The TPM: Technical Overview of Microsoft's Interim Measures against CVE-2017-15361

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), 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

This document provides a technical overview of Microsoft's interim measures against a vulnerability in Trusted Platform Module (TPM) chipsets manufactured by Infineon ([Common Vulnerabilities and Exposures] CVE-2017-15361).[1] This vulnerability affects the generation of Rivest–Shamir–Adleman (RSA) keys by vulnerable TPM chipsets making keys susceptible to the Return of Coppersmith's attack (ROCA). This attack enables attackers to recover the private key from the public key of a weak public-private key pair. The CVE-2017-15361 vulnerability is a firmware vulnerability, and not a vulnerability in the operating system. However, because of its criticality, Microsoft has implemented interim measures against this vulnerability until firmware vendors provide a patch addressing it. These measures are implemented as part of the KB4041691 update for the Windows version that is in the focus of this work.[2]

[1] https://nvd.nist.gov/vuln/detail/CVE-2017-15361 [Retrieved: 17/12/2019]

[2] https://support.microsoft.com/en-us/help/4041691/windows-10-update-kb4041691 [Retrieved: 17/12/2019]

Among other things, the KB4041691 update modifies Windows 10 such that it prevents the generation of weak RSA keys by the TPM and generates EventLog log entries when a vulnerable TPM chipset is detected.[3] In summary, the KB4041691 update:

- introduces in the *ncrypt* library the *PCP_TPM_IFX_RSA_KEYGEN_PROHIBITED* and *PCP_TPM_IFX_RSA_KEYGEN_VULNERABILITY* properties of the provider named *Platform Cryptographic Provider*. This provider, characterized by named properties, abstracts the TPM device and is used by applications as an abstraction layer for communicating with the TPM. The *PCP_TPM_IFX_RSA_KEYGEN_PROHIBITED* property enables system administrators to prohibit the generation of weak RSA keys by a vulnerable TPM chipset at system-level by setting the registry value *HKEY_LOCAL_MACHINE\Software\Policies\Microsoft\Tpm\IFX_VULNERABLE_RSA_KEY_GEN_PROHIBITED* to *0x1*. The *PCP_TPM_IFX_RSA_KEYGEN_VULNERABILITY* property enables applications to detect vulnerable TPM chipsets and react appropriately, independently of system configuration. This may involve, for example, generating keys in software instead of hardware (i.e., by the TPM);
- implements the generation of an EventLog log entry with an ID of 1794 when the *Tpm-Maintenance* scheduled task is executed. This task is executed as part of the TPM auto-provisioning process. If TPM auto-provisioning is enabled, this process is conducted at every system startup.

Section 2 and Section 3 of this work provide more technical information on the points listed above.

It is important to emphasize that in order to fully remediate the CVE-2017-15361 vulnerability, users have to install the firmware update provided by the hardware original equipment manufacturer (OEM) for patching the vulnerability. In addition, users have to clear and re-provision the TPM. This may render software that uses TPM-generated keys unstable and may lead to loss of TPM-protected data. Therefore, users have to make in advance remediation plans for such software and data. This includes, for example, a temporary transition to the use of software-generated keys. After the TPM has been re-provisioned, users may generate new keys with it and re-enroll software that uses TPM-generated keys and data that needs to be protected by the TPM.

2 Generation of RSA keys

Windows 10 provides the Cryptography API: Next Generation (CNG) library as an abstraction layer for communicating with the TPM. CNG uses the concept of cryptographic providers, where providers are entities performing cryptographic operations (e.g., hashing, digital signature verification). There are two main types of CNG providers: algorithm and key storage providers. The former are used for performing basic cryptographic operations, such as hashing and signing, whereas the latter are used for performing key operations, such as creating and storing keys. CNG abstracts the TPM device in the form of a cryptographic provider, referred to as the *Platform Cryptographic Provider*.

Each key storage provider is characterized by named provider properties. Applications may obtain the value of a given property by invoking the *NCryptGetProperty* function,[4] specifying the name of the property as the function's second parameter.

The KB4041691 update introduces two new properties of the *Platform Cryptographic Provider*: *PCP_TPM_IFX_RSA_KEYGEN_PROHIBITED* and *PCP_TPM_IFX_RSA_KEYGEN_VULNERABILITY*. For TPMs of version 2.0, these provider properties are implemented in the *PLATFORM_20_COMMON_PROVIDER::GetProperty* function, in the *PCPTpm12.dll* library file. This library file is part of the CNG functionalities related to the TPM.

