Device Guard Image Integrity: Architecture Overview

Aleksandar Milenkoski✉ amilenkoski@ernw.de
Dominik Phillips dphillips@ernw.de

This work is part of the *Windows Insight* series. This series aims to assist efforts on analysing inner working principles, functionalities, and properties of the Microsoft Windows operating system. For general inquiries contact Aleksandar Milenkoski (amilenkoski@ernw.de) or Dominik Phillips (dphillips@ernw.de). For inquiries on this work contact the corresponding author (✉).

The content of this work has been created in the course of the project named 'Studie zu Systemaufbau, Protokollierung, Härtung und Sicherheitsfunktionen in Windows 10 (SiSyPHuS Win10)' (ger.) - 'Study of system design, logging, hardening, and security functions in Windows 10' (eng.). This project has been contracted by the German Federal Office for Information Security (ger., Bundesamt für Sicherheit in der Informationstechnik - BSI).

Required Reading

In addition to referenced work, related work focussing on Windows Architecture, the Trusted Platform Module (TPM), and Virtual Secure Mode (VSM), part of the *Windows Insight* series, are relevant for understanding concepts and terms mentioned in this document.

Technology Domain

The operating system in focus is Windows 10, build 1607, 64-bit, long-term servicing branch (LTSB).

1 Introduction

The Device Guard component of Windows 10 implements a feature for preventing the execution of untrusted code. Untrusted code is program code whose integrity and authenticity cannot be verified. For example, this is code that has been tampered with in an unauthorized manner, or originates from untrusted sources.

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.[1] These criteria may involve cryptographic information (e.g., hash values) or non-cryptographic information (e.g., file names). In addition to configurable code integrity, Windows 10 implements code integrity functionalities that do not take user-defined criteria into account. These are implemented as part of the Windows boot manager, the Windows loader, and the kernel. This work refers to these functionalities as non-configurable code integrity. When enabled, the VSM feature – hypervisor code integrity (HVCI), protects configurable and non-configurable code integrity functionalities by executing them in the secure environment. If the Unified Extensible Firmware Interface (UEFI) is present, the UEFI SecureBoot feature may be deployed for

[1] https://blogs.technet.microsoft.com/ash/2016/03/02/windows-10-device-guard-and-credential-guard-demystified/ [Retrieved: 17/7/2018]

the verification of the integrity of the UEFI firmware and the Windows boot entities, the boot manager and the Windows loader.

The configurable code integrity features can be structured into two categories: user-mode code integrity (UMCI) and kernel-mode code integrity (KMCI) ([YIRS17], 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. The UMCI and KMCI implementations of the configurable code integrity feature are also known as Windows Defender Application Control (WDAC).

2 Architecture Overview

Figure 1 depicts a compact overview of the architecture of the Device Guard and Windows 10 code integrity features. Configurable code integrity is based on user-defined rules. Among other things, these rules may specify file names, file versions, and hashes of images. An image is verified based on comparing rule-specified data with relevant data associated with the image. For example, a rule may specify the image's hash value. When the image is verified, Windows compares the rule-specified hash value with a hash value that it has calculated. In the case of a mismatch, the image may not be allowed to execute.

