt
vwifimp
WacomPen
wanarp
wanarpv6
wcifs
wcnfs
WdBoot
Wdf01000
WdFilter
wdiwifi
WdNisDrv
WFPLWFS
WIMMount
WindowsTrust
WindowsTrust
WinMad
WinNat
WINUSB
WinVerbs
wmbclass
WmiAcpi
Wof
WpdUpFltr
ws2ifsl
WudfPf
WUDFRd
WUDFWpdFs
xboxgip
xinputhid

Audit Policy Categories and Event IDs

Account Logon

<table>
<tr><th>Sub-category</th><th>Event IDs</th></tr>
<tr><td>Audit Credential Validation</td><td>
<table>
<tr><td>Event ID</td><td>Event message</td></tr>
<tr><td>4774</td><td>An account was mapped for logon</td></tr>
<tr><td>4775</td><td>An account could not be mapped for logon</td></tr>
<tr><td>4776</td><td>The computer attempted to validate the credentials for an account</td></tr>
<tr><td>4777</td><td>The domain controller failed to validate the credentials for an account</td></tr>
</table>
Detailed description:
https://docs.microsoft.com/en-us/windows/device-security/a
</td></tr>
</table>

<table>
<tr><th>Sub-category</th><th>Event IDs</th></tr>
<tr><td></td><td>uditing/audit-credential-validation [Retrieved: 7/5/2017]</td></tr>
<tr><td>Audit Kerberos Authentication Service</td><td><table><tr><td>Event ID</td><td>Event message</td></tr><tr><td>4768</td><td>A Kerberos authentication ticket (TGT) was requested</td></tr><tr><td>4771</td><td>Kerberos pre-authentication failed</td></tr><tr><td>4772</td><td>A Kerberos authentication ticket request failed.</td></tr></table>
Detailed description: https://docs.microsoft.com/en-us/windows/device-security/auditing/audit-kerberos-authentication-service [Retrieved: 7/5/2017]</td></tr>
<tr><td>Audit Kerberos Service Ticket Operations</td><td><table><tr><td>Event ID</td><td>Event message</td></tr><tr><td>4769</td><td>A Kerberos service ticket was requested.</td></tr><tr><td>4770</td><td>A Kerberos service ticket was renewed</td></tr><tr><td>4773</td><td>A Kerberos service ticket request failed</td></tr></table>
Detailed description: https://docs.microsoft.com/en-us/windows/device-security/auditing/audit-kerberos-service-ticket-operations [Retrieved: 7/5/2017]</td></tr>
<tr><td>Audit Other Account Logon Events</td><td>This subcategory does not contain any events. It is intended for future policies.

Detailed description: https://docs.microsoft.com/en-us/windows/device-security/auditing/audit-other-account-logon-events [Retrieved: 7/5/2017]</td></tr>
</table>

Account Management

<table>
<tr><th>Sub-category</th><th>Event IDs</th></tr>
<tr><td>Audit Application Group Management</td><td><table><tr><td>Event ID</td><td>Event message</td></tr><tr><td>4783</td><td>A basic application group was created</td></tr><tr><td>4784</td><td>A basic application group was changed</td></tr><tr><td>4785</td><td>A member was added to a basic application group</td></tr><tr><td>4786</td><td>A member was removed from a basic application group</td></tr><tr><td>4787</td><td>A non-member was added to a basic</td></tr></table></td></tr>
</table>

<table>
<tr><th>Sub-category</th><th>Event IDs</th></tr>
<tr><td></td><td>
<table>
<tr><td></td><td>application group</td></tr>
<tr><td>4788</td><td>A non-member was removed from a basic application group</td></tr>
<tr><td>4789</td><td>A basic application group was deleted</td></tr>
<tr><td>4790</td><td>An LDAP query group was created</td></tr>
<tr><td>4791</td><td>An LDAP query group was changed</td></tr>
<tr><td>4792</td><td>An LDAP query group was delete</td></tr>
</table>

Detailed description: https://docs.microsoft.com/en-us/windows/device-security/auditing/audit-application-group-management [Retrieved: 7/5/2017]
</td></tr>
<tr><td>Audit Computer Account Management</td><td>
<table>
<tr><td>Event ID</td><td>Event message</td></tr>
<tr><td>4741</td><td>A computer account was created</td></tr>
<tr><td>4742</td><td>A computer account was changed</td></tr>
<tr><td>4743</td><td>A computer account was deleted</td></tr>
</table>

Detailed description: https://docs.microsoft.com/en-us/windows/device-security/auditing/audit-computer-account-management [Retrieved: 7/5/2017]
</td></tr>
<tr><td>Audit Distribution Group Management</td><td>
<table>
<tr><td>Event ID</td><td>Event message</td></tr>
<tr><td>4749</td><td>A security-disabled global group was created</td></tr>
<tr><td>4750</td><td>A security-disabled global group was changed</td></tr>
<tr><td>4751</td><td>A member was added to a security-disabled global group</td></tr>
<tr><td>4752</td><td>A member was removed from a security-disabled global group</td></tr>
<tr><td>4753</td><td>A security-disabled global group was deleted</td></tr>
<tr><td>4759</td><td>A security-disabled universal group was created</td></tr>
<tr><td>4760</td><td>A security-disabled universal group was changed</td></tr>
<tr><td>4761</td><td>A member was added to a security-disabled universal group</td></tr>
<tr><td>4762</td><td>A member was removed from a security-disabled universal group</td></tr>
<tr><td>4763</td><td>A security-disabled universal group was deleted</td></tr>
<tr><td>4744</td><td>A security-disabled local group was created</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Sub-category</th><th>Event IDs</th></tr>
<tr><td></td><td>
<table>
<tr><td>4745</td><td>A security-disabled local group was changed</td></tr>
<tr><td>4746</td><td>A member was added to a security-disabled local group</td></tr>
<tr><td>4747</td><td>A member was removed from a security-disabled local group</td></tr>
<tr><td>4748</td><td>A security-disabled local group was deleted</td></tr>
</table>

Detailed description:
https://docs.microsoft.com/en-us/windows/device-security/auditing/audit-distribution-group-management [Retrieved: 7/5/2017]
</td></tr>
<tr><td>Audit Other Account Management Events</td><td>
<table>
<tr><td>Event ID</td><td>Event message</td></tr>
<tr><td>4782</td><td>The password hash an account was accessed</td></tr>
<tr><td>4793</td><td>The Password Policy Checking API was called</td></tr>
</table>