The *PCP_TPM_IFX_RSA_KEYGEN_VULNERABILITY* property is used for detecting vulnerable versions of TPM chipsets;[5] that is, this property is set if a vulnerable TPM chipset is deployed. The *IsIFXRSAKeyGenVulnerabilityAffected* function, implemented in *PCPTpm12.dll*, identifies vulnerable versions of TPM chipsets.

[3] https://portal.msrc.microsoft.com/en-us/security-guidance/advisory/ADV170012 [Retrieved: 17/12/2019]

[4] https://docs.microsoft.com/en-us/windows/win32/api/ncrypt/nf-ncrypt-ncryptgetproperty [Retrieved: 17/12/2019]

[5] https://nvd.nist.gov/vuln/detail/CVE-2017-15361 [Retrieved: 17/12/2019]

The *PCP_TPM_IFX_RSA_KEYGEN_PROHIBITED* property is used for system administrators to configure Windows 10 to prohibit the generation of weak RSA keys. This property is set if the registry value *HKEY_LOCAL_MACHINE\Software\Policies\Microsoft\Tpm\IFX_VULNERABLE_RSA_KEY_GEN_PROHIBITED* is set to *0x1*. The evaluation of this registry value is performed by the *IsIFXRSAVulnerabilityKeyGenProhibitedByPolicy* function, implemented in *PCPTpm12.dll* (see Figure 1). The setting of the registry value prohibits the generation of RSA keys by vulnerable TPMs at system-level. We observed that this value is not set by default when the KB4041691 update is applied.

Figure 1: Evaluation of the registry value HKEY_LOCAL_MACHINE\Software\Policies\Microsoft\Tpm\IFX_VULNERABLE_RSA_KEY_GEN_PROHIBITED

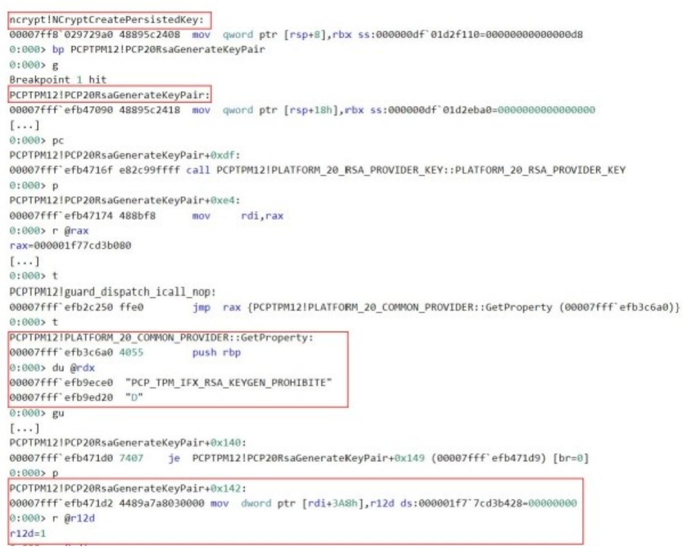

Figure 2: Operation of NCryptCreatePersistedKey and PCP20RsaGenerateKeyPair

Next, we discuss how the *PCP_TPM_IFX_RSA_KEYGEN_PROHIBITED* property prohibits the generation of weak RSA keys. In order to observe the impact of this property, we attempted to generate an RSA key using the *Platform Cryptographic Provider* (i.e., the TPM) with the *PCPTool*.[6]

Applications invoke the *NCryptCreatePersistedKey* function, part of the *ncrypt* library (library file: *ncrypt.dll*), to instruct the TPM to generate an RSA key.[7] In the context of the *ncrypt* library, keys are represented as key objects. *NCryptCreatePersistedKey* invokes the *PCP20RsaGenerateKeyPair* function implemented in *PCPTpm12.dll*. The KB4041691 update modifies *PCP20RsaGenerateKeyPair* such that it generates a key object and invokes the *PLATFORM_20_COMMON_PROVIDER::GetProperty* function in order to obtain the value of the *PCP_TPM_IFX_-*

[6]https://github.com/microsoft/TSS.MSR/tree/master/PCPTool.v11 [Retrieved: 17/12/2019]

[7]https://docs.microsoft.com/en-us/windows/win32/api/ncrypt/nf-ncrypt-ncryptcreatepersistedkey [Retrieved: 17/12/2019]