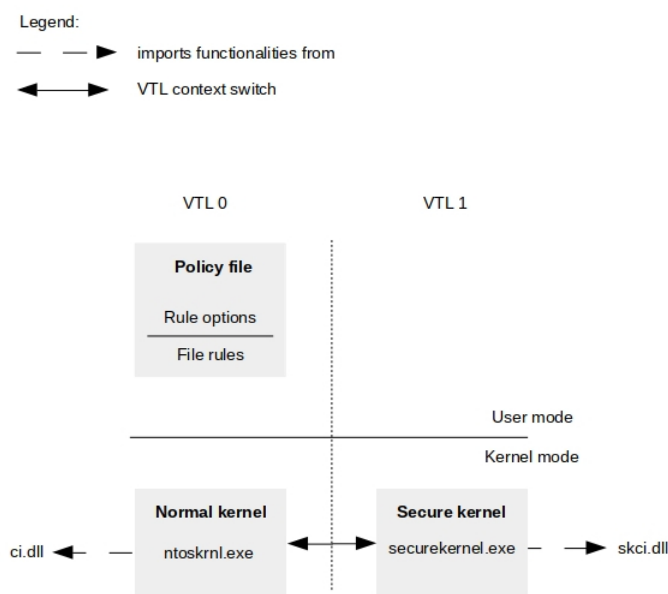

Figure 1: The architecture of the Device Guard and Windows 10 code integrity features

User-defined rules are stored in a policy file, referred to as WDAC policy in this work (Policy file in Figure 1). This file is written in the Extensible Markup Language (XML) format and then converted into binary format for deployment. The WDAC policy can be digitally signed in order to prevent modifications after it is deployed. In addition, the TPM measures WDAC policies for integrity measurement purposes.

A WDAC policy consists of rules grouped into sections. In a WDAC policy, a user may define:

- policy rule options (*Rule options* in Figure 1): Policy rule options configure the overall functionality of WDAC. An example is the *Enabled: UMCI option*, which enables UMCI. Table 1 lists the different policy rule options and provides descriptions. The descriptions presented in Table 1 are based on the information 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: 17/7/2018].
- file rules (*File rules* in Figure 1): File rules 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. Table 2 lists the different policy levels and provides descriptions. The descriptions presented in Table 2 are based on the information 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: 17/7/2018].

The policy rule options and policy levels that are available on a given Windows 10 instance can be observed by investigating the policy XML schema. The schema is stored in the *Windows\schemas\CodeIntegrity\cipolicy.xsd* file. Table 1 and Table 2 present only the information about policy rule options and levels 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: 17/7/2018].

The policy levels make WDAC highly configurable and allow for administrators to decide on a trade-off between policy manageability and verification strictness. For example, in contrast to *FileName*, the policy level *Hash* reports any modification of a file's content. However, the policy in which this level is specified has to be updated every time the content of the file is modified. This makes *Hash* an operationally challenging policy level for verifying files that are frequently modified.

Policy rule option	**Description**
Enabled: UMCI	This option applies the deployed WDAC policy to entities that operate in user- and in kernel-mode. By default, a WDAC policy applies only to entities that operate in kernel-mode.
Enabled: Boot Menu Protection	Currently not supported.
Required: WHQL	This option requires every driver specified in the WDAC policy to be signed by the Windows Hardware Quality Labs (WHQL). WHQL signs images that have passed Windows Hardware Certification Kit (WHCK) tests.
Enabled: Audit Mode	This option enables the audit mode of a WDAC policy – the option allows for all images to execute, but logs relevant events. By default, a WDAC policy operates in audit mode. If this policy rule option is not set, a WDAC policy operates in enforcement mode – images whose execution is not allowed by the policy are not executed.
Disabled: Flight Signing	This option restricts the execution of images that are flight signed. Flight signed images are images that are signed during their development. Flight signed images are typically release candidates, for example, Windows Insider Preview image builds.[2]
Enabled: Inherit Default Policy	Currently not supported.
Enabled: Unsigned System Integrity Policy	This option allows the WDAC policy to be deployed unsigned. By default, a WDAC policy has to be signed.
Allowed: Debug Policy Augmented	Currently not supported.
Required: EV Signers	This option requires for driver images to be signed by the WHQL and by Extended Validation (EV) certificates. For an EV certificate to be issued to a given entity, the entity is subjected to a rigorous vetting by a certificate authority.
Enabled: Advanced Boot Options Menu	This option configures the Windows advanced boot menu to be presented to physically present users when a WDAC policy is deployed. By default, this menu is not presented.