Detailed description:
https://docs.microsoft.com/en-us/windows/device-security/auditing/audit-other-account-management-events [Retrieved: 7/5/2017]
</td></tr>
<tr><td>Audit Security Group Management</td><td>
<table>
<tr><td>Event ID</td><td>Event message</td></tr>
<tr><td>4731</td><td>A security-enabled local group was created</td></tr>
<tr><td>4732</td><td>A member was added to a security-enabled local group</td></tr>
<tr><td>4733</td><td>A member was removed from a security-enabled local group</td></tr>
<tr><td>4734</td><td>A security-enabled local group was deleted</td></tr>
<tr><td>4735</td><td>A security-enabled local group was deleted</td></tr>
<tr><td>4764</td><td>A security-enabled local group was deleted</td></tr>
<tr><td>4799</td><td>A security-enabled local group membership was enumerated</td></tr>
<tr><td>4727</td><td>A security-enabled global group was created</td></tr>
<tr><td>4737</td><td>A security-enabled global group was changed</td></tr>
<tr><td>4728</td><td>A member was added to a security-enabled global group</td></tr>
<tr><td>4729</td><td>A member was removed from a security-enabled global group</td></tr>
<tr><td>4730</td><td>A security-enabled global group was deleted</td></tr>
<tr><td>4754</td><td>A security-enabled universal group was created.</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Sub-category</th><th>Event IDs</th></tr>
<tr><td></td><td>
<table>
<tr><td></td><td></td></tr>
<tr><td>4755</td><td>A security-enabled universal group was changed</td></tr>
<tr><td>4756</td><td>A member was added to a security-enabled universal group</td></tr>
<tr><td>4757</td><td>A member was removed from a security-enabled universal group</td></tr>
<tr><td>4758</td><td>A security-enabled universal group was deleted</td></tr>
</table>

Detailed description: https://docs.microsoft.com/en-us/windows/device-security/auditing/audit-security-group-management [Retrieved: 7/5/2017]
</td></tr>
<tr><td>Audit User Account Management</td><td>
<table>
<tr><td>Event ID</td><td>Event message</td></tr>
<tr><td>4720</td><td>A user account was created</td></tr>
<tr><td>4722</td><td>A user account was enabled</td></tr>
<tr><td>4723</td><td>An attempt was made to change an account's password</td></tr>
<tr><td>4724</td><td>An attempt was made to reset an account's password</td></tr>
<tr><td>4725</td><td>A user account was disabled</td></tr>
<tr><td>4726</td><td>A user account was deleted</td></tr>
<tr><td>4738</td><td>A user account was changed</td></tr>
<tr><td>4740</td><td>A user account was locked out</td></tr>
<tr><td>4765</td><td>SID History was added to an account</td></tr>
<tr><td>4766</td><td>An attempt to add SID History to an account failed</td></tr>
<tr><td>4767</td><td>A user account was unlocked</td></tr>
<tr><td>4780</td><td>The ACL was set on accounts which are members of administrators groups</td></tr>
<tr><td>4781</td><td>The name of an account was changed</td></tr>
<tr><td>4794</td><td>An attempt was made to set the Directory Services Restore Mode administrator password</td></tr>
</table>
</td></tr>
</table>

Sub-category	Event IDs
	4798 — A user's local group membership was enumerated 5376 — Credential Manager credentials were backed up 5377 — Credential Manager credentials were restored from a backup Detailed description: https://docs.microsoft.com/en-us/windows/device-security/auditing/audit-user-account-management [Retrieved: 7/5/2017]

Detailed Tracking

Sub-category	Event IDs
Audit DPAPI Activity	Event ID — Event message 4692 — Backup of data protection master key was attempted 4693 — Recovery of data protection master key was attempted 4694 — Protection of auditable protected data was attempted 4695 — Unprotection of auditable protected data was attempted Detailed description: https://docs.microsoft.com/en-us/windows/device-security/auditing/audit-dpapi-activity [Retrieved: 7/5/2017]
Audit PNP activity	Event ID — Event message 6416 — A new external device was recognized by the System 6419 — A request was made to disable a device 6420 — A device was disabled 6421 — A request was made to enable a device 6422 — A device was enabled 6423 — The installation of this device is forbidden by system policy

Sub-category	Event IDs
	6424 — The installation of this device was allowed, after having previously been forbidden by policy Detailed description: https://docs.microsoft.com/en-us/windows/device-security/auditing/audit-pnp-activity [Retrieved: 7/5/2017]
Audit Process Creation	Event ID — Event message 4688 — A new process has been created 4696 — A primary token was assigned to process Detailed description: https://docs.microsoft.com/en-us/windows/device-security/auditing/audit-kerberos-service-ticket-operations [Retrieved: 7/5/2017]
Audit Process Termination	Event ID — Event message 4689 — A process has exited Detailed description: https://docs.microsoft.com/en-us/windows/device-security/auditing/audit-process-termination [Retrieved: 7/5/2017]
Audit RPC Events	Event ID — Event message 5712 — An RPC was attempted Detailed description: https://docs.microsoft.com/en-us/windows/device-security/auditing/audit-rpc-events [Retrieved: 7/5/2017]

DS Access

Sub-category	Event IDs
Audit Detailed Directory Service Replication	Event ID — Event message 4928 — An Active Directory replica source naming context was established 4929 — An Active Directory replica source naming context was removed 4930 — An Active Directory replica source naming context was modified 4931 — An Active Directory replica destination naming

<table>
<tr><th>Sub-category</th><th>Event IDs</th></tr>
<tr><td></td><td>
<table>
<tr><td></td><td>context was modified</td></tr>
<tr><td>4934</td><td>Attributes of an Active Directory object were replicated</td></tr>
<tr><td>4935</td><td>Replication failure begins</td></tr>
<tr><td>4936</td><td>Replication failure ends</td></tr>
<tr><td>4937</td><td>A lingering object was removed from a replica</td></tr>
</table>

Detailed description:
https://docs.microsoft.com/en-us/windows/device-security/auditing/audit-detailed-directory-service-replication [Retrieved: 7/5/2017]
</td></tr>
<tr><td>Audit Directory Service Access</td><td>
<table>
<tr><td>Event ID</td><td>Event message</td></tr>
<tr><td>4662</td><td>An operation was performed on an object</td></tr>
<tr><td>4661</td><td>A handle to an object was requested</td></tr>
</table>

Detailed description:
https://docs.microsoft.com/en-us/windows/device-security/auditing/audit-kerberos-authentication-service [Retrieved: 7/5/2017]
</td></tr>
<tr><td>Audit Directory Service Changes</td><td>
<table>
<tr><td>Event ID</td><td>Event message</td></tr>
<tr><td>5136</td><td>A directory service object was modified</td></tr>
<tr><td>5137</td><td>A directory service object was created</td></tr>
<tr><td>5138</td><td>A directory service object was undeleted</td></tr>
<tr><td>5139</td><td>A directory service object was moved</td></tr>
<tr><td>5141</td><td>A directory service object was deleted</td></tr>
</table>

