## Virtual Secure Mode und Device Guard

Das jetzt nicht mehr neue Sicherheitskonzept von Microsoft namens Device Guard läuft ab Windows 10 Enterprise und Server 2016.

Mit dieser Technik können Hardware- und Softwarefeatures so eingesetzt, das z.B. nur vertrauenswürdige Anwendungen ausgeführt werden können.

Device Guard setzt isolierten Umgebungen (Container) ein, um Anwendungen auszuführen und voneinander zu trennen. Als Basis für die Umsetzung dient der Hypervisor von Microsoft.

Hardwarevoraussetzungen:

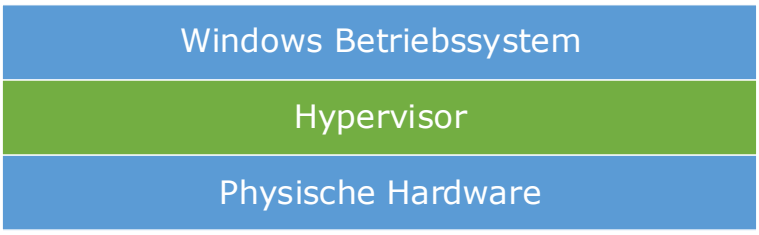
- UEFI ab v2.3.1 im einheitlichen Modus
- Windows 64-Bit
- Second Layer Address Translation SLAT
- Virtualisierungserweiterung z.B. Intel VT oder AMD-V
- Optional: Trusted Platform Modul TPM
- Optional: Andere Bootmedien sollten deaktiviert werden

Für die Nutzung von Device Guard muss zuerst der Virtual Secure Mode aktiviert werden. Der VSM ist eine Funktion der die Virtualisierungserweiterung des Prozessors einsetzt, um so die Sicherheit zu erhöhen. Geschützt werden dabei unteranderem die Daten im Speicher und es wird sichergestellt, dass jede Instanz nur auf die eigenen Daten zugreifen kann.

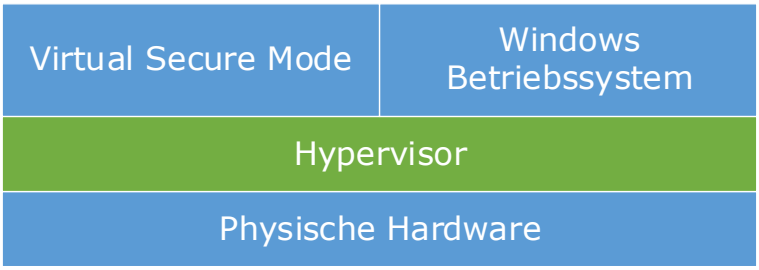
Auch der Credential Guard benötigt für die Ausführung den Virtual Secure Mode, aber der ist heute nicht mein Thema.

### Der Aufbau wäre bildlich so darzustellen:

Der Hypervisor dient als Zwischenschicht, um so die sicherheitsrelevanten Prozesse getrennt vom Betriebssystem auslagern zu können.




Auf diese Weise ist die Virtual Secure Mode Instanz geschützt und wird getrennt vom Betriebssystem (Kernel) verwaltet.

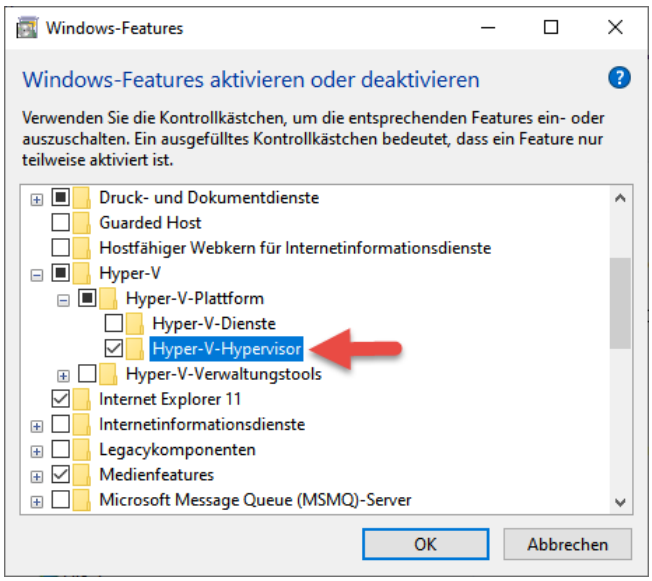


## Virtual Secure Mode und Device Guard

Kommen wir zur Aktivierung des Virtual Secure Mode. Zunächst muss der Hypervisor installiert werden. Die Managementtools werden nicht benötigt.