RSA_KEYGEN_PROHIBITED property (see Figure 2). If the property is set, *PCP20RsaGenerateKeyPair* sets the offset *0x3A8* of the generated key object to *1*. This indicates that the generation of weak RSA keys is prohibited.

```
ncrypt!NCryptFinalizeKey:
00007ff8`02972b00 48895c2408      mov     qword ptr [rsp+8],rbx ss:000000df`01d2f110=0000000000000000
0:000> g
Breakpoint 3 hit
PCPTPM12!PLATFORM_20_RSA_PROVIDER_KEY::FinalizeRsaKeyPair:
00007fff`efb43270 89542410        mov     dword ptr [rsp+10h],edx ss:000000df`01d2eda8=00000000
0:000> pc
PCPTPM12!PLATFORM_20_RSA_PROVIDER_KEY::FinalizeRsaKeyPair+0x53:
00007fff`efb432c3 e884110400      call    PCPTPM12!tpm12class::TPMW8_Create::TPMW8_Create
0:000> p

[...]

0:000> p
PCPTPM12!PLATFORM_20_RSA_PROVIDER_KEY::FinalizeRsaKeyPair+0x236:
00007fff`efb434a6 4439bea8030000  cmp     dword ptr [rsi+3A8h],r15d ds:000001f7`7cd3b428=00000001
0:000> r @rsi
rsi=000001f77cd3b080
0:000> p
PCPTPM12!PLATFORM_20_RSA_PROVIDER_KEY::FinalizeRsaKeyPair+0x23d:
00007fff`efb434ad 7414            je      PCPTPM12!PLATFORM_20_RSA_PROVIDER_KEY::FinalizeRsaKeyPair+0x253
0:000> p
PCPTPM12!PLATFORM_20_RSA_PROVIDER_KEY::FinalizeRsaKeyPair+0x23f:
00007fff`efb434af bb1f2029c0      mov     ebx,0C029201Fh
0:000> p
PCPTPM12!PLATFORM_20_RSA_PROVIDER_KEY::FinalizeRsaKeyPair+0x244:
00007fff`efb434b4 8bcb            mov     ecx,ebx
0:000> gu
PCPTPM12!PLATFORM_20_RSA_PROVIDER_KEY::FinalizeKeyPair+0xfe:
00007fff`efb431fe 8bf0            mov     esi,eax
0:000> r @eax
eax=c029201f
```

Figure 3: Operation of NCryptFinalizeKey and FinalizeRsaKeyPair

Applications must invoke the *NCryptFinalizeKey* function, also part of the *ncrypt* library, such that a key generated by *NCryptCreatePersistedKey* is completed and ready for use.[8] When a TPM of version 2.0 has been instructed to generate an RSA key, *NCryptFinalizeKey* invokes the *PLATFORM_20_RSA_PROVIDER_KEY::FinalizeRsaKeyPair* function implemented in *PCPTpm12.dll* (see Figure 3). This function evaluates the value stored at the offset *0x3A8* of the key object generated by *NCryptCreatePersistedKey*. If this value is *1*, *FinalizeRsaKeyPair* does not finalize the key and returns the status (error) code *0xC029201f* (*STATUS_PCP_IFX_RSA_KEY_CREATION_BLOCKED*, function return values are stored in the *rax* register, see Figure 3). This effectively prohibits the generation of RSA keys by the TPM if the registry value *HKEY_LOCAL_MACHINE\Software\Policies\Microsoft\Tpm\IFX_VULNERABLE_RSA_KEY_GEN_PROHIBITED* is set to *0x1*. Figure 4 depicts the definition of the status code *0xC029201f* implemented in *ntstatus.h*, a library header file distributed as part of Microsoft's Software Development Kit (SDK).

```
//
// MessageId: STATUS_PCP_IFX_RSA_KEY_CREATION_BLOCKED
//
// MessageText:
//
// The RSA key creation is blocked on this TPM due to known security vulnerabilities.
//
#define STATUS_PCP_IFX_RSA_KEY_CREATION_BLOCKED ((NTSTATUS)0xC029201FL)
```

Figure 4: Definition of the status code STATUS_PCP_IFX_RSA_KEY_CREATION_BLOCKED

This section shows that the KB4041691 update, by introducing the *PCP_TPM_IFX_RSA_KEYGEN_PROHIBITED* and *PCP_TPM_IFX_RSA_KEYGEN_VULNERABILITY* provider properties in the *ncrypt* library, enables applications to detect vulnerable TPM chipsets and system administrators to prohibit the generation of weak RSA keys at system-level. If the generation of weak RSA keys is not explicitly prohibited by administrators, applications may use the*PCP_TPM_IFX_RSA_KEYGEN_VULNERABILITY* property to detect vulnerable TPM chipsets and react appropriately, independently of system configuration. This may involve, for example, generating keys in software instead of hardware (i.e., by the TPM). The KB4041691 update introduces such changes to applications developed by Microsoft and distributed with Windows 10. Section 2.1 shows how the Windows Hello component of Windows 10 addresses the generation of weak RSA keys by the TPM.