Detailed description:
https://docs.microsoft.com/en-us/windows/device-security/auditing/audit-directory-service-changes [Retrieved: 7/5/2017]
</td></tr>
<tr><td>Audit Directory Service Replication</td><td>
<table>
<tr><td>Event ID</td><td>Event message</td></tr>
<tr><td>4932</td><td>Synchronization of a replica of an Active Directory naming context has begun</td></tr>
<tr><td>4933</td><td>Synchronization of a replica of an Active Directory naming context has ended</td></tr>
</table>

Detailed description:
</td></tr>
</table>

Sub-category	Event IDs
	https://docs.microsoft.com/en-us/windows/device-security/auditing/audit-directory-service-replication [Retrieved: 7/5/2017]

Logon/Logoff

Sub-category	Event IDs
Audit Account Lockout	Event ID: 4625 — Event message: An account failed to log on Detailed description: https://docs.microsoft.com/en-us/windows/device-security/auditing/audit-credential-validation [Retrieved: 7/5/2017]
Audit User/Device Claims	Event ID: 4626 — Event message: User/Device claims information Detailed description: https://docs.microsoft.com/en-us/windows/device-security/auditing/audit-kerberos-authentication-service [Retrieved: 7/5/2017]
Audit IPsec Extended Mode	Event ID: 4978 — Event message: During Extended Mode negotiation, IPsec received an invalid negotiation packet. If this problem persists, it could indicate a network issue or an attempt to modify or replay this negotiation Event ID: 4979 — Event message: IPsec Main Mode and Extended Mode security associations were established Event ID: 4980 — Event message: IPsec Main Mode and Extended Mode security associations were established Event ID: 4981 — Event message: IPsec Main Mode and Extended Mode security associations were established. Event ID: 4982 — Event message: IPsec Main Mode and Extended Mode security associations were established. Event ID: 4983 — Event message: An IPsec Extended Mode negotiation failed. The corresponding Main Mode security association has been deleted. Event ID: 4984 — Event message: An IPsec Extended Mode negotiation failed. The corresponding Main Mode security

<table>
<tr><th>Sub-category</th><th>Event IDs</th></tr>
<tr><td></td><td>
<table>
<tr><td></td><td>association has been deleted.</td></tr>
</table>

Detailed description:
https://docs.microsoft.com/en-us/windows/device-security/auditing/audit-ipsec-extended-mode [Retrieved: 7/5/2017]
</td></tr>
<tr><td>Audit Group Membership</td><td>
<table>
<tr><td>Event ID</td><td>Event message</td></tr>
<tr><td>4627</td><td>Group membership information</td></tr>
</table>

Detailed description:
https://docs.microsoft.com/en-us/windows/device-security/auditing/audit-group-membership [Retrieved: 7/5/2017]
</td></tr>
<tr><td>Audit IPsec Main Mode</td><td>
<table>
<tr><td>Event ID</td><td>Event message</td></tr>
<tr><td>4646</td><td>Security ID: %1</td></tr>
<tr><td>4650</td><td>An IPsec Main Mode security association was established. Extended Mode was not enabled. Certificate authentication was not used</td></tr>
<tr><td>4651</td><td>An IPsec Main Mode security association was established. Extended Mode was not enabled. A certificate was used for authentication</td></tr>
<tr><td>4652</td><td>An IPsec Main Mode negotiation failed</td></tr>
<tr><td>4653</td><td>An IPsec Main Mode negotiation failed</td></tr>
<tr><td>4655</td><td>An IPsec Main Mode security association ended</td></tr>
<tr><td>4976</td><td>During Main Mode negotiation, IPsec received an invalid negotiation packet. If this problem persists, it could indicate a network issue or an attempt to modify or replay this negotiation</td></tr>
<tr><td>5049</td><td>An IPsec Security Association was deleted</td></tr>
<tr><td>5453</td><td>An IPsec negotiation with a remote computer failed because the IKE and AuthIP IPsec Keying Modules (IKEEXT) service is not started</td></tr>
</table>

Detailed description:
https://docs.microsoft.com/en-us/windows/device-security/auditing/audit-ipsec-main-mode [Retrieved: 7/5/2017]
</td></tr>
<tr><td>Audit IPsec Quick Mode</td><td>
<table>
<tr><td>Event ID</td><td>Event message</td></tr>
<tr><td>4977</td><td>During Quick Mode negotiation, IPsec received an invalid negotiation packet. If this problem</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Sub-category</th><th>Event IDs</th></tr>
<tr><td></td><td>
<table>
<tr><td></td><td>persists, it could indicate a network issue or an attempt to modify or replay this negotiation</td></tr>
<tr><td>5451</td><td>An IPsec Quick Mode security association was established</td></tr>
<tr><td>5452</td><td>An IPsec Quick Mode security association ended</td></tr>
</table>

Detailed description:
https://docs.microsoft.com/en-us/windows/device-security/auditing/audit-ipsec-quick-mode [Retrieved: 7/5/2017]
</td></tr>
<tr><td>Audit Logoff</td><td>
<table>
<tr><td>Event ID</td><td>Event message</td></tr>
<tr><td>4634</td><td>An account was logged off</td></tr>
<tr><td>4647</td><td>User initiated logoff</td></tr>
</table>

Detailed description:
https://docs.microsoft.com/en-us/windows/device-security/auditing/audit-logoff [Retrieved: 7/5/2017]
</td></tr>
<tr><td>Audit Logon</td><td>
<table>
<tr><td>Event ID</td><td>Event message</td></tr>
<tr><td>4624</td><td>An account was successfully logged on</td></tr>
<tr><td>4625</td><td>An account failed to log on</td></tr>
<tr><td>4648</td><td>A logon was attempted using explicit credentials</td></tr>
<tr><td>4675</td><td>SIDs were filtered</td></tr>
</table>

Detailed description:
https://docs.microsoft.com/en-us/windows/device-security/auditing/audit-logon [Retrieved: 7/5/2017]
</td></tr>
<tr><td>Audit Network Policy Server</td><td>
<table>
<tr><td>Event ID</td><td>Event message</td></tr>
<tr><td>6272</td><td>Network Policy Server granted access to a user</td></tr>
<tr><td>6273</td><td>Network Policy Server denied access to a user</td></tr>
<tr><td>6274</td><td>Network Policy Server discarded the request for a user</td></tr>
<tr><td>6275</td><td>Network Policy Server discarded the accounting request for a user</td></tr>
<tr><td>6276</td><td>Network Policy Server quarantined a user</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Sub-category</th><th>Event IDs</th></tr>
<tr><td></td><td>
<table>
<tr><td></td><td></td></tr>
<tr><td>6277</td><td>Network Policy Server granted access to a user but put it on probation because the host did not meet the defined health policy</td></tr>
<tr><td>6278</td><td>Network Policy Server granted full access to a user because the host met the defined health policy</td></tr>
<tr><td>6279</td><td>Network Policy Server locked the user account due to repeated failed authentication attempts</td></tr>
<tr><td>6280</td><td>Network Policy Server unlocked the user account</td></tr>
</table>