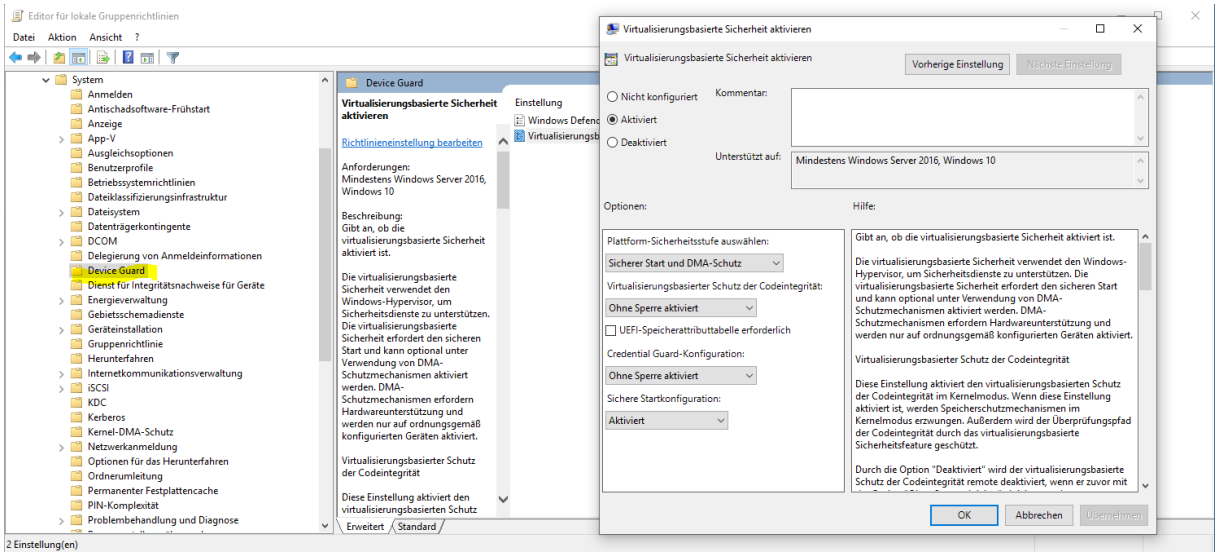
Install-WindowsFeature -Name Hyper-V

-IncludeManagementTools



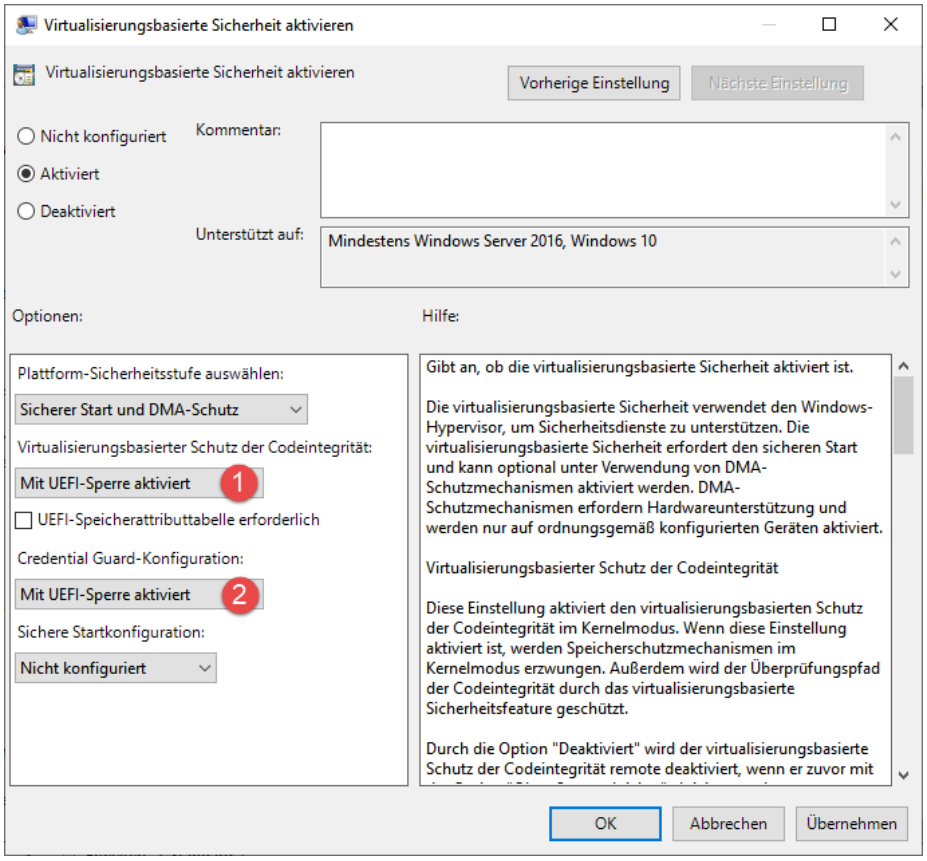

Neustart des Systems.

Zum konfigurieren des Secure Modes muss noch eine Gruppenrichtlinie erstellt bzw. aktiviert werden.

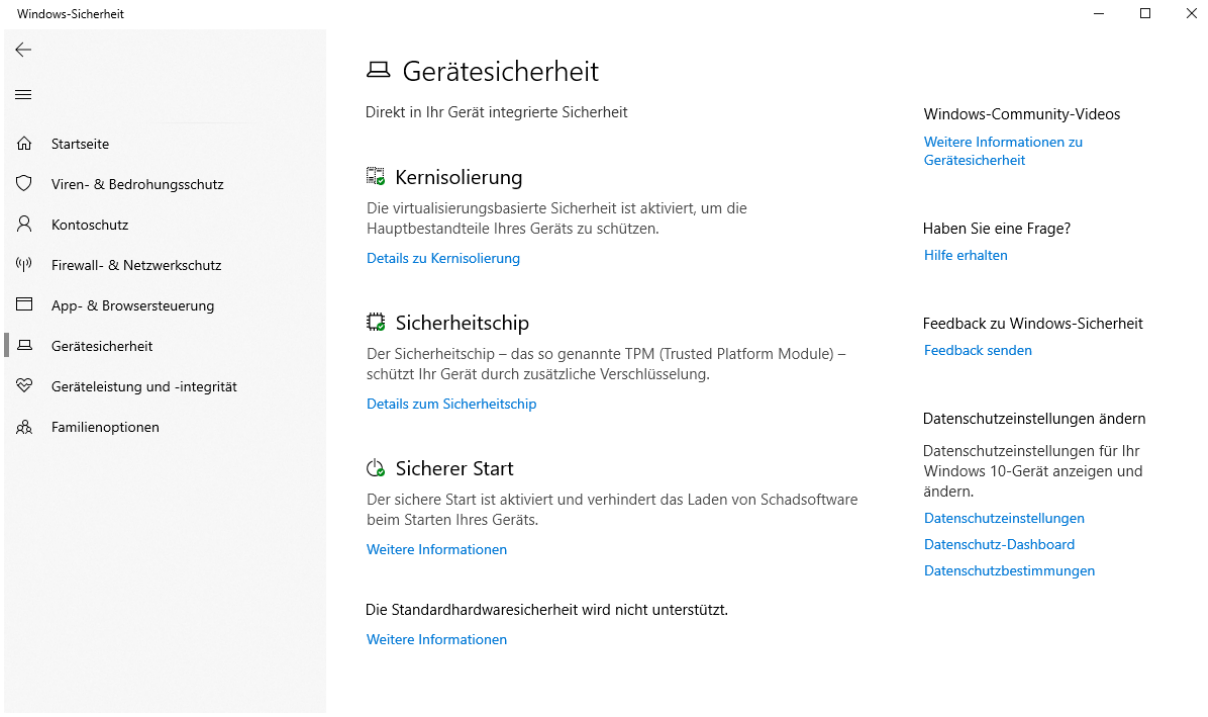


## Virtual Secure Mode und Device Guard

Würden diese beiden Optionen aktiviert werden, würde man damit eine Remoteabschaltung verhindern.



