ERNW
providing security.

The TPM: Integrity Measurement

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 the Trusted Platform Module (TPM) and early launch anti-malware (ELAM), 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).

The TPM standard in focus is version 2.0.

1 Introduction

In this work, we discuss the integrity measurement mechanism of Windows 10 and the role that the TPM plays as part of it. This mechanism, among other things, implements the production of measurement data. This involves calculation of hashes of relevant executable files or of code sequences at every system startup. It also involves the storage of these hashes and relevant related data in the TPM device and in log files for later analysis.

The analysis of measurement data is normally performed by a trusted remote platform, a platform different than the one where hashes have been calculated. The remote platform can be reached over a secure network connection. A typical analysis of measurement data consists of, for example, comparing the most recently calculated hashes with hashes calculated at a previous time, or with hashes known as good hashes. A mismatch in the hash values indicates platform corruption. The verification of platform integrity by a remote platform is known as remote attestation.

2 The Integrity Measurement Mechanism of Windows 10

Figure 1 depicts the architecture of the integrity measurement mechanism implemented in Windows 10. During the booting process of a given platform (*Platform* in Figure 1), the Unified Extensible Firmware Interface (UEFI) firmware, the boot manager, and the Windows loader measure relevant entities. They then store a processed form of the produced measurement data in the platform configuration registers (PCRs) of the TPM installed on the platform (*measured into* in Figure 1). In Figure 1, we refer to the UEFI firmware, the boot manager, and the Windows loader as the pre-operating system (OS) environment (*Pre-OS* in Figure 1). We discuss in greater detail the measurement performed in the pre-OS environment in Section 4.

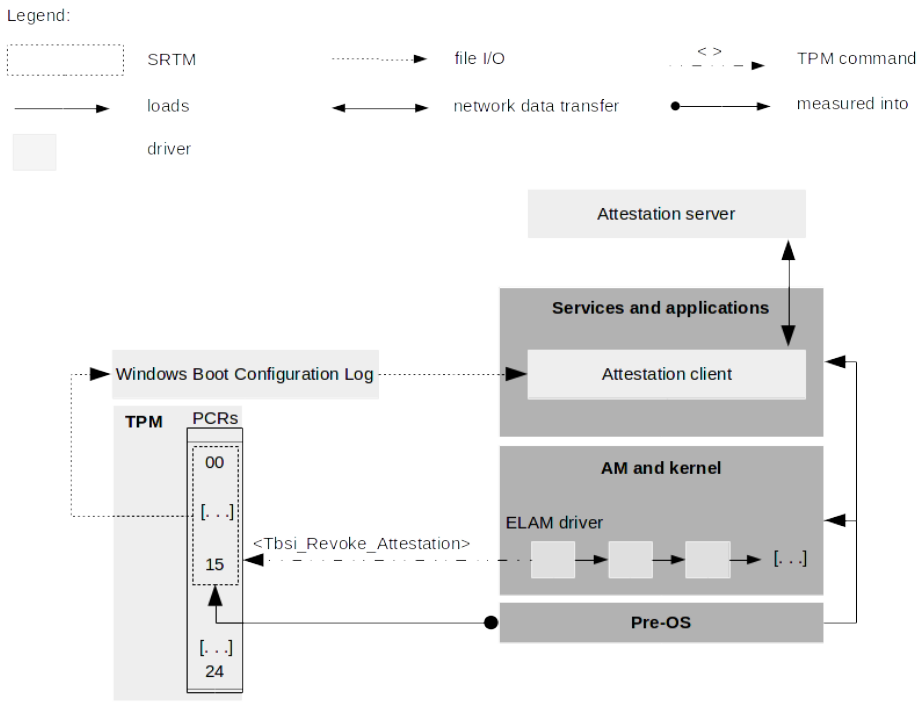

Figure 1: The architecture of the integrity measurement mechanism of Windows 10

The platform stores the hashes calculated in the pre-OS environment and relevant related data into a context known as the Windows Boot Configuration Log (WBCL) ([Ste16], Section 'Windows Boot Configuration Log'). A new WBCL is generated at every system startup since this is when new integrity measurements are made. Each WBCL is archived into a log file, referred to as the WBCL file. WBCL files are stored in the *%System-Root%\Logs\MeasuredBoot* directory. We discuss the content and format of WBCL files in Section 3.

