Windows Defender Application Control: Image verification

Aleksandar Milenkoski[✉]
amilenkoski@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 Device Guard Image Integrity, 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

Windows 10 performs WDAC verification (i.e., verification of images conducted by WDAC) in the *CiEvaluatePolicyInfo* and *CipApplySIPolicyUMCI* functions, implemented in *ci.dll*. Both functions ultimately invoke the *ci.dll* function *SIPolicyValidateImage*. Figure 1 and Figure 2 depict sample function call stacks resulting in the invocation of *SIPolicyValidateImage* by *CiEvaluatePolicyInfo* and *CipApplySIPolicyUMCI*.

All digitally signed critical system images, which include the Windows kernel and drivers loaded during the Windows boot process, are subjected to non-configurable code integrity verification when loaded. If the non-configurable code integrity verification succeeds, WDAC verification is performed. This is done by invoking *CiEvaluatePolicyInfo* which invokes *SIPolicyValidateImage*. WDAC verification takes place only if WDAC is enabled (i.e., if a WDAC policy is deployed). If the non-configurable or the WDAC verification fails, the verified image is not loaded. All critical system images are digitally signed by Microsoft.

All other digitally signed images, such as third-party images operating in user-mode, are also subjected to non-configurable code integrity verification when loaded. After non-configurable integrity verification, WDAC verification is performed. This is done by conditionally invoking *CiEvaluatePolicyInfo* and/or *CipApplySIPolicyUMCI*, which invoke *SIPolicyValidateImage*. WDAC verification takes place only if WDAC is enabled (i.e., if a WDAC policy is deployed). If the non-configurable integrity verification had failed, for all policy levels, except *Hash* and

```
Breakpoint 0 hit
CI!SIPolicyValidateImage:
fffff80a`76d9f084 4c894c2420      mov     qword ptr [rsp+20h],r9
kd> kc
 # Call Site
00 CI!SIPolicyValidateImage
01 CI!CipApplySiPolicyEx
02 CI!CiEvaluatePolicyInfo
[...]
06 CI!CipValidateImageHash
07 CI!CiValidateImageHeader
[...]
19 nt!Phase1Initialization
1a nt!PspSystemThreadStartup
1b nt!KiStartSystemThread
```

Figure 1: Function stack: Invoking SIPolicyValidateImage by CiEvaluatePolicyInfo

```
Breakpoint 1 hit
CI!SIPolicyValidateImage:
fffff803`c3c3f084 4c894c2420      mov     qword ptr [rsp+20h],r9
kd> kc
 # Call Site
00 CI!SIPolicyValidateImage
01 CI!CipApplySiPolicyEx
02 CI!CipApplySIPolicyUMCI
03 CI!CipValidateImageHash
[....]
08 nt!MmCreateSection
[....]
11 ntdll!LdrLoadDll
[....]
```

Figure 2: Function stack: Invoking SIPolicyValidateImage by CipApplySiPolicyUMCI

FileName, WDAC blocks the execution of the image in *CipApplySIPolicyUMCI*. For the policy levels *Hash* and *FileName*, WDAC blocks, or allows, the execution of the image in *CipApplySIPolicyUMCI* only if WDAC verification fails, or succeeds, respectively. *CipApplySIPolicyUMCI* is not invoked when a critical system image is verified.

If a verified image is not digitally signed, WDAC performs image verification in an identical manner as in the scenario when the image is signed and the policy level *Hash* or *FileName* is configured. That is, WDAC blocks, or allows, the execution of the image in *CipApplySIPolicyUMCI* if WDAC verification fails, or succeeds, respectively. For unsigned images, only the policy levels *Hash* or *FileName* may be configured.

2 Image verification in *SIPolicyValidateImage*

SIPolicyValidateImage verifies images based on data stored in an initialized WDAC policy and brings the decision whether an image is allowed to execute. This section provides an overview of the working principles of *SIPolicyValidateImage*.

SIPolicyValidateImage verifies an image based on comparing:

- data stored in a deployed WDAC policy as part of file rules; with
- verification data associated with the image being verified. This work refers to this data as image verification data.

What image verification data is compared in *SIPolicyValidateImage* depends on the policy levels configured in the deployed WDAC policy. For example, this data involves the image's file name if the policy level *FileName* is configured.