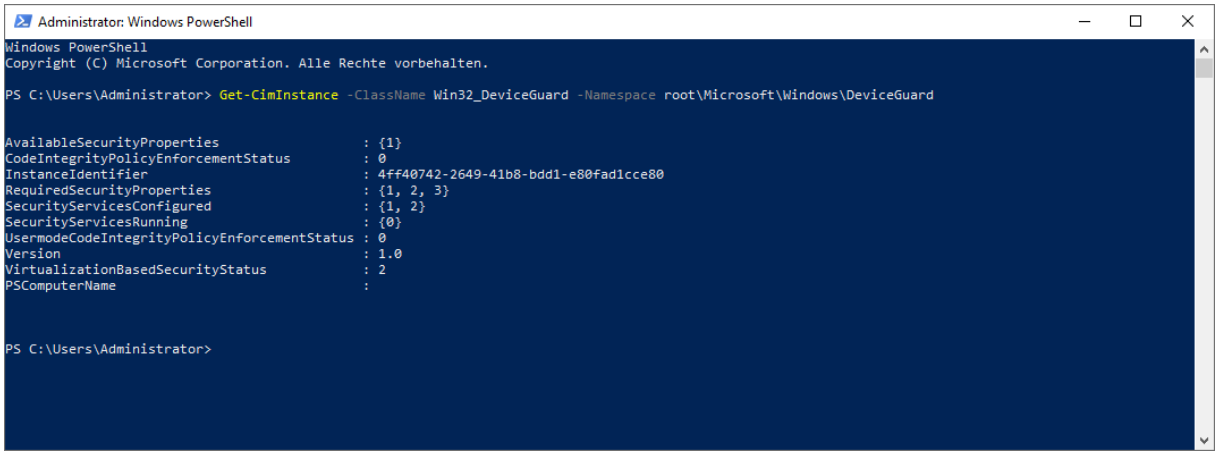

Device Guard ist nun für den Einsatz bereit.



# Virtual Secure Mode und Device Guard

## Prüfen ob Device Guard läuft:

Get-CimInstance –ClassName Win32_DeviceGuard –Namespace root\Microsoft\Windows\DeviceGuard




| Properties | Description | Valid values |
|---|---|---|
| **AvailableSecurityProperties** | This field helps to enumerate and report state on the relevant security properties for Device Guard. | ○ **0.** If present, no relevant properties exist on the device.<br>○ **1.** If present, hypervisor support is available.<br>○ **2.** If present, Secure Boot is available.<br>○ **3.** If present, DMA protection is available.<br>○ **4.** If present, Secure Memory Overwrite is available.<br>○ **5.** If present, NX protections are available.<br>○ **6.** If present, SMM mitigations are available.<br><br>**Note**: 4, 5, and 6 were added as of Windows 10, version 1607. |
| **InstanceIdentifier** | A string that is unique to a particular device. | Determined by WMI. |
| **RequiredSecurityProperties** | This field describes the required security properties to enable virtualization-based security. | ○ **0.** Nothing is required.<br>○ **1.** If present, hypervisor support is needed. |

| | | |
|---|---|---|
| | | ○ **2.** If present, Secure Boot is needed.<br>○ **3.** If present, DMA protection is needed.<br>○ **4.** If present, Secure Memory Overwrite is needed.<br>○ **5.** If present, NX protections are needed.<br>○ **6.** If present, SMM mitigations are needed.<br><br>**Note**: 4, 5, and 6 were added as of Windows 10, version 1607. |
| **SecurityServicesConfigured** | This field indicates whether the Credential Guard or HVCI service has been configured. | ○ **0.** No services configured.<br>○ **1.** If present, Credential Guard is configured.<br>○ **2.** If present, HVCI is configured. |
| **SecurityServicesRunning** | This field indicates whether the Credential Guard or HVCI service is running. | ○ **0.** No services running.<br>○ **1.** If present, Credential Guard is running.<br>○ **2.** If present, HVCI is running. |
| **Version** | This field lists the version of this WMI class. | The only valid value now is **1.0**. |
| **VirtualizationBasedSecurityStatus** | This field indicates whether VBS is enabled and running. | ○ **0.** VBS is not enabled.<br>○ **1.** VBS is enabled but not running.<br>○ **2.** VBS is enabled and running. |
| **PSComputerName** | This field lists the computer name. | All valid values for computer name. |

**Registry – die Einstellungen finden wir hier:**

Computer\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\DeviceGuard

Das Ziel ist es nun, nur noch die Anwendungen laufen zu lassen, die über eine Richtlinie vorab definiert wurden (Whitelisting).

Solche Richtlinien heißen „Codeintegritätsrichtlinien". Codeintegritätsrichtlinien schützen uns vor ungewollten und gefährlichen Anwendungen.

Code-Integrität = Windows Defender Application Control

Microsoft stellt eine Anleitung zum Aufbau einer CI Policy bereit:

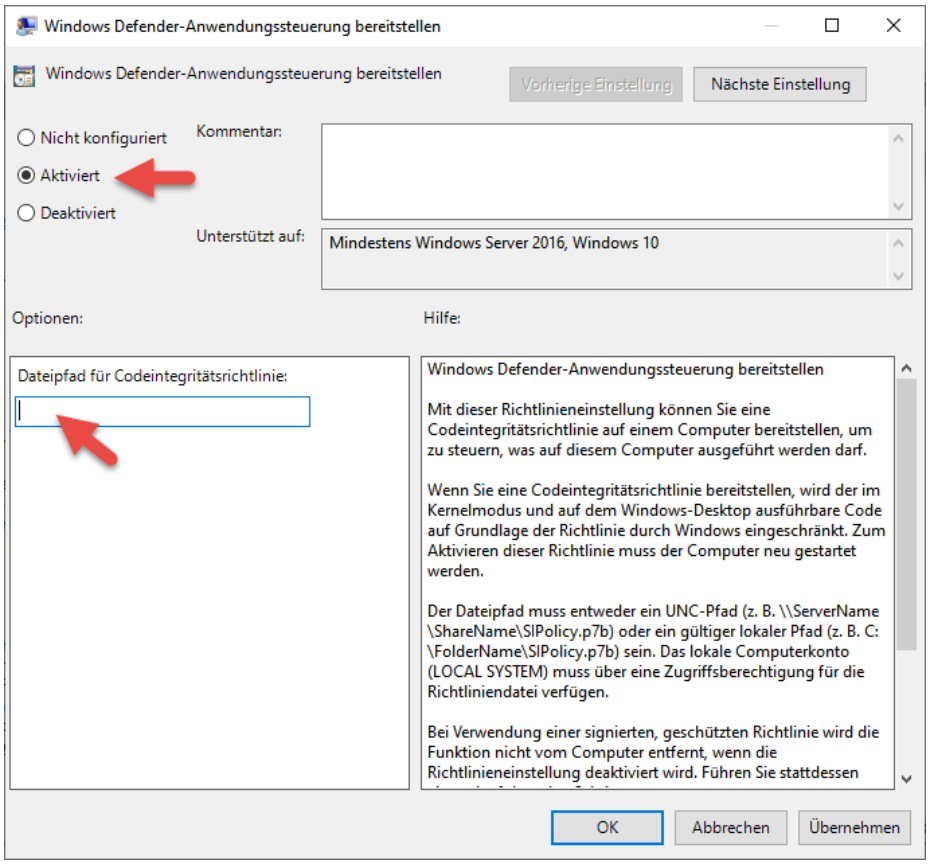
https://docs.microsoft.com/en-us/powershell/module/configci/new-cipolicy?view=win10-ps

https://docs.microsoft.com/en-us/powershell/module/configci/set-ruleoption?view=win10-ps

https://docs.microsoft.com/de-de/windows/security/threat-protection/windows-defender-application-control/create-initial-default-policy

Nach dem Erstellen einer solchen Richtlinie, muss diese auf dem oder die Computer bereitgestellt gestellt werden.