The Windows loader loads the kernel, which may implement ELAM technology in the form of an ELAM driver (*ELAM and kernel* in Figure 1. In case it detects the loading of a malicious driver, the ELAM driver may revoke the current WBCL using the *Tbsi_Revoke_Attestation* function of the TBS library ([Ste16], Section 'Invalidating the System Trust State').[1,2] Among other things, a revocation of a WBCL consists of storing an unspecified value in the PCR with index 12. This indicates system corruption to the remote entity verifying platform integrity.

We analyzed the Windows Defender ELAM driver revoking the WBCL in a scenario where a given boot driver is considered malicious. We first configured the policy at *Computer Configuration -> Administrative Templates -> System -> Early Launch Antimalware* such that the kernel initializes only known good images. We then set

[1] https://msdn.microsoft.com/en-us/library/windows/desktop/jj553829(v=vs.85).aspx [Retrieved: 22/9/2017]

[2] https://docs.microsoft.com/en-us/windows-hardware/drivers/install/elam-driver-requirements [Retrieved: 22/9/2017]

breakpoints at the functions for submitting and processing TPM commands implemented as part of the export TPM driver *tbs.sys* and the TPM driver *tpm.sys*. Finally, we modified the return value of the *EbLookupProperty* function to *1* when the Windows Defender ELAM driver was checking a boot driver for malware. This return value indicates a known bad image. To remind, the return value of *EbLookupProperty* represents the decision of the ELAM driver on the maliciousness of a given boot driver.

We did not observe the Windows Defender ELAM driver or the kernel invoking *Tbsi_Revoke_Attestation* in order to revoke the current WBCL. They also did not invoke any other function of the TBS library or sent any TPM command to the TPM device after a decision on the maliciousness of the driver was made. It remains to be investigated whether the WBCL is revoked using means other than the ones we were focusing on, those specified in the Microsoft's development guidelines for ELAM drivers.[3] Although we did not observe the revocation of the WBCL, we observed that the kernel did not load the boot driver designated as a known bad image; that is, we observed that the Windows Defender ELAM driver effectively blocks the loading of malicious drivers.

The Windows kernel loads drivers and eventually the Windows subsystem, enabling the execution of system services and user applications (*Services and applications* in Figure 1). At this point, the content of WBCL files may be read by an application that transfers relevant content of these files to a remote entity verifying platform integrity (*Remote location* in Figure 1).[4] In Figure 1, we refer to the former as *attestation client* and to the latter as *attestation server*. The attestation client may obtain the most recent WBCL by invoking the *Tbsi_Get_TCG_Log* function of the TBS library.[5]

3 Windows Boot Configuration Log

WBCL files contain data in binary form. This data can be translated into Extensible Markup Language (XML) format using the *PCPTool* utility. Figure 2 depicts an excerpt of a WBCL file in XML format. This WBCL file was generated after a regular system reboot.

A WBCL file consists of multiple entries, where each entry contains relevant information on a given measured entity in the form of a *TCG_PCR_EVENT* structure. This structure is defined in the TCG (Extensible Firmware Interface) EFI Protocol Specification, family 2.0, level 00, revision 00.13 ([Tru16a], Section 5), which is the latest TCG EFI specification at the time of writing. It represents each measurement of an entity as a single 'measurement event' in TCG terminology (see the XML tags starting with *EV_* in Figure 2). Some relevant fields of *TCG_PCR_EVENT* are *PCRIndex* and *Digest*. *PCRIndex* is the number of the PCR into which the entity has been measured (*PCR Index* in Figure 2). *Digest* is the calculated hash of the measured entity (*Digest* in Figure 2).

Hashes of measured entities may be Secure Hash Algorithm (SHA)-1 hashes or hashes of other types, referred to as crypto agile hashes in the TCG EFI Protocol Specification ([Tru16a], Section 5.2). They are extended into specific PCRs of the TPM (see the values of the *PCR* XML tags in Figure 2). Extension of a hash into a given PCR is done by updating the value already stored in the PCR as follows: *PCRnew = H (PCRold || Digest)*, where *PCRold* is the old value stored in the PCR, *PCRnew* is the new value to be stored in the PCR, and *H* is a hash algorithm. In summary, the extension of a hash of a measured entity into a PCR consists of:

- oncatenating the old value stored in the PCR with the hash of the entity;
- hashing the resulting value of the above operation; and
- storing the resulting hash into the PCR.

The extension of hashes into PCRs is described in detail in the Trusted Platform Module Library Part 3: Commands, family 2.0, level 00, revision 01.16 ([Tru16b], Section 22.2.1). Given that the values stored in the TPM's PCRs are hashes of measurement data, they serve primarily for verification of the integrity of WBCLs.