SIPolicyValidateImage compares data stored in the deployed WDAC policy with image verification data using standard data comparison functions, such as memory and string comparison functions. Examples include *memcmp* and *RtlEqualUnicodeString*.[1,2] *SIPolicyValidateImage* accesses the content of the deployed WDAC policy through the *ci.dll* variable *g_SiPolicyHandles*. This variable stores at the offset *0x6C* the content of the policy in binary format. *g_SiPolicyHandles* is passed as the first parameter of *SIPolicyValidateImage*.

g_SiPolicyHandles is populated with the content of the deployed WDAC policy in the *SIPolicyInitialize* and *SIPolicySetActivePolicy* functions. These functions are invoked during the initialization of code integrity. Label [1] in Figure 3 depicts *SIPolicyInitialize* storing the content of a policy in *g_SiPolicyHandles*. Label [2] in Figure 3 depicts the policy content, stored in *g_SiPolicyHandles*, passed as a parameter to *SIPolicyValidateImage*. Label [3] in Figure 3 depicts the policy content as viewed with the *HxD* hex editor, in the context of the Windows 10 instance where the policy is deployed.

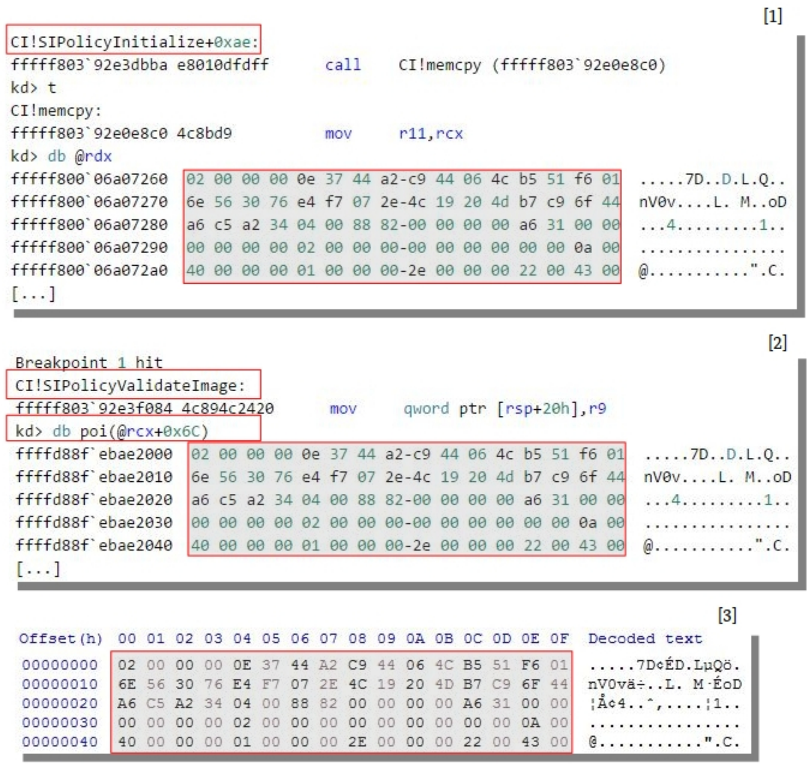


Figure 3: The content of a deployed DeviceGuard policy in different contexts

Image verification data is passed to *SIPolicyValidateImage* in the form of a structure, referred to as validation context. The validation context stores the image verification data that may be relevant when verifying an image based on any policy level. For example, the validation context stores:

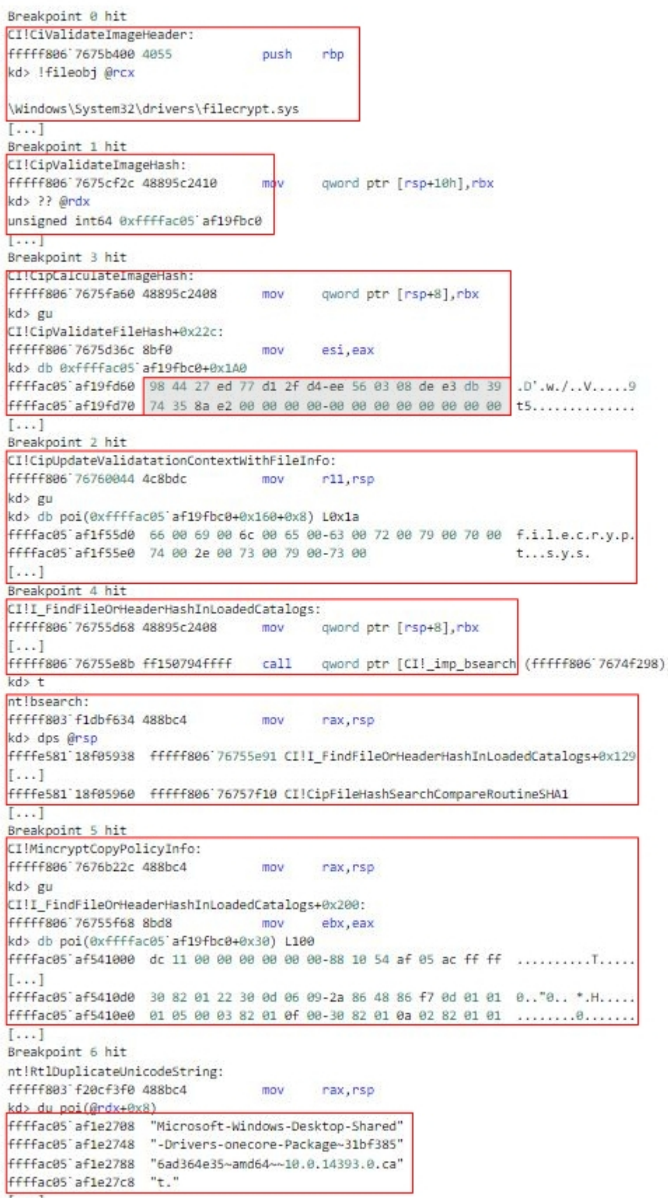
- at offset *0x30* data related to the certificate chain used to sign the image being verified, including the leaf (i.e., the signer's certificate) and the *PCAcertificate*. This data is relevant when an image is verified based on the *Leaf* and *Publisher* policy levels;
- at offset *0x160* the name of the image being verified. This data is relevant when an image is verified based on the *FileName* policy level;
- at offset *0x1A0* the hash value of the image being verified. This data is relevant when an image is verified based on the *Hash* policy level.

---

[1] https://docs.microsoft.com/en-us/cpp/c-runtime-library/reference/memcmp-wmemcmp?view=vs-2019 [Retrieved: 6/10/2019]

[2] https://docs.microsoft.com/en-us/windows-hardware/drivers/ddi/content/wdm/nf-wdm-rtlequalunicodestring [Retrieved: 6/10/2019]