Bei einer zentralisierten Bereitstellung sollte ein Netzwerkpfad angegeben werden. Handelt es sich um eine Standalone Maschine z.B. einem Kiosk-Rechner, so kann die Policy lokal liegen, aber bitte mit den entsprechenden NTFS-Rechten geschützt.



**Beispiel:**

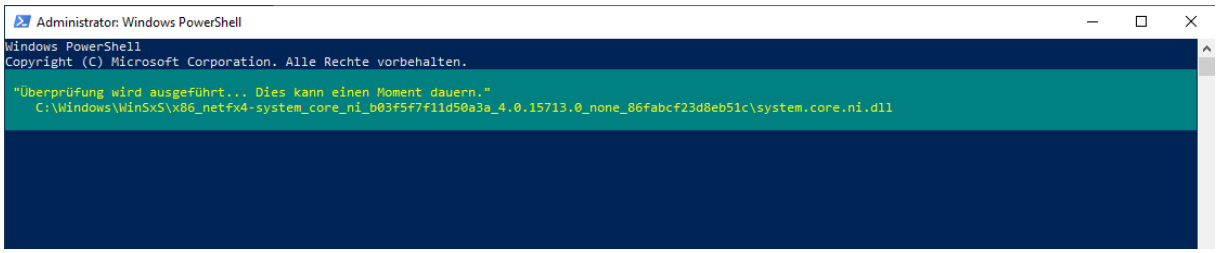
Möchten ich einen Kiosk-Rechner schützen, der fertig installiert und konfiguriert ist, so kann ich mir aus dem aktuellen Stand der Maschine eine Policy erstellen lassen:

Die Powershell mit administrativen Rechten öffnen.

Schritt 1)

Erstellen einer neuen Policy.

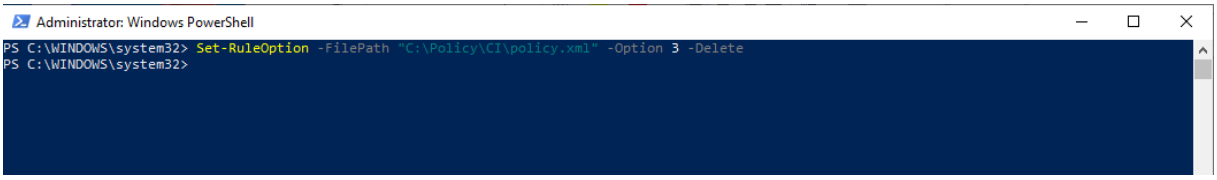
```
$dg_policy = "C:\Policy\CI\policy.xml"
$dg_policyBIN = "C:\Policy\CI\policy.p7b"
New-CIPolicy -FilePath $dg_policy -UserPEs -Level Publisher -Fallback Hash
```




Schritt 2)

Nun deaktivieren wir den Audit Mode

```
Set-RuleOption -FilePath "C:\Policy\CI\policy.xml" -Option 3 -Delete
```



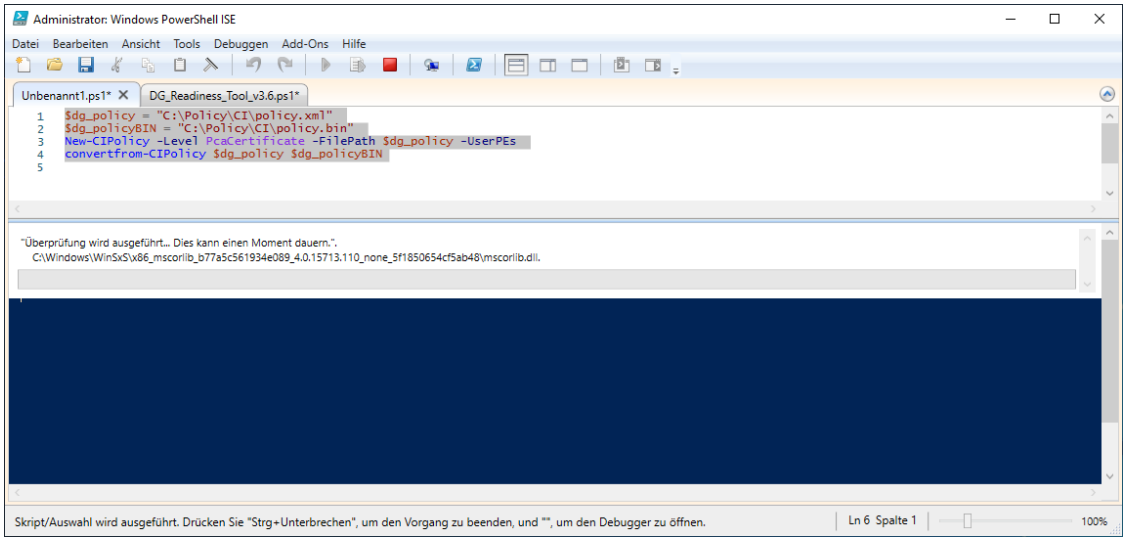

Schritt 3)

Konvertieren die neue Policy *.xml in eine binäre Datei.