---

[3] https://docs.microsoft.com/en-us/windows-hardware/drivers/install/elam-driver-requirements [Retrieved: 22/9/2017]

[4] https://docs.microsoft.com/en-us/windows/device-security/protect-high-value-assets-by-controlling-the-health-of-windows-10-based-devices [Retrieved: 22/9/2017]

[5] https://msdn.microsoft.com/de-de/library/windows/desktop/bb530712(v=vs.85).aspx [Retrieved: 22/9/2017]

```
<TCGLog>

[...]

<EV_Separator PCR="04" EventDigest="9069ca78e7450a285173431b3e52c5c25299e473" Size="4">
    00000000
<!-- .... -->
</EV_Separator>

[...]

<EV_EFI_Boot_Services_Application PCR="04" Digest="cc1d1c1e3ee18f559666b941bd10558063cb779d" Size="176">
    18c0209300000000600f1200000000000000001000000000900000000000000002010c00d041030a00000000001010600021f
    03120a0000000000000004012a00020000000180e0000000000020030000000005683625c6a4cf24a95d1d7fbc1364a4e
    0202040446005c0045004600490005c004d00690063007200 6f0073006f00660074005c0042006f006f0074005c0062006f00
    6f0074006d00670066007700 2e00650066006900000007fff0400
    <!-- .....A.....V.b.jL.J...
    ..6JN....F...E.F.I...M.i.c.r.o.s.o.f.t...B.o.o.t...b.o.o.t.m.g.f.w...e.f.i..... -->
</EV_EFI_Boot_Services_Application>

[...]

<PCRs>
    <PCR Index="00">4559d082d1de3e7c82554a734901d462f5846dfd</PCR>
    <PCR Index="01">3917e16e21826261c8e9bfa5b0ee91e0354f5f11</PCR>
    <PCR Index="02">b2a83b0ebf2f8374299a5b2bdfc31ea955ad7236</PCR>
    <PCR Index="03">b2a83b0ebf2f8374299a5b2bdfc31ea955ad7236</PCR>
    <PCR Index="04">435ead75281c162049e26d5685b508feebb57d23</PCR>
    <PCR Index="05">45a323382bd933f08e7f0e256bc8249e4095b1ec</PCR>
    <PCR Index="06">a9cdbe970aa5ddaaa8c20728a0eb1644dc4d50fe</PCR>
    <PCR Index="07">2608f8e106de28e13d81d37902082bda04a5db2c</PCR>
    <PCR Index="11">ebb98df76613280f20dc38221143a9e727399486</PCR>
    <PCR Index="12">06902aa5f773bd8c7d67499867fe4fc05a9c5ef0</PCR>
    <PCR Index="13">4f451830ea216fd4b90616d3ccec975b0bd153c7</PCR>
    <PCR Index="14">62aabb45314db6a801d1695e64b0d604d20889f8</PCR>
</PCRs>
</TCGLog>
```

Figure 2: An excerpt of a WBCL file

Into what PCRs hashes are extended depends on what is measured. Table 1 of the TCG PC Client Platform Firmware Profile Specification, family 2.0, level 00, revision 00.21 [Tru17] presents a mapping between measured entities and PCR indexes. In summary, the PCRs with indexes between 0 and 7 are used when extending hashes of firmware-related entities. Example such entities are UEFI variables. PCRs with indexes between 8 and 15 are used for measuring entities related to the installed operating system. What is stored in these PCRs is left to the discretion of the operating system's vendor. The PCRs with indexes between 0 and 15 are non-resettable PCRs, that is, the values stored in them cannot be cleared by the operating system, but only by hardware at each system reboot (*non-resettable PCRs* in Figure 1, [Tru13], Section 5.3).[6]

Each measurement event is of a specific type, which indicates what has been measured. For example, the measurement event of type *EV_EFI_VARIABLE_BOOT* contains measurement of a UEFI variable ([Tru17], Table 5). Table 2 presents a mapping between PCR indexes and types of measurement events. The events were extended into the PCRs on the Windows 10 system after a regular system reboot. Table 2 lists only events specified in the TCG PC Client Platform Firmware Profile Specification. We obtained the results presented in Table 2 using a parser of WBCL files translated into XML format, which we developed. Some event types listed in Table 2 have multiple sub-types storing comprehensive information on measured entities. We refer to ([Tru17], Section 9.3.1) for detailed descriptions of event types.