The validation context is populated with data in multiple functions that are invoked before *SIPolicyValidateImage*. As an example, Figure 4 depicts Windows 10 populating the validation context with image verification data when the image *filecrypt.sys* is verified. This image is verified against a WDAC policy with the *PcaCertificate* policy level configured. The validation context is initialized in the *CipValidateImageHash* function, at the address *0xffffac05af19fbc0*. Once the validation context is initialized, the functions *CipCalculateImageHash*, *CipUpdateValidatationContextWithFileInfo*, and *MinCryptCopyPolicyInfo* populate the offsets *0x1A0*, *0x160*, and *0x30* of the validation context, respectively. *CipCalculateImageHash* calculates the image's hash value and stores it at the offset *0x1A0* of the validation context (see Figure 4, function *CipCalculateImageHash*). *CipUpdateValidatationContextWithFileInfo* extracts the name of the image from its properties and stores it at the offset *0x160* of the validation context (see Figure 4, function *CipUpdateValidatationContextWithFileInfo*). *CipUpdateValidatationContextWithFileInfo* extracts the image's name and version from the image itself, by invoking the *SIPolicyGetOriginalFilenameAndVersionFromImage* function.



```
Breakpoint 0 hit
CI!CiValidateImageHeader:
fffff806`7675b400 4055            push    rbp
kd> !fileobj @rcx

\Windows\System32\drivers\filecrypt.sys
[...]
Breakpoint 1 hit
CI!CipValidateImageHash:
fffff806`7675cf2c 48895c2410      mov     qword ptr [rsp+10h],rbx
kd> ?? @rdx
unsigned int64 0xffffac05`af19fbc0
[...]
Breakpoint 3 hit
CI!CipCalculateImageHash:
fffff806`7675fa60 48895c2408      mov     qword ptr [rsp+8],rbx
kd> gu
CI!CipValidateFileHash+0x22c:
fffff806`7675d36c 8bf0            mov     esi,eax
kd> db 0xffffac05`af19fbc0+0x1A0
ffffac05`af19fd60  98 44 27 ed 77 d1 2f d4-ee 56 03 08 de e3 db 39  .D'.w./..V.....9
ffffac05`af19fd70  74 35 8a e2 00 00 00 00-00 00 00 00 00 00 00 00  t5..............
[...]
Breakpoint 2 hit
CI!CipUpdateValidatationContextWithFileInfo:
fffff806`76760044 4c8bdc          mov     r11,rsp
kd> gu
kd> db poi(0xffffac05`af19fbc0+0x160+0x8) L0x1a
ffffac05`af1f55d0  66 00 69 00 6c 00 65 00-63 00 72 00 79 00 70 00  f.i.l.e.c.r.y.p.
ffffac05`af1f55e0  74 00 2e 00 73 00 79 00-73 00                    t...s.y.s.
[...]
Breakpoint 4 hit
CI!I_FindFileOrHeaderHashInLoadedCatalogs:
fffff806`76755d68 48895c2408      mov     qword ptr [rsp+8],rbx
[...]
fffff806`76755e8b ff150794ffff    call    qword ptr [CI!_imp_bsearch (fffff806`7674f298)]
kd> t
nt!bsearch:
fffff803`f1dbf634 488bc4          mov     rax,rsp
kd> dps @rsp
ffffe581`18f05938  fffff806`76755e91 CI!I_FindFileOrHeaderHashInLoadedCatalogs+0x129
[...]
ffffe581`18f05960  fffff806`76757f10 CI!CipFileHashSearchCompareRoutineSHA1
[...]
Breakpoint 5 hit
CI!MincryptCopyPolicyInfo:
fffff806`7676b22c 488bc4          mov     rax,rsp
kd> gu
CI!I_FindFileOrHeaderHashInLoadedCatalogs+0x200:
fffff806`76755f68 8bd8            mov     ebx,eax
kd> db poi(0xffffac05`af19fbc0+0x30) L100
ffffac05`af541000  dc 11 00 00 00 00 00 00-88 10 54 af 05 ac ff ff  ..........T.....
[...]
ffffac05`af5410d0  30 82 01 22 30 0d 06 09-2a 86 48 86 f7 0d 01 01  0.."0...*.H.....
ffffac05`af5410e0  01 05 00 03 82 01 0f 00-30 82 01 0a 02 82 01 01  ........0.......
[...]
Breakpoint 6 hit
nt!RtlDuplicateUnicodeString:
fffff803`f20cf3f0 488bc4          mov     rax,rsp
kd> du poi(@rdx+0x8)
ffffac05`af1e2708  "Microsoft-Windows-Desktop-Shared"
ffffac05`af1e2748  "-Drivers-onecore-Package~31bf385"
ffffac05`af1e2788  "6ad364e35~amd64~~10.0.14393.0.ca"
ffffac05`af1e27c8  "t."
[...]
```

Figure 4: Populating the validation context with image integrity verification data

filecrypt.sys is signed through a catalog file. Windows 10 uses catalogs to associate Authenticode signatures with a given image.[3] Catalogs are files that contain a set of file hashes such that each hash identifies a specific image. The catalog file itself is signed with an Authenticode signature. Therefore, a single catalog file serves as a detached signature that may be associated with multiple images.[4] The data related to the certificate chain used to sign the *filecrypt.sys* image is extracted from the Authenticode signature embedded in the catalog. This data is relevant for image verification, for example, when the *PcaCertificate* policy level is configured. The *I_FindFileOrHeaderHashInLoadedCatalogs* function searches through deployed catalog files for the image's hash value previously calculated by *CipCalculateImageHash*. In Windows 10, deployed catalog files are stored in the *%SystemRoot%\System32\CatRoot* folder. *I_FindFileOrHeaderHashInLoadedCatalogs* performs binary search in order to locate the image's hash value stored in a catalog (see *bsearch* in Figure 4).[5]

Since *CipCalculateImageHash* has calculated a Secure Hash Algorithm (SHA)-1 hash value of the image, *bsearch* invokes the *CipFileHashSearchCompareRoutineSHA1* function in order to locate the SHA-1 hash in a catalog. The SHA-1 hash of *filecrypt.sys* is stored in the catalog *Microsoft-Windows-Desktop-Shared-Drivers-onecore-Package~31bf3856ad364e35~amd64~~10.0.14393.0.cat* (see Figure 4, function *RtlDuplicateUnicodeString*). Figure 5 depicts the SHA-1 hash of *filecrypt.sys* stored in this catalog as viewed with the *HxD* hex editor (see also Figure 4, function *CipCalculateImageHash*).