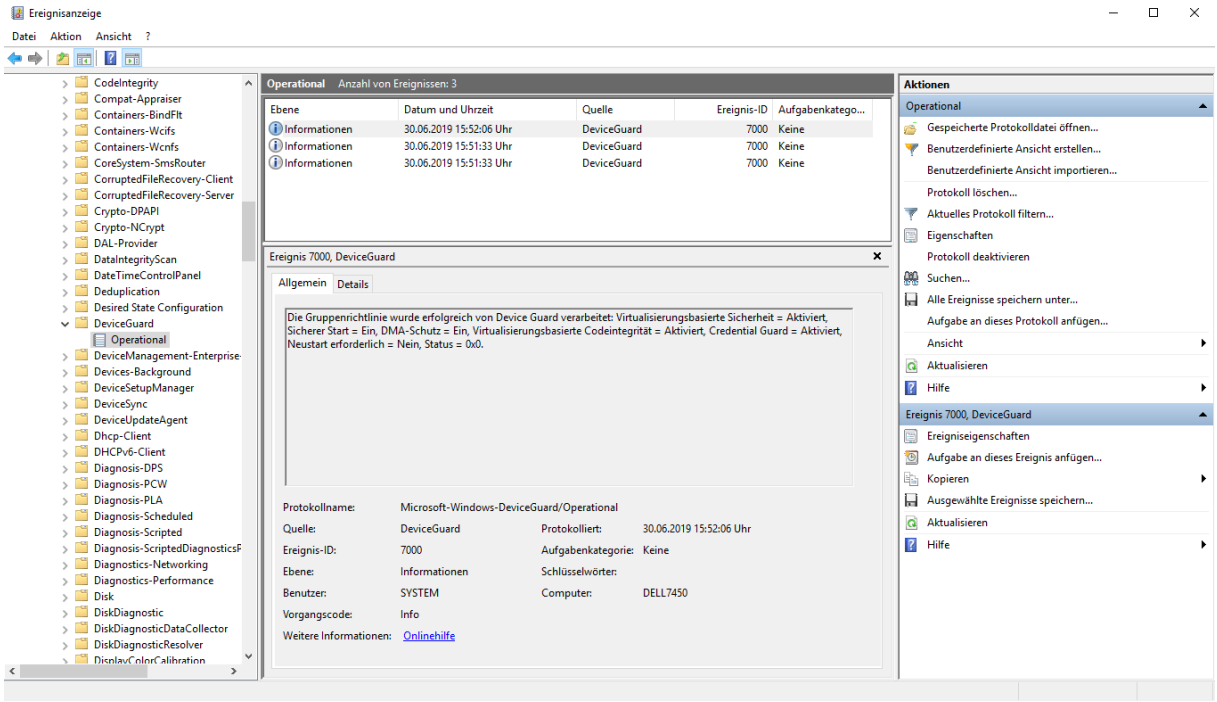
```
convertfrom-CIPolicy $dg_policy $dg_policyBIN

#ConvertFrom-CIPolicy -XmlFilePath "C:\Policy\CI\policy.xml"  -BinaryFilePath
"C:\Policy\CI\policy.p7b"
```

Mit diesem Befehl wird nach signierten Binärdaten gesucht, ausführbaren Dateien oder DLLs.

```
$dg_policy = "C:\Policy\CI\policy.xml"
$dg_policyBIN = "C:\Policy\CI\policy.bin"
New-CIPolicy -Level PcaCertificate -FilePath $dg_policy -UserPEs
convertfrom-CIPolicy $dg_policy $dg_policyBIN
```

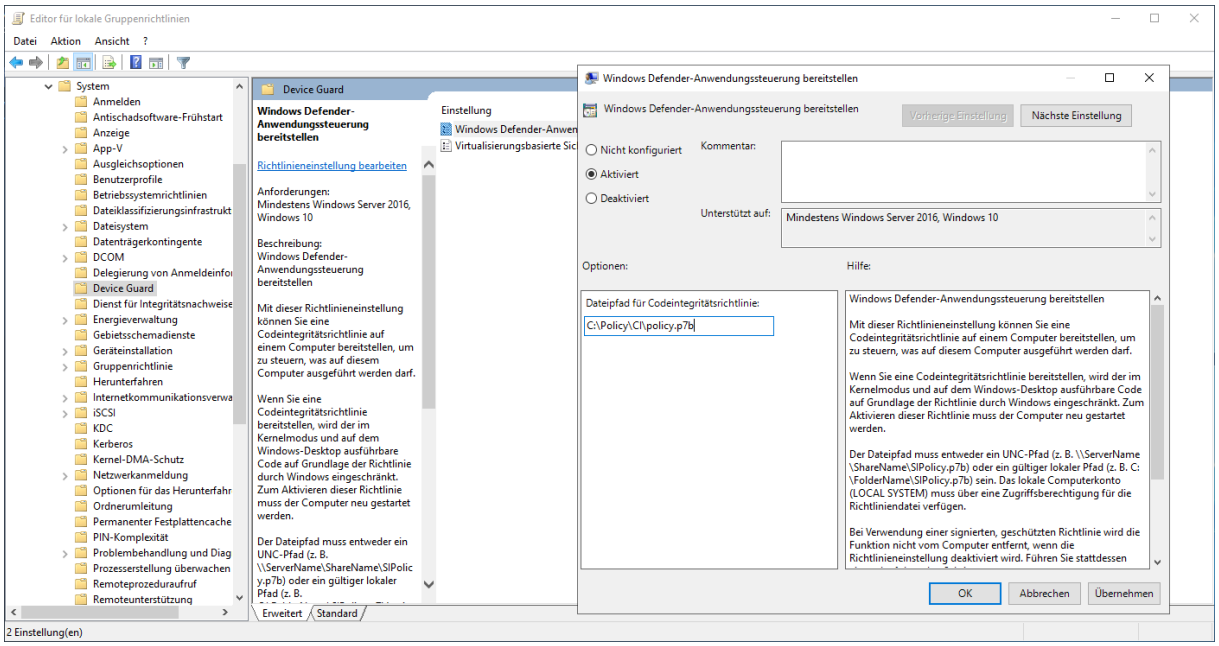



Die Events in der Ereignisanzeige finden wir an diesen Stellen:

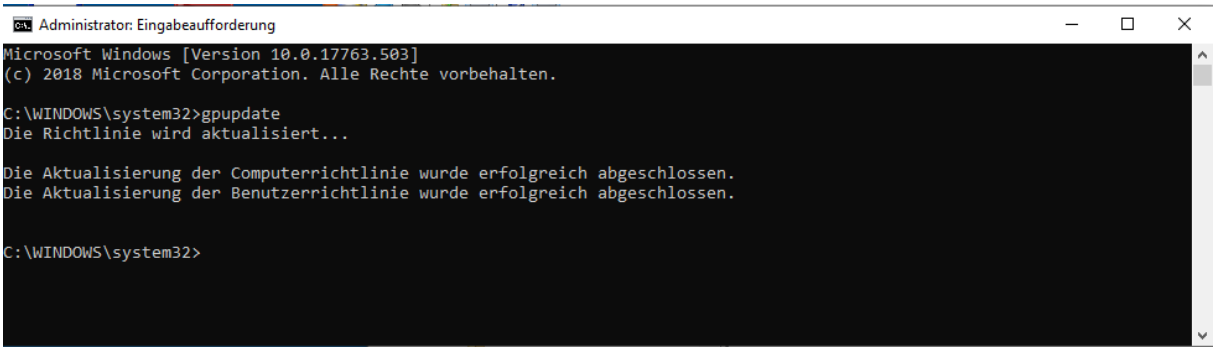


## Virtual Secure Mode und Device Guard

Nachdem die erhobenen Informationen als .xml Datei vorliegt, wird diese im letzten Schritt in eine Binärdatei *.p7b umgewandelt.
Die Policy ist nun fertig für den Einsatz in einer Gruppenrichtlinie.