Detailed description:
https://docs.microsoft.com/en-us/windows/device-security/auditing/audit-network-policy-server [Retrieved: 7/5/2017]
</td></tr>
<tr><td>Audit Other Logon/Logoff Events</td><td>
<table>
<tr><td>Event ID</td><td>Event message</td></tr>
<tr><td>4649</td><td>A replay attack was detected</td></tr>
<tr><td>4778</td><td>A session was reconnected to a Window Station</td></tr>
<tr><td>4779</td><td>A session was disconnected from a Window Station</td></tr>
<tr><td>4800</td><td>The workstation was locked</td></tr>
<tr><td>4801</td><td>The workstation was unlocked</td></tr>
<tr><td>4802</td><td>The screen saver was invoked</td></tr>
<tr><td>4803</td><td>The screen saver was dismissed</td></tr>
<tr><td>5378</td><td>The requested credentials delegation was disallowed by policy</td></tr>
<tr><td>5632</td><td>A request was made to authenticate to a wireless network</td></tr>
<tr><td>5633</td><td>A request was made to authenticate to a wired network</td></tr>
</table>

Detailed description:
https://docs.microsoft.com/en-us/windows/device-security/auditing/audit-other-logonlogoff-events [Retrieved: 7/5/2017]
</td></tr>
<tr><td>Audit Special Logon</td><td>
<table>
<tr><td>Event ID</td><td>Event message</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Sub-category</th><th>Event IDs</th></tr>
<tr><td></td><td>
<table>
<tr><td>4964</td><td>Special groups have been assigned to a new logon</td></tr>
<tr><td>4672</td><td>Special privileges assigned to new logon</td></tr>
</table>

Detailed description:
https://docs.microsoft.com/en-us/windows/device-security/auditing/audit-group-membership [Retrieved: 7/5/2017]
</td></tr>
</table>

Object Access

<table>
<tr><th>Sub-category</th><th>Event IDs</th></tr>
<tr><td>Audit Application Generated</td><td>
<table>
<tr><td>Event ID</td><td>Event message</td></tr>
<tr><td>4665</td><td>An attempt was made to create an application client context</td></tr>
<tr><td>4666</td><td>An application attempted an operation</td></tr>
<tr><td>4667</td><td>An application client context was deleted</td></tr>
<tr><td>4668</td><td>An application was initialized</td></tr>
</table>

Detailed description:
https://docs.microsoft.com/en-us/windows/device-security/auditing/audit-application-generated [Retrieved: 7/5/2017]
</td></tr>
<tr><td>Audit Certification Services</td><td>
<table>
<tr><td>Event ID</td><td>Event message</td></tr>
<tr><td>4868</td><td>The certificate manager denied a pending certificate request</td></tr>
<tr><td>4869</td><td>Certificate Services received a resubmitted certificate request</td></tr>
<tr><td>4870</td><td>Certificate Services revoked a certificate.</td></tr>
<tr><td>4871</td><td>Certificate Services received a request to publish the certificate revocation list (CRL).</td></tr>
<tr><td>4872</td><td>Certificate Services published the certificate revocation list (CRL)</td></tr>
<tr><td>4873</td><td>A certificate request extension changed</td></tr>
<tr><td>4874</td><td>One or more certificate request attributes changed</td></tr>
<tr><td>4875</td><td>Certificate Services received a request to shut down</td></tr>
</table>
</td></tr>
</table>

Sub-category	Event IDs	
	4876	Certificate Services backup started
	4877	Certificate Services backup completed
	4878	Certificate Services restore started
	4879	Certificate Services restore completed
	4880	Certificate Services started
	4881	Certificate Services stopped
	4882	The security permissions for Certificate Services changed
	4883	Certificate Services retrieved an archived key
	4884	Certificate Services imported a certificate into its database
	4885	The audit filter for Certificate Services changed
	4886	Certificate Services received a certificate request
	4887	Certificate Services approved a certificate request and issued a certificate
	4888	Certificate Services denied a certificate request
	4889	Certificate Services set the status of a certificate request to pending
	4890	The certificate manager settings for Certificate Services changed
	4891	A configuration entry changed in Certificate Services
	4892	A property of Certificate Services changed
	4893	Certificate Services archived a key
	4894	Certificate Services imported and archived a key
	4895	Certificate Services published the CA certificate to Active Directory Domain Services
	4896	One or more rows have been deleted from the certificate database.
	4897	Role separation enabled.

<table>
<tr><th>Sub-category</th><th>Event IDs</th></tr>
<tr><td></td><td>
<table>
<tr><td>4898</td><td>Certificate Services loaded a template.</td></tr>
</table>
Detailed description:
https://docs.microsoft.com/en-us/windows/device-security/auditing/audit-certification-services [Retrieved: 7/5/2017]
</td></tr>
<tr><td>Audit Detailed File Share</td><td>
<table>
<tr><td>Event ID</td><td>Event message</td></tr>
<tr><td>5145</td><td>A network share object was checked to see whether client can be granted desired access</td></tr>
</table>
Detailed description:
https://docs.microsoft.com/en-us/windows/device-security/auditing/audit-detailed-file-share [Retrieved: 7/5/2017]
</td></tr>
<tr><td>Audit File Share</td><td>
<table>
<tr><td>Event ID</td><td>Event message</td></tr>
<tr><td>5140</td><td>A network share object was accessed.</td></tr>
<tr><td>5142</td><td>A network share object was added.</td></tr>
<tr><td>5143</td><td>A network share object was modified.</td></tr>
<tr><td>5144</td><td>A network share object was deleted</td></tr>
<tr><td>5168</td><td>SPN check for SMB/SMB2 failed</td></tr>
</table>
Detailed description:
https://docs.microsoft.com/en-us/windows/device-security/auditing/audit-other-account-logon-events [Retrieved: 7/5/2017]
</td></tr>
<tr><td>Audit File System</td><td>
<table>
<tr><td>Event ID</td><td>Event message</td></tr>
<tr><td>4656</td><td>A handle to an object was requested</td></tr>
<tr><td>4658</td><td>The handle to an object was closed</td></tr>
<tr><td>4660</td><td>An object was deleted</td></tr>
<tr><td>4663</td><td>An attempt was made to access an object</td></tr>
<tr><td>4664</td><td>An attempt was made to create a hard link</td></tr>
<tr><td>4985</td><td>The state of a transaction has changed</td></tr>
<tr><td>5051</td><td>A file was virtualized</td></tr>
<tr><td>4670</td><td>Permissions on an object were changed</td></tr>
</table>
Detailed description:
https://docs.microsoft.com/en-us/windows/device-security/a
</td></tr>
</table>