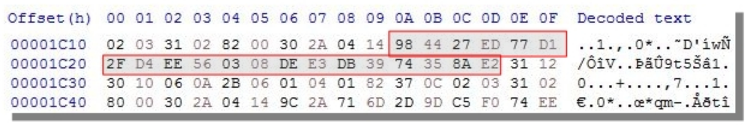


Figure 5: SHA-1 hash of filecrypt.sys stored in a catalog

Once *FindFileOrHeaderHashInLoadedCatalogs* has located the catalog, *MinCryptCopyPolicyInfo* stores data related to the certificate chain used to sign the catalog at the offset *0x30* of validation context (see Figure 4, function *MinCryptCopyPolicyInfo*). This includes the certificate chain itself. The binary sequence beginning with *30 82* depicted in Figure 4 mark certificate content encoded in the *ASN.1* format.

Once *MinCryptCopyPolicyInfo* has stored in the validation context data related to the certificate chain used to sign the catalog, *SIPolicyValidateImage* compares this data with data stored in the deployed WDAC policy (see Figure 6, label [1], function *memcmp*). The data *SIPolicyValidateImage* compares is a hash of the *TbsCertificate* field of the PCAcertificate used to sign *Microsoft-Windows-Desktop-Shared-Drivers-onecore-Package~31bf3856ad364e35~amd64~~10.0.14393.0.cat* (*4e 80 be...* in Figure 6, label [1]).[6] Figure 6, label [2], depicts the extraction of the *TbsCertificate* field from this certificate with the *openssl* and *dd* utilities. It also depicts the calculated SHA-256 hash of the extracted *TbsCertificate* field with the *sha256sum* utility.

Figure 6 shows that the data related to the certificate chain used to sign *filecrypt.sys*, compared in *SIPolicyValidateImage*, originates from the catalog that represents a detached signature of *filecrypt.sys*. In general, the discussion above shows that the image verification data compared in *SIPolicyValidateImage* originates either from the image being verified itself, or from the catalog serving as the image's detached signature.

Once the validation context has been populated with data, it is passed to *SIPolicyValidateImage* for comparison with data originating from the deployed WDAC policy. The text blocks below provide an insight into the operation of *SIPolicyValidateImage*. They provide an overview of *SIPolicyValidateImage* comparing image verification data with data stored in a deployed WDAC policy when the policy levels *Hash*, *PcaCertificate*, and *Publisher* are configured (see also Figure 6). In the text blocks below:

- in the label *Policy* and *integrity verification*:

[3] https://docs.microsoft.com/en-us/windows-hardware/drivers/install/authenticode [Retrieved: 6/10/2019]

[4] https://docs.microsoft.com/en-us/windows-hardware/drivers/install/catalog-files [Retrieved: 6/10/2019]

[5] https://docs.microsoft.com/en-us/cpp/c-runtime-library/reference/bsearch?view=vs-2019 [Retrieved: 6/10/2019]