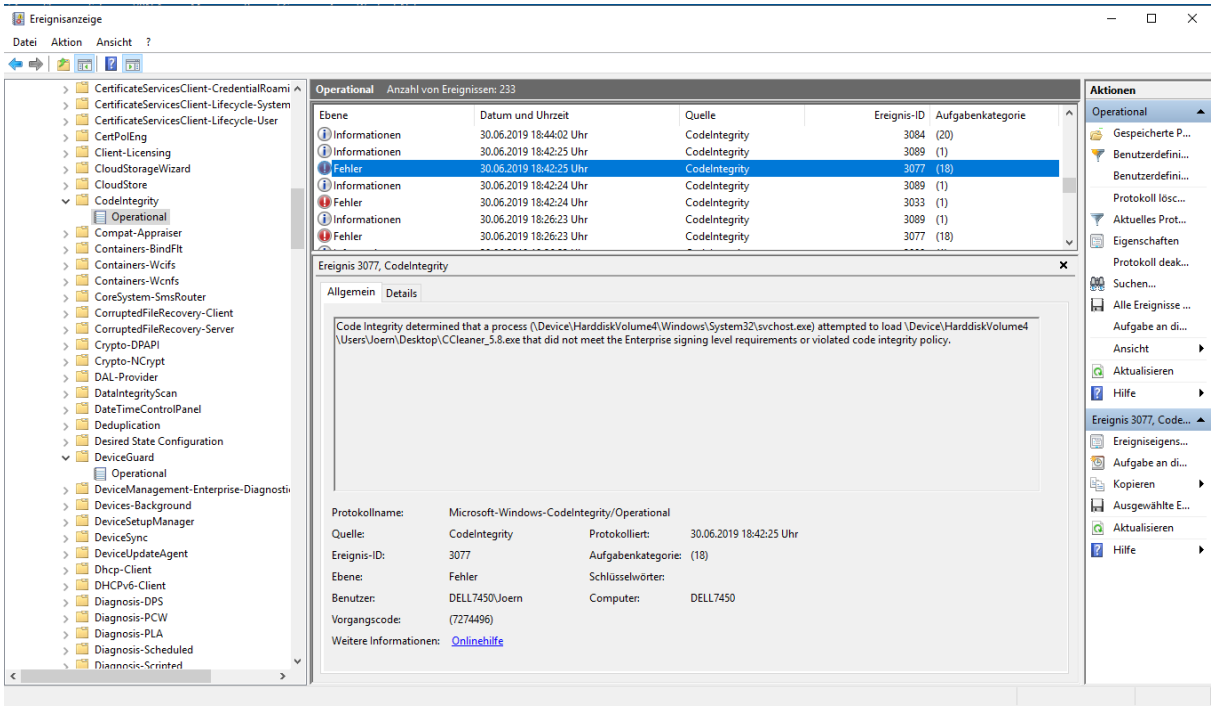

Gruppenrichtlinienupdate durchführen und neu starten. Doppelt hält besser!




Starte ich jetzt eine Applikation die nicht in der Policy als Whitelisting enthalten ist, so wird die Ausführung blockiert.

## Virtual Secure Mode und Device Guard

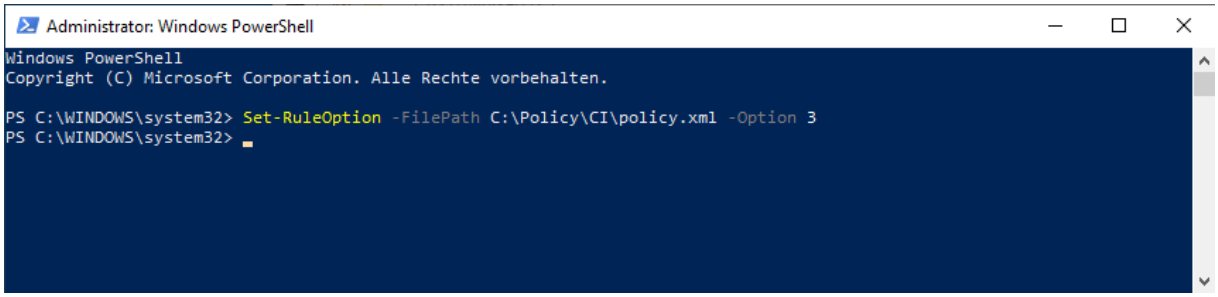

Im Log erscheint eine entsprechende Fehlermeldung.



### Update der Policy.xml

Soll auf der Maschine eine neue Software installiert werden, was ja aktuell nicht mehr funktionieren würde, so muss man Code Integrity wieder in den Audit Mode versetzen.