<table>
<tr><th>Sub-category</th><th>Event IDs</th></tr>
<tr><td></td><td>uditing/audit-file-system [Retrieved: 7/5/2017]</td></tr>
<tr><td>Audit Filtering Platform Connection</td><td>
<table>
<tr><td>Event ID</td><td>Event message</td></tr>
<tr><td>5031</td><td>The Windows Firewall Service blocked an application from accepting incoming connections on the network</td></tr>
<tr><td>5150</td><td>The Windows Filtering Platform blocked a packet</td></tr>
<tr><td>5151</td><td>A more restrictive Windows Filtering Platform filter has blocked a packet</td></tr>
<tr><td>5154</td><td>The Windows Filtering Platform has permitted an application or service to listen on a port for incoming connections</td></tr>
<tr><td>5155</td><td>The Windows Filtering Platform has blocked an application or service from listening on a port for incoming connections</td></tr>
<tr><td>5156</td><td>The Windows Filtering Platform has permitted a connection</td></tr>
<tr><td>5157</td><td>The Windows Filtering Platform has blocked a connection</td></tr>
<tr><td>5158</td><td>The Windows Filtering Platform has permitted a bind to a local port</td></tr>
<tr><td>5159</td><td>The Windows Filtering Platform has blocked a bind to a local port</td></tr>
</table>

Detailed description: https://docs.microsoft.com/en-us/windows/device-security/auditing/audit-filtering-platform-connection [Retrieved: 7/5/2017]</td></tr>
<tr><td>Audit Filtering Platform Packet Drop</td><td>
<table>
<tr><td>Event ID</td><td>Event message</td></tr>
<tr><td>5152</td><td>The Windows Filtering Platform blocked a packet</td></tr>
<tr><td>5153</td><td>A more restrictive Windows Filtering Platform filter has blocked a packet</td></tr>
</table>

Detailed description: https://docs.microsoft.com/en-us/windows/device-security/auditing/audit-filtering-platform-packet-drop [Retrieved: 7/5/2017]</td></tr>
<tr><td>Audit Handle Manipulation</td><td>
<table>
<tr><td>Event ID</td><td>Event message</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Sub-category</th><th>Event IDs</th></tr>
<tr><td></td><td>
<table>
<tr><td>4658</td><td>The handle to an object was closed</td></tr>
<tr><td>4690</td><td>An attempt was made to duplicate a handle to an object</td></tr>
</table>

Detailed description: https://docs.microsoft.com/en-us/windows/device-security/auditing/audit-handle-manipulation [Retrieved: 7/5/2017]
</td></tr>
<tr><td>Audit Kernel Object</td><td>
<table>
<tr><td>Event ID</td><td>Event message</td></tr>
<tr><td>4656</td><td>A handle to an object was requested</td></tr>
<tr><td>4658</td><td>The handle to an object was closed</td></tr>
<tr><td>4660</td><td>An object was deleted</td></tr>
<tr><td>4663</td><td>An attempt was made to access an object</td></tr>
</table>

Detailed description: https://docs.microsoft.com/en-us/windows/device-security/auditing/audit-kernel-object [Retrieved: 7/5/2017]
</td></tr>
<tr><td>Audit Other Object Access Events</td><td>
<table>
<tr><td>Event ID</td><td>Event message</td></tr>
<tr><td>4671</td><td>An application attempted to access a blocked ordinal through the TBS</td></tr>
<tr><td>4691</td><td>Indirect access to an object was requested</td></tr>
<tr><td>5148</td><td>The Windows Filtering Platform has detected a DoS attack and entered a defensive mode; packets associated with this attack will be discarded</td></tr>
<tr><td>5149</td><td>The DoS attack has subsided and normal processing is being resumed</td></tr>
<tr><td>4698</td><td>A scheduled task was created</td></tr>
<tr><td>4699</td><td>A scheduled task was deleted</td></tr>
<tr><td>4700</td><td>A scheduled task was enabled</td></tr>
<tr><td>4701</td><td>A scheduled task was disabled</td></tr>
<tr><td>4702</td><td>A scheduled task was updated</td></tr>
<tr><td>5888</td><td>An object in the COM+ Catalog was modified</td></tr>
<tr><td>5889</td><td>An object was deleted from the COM+ Catalog</td></tr>
<tr><td>5890</td><td>An object was added to the COM+ Catalog</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Sub-category</th><th>Event IDs</th></tr>
<tr><td></td><td>
<table>
<tr><td></td><td></td></tr>
</table>

Detailed description:
https://docs.microsoft.com/en-us/windows/device-security/auditing/audit-other-object-access-events [Retrieved: 7/5/2017]
</td></tr>
<tr><td>Audit Registry</td><td>
<table>
<tr><td>Event ID</td><td>Event message</td></tr>
<tr><td>4663</td><td>An attempt was made to access an object</td></tr>
<tr><td>4656</td><td>A handle to an object was requested</td></tr>
<tr><td>4658</td><td>The handle to an object was closed</td></tr>
<tr><td>4660</td><td>An object was deleted</td></tr>
<tr><td>4657</td><td>A registry value was modified</td></tr>
<tr><td>5039</td><td>A registry key was virtualized</td></tr>
<tr><td>4670</td><td>Permissions on an object were changed.</td></tr>
</table>

Detailed description:
https://docs.microsoft.com/en-us/windows/device-security/auditing/audit-registry [Retrieved: 7/5/2017]
</td></tr>
<tr><td>Audit Removable Storage</td><td>
<table>
<tr><td>Event ID</td><td>Event message</td></tr>
<tr><td>4656</td><td>A handle to an object was requested</td></tr>
<tr><td>4658</td><td>The handle to an object was closed</td></tr>
<tr><td>4663</td><td>An attempt was made to access an object</td></tr>
</table>

Detailed description:
https://docs.microsoft.com/en-us/windows/device-security/auditing/audit-removable-storage [Retrieved: 7/5/2017]
</td></tr>
<tr><td>Audit SAM</td><td>
<table>
<tr><td>Event ID</td><td>Event message</td></tr>
<tr><td>4661</td><td>A handle to an object was requested</td></tr>
</table>

Detailed description:
https://docs.microsoft.com/en-us/windows/device-security/auditing/audit-sam [Retrieved: 7/5/2017]
</td></tr>
<tr><td>Audit Central Access Policy Staging</td><td>
<table>
<tr><td>Event ID</td><td>Event message</td></tr>
<tr><td>4818</td><td>Proposed Central Access Policy does not grant the same access permissions as the current Central Access Policy</td></tr>
</table>

Detailed description:
</td></tr>
</table>

Sub-category	Event IDs
	https://docs.microsoft.com/en-us/windows/device-security/auditing/audit-central-access-policy-staging [Retrieved: 7/5/2017]