In addition to the data presented in Table 2, we extracted from the WBCL file a list of measured executables. This includes system drivers, system services, and driver executables. Measurements of executables are stored in WBCL files as events of type *EV_Event_Tag*. This event type contains information on the hashes of the executables. Measurements of executables are extended into the PCRs with indexes 12 and 13 ([Ste16], Section 'Windows Integrity Measurements'). The list of executables we extracted is placed in the Appendix, section 'Measured Executables'. It contains the filenames of the measured executables.

[6] ([Ste16], Section 'Root of Trust Overview') presents a mapping between measured Windows 8 entities and PCR indexes. To the best of our knowledge, such a mapping for Windows 10 entities is not available at the time of writing. Based on our analysis of the content of WBCL files, we assume that the mapping between Windows 10 entities and PCR indexes is to a great extent the same as that specified in [Ste16].

PCR	Event Type
0	*EV_CRTM_Contents*; *EV_CRTM_Version*; *EV_Post_Code*; *EV_EFI_Handoff_Tables*; *EV_Separator*
1	*EV_Event_Tag*;*EV_EFI_Handoff_Tables*; *EV_Separator*; *EV_EFI_Variable_Boot*
2	*EV_Separator*
3	*EV_Separator*
4	*EV_Separator*; *EV_EFI_Boot_Services_Application*
5	*EV_Separator*;*EV_EFI_Action*
6	*EV_Separator*;*EV_Action*
7	*EV_EFI_Variable_Driver_Config*;*EV_Separator*
11	*EV_Compact_Hash*
12	*EV_Event_Tag*;*EV_Separator*
13	*EV_Event_Tag*;*EV_Separator*

Table 1: A mapping between PCR indexes and types of measurement events

4 Implementation of Integrity Measurement

Measurements of Windows entities are performed by the boot manager and the Windows loader (*Pre-OS* in Figure 1). In this paragraph, we focus on the implementation of the integrity measurement mechanism in the Windows loader. We observed that the implementation of this mechanism in the boot manager is conceptually identical to the one presented in this section.

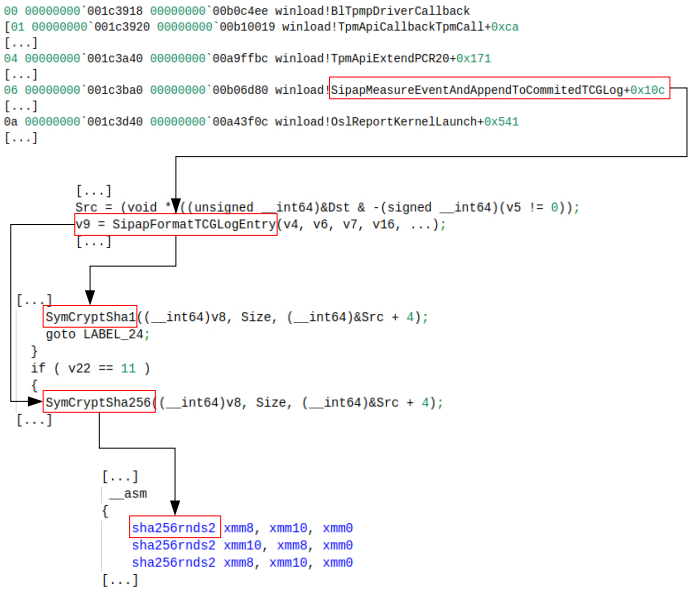

Figure 3: Integrity measurement in the Windows loader

Figure 3 depicts the stack and code snippets of functions executed as part of the integrity measurement mechanism implemented in the Windows loader. The content depicted in Figure 3 is as displayed by the *windbg* debugger and by the *IDA* disassembler in the form of pseudo-code. The *OslReportKernelLaunch* function is one of the functions implemented in the Windows loader that triggers integrity measurement. This is done by queueing measurement events for processing by submitting them to the *SipapMeasureEventAndAppendToCommitedTCGLog* function. This indicates that integrity measurements are conducted in an asynchronous manner.

SipapMeasureEventAndAppendToCommitedTCGLog first calculates hashes and therefore conducts the actual measurements. It then extends the measurements into PCRs by invoking *TpmApiExtendPCR*.*TpmApiExtendPCR* constructs a TPM command buffer and invokes *TpmApiCallbackTpmCall*. This function communicates with the TPM

by invoking *BlTpmpDriverCallback*.

Operations for hash calculation are implemented as functions of the Windows loader, that is, they are software-implemented (see *SipapFormatTCGLogEntry*, *SymCryptSha1/256* in Figure 3). For example, if the Intel SHA extensions are present, hash calculation is performed by executing CPU instructions specifically developed for that purpose (*sha256rnds2* in Figure 3).