[2] https://insider.windows.com/en-us/ [Retrieved: 17/7/2018]

Enabled: Boot Audit on Failure	This option configures the WDAC policy operating in enforcement mode to switch to audit mode if image verification fails during system startup.
Disabled: Script Enforcement	Currently not supported.
Required: Enforce Store Applications	This option applies the WDAC policy to Universal Windows Applications (UWAs). Otherwise, WDAC policies are not applied to UWAs.
Enabled: Managed Installer	This option allows for images installed by a software distribution solution, such as the *System Center Configuration Manager*, to execute. [3]
Enabled: Intelligent Security Graph Authorization	This option allows for images classified as "known good" by the *Intelligent Security Graph* to execute.[4]
Enabled: Invalidate EAs on Reboot	This option invalidates cached image classifications by the *Intelligent Security Graph* on system reboot. Therefore, it forces re-evaluation of images that have been allowed to execute with the *Enabled: Intelligent Security Graph Authorization* option configured.
Enabled: Update Policy No Reboot	This option allows for modifications to an already deployed WDAC policy to be applied without system reboot. By default, for changes to a deployed WDAC policy to take effect, the system at which the policy is deployed has to be rebooted.

Table 1: Policy rule options

Policy level	**Description**
Hash	This level verifies an image based on the image's hash value.
FileName	This level verifies an image based on the image's name. This name is stored as part of the image as an image property.
LeafCertificate	This level verifies an image based on a hash value of a portion of the certificate issued to the image's signer. This certificate is the leaf of the certificate chain used to sign the image.
PcaCertificate	This level verifies an image based on a hash value of a portion of the certificate that is at the highest position in the certificate chain used to sign the image, with the exception of the root certificate. This is the certificate below the root certificate in the certificate chain. We refer to it as the PCAcertificate.
RootCertificate	Currently not supported.
Publisher	This level verifies an image based on a hash value of a portion of the PCAcertificate and the common name (CN) field of the leaf certificate in the certificate chain used to sign the image. This level is a combination of the *PcaCertificate* level with a verification based on the previously mentioned CN field.
SignedVersion	This level verifies an image based on a hash value of a portion of the PCAcertificate, the CN field of the leaf certificate in the certificate chain used to sign the image, and the image's file version. The image's file version has to be at, or above, a minimum version specified in the WDAC policy. This level is a combination of the Publisher level with a verification based on the image's file version.

[3] http://download.microsoft.com/download/5/D/B/5DBEBA38-8D5D-4119-B2E8-B8369B74BF43/system_center_configuration_manager_and_microsoft_intune_datasheet.pdf [Retrieved: 17/7/2018]

[4] http://cloud-platform-assets.azurewebsites.net/intelligent-security-graph/ [Retrieved: 17/7/2018]

FilePublisher	This level verifies an image based on its name, a hash value of a portion of the PCAcertificate, the common name (CN) field of the leaf certificate in the certificate chain used to sign the image, and the image's file version. This level is a combination of the *SignedVersion* level with a verification based on the image's name.
WHQL	This level allows an image to execute if it has been signed by the WHQL.
WHQLPublisher	This level allows an image to execute if it has been signed by the WHQL and verified based on the CN field of the leaf certificate in the certificate chain used to sign the image. This level is a combination of the *WHQL* level with a verfication based on the previously mentioned CN field.
WHQLFilePublisher	This level allows an image to execute if it has been signed by the WHQL, verified based on the CN field of the leaf certificate in the certificate chain used to sign the image, and verified based on the image's file version. The image's file version has to be at, or above, a minimum version specified in the WDAC policy. This level is a combination of the *WHQLPublisher* level with a verfication based on the image's file version.

Table 2: Policy levels

Once a WDAC policy in XML format is converted into binary format, it can be deployed. For example, the group policy at the *Administrative Templates\System\Device Guard* policy path may be used for policy deployment. Windows 10 stores WDAC policies in the *SIPolicy.p7b* file. On non-UEFI platforms, Windows 10 places the *SIPolicy.p7b* file in the *%System%\System32\CodeIntegrity* directory. On UEFI-based platforms, Windows 10 places the *SIPolicy.p7b* file additionally in the *\EFI\Microsoft\Boot* directory of the boot partition.