Policy Change

Sub-category	Event IDs
Audit Audit Policy Change	<table><tr><td>Event ID</td><td>Event message</td></tr><tr><td>4715</td><td>The audit policy (SACL) on an object was changed</td></tr><tr><td>4719</td><td>System audit policy was changed</td></tr><tr><td>4817</td><td>Auditing settings on object were changed</td></tr><tr><td>4902</td><td>The Per-user audit policy table was created</td></tr><tr><td>4906</td><td>The CrashOnAuditFail value has changed</td></tr><tr><td>4907</td><td>Auditing settings on object were changed</td></tr><tr><td>4908</td><td>Special Groups Logon table modified</td></tr><tr><td>4912</td><td>Per User Audit Policy was changed</td></tr><tr><td>4904</td><td>An attempt was made to register a security event source</td></tr><tr><td>4905</td><td>An attempt was made to unregister a security event source</td></tr></table> Detailed description: https://docs.microsoft.com/en-us/windows/device-security/auditing/audit-audit-policy-change [Retrieved: 7/5/2017]
Audit Authentication Policy Change	<table><tr><td>Event ID</td><td>Event message</td></tr><tr><td>4670</td><td>Permissions on an object were changed</td></tr><tr><td>4706</td><td>A new trust was created to a domain</td></tr><tr><td>4707</td><td>A trust to a domain was removed</td></tr><tr><td>4716</td><td>Trusted domain information was modified</td></tr><tr><td>4713</td><td>Kerberos policy was changed</td></tr><tr><td>4717</td><td>System security access was granted to an account</td></tr><tr><td>4718</td><td>System security access was removed from an</td></tr></table>

<table>
<tr><th>Sub-category</th><th>Event IDs</th></tr>
<tr><td></td><td>
<table>
<tr><td></td><td>account</td></tr>
<tr><td>4739</td><td>Domain Policy was changed</td></tr>
<tr><td>4864</td><td>A namespace collision was detected</td></tr>
<tr><td>4865</td><td>A trusted forest information entry was added</td></tr>
<tr><td>4866</td><td>A trusted forest information entry was removed</td></tr>
<tr><td>4867</td><td>A trusted forest information entry was modified</td></tr>
</table>

Detailed description: https://docs.microsoft.com/en-us/windows/device-security/auditing/audit-authentication-policy-change [Retrieved: 7/5/2017]
</td></tr>
<tr><td>Audit Authorization Policy Change</td><td>
<table>
<tr><td>Event ID</td><td>Event message</td></tr>
<tr><td>4703</td><td>A user right was adjusted</td></tr>
<tr><td>4704</td><td>A user right was assigned</td></tr>
<tr><td>4705</td><td>A user right was removed</td></tr>
<tr><td>4670</td><td>Permissions on an object were changed</td></tr>
<tr><td>4911</td><td>Resource attributes of the object were changed</td></tr>
<tr><td>4913</td><td>Central Access Policy on the object was changed</td></tr>
</table>

Detailed description: https://docs.microsoft.com/en-us/windows/device-security/auditing/audit-authorization-policy-change [Retrieved: 7/5/2017]
</td></tr>
<tr><td>Audit Filtering Platform Policy Change</td><td>
<table>
<tr><td>Event ID</td><td>Event message</td></tr>
<tr><td>4709</td><td>IPsec Services was started</td></tr>
<tr><td>4710</td><td>IPsec Services was disabled</td></tr>
<tr><td>4712</td><td>IPsec Services encountered a potentially serious failure</td></tr>
<tr><td>5040</td><td>A change has been made to IPsec settings. An Authentication Set was added</td></tr>
<tr><td>5041</td><td>A change has been made to IPsec settings. An Authentication Set was modified</td></tr>
</table>
</td></tr>
</table>

Sub-category	Event IDs	
	5042	A change has been made to IPsec settings. An Authentication Set was deleted
	5043	A change has been made to IPsec settings. A Connection Security Rule was added
	5044	A change has been made to IPsec settings. A Connection Security Rule was modified
	5045	A change has been made to IPsec settings. A Connection Security Rule was deleted.
	5046	A change has been made to IPsec settings. A Crypto Set was added.
	5047	A change has been made to IPsec settings. A Crypto Set was modified
	5048	A change has been made to IPsec settings. A Crypto Set was deleted
	5440	The following callout was present when the Windows Filtering Platform Base Filtering Engine started
	5441	The following filter was present when the Windows Filtering Platform Base Filtering Engine started.
	5442	The following provider was present when the Windows Filtering Platform Base Filtering Engine started.
	5443	The following provider context was present when the Windows Filtering Platform Base Filtering Engine started.
	5444	The following sub-layer was present when the Windows Filtering Platform Base Filtering Engine started.
	5446	A Windows Filtering Platform callout has been changed.
	5448	A Windows Filtering Platform provider has been changed.
	5449	A Windows Filtering Platform provider context has been changed.
	5450	A Windows Filtering Platform sub-layer has been changed.

Sub-category	Event IDs	
	5456	PAStore Engine applied Active Directory storage IPsec policy on the computer.
	5457	PAStore Engine failed to apply Active Directory storage IPsec policy on the computer.
	5458	PAStore Engine applied locally cached copy of Active Directory storage IPsec policy on the computer.
	5459	PAStore Engine failed to apply locally cached copy of Active Directory storage IPsec policy on the computer.
	5460	PAStore Engine applied local registry storage IPsec policy on the computer.
	5461	PAStore Engine failed to apply local registry storage IPsec policy on the computer
	5462	PAStore Engine failed to apply some rules of the active IPsec policy on the computer. Use the IP Security Monitor snap-in to diagnose the problem
	5463	PAStore Engine polled for changes to the active IPsec policy and detected no changes
	5464	PAStore Engine polled for changes to the active IPsec policy, detected changes, and applied them to IPsec Services.
	5465	PAStore Engine received a control for forced reloading of IPsec policy and processed the control successfully.
	5466	PAStore Engine polled for changes to the Active Directory IPsec policy, determined that Active Directory cannot be reached, and will use the cached copy of the Active Directory IPsec policy instead. Any changes made to the Active Directory IPsec policy since the last poll could not be applied.
	5467	PAStore Engine polled for changes to the Active Directory IPsec policy, determined that Active Directory can be reached, and found no changes to the policy. The cached copy of the Active Directory IPsec policy is no longer being used.
	5468	PAStore Engine polled for changes to the Active