[6] https://tools.ietf.org/html/rfc5280#section-4.1.1.1

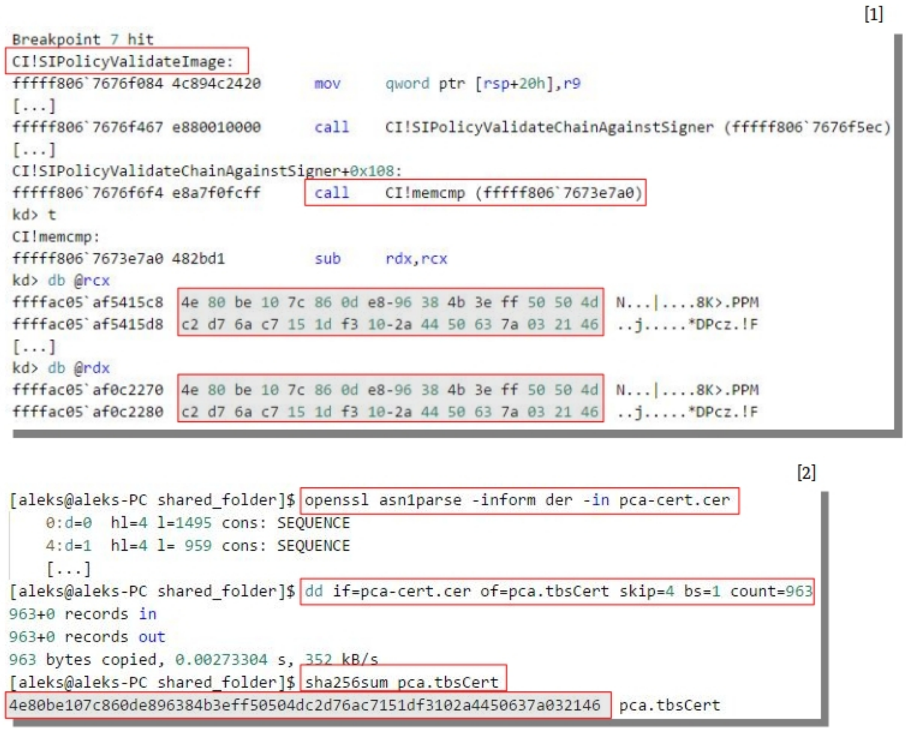


Figure 6: SIPolicyValidateImage comparing certificate data

- the field *Policy level* holds the configured policy level and the field *Policy generation* holds the command used to create the WDAC policy.[7] *$InitialCIPolicy* is a variable that specifies the path to the file in which the generated policy is to be stored;
- the field *Verified data* holds the data based on which the image is verified;
- the field *Verification function* holds the name of the function invoked by *SIPolicyValidateImage* in which data originating from the WDAC policy and image verification data is compared; and
- the field *Comparison function(s)* holds the name of the function, or functions, that compare data originating from the WDAC policy with image verification data;

• in the label *Depiction* is placed a figure:
  - the label [1] of this figure depicts the execution of *SIPolicyValidateImage*, with a focus on what is documented in the fields *Verified data*, *Verification function*, and *Comparison function(s)*. This includes data stored in the validation context, data stored in the deployed WDAC policy, the function invoked by *SIPolicyValidateImage* in which this data is compared, and the function(s) with which the data is compared;
  - the label [2] of this figure depicts the data compared in *SIPolicyValidateImage*, extracted from the deployed WDAC policy in XML format with the *findstr* utility.

[7] https://docs.microsoft.com/en-us/powershell/module/configci/new-cipolicy?view=win10-ps [Retrieved: 6/10/2019]

Policy and integrity verification

Policy level:

Hash

Policy generation:

New-CIPolicy -Level Hash -FilePath $InitialCIPolicy –UserPEs

Verified data:

the image's hash value

Verification function:

SIPolicyMatchFileRules

Comparison function(s):

memcmp

Depiction

[1]

```
Breakpoint 0 hit
CI!SIPolicyValidateImage:
[...]
kd> db poi ( poi(@rsp+0x38+0x8) + 0x8 ) L0x16
ffffb181`52dea060  64 00 75 00 6d 00 70 00-66 00 76 00 65 00 2e 00  d.u.m.p.f.v.e...
ffffb181`52dea070  73 00 79 00 73 00                                s.y.s.
[...]

CI!SIPolicyValidateImage+0x36c:
fffff804`d6cdf3f0 e853fafff       call    CI!SIPolicyMatchFileRules (fffff804`d6cdee48)
[...]
CI!SIPolicyMatchFileRules+0x1fc:
fffff804`d6cdf044 e857f7fcff      call    CI!memcmp (fffff804`d6cae7a0)
kd> t
CI!memcmp:
fffff804`d6cae7a0 482bd1          sub     rdx,rcx
kd> db @rcx
ffffb181`52d89eb0  21 17 ab 0b 78 6c 1e 89-5c 4e 28 33 0e 4a db 23  !...xl..\N(3.J.#
ffffb181`52d89ec0  d1 f5 de 41 71 ce 0f dd-f9 3c d3 1e 41 7d 91 a4  ...Aq....<..A}..
[...]
kd> db @rdx
ffffb181`528000ec  00 01 3a f8 35 30 1e 03-57 0c 85 20 67 fc 0f a0  ..:.50..W.. g...
ffffb181`528000fc  e7 17 65 94 67 f2 e2 ba-eb 39 30 47 ec 21 d7 54  ..e.g....90G.!.T
[...]
CI!memcmp:
fffff804`d6cae7a0 482bd1          sub     rdx,rcx
kd> db @rcx
ffffb181`52d89eb0  21 17 ab 0b 78 6c 1e 89-5c 4e 28 33 0e 4a db 23  !...xl..\N(3.J.#
ffffb181`52d89ec0  d1 f5 de 41 71 ce 0f dd-f9 3c d3 1e 41 7d 91 a4  ...Aq....<..A}..
[...]
kd> db @rdx
ffffb181`52800250  00 05 4c c3 27 30 2f 4e-35 2f d4 0d 7c 85 8c 02  ..L.'0/N5/..|...
ffffb181`52800260  7b 50 81 b1 7e 50 fd 39-71 73 11 8b da 4e 18 5e  {P..~P.9qs...N.^
[...]
```