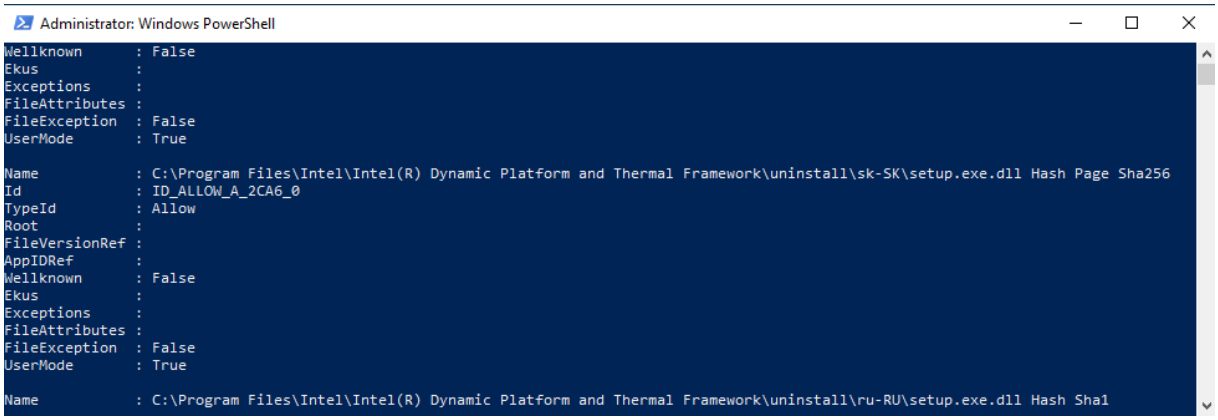
Set-RuleOption -FilePath C:\Policy\CI\policy.xml -Option 3




```
PS C:\Policy\ci> Set-RuleOption -help
0 Enabled:UMCI
1 Enabled:Boot Menu Protection
2 Required:WHQL
3 Enabled:Audit Mode
4 Disabled:Flight Signing
5 Enabled:Inherit Default Policy
6 Enabled:Unsigned System Integrity Policy
7 Allowed:Debug Policy Augmented
8 Required:EV Signers
9 Enabled:Advanced Boot Options Menu
10 Enabled:Boot Audit On Failure
11 Disabled:Script Enforcement
12 Required:Enforce Store Applications
13 Enabled:Managed Installer
14 Enabled:Intelligent Security Graph Authorization
15 Enabled:Invalidate EAs on Reboot
16 Enabled:Update Policy No Reboot
```

Als nächstes erstellen wir eine neue Policy namens „Install" um mergen diese danach mit der bereits vorhanden Policy namens Policy.xml.

Zur Demonstration habe ich unter C:\ einen neuen Ordner namens Tools erstellt und 2 Portable Programme abgelegt, die gerade eben noch blockiert wurden. Diese möchte ich nun in meine bereits erstellte Policy mit aufnehmen, damit sie später benutzbar werden, also nicht mehr blockiert werden.

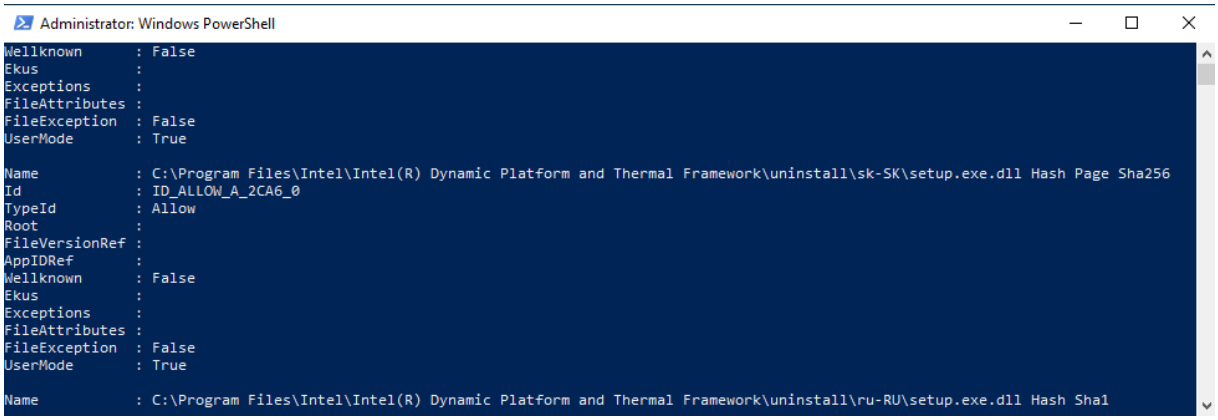


New-CIPolicy -FilePath "C:\Install\Install.xml" -Level FilePublisher -Fallback Hash -UserPEs -ScanPath "C:\Tools\"

Der Parameter -OmitPaths .\Install\NoScan\ schließt zu scannende Ordner aus.


Den Merge Vorgang starten.

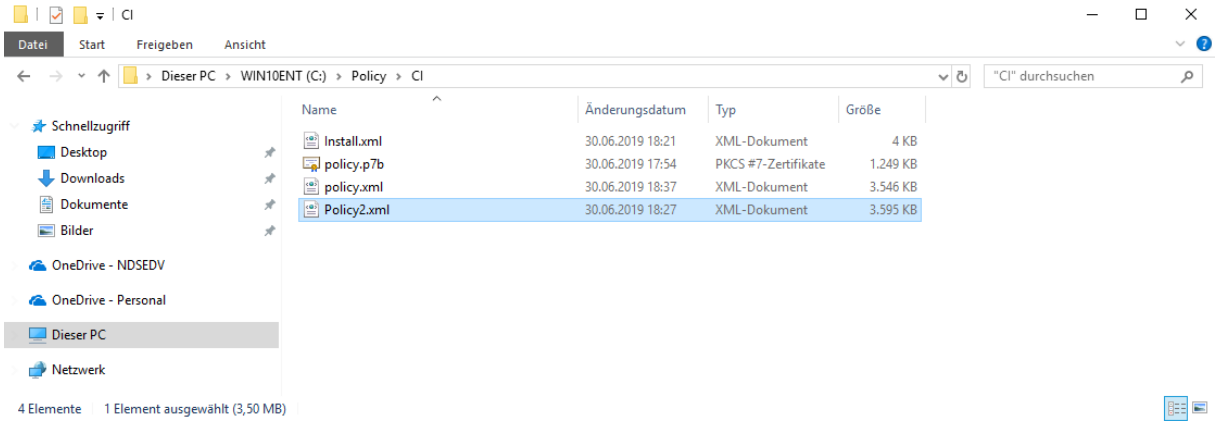
Merge-CIPolicy Policy.xml, Install.xml -OutputFilePath Policy2.xml

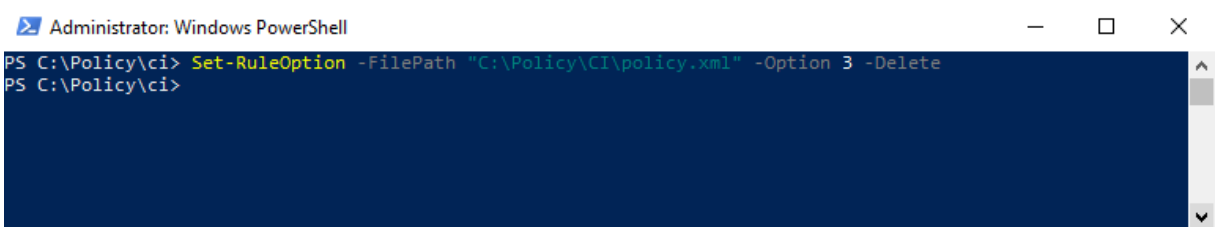



Den Audit Mode wieder deaktivieren.