<table>
<tr><th>Sub-category</th><th>Event IDs</th></tr>
<tr><td></td><td>
<table>
<tr><td></td><td>Directory IPsec policy, determined that Active Directory can be reached, found changes to the policy, and applied those changes. The cached copy of the Active Directory IPsec policy is no longer being used.</td></tr>
<tr><td>5471</td><td>PAStore Engine loaded local storage IPsec policy on the computer.</td></tr>
<tr><td>5472</td><td>PAStore Engine failed to load local storage IPsec policy on the computer.</td></tr>
<tr><td>5473</td><td>PAStore Engine loaded directory storage IPsec policy on the computer.</td></tr>
<tr><td>5474</td><td>PAStore Engine failed to load directory storage IPsec policy on the computer.</td></tr>
<tr><td>5477</td><td>PAStore Engine failed to add quick mode filter.</td></tr>
</table>
Detailed description: https://docs.microsoft.com/en-us/windows/device-security/auditing/audit-filtering-platform-policy-change [Retrieved: 7/5/2017]
</td></tr>
<tr><td>Audit MPSSVC Rule-Level Policy Change</td><td>
<table>
<tr><td>Event ID</td><td>Event message</td></tr>
<tr><td>4944</td><td>The following policy was active when the Windows Firewall started</td></tr>
<tr><td>4945</td><td>A rule was listed when the Windows Firewall started</td></tr>
<tr><td>4946</td><td>A change has been made to Windows Firewall exception list. A rule was added</td></tr>
<tr><td>4947</td><td>A change has been made to Windows Firewall exception list. A rule was modified</td></tr>
<tr><td>4948</td><td>A change has been made to Windows Firewall exception list. A rule was deleted</td></tr>
<tr><td>4949</td><td>Windows Firewall settings were restored to the default values</td></tr>
<tr><td>4950</td><td>A Windows Firewall setting has changed</td></tr>
<tr><td>4951</td><td>A rule has been ignored because its major version number was not recognized by Windows Firewall</td></tr>
<tr><td>4952</td><td>Parts of a rule have been ignored because its</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Sub-category</th><th>Event IDs</th></tr>
<tr><td></td><td>
<table>
<tr><td></td><td>minor version number was not recognized by Windows Firewall. The other parts of the rule will be enforced</td></tr>
<tr><td>4953</td><td>A rule has been ignored by Windows Firewall because it could not parse the rule</td></tr>
<tr><td>4954</td><td>Windows Firewall Group Policy settings have changed. The new settings have been applied</td></tr>
<tr><td>4956</td><td>Windows Firewall has changed the active profile</td></tr>
<tr><td>4957</td><td>Windows Firewall did not apply the following rule:</td></tr>
<tr><td>4958</td><td>Windows Firewall did not apply the following rule because the rule referred to items not configured on this computer</td></tr>
</table>

Detailed description:
https://docs.microsoft.com/en-us/windows/device-security/auditing/audit-mpssvc-rule-level-policy-change [Retrieved: 7/5/2017]
</td></tr>
<tr><td>Audit Other Policy Change Events</td><td>
<table>
<tr><td>Event ID</td><td>Event message</td></tr>
<tr><td>4714</td><td>Encrypted data recovery policy was changed</td></tr>
<tr><td>4819</td><td>Central Access Policies on the machine have been changed</td></tr>
<tr><td>4826</td><td>Boot Configuration Data loaded</td></tr>
<tr><td>4909</td><td>The local policy settings for the TBS were changed</td></tr>
<tr><td>4910</td><td>The group policy settings for the TBS were changed</td></tr>
<tr><td>5063</td><td>A cryptographic provider operation was attempted</td></tr>
<tr><td>5064</td><td>A cryptographic context operation was attempted</td></tr>
<tr><td>5065</td><td>A cryptographic context modification was attempted</td></tr>
<tr><td>5066</td><td>A cryptographic function operation was attempted</td></tr>
</table>
</td></tr>
</table>

Sub-category	Event IDs
	5067 — A cryptographic function modification was attempted 5068 — A cryptographic function provider operation was attempted 5069 — A cryptographic function property operation was attempted 5070 — A cryptographic function property modification was attempted 5447 — A Windows Filtering Platform filter has been changed 6144 — Security policy in the group policy objects has been applied successfully 6145 — One or more errors occurred while processing security policy in the group policy objects Detailed description: https://docs.microsoft.com/en-us/windows/device-security/auditing/audit-other-policy-change-events [Retrieved: 7/5/2017]

Privilege Use

Sub-category	Event IDs
Audit Non-Sensitive Privilege Use	Event ID — Event message 4673 — A privileged service was called 4674 — An operation was attempted on a privileged object 4985 — The state of a transaction has changed Detailed description: https://docs.microsoft.com/en-us/windows/device-security/auditing/audit-non-sensitive-privilege-use [Retrieved: 7/5/2017]
Audit Sensitive Privilege Use	Event ID — Event message 4673 — A privileged service was called 4674 — An operation was attempted on a privileged object

<table>
<tr><th>Sub-category</th><th>Event IDs</th></tr>
<tr><td></td><td><table><tr><td>4985</td><td>The state of a transaction has changed</td></tr></table>
Detailed description:
https://docs.microsoft.com/en-us/windows/device-security/auditing/audit-sensitive-privilege-use [Retrieved: 7/5/2017]</td></tr>
<tr><td>Audit Other Privilege Use Events</td><td><table><tr><td>Event ID</td><td>Event message</td></tr><tr><td>4985</td><td>The state of a transaction has changed</td></tr></table>
Detailed description:
https://docs.microsoft.com/en-us/windows/device-security/auditing/audit-other-privilege-use-events [Retrieved: 7/5/2017]</td></tr>
</table>

System

<table>
<tr><th>Sub-category</th><th>Event IDs</th></tr>
<tr><td>Audit IPsec Driver</td><td><table><tr><td>Event ID</td><td>Event message</td></tr><tr><td>4960</td><td>IPsec dropped an inbound packet that failed an integrity check. If this problem persists, it could indicate a network issue or that packets are being modified in transit to this computer. Verify that the packets sent from the remote computer are the same as those received by this computer. This error might also indicate interoperability problems with other IPsec implementations</td></tr><tr><td>4961</td><td>IPsec dropped an inbound packet that failed a replay check. If this problem persists, it could indicate a replay attack against this computer</td></tr><tr><td>4962</td><td>IPsec dropped an inbound packet that failed a replay check. The inbound packet had too low a sequence number to ensure it was not a replay</td></tr><tr><td>4963</td><td>IPsec dropped an inbound clear text packet that should have been secured. This is usually due to the remote computer changing its IPsec policy without informing this computer. This could also be a spoofing attack attempt</td></tr><tr><td>4965</td><td>IPsec received a packet from a remote computer with an incorrect Security Parameter Index (SPI). This is usually caused by malfunctioning hardware that is corrupting packets. If these errors persist, verify that the packets sent from the remote computer are the same as those received by this computer. This</td></tr></table></td></tr>
</table>