[2]

```
C:\Users\ernw\Desktop>findstr /i "2117ab0b786c1e895c4e28330e4adb23" EnforcedPolicy.xml
    <Allow [...] FriendlyName="[...]dumpfve.sys Hash Sha256" Hash="2117AB0B786C1E895C4E28330E4ADB23
    D1F5DE4171CE0FDDF93CD31E417D91A4" />
    <Allow ID="ID_ALLOW_A_C946" FriendlyName="[...]dumpfve.sysdumpfve.sys Hash Sha256" Hash="2117AB0B786C1E895C4E28330E4ADB23
    D1F5DE4171CE0FDDF93CD31E417D91A4" />

C:\Users\ernw\Desktop>findstr /i "00013af835301e03570c852067fc0fa0" EnforcedPolicy.xml
    <Allow [...] FriendlyName="C:\Windows\[...]Activities.dll Hash Sha256" Hash="00013AF835301E03570C852067FC0FA0E717659
    467F2E2BAEB393047EC21D754" />
    <Allow [...] FriendlyName="C:\Windows\[...]Activities.dll Hash Sha256" Hash="00013AF835301E03570C852067FC0FA0E717659
    467F2E2BAEB393047EC21D754" />

C:\Users\ernw\Desktop>findstr /i "00054cc327302f4e352fd40d7c858c02" EnforcedPolicy.xml
    <Allow [...] FriendlyName="C:\Windows\[..]cemapi.dll.mui Hash Page Sha256" Hash="00054CC327302F4E352FD40D7C858C027B5081
    B17E50FD397173118BDA4E185E" />
    <Allow [...] FriendlyName="C:\Windows\[...]cemapi.dll.mui Hash Page Sha256" Hash="00054CC327302F4E352FD40D7C858C027B5081
    B17E50FD397173118BDA4E185E" />
```

Policy and integrity verification

Policy level:

PcaCertificate

Policy generation:

New-CIPolicy -Level PcaCertificate -FilePath $InitialCIPolicy –UserPEs

Verified data:

the hash value of the *TcbCertificate* field of the PCAcertificate

Verification function:

SIPolicyValidateChainAgainstSigner

Comparison function(s):

memcmp

Depiction

[1]

```
Breakpoint 0 hit
CI!SIPolicyValidateImage:
fffff806`1706f084 4c894c2420      mov     qword ptr [rsp+20h],r9
kd> db poi ( poi(@rsp+0x38+0x8) + 0x8 ) L0x18
ffff800d`79221b80  63 00 72 00 61 00 73 00-68 00 64 00 6d 00 70 00  c.r.a.s.h.d.m.p.
ffff800d`79221b90  2e 00 73 00 79 00 73 00                          ..s.y.s.
[...]
CI!SIPolicyValidateImage+0x3e3:
fffff806`1706f467 e880010000      call    CI!SIPolicyValidateChainAgainstSigner (fffff806`1706f5ec)
kd> t
[...]
CI!SIPolicyValidateChainAgainstSigner+0x108:
fffff806`1706f6f4 e8a7f0fcff      call    CI!memcmp (fffff806`1703e7a0)
kd> t
CI!memcmp:
fffff806`1703e7a0 482bd1          sub     rdx,rcx
kd> db @rcx
ffff800d`791a25c8  4e 80 be 10 7c 86 0d e8-96 38 4b 3e ff 50 50 4d  N...|....8K>.PPM
ffff800d`791a25d8  c2 d7 6a c7 15 1d f3 10-2a 44 50 63 7a 03 21 46  ..j.....*DPcz.!F
[...]
kd> db @rdx
ffff800d`790b588c  4e 80 be 10 7c 86 0d e8-96 38 4b 3e ff 50 50 4d  N...|....8K>.PPM
ffff800d`790b589c  c2 d7 6a c7 15 1d f3 10-2a 44 50 63 7a 03 21 46  ..j.....*DPcz.!F
[...]
```