Set-RuleOption -FilePath "C:\Policy\CI\policy.xml" -Option 3 -Delete

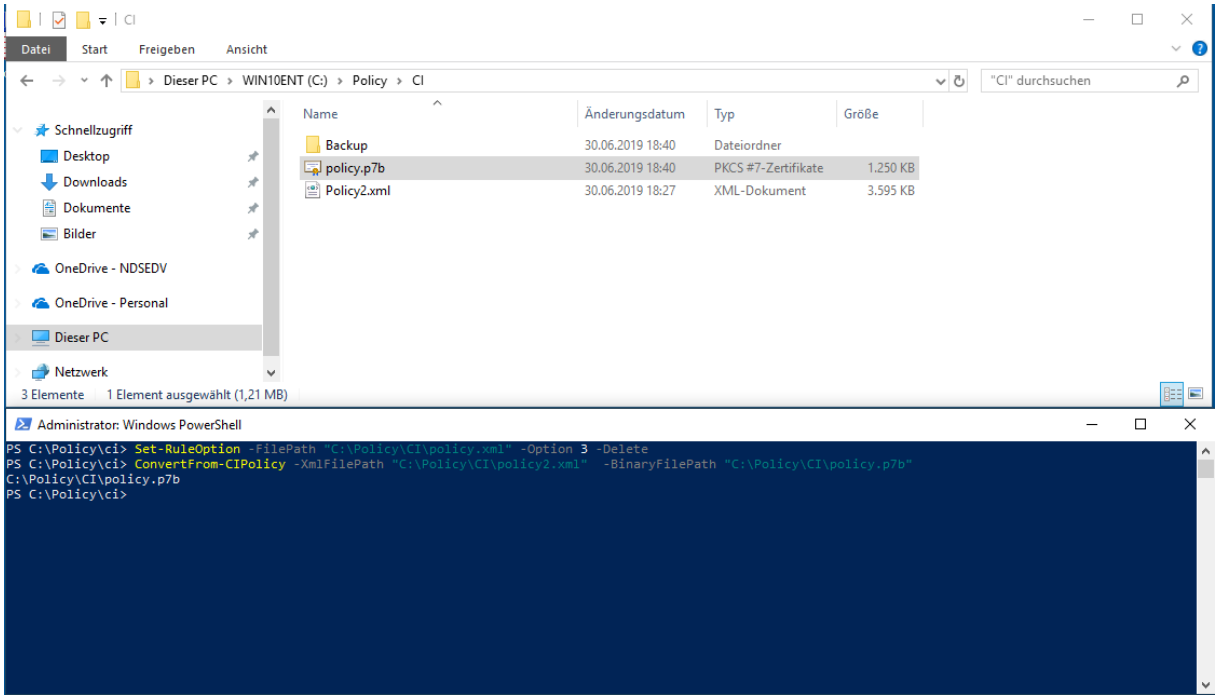
Es ist auch klar zu erkennen, das die Policy nach dem mergen größer geworden ist.



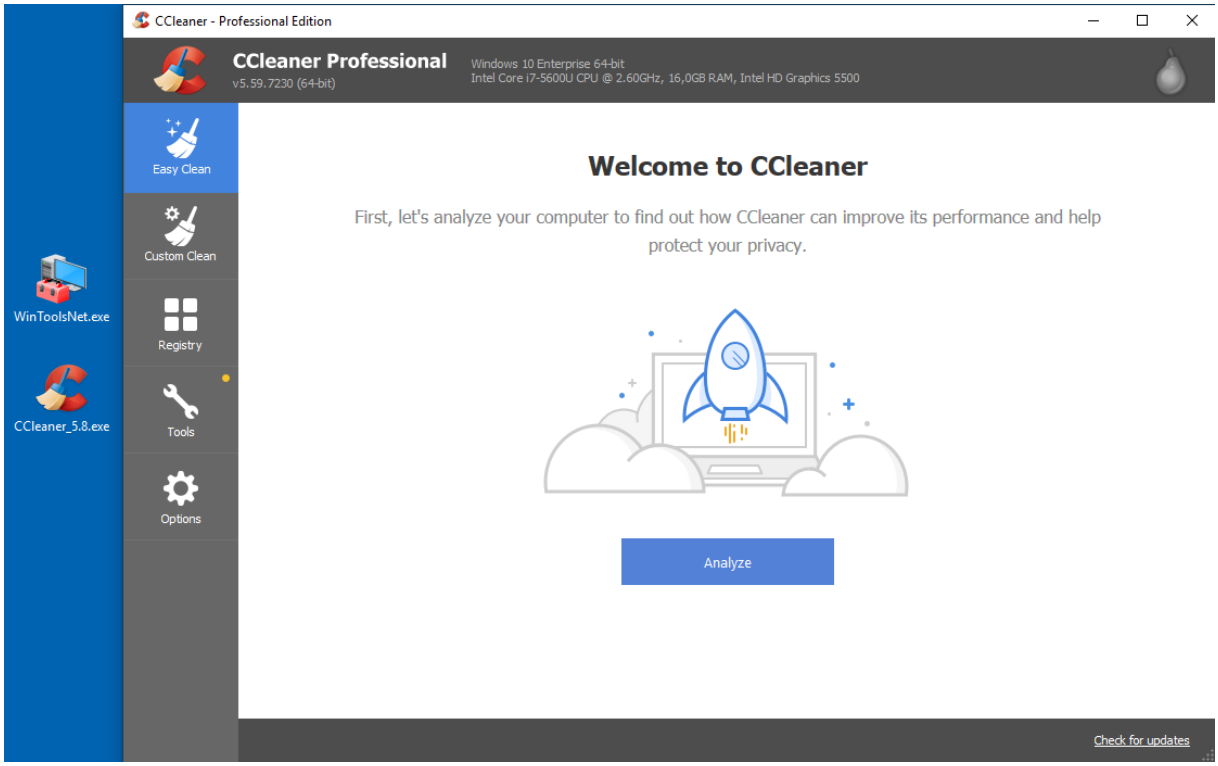
Abschließend wird die neue Policy2.xml wieder in eine binäre Datei umgewandelt.

```
ConvertFrom-CIPolicy -XmlFilePath "C:\Policy\CI\policy2.xml"  -BinaryFilePath "C:\Policy\CI\policy.p7b"
```




Starte den Rechner einmal durch.

Voila, die beiden Apps lassen sich nun ausführen.



# Virtual Secure Mode und Device Guard

**Erstellt von Jörn Walter**

**30.06.2019**