<table>
<tr><th>Sub-category</th><th>Event IDs</th></tr>
<tr><td></td><td>
<table>
<tr><td></td><td>error may also indicate interoperability problems with other IPsec implementations. In that case, if connectivity is not impeded, then these events can be ignored</td></tr>
<tr><td>5478</td><td>IPsec Services has started successfully</td></tr>
<tr><td>5479</td><td>IPsec Services has been shut down successfully. The shutdown of IPsec Services can put the computer at greater risk of network attack or expose the computer to potential security risks</td></tr>
<tr><td>5480</td><td>IPsec Services failed to get the complete list of network interfaces on the computer. This poses a potential security risk because some of the network interfaces may not get the protection provided by the applied IPsec filters. Use the IP Security Monitor snap-in to diagnose the problem</td></tr>
<tr><td>5483</td><td>IPsec Services failed to initialize RPC server. IPsec Services could not be started</td></tr>
<tr><td>5484</td><td>IPsec Services has experienced a critical failure and has been shut down. The shutdown of IPsec Services can put the computer at greater risk of network attack or expose the computer to potential security risks</td></tr>
<tr><td>5485</td><td>IPsec Services failed to process some IPsec filters on a plug-and-play event for network interfaces. This poses a potential security risk because some of the network interfaces may not get the protection provided by the applied IPsec filters. Use the IP Security Monitor snap-in to diagnose the problem</td></tr>
</table>

Detailed description:
https://docs.microsoft.com/en-us/windows/device-security/auditing/audit-ipsec-driver [Retrieved: 7/5/2017]
</td></tr>
<tr><td>Audit Other System Events</td><td>
<table>
<tr><td>Event ID</td><td>Event message</td></tr>
<tr><td>5024</td><td>The Windows Firewall Service has started successfully</td></tr>
<tr><td>5025</td><td>The Windows Firewall Service has been stopped</td></tr>
<tr><td>5027</td><td>The Windows Firewall Service was unable to retrieve the security policy from the local storage. The service will continue enforcing the</td></tr>
</table>
</td></tr>
</table>

Sub-category	Event IDs	
		current policy
	5028	The Windows Firewall Service was unable to parse the new security policy. The service will continue with currently enforced policy
	5029	The Windows Firewall Service failed to initialize the driver. The service will continue to enforce the current policy
	5030	The Windows Firewall Service failed to start
	5032	Windows Firewall was unable to notify the user that it blocked an application from accepting incoming connections on the network
	5033	The Windows Firewall Driver has started successfully
	5034	The Windows Firewall Driver was stopped
	5035	The Windows Firewall Driver failed to start
	5037	The Windows Firewall Driver detected critical runtime error. Terminating
	5058	Key file operation
	5059	Key migration operation
	6400	BranchCache: Received an incorrectly formatted response while discovering availability of content
	6401	BranchCache: Received invalid data from a peer. Data discarded.
	6402	BranchCache: The message to the hosted cache offering it data is incorrectly formatted
	6403	BranchCache: The hosted cache sent an incorrectly formatted response to the client
	6404	BranchCache: Hosted cache could not be authenticated using the provisioned SSL certificate
	6405	BranchCache: %2 instance(s) of event id %1 occurred.
	6406	%1 registered to Windows Firewall to control filtering for the following: %2

Sub-category	Event IDs
	6407 — 1.00% 6408 — Registered product %1 failed and Windows Firewall is now controlling the filtering for %2 6409 — BranchCache: A service connection point object could not be parsed. Detailed description: https://docs.microsoft.com/en-us/windows/device-security/auditing/audit-other-system-events [Retrieved: 7/5/2017]
Audit Security State Change	Event ID — Event message 4608 — Windows is starting up. 4616 — The system time was changed 4621 — Administrator recovered system from CrashOnAuditFail Detailed description: https://docs.microsoft.com/en-us/windows/device-security/auditing/audit-security-state-change [Retrieved: 7/5/2017]
Audit Security System Extension	Event ID — Event message 4610 — An authentication package has been loaded by the Local Security Authority 4611 — A trusted logon process has been registered with the Local Security Authority 4614 — A notification package has been loaded by the Security Account Manager 4622g — A security package has been loaded by the Local Security Authority 4697 — A service was installed in the system Detailed description: https://docs.microsoft.com/en-us/windows/device-security/auditing/audit-security-system-extension [Retrieved: 7/5/2017]
Audit System Integrity	Event ID — Event message 4612 — Internal resources allocated for the queuing of audit messages have been exhausted, leading to the loss of some audits 4615 — Invalid use of LPC port

Sub-category	Event IDs
	4618 — A monitored security event pattern has occurred 4816 — RPC detected an integrity violation while decrypting an incoming message. 5038 — Code integrity determined that the image hash of a file is not valid. The file could be corrupt due to unauthorized modification or the invalid hash could indicate a potential disk device error. 5056 — A cryptographic self-test was performed. 5062 — A kernel-mode cryptographic self-test was performed. 5057 — A cryptographic primitive operation failed. 5060 — Verification operation failed. 5061 — Cryptographic operation. 6281 — Code Integrity determined that the page hashes of an image file are not valid. The file could be improperly signed without page hashes or corrupt due to unauthorized modification. The invalid hashes could indicate a potential disk device error. 6410 — Code integrity determined that a file does not meet the security requirements to load into a process. Detailed description: https://docs.microsoft.com/en-us/windows/device-security/auditing/audit-system-integrity [Retrieved: 7/5/2017]

Global Object Access Auditing

Sub-category	Event IDs
File System (Global Object Access Auditing)	Information on Event IDs not disclosed by Microsoft. Detailed description: https://docs.microsoft.com/en-us/windows/device-security/auditing/file-system-global-object-access-auditing [Retrieved: 7/5/2017]
Registry (Global Object Access Auditing)	Information on Event IDs not disclosed by Microsoft. Detailed description:

Sub-category	Event IDs
	https://docs.microsoft.com/en-us/windows/device-security/auditing/registry-global-object-access-auditing [Retrieved: 7/5/2017]

Reference Documentation

Yosif 2017	Yosifovic, Pavel; Ionescu, Alex; Russinovich, Mark E.; Solomon, David A. : Windows Internals, Part 1 and Part 2
Lissoir 2013	Lissoir, Alain: Understanding WMI Scripting: Exploiting Microsoft's Windows Management Instrumentation in Mission-Critical Computing Infrastructures
Ionescu 2015	Ionescu, Alex: Battle of SKM and IUM
Mic 2017	Microsoft Corporation: Hypervisor Top Level Functional Specification
Finn 2013	Finn, Aidan; Lownds, Patrick; Luescher, Michel; Flynn, Damian : Windows Server 2012 Hyper-V Installation and Configuration Guide
UEFIF 2017	UEFI Forum: Unified Extensible Firmware Interface (UEFI) Specification, v. 2.7
TCGM 2011	Trusted Computing Group (TCG): TPM Main: Part 1, Design Principles
Tomlinson 2008	Tomlinson, Allan: Introduction to the TPM
TCGE 2014	Trusted Computing Group (TCG): TCG EFI Platform Specification For TPM Family 1.1 or 1.2
Richter 2013	Richter, Jeffrey; van de Bospoort, Maarten: Windows Runtime via C#
Tulloch 2009	Tulloch, Mitch; Northrup, Tony; Honeycutt, Jerry: Windows® 7 Resource Kit
Soulami 2012	Tarik Soulami: Inside Windows Debugging

Keywords and Abbreviations