[2]

```
C:\Users\ernw\Desktop>findstr /i "4e80be107c860de896384b3eff50504d" EnforcedPolicy_Pca.xml
      <CertRoot Type="TBS" Value="4E80BE107C860DE896384B3EFF50504DC2D76AC
      7151DF3102A4450637A032146" />
      <CertRoot Type="TBS" Value="4E80BE107C860DE896384B3EFF50504DC2D76AC
      7151DF3102A4450637A032146" />
```

Policy and integrity verification

Policy level:

Publisher

Policy generation:

New-CIPolicy -Level Publisher -FilePath $InitialCIPolicy –UserPEs

Verified data:

the CN field of the certificate of the image's signer the hash value of the *TcbCertificate* field of the PCAcertificate

Verification function:

SIPolicyValidateChainAgainstSigner

Comparison function(s):

memcmp, *RtlEqualUnicodeString*

Depiction

[1]

```
Breakpoint 0 hit
CI!SIPolicyValidateImage:
fffff808`0b21f084 4c894c2420      mov     qword ptr [rsp+20h],r9
kd> db poi ( poi(@rsp+0x38+0x8) + 0x8 ) L0x18
ffffd907`fcf172a0  73 00 74 00 6f 00 72 00-61 00 68 00 63 00 69 00  s.t.o.r.a.h.c.i.
ffffd907`fcf172b0  2e 00 73 00 79 00 73 00                          ..s.y.s.
[...]
CI!SIPolicyValidateImage+0x3e3:
fffff808`0b21f467 e880010000      call    CI!SIPolicyValidateChainAgainstSigner (fffff808`0b21f5ec)
[...]
CI!SIPolicyValidateChainAgainstSigner+0x108:
fffff808`0b21f6f4 e8a7f0fcff      call    CI!memcmp (fffff808`0b1ee7a0)
kd> t
CI!memcmp:
fffff808`0b1ee7a0 482bd1          sub     rdx,rcx
kd> db @rcx
ffffd907`fcde15c8  4e 80 be 10 7c 86 0d e8-96 38 4b 3e ff 50 50 4d  N...|....8K>.PPM
ffffd907`fcde15d8  c2 d7 6a c7 15 1d f3 10-2a 44 50 63 7a 03 21 46  ..j.....*DPcz.!F
[...]
kd> db @rdx
ffffd907`fccc6270  4e 80 be 10 7c 86 0d e8-96 38 4b 3e ff 50 50 4d  N...|....8K>.PPM
ffffd907`fccc6280  c2 d7 6a c7 15 1d f3 10-2a 44 50 63 7a 03 21 46  ..j.....*DPcz.!F
[...]
CI!SIPolicyValidateChainAgainstSigner+0x14a:
fffff808`0b21f736 e8abdffcff      call    CI!RtlEqualUnicodeString (fffff808`0b1ed6e6)
kd> t
CI!RtlEqualUnicodeString:
fffff808`0b1ed6e6 ff25141e0100    jmp     qword ptr [CI!_imp_RtlEqualUnicodeString (fffff808`0b1ff500)]
kd> t
nt!RtlEqualUnicodeString:
fffff800`75c72ae0 4883ec08        sub     rsp,8
kd> du poi(@rcx+0x8)
ffffd907`fcc94010  "Microsoft Windows"
kd> du poi(@rdx+0x8)
ffffd907`fccc62a0  "Microsoft Windows"
[...]
```

[2]

```
C:\Users\ernw\Desktop>findstr /i "4e80be107c860de896384b3eff50504d" EnforcedPolicy_Publisher.xml
        <CertRoot Type="TBS" Value="4E80BE107C860DE896384B3EFF50504DC2D76AC7151DF3102A4450637A032146" />
        <CertRoot Type="TBS" Value="4E80BE107C860DE896384B3EFF50504DC2D76AC7151DF3102A4450637A032146" />
        <CertRoot Type="TBS" Value="4E80BE107C860DE896384B3EFF50504DC2D76AC7151DF3102A4450637A032146" />
[...]

C:\Users\ernw\Desktop>findstr /i "CertPublisher.*Value.*Microsoft.*Windows.*" EnforcedPolicy_Publisher.xml
        <CertPublisher Value="Microsoft Windows" />
        <CertPublisher Value="Microsoft Windows" />
        <CertPublisher Value="Microsoft Windows 3rd party Component" />
        [...]
        <CertPublisher Value="Microsoft Windows Kits Publisher" />
        <CertPublisher Value="Microsoft Windows" />
```