Figure 2 depicts the placement of a WDAC policy stored in the binary file *C:\Users\ernw\Desktop\DeviceGuardPolicy.bin*. This file is deployed by configuring the *Administrative Templates\System\Device Guard* group policy with the *Group Policy Object Editor* utility. Once a user configures this group policy, the *Group Policy Object Editor* utility loads the *dggpext.dll* library file and invokes the *InstallConfigCIPolicy* function. This function copies the content of *DeviceGuardPolicy.bin* to the *%System%\System32\CodeIntegrity\SIPolicy.p7b* and the *\EFI\Microsoft\Boot\SIPolicy.p7b* file, depending on the presence of UEFI. The analysis presented in this work was conducted on a platform where UEFI is not present.

```
dggpext!InstallConfigCIPolicy:
00007ffa`3b4e1afc 488bc4          mov     rax,rsp
[...]
dggpext!InstallConfigCIPolicy+0x99:
00007ffa`3b4e1b95 e812f6ffff      call    dggpext!GetSystemFirmwareType (00007ffa`3b4e11ac)
[...]
KERNEL32!CopyFileExWStub:
00007ffa`4cdff910 48ff25894f0500  jmp     qword ptr [KERNEL32!_imp_CopyFileExW (00007ffa`4ce548a0)]
0:016> du @rcx
00000061`48f7e080  "C:\Users\ernw\Desktop\DeviceGuar"
00000061`48f7e0c0  "dPolicy.bin"
0:016> du @rdx
000001f7`c8902480  "C:\Windows\System32\CodeIntegrit"
000001f7`c89024c0  "y\SIPolicy.p7b"
0:016> gu
[...]
KERNEL32!CopyFileExWStub:
00007ffa`4cdff910 48ff25894f0500  jmp     qword ptr [KERNEL32!_imp_CopyFileExW (00007ffa`4ce548a0)]
0:016> du @rcx
00000061`48f7e080  "C:\Users\ernw\Desktop\DeviceGuar"
00000061`48f7e0c0  "dPolicy.bin"
0:016> du @rdx
000001f7`c8902510  "\\?\GLOBALROOT\Device\HarddiskVo"
000001f7`c8902550  "lume2\EFI\Microsoft\Boot\SIPolic"
000001f7`c8902590  "y.p7b"
[...]
```

Figure 2: Placement of a WDAC policy

The configurable and non-configurable code integrity features implement functionalities in the boot manager, the Windows loader, and the Windows kernel. In the context of the boot manager and the Windows loader, code integrity functionalities are implemented as part of their executables. In the context of the Windows kernel, code integrity functionalities are implemented as kernel routines in external library files. If the VSM feature HVCI is disabled, code integrity functionalities are executed in the context of the *ci.dll* library file. This file is loaded by the *ntoskrnl.exe* executable, which implements the normal kernel (Normal kernel in Figure 1). The *ci.dll* library file exposes an interface of functions to the kernel for use.

If HVCI is enabled, Windows routes code integrity functionalities to the secure environment, that is, to the virtual trust level (VTL) 1, for execution (VTL 0, VTL 1, and VTL context switch in Figure 1). Code integrity functionalities are then executed in the context of the *skci.dll* library file. This prevents attackers that have gained access to the normal environment to tamper with code integrity functionalities. *skci.dll* is loaded by the *securekernel.exe* executable, which implements the secure kernel (Secure kernel in Figure 1).

References

[YIRS17] Pavel Yosifovic, Alex Ionescu, Mark E. Russinovich, and David A. Solomon. *Windows Internals, Part 1 and Part 2*. 2017. Microsoft Press.