Appendix

Measured Executables

bootmgfw.efi.MUI
tcpip.sys
WdBoot.sys
stexstor.sys
EhStorClass.sys
amdsata.sys
kd.dll
vdrvroot.sys
nvstor.sys
megasr.sys
bxvbda.sys
lsi_sss.sys
CLFS.SYS
werkernel.sys
WdFilter.sys
fvevol.sys
USBPORT.SYS
sdbus.sys
bootres.dll
lsi_sas3i.sys
ntoskrnl.exe
usbhub.sys
pdc.sys
3ware.sys
vmstorfl.sys
amdxata.sys
winhv.sys
Fs_Rec.sys
cnghwassist.sys
WppRecorder.sys
winload.efi.MUI
lsi_sas2i.sys

winload.efi
NETIO.SYS
hvsocket.sys
vmbkmcl.sys
rdyboost.sys
mountmgr.sys
sisraid4.sys
USBD.SYS
ucx01000.sys
mcupdate_GenuineIntel.dll
arcsas.sys
FLTMGR.SYS
storufs.sys
ADP80XX.SYS
ksecdd.sys
uaspstor.sys
storvsc.sys
BOOTVID.dll
CEA.sys
vstxraid.sys
iaStorV.sys
USBSTOR.SYS
pci.sys
ksecpkg.sys
fwpkclnt.sys
ACPI.sys
iaStorAV.sys
spaceport.sys
percsas2i.sys
WMILIB.SYS
percsas3i.sys
volume.sys

cng.sys
UsbHub3.sys
CI.dll
msisadrv.sys
Wdf01000.sys
usbehci.sys
mvumis.sys
partmgr.sys
volsnap.sys
acpiex.sys
volmgr.sys
hal.dll
PCIIDEX.SYS
volmgrx.sys
atapi.sys
pcw.sys
sbp2port.sys
NTFS.sys
ataport.SYS
amdsbs.sys
Wof.sys
storport.sys
pci.sys
cht4sx64.sys
disk.sys
WDFLDR.SYS
WindowsTrustedRTProxy.sys
megasas.sys
scmbus.sys
pciide.sys
wfplwfs.sys

vmbus.sys
NDIS.SYS
isapnp.sys
EhStorClass.sys
tm.sys
cmimcext.sys
HpSAMD.sys
SiSRaid2.sys
usbccgp.sys
evbda.sys
msrpc.sys
clipsp.sys
ntosext.sys
vsmraid.sys
USBXHCI.SYS
PSHED.dll
iorate.sys
tpm.sys
intelpep.sys
hwpolicy.sys
mup.sys
stornvme.sys
fileinfo.sys
intelide.sys
storahci.sys
pcmcia.sys
ApiSetSchema.dll
EhStorTcgDrv.sys
nvraid.sys
WindowsTrustedRT.sys
lsi_sas.sys

References

[Ste16] Stefan Thom and Jork Loeser and Ron Aigner and Paul England and Rob Spiger and Jim Morgan. Using the Windows 8 Platform Crypto Provider and Associated TPM Functionality. 2016. https://github.com/Microsoft/TSS.MSR/blob/master/PCPTool.v11/Using%20the%20Windows%208%20Platform%20Crypto%20Provider%20and%20Associated%20TPM%20Functionality.pdf.

[Tru13] Trusted Computing Group (TCG). TCG PC Client Specific TPM Interface Specification (TIS). 2013. Version 1.3; https://trustedcomputinggroup.org/wp-content/uploads/TCG_PCClientTPMInterfaceSpecification_TIS__1-3_27_03212013.pdf.

[Tru16a] Trusted Computing Group (TCG). Trusted Computing Group (TCG): TCG EFI Protocol Specification. 2016. https://trustedcomputinggroup.org/wp-content/uploads/EFI-Protocol-Specification-rev13-160330final.pdf.

[Tru16b] Trusted Computing Group (TCG). Trusted Platform Module Library Part 3: Commands. 2016. Family 2.0, Level 00, Revision 01.38; https://trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-2.0-Part-3-Commands-01.38.pdf.

[Tru17] Trusted Computing Group (TCG). TCG PC Client Platform Firmware Profile Specification. 2017. Family 2.0, Level 00, Revision 00.21; https://trustedcomputinggroup.org/wp-content/uploads/PC-ClientSpecific_Platform_Profile_for_TPM_2p0_Systems_v21.pdf.