[8]https://docs.microsoft.com/en-us/windows/win32/api/ncrypt/nf-ncrypt-ncryptfinalizekey [Retrieved: 17/12/2019]

2.1 Windows Hello

The Windows Hello component of Windows 10 enables users to authenticate themselves using biometric data or PIN. The PIN can be bound to the TPM and hence to the device where the PIN is generated. Therefore, for an attacker to compromise a PIN-protected Windows account set on a given device, the attacker would have to know the account's PIN and to have physical access to the device itself.[9]

If a TPM is available, Windows Hello (if not explicitly configured otherwise) will use the TPM to bind the PIN to the TPM through a process called TPM sealing. This process involves encrypting data using a private key protected by the TPM (i.e., stored in the TPM or wrapped by the TPM) and taking the state of the TPM's Platform Configuration Registers (PCRs) into account.[10] Encrypted data may then be decrypted only by the TPM device that has encrypted it and only if the PCR states match. Windows Hello binds a PIN to a TPM such that the TPM seals a randomly generated sequence, referred to as authentication key in this work, using the PIN as a decryption (unsealing) passphrase. When a user attempts to log in using a PIN, the PIN is used for the TPM to unseal the authentication key. If the unsealing process succeeds, which implies that the PIN is correct, the user is logged in. This ensures that the PIN-based login process takes place at the device where the PIN has been created.

The *Microsoft Passport Container* service, with functionalities implemented in the *NgcCtnr.dll* library file, manages the PIN-based login process. The KB4041691 update modifies the *PolicyManager::GetProviderName* function, implemented in *NgcCtnr.dll*, such that it invokes:

- the *PolicyManager::IsInsecureTpm* function, which evaluates the value of the *PCP_TPM_IFX_RSA_KEYGEN_-VULNERABILITY* property (see Section 2);
- the *PolicyManager::IsInsecureTpmBlockedByTpmPolicy* function, which evaluates the value of the *PCP_-TPM_IFX_RSA_KEYGEN_PROHIBITED* property (see Section 2); and
- the *PolicyManager::IsInsecureTpmBlockedByWHfBPolicy* function, which evaluates the registry value *HKEY_-LOCAL_MACHINE\System\CurrentControlSet\Control\Cryptography\NGC\InsecureTPM*; this is a Windows Hello-specific configuration interface for prohibiting Windows Hello to use vulnerable TPM chipsets.

If the first parameter of one of these functions is set to *0x1*, Windows Hello is configured to use a software-based cryptographic provider, that is, the *Microsoft Software Key Storage Provider*[11] – this indicates that the TPM chipset is vulnerable or its use has been prohibited by a system administrator (see Section 2). If the first parameter of all of these functions is set to *0x0*, Windows Hello is configured to use the TPM, that is the *Platform Cryptographic Provider*.

Figure 5 depicts relevant aspects of the workflow of the *PolicyManager::GetProviderName* function when executed on a device with a vulnerable TPM chipset. The first parameter of *PolicyManager::IsInsecureTpm* is set to *0x1*, and therefore, Windows Hello is configured to use the *Microsoft Software Key Storage* provider.

Figure 6 depicts relevant aspects of the workflow of the *PolicyManager::GetProviderName* function when executed on the same device, however, in a scenario where the value of the first parameter of *PolicyManager::IsInsecureTpm* has been manually changed (patched) to *0x0* and the generation of weak RSA keys has not been prohibited. This indicates that the TPM chipset is not affected by CVE-2017-15361. Therefore, Windows Hello is configured to use the TPM, that is, the *Platform Cryptographic Provider* (*Microsoft Platform Crypto Provider* in Figure 6).

If Windows Hello is configured to use the *Platform Cryptographic Provider*, when a user configures a new PIN, it binds the PIN to the TPM by sealing a previously generated authentication key specifying the PIN as a decryption (unsealing) passphrase. Figure 7, Figure 8, and Figure 9 depict relevant aspects of this process.

---

[9] https://docs.microsoft.com/en-us/windows/security/identity-protection/hello-for-business/hello-why-pin-is-better-than-password [Retrieved: 17/12/2019]

[10] https://docs.microsoft.com/en-us/windows/security/information-protection/tpm/switch-pcr-banks-on-tpm-2-0-devices [Retrieved: 17/12/2019]

[11] https://www.pkisolutions.com/understanding-microsoft-crypto-providers/ [Retrieved: 17/12/2019]

```
Breakpoint 0 hit
NgcCtnr!PolicyManager::IsInsecureTpm:
00007ff8`372dfd8c 48895c2410      mov     qword ptr [rsp+10h],rbx ss:00000081`7c77ddb8=0000000000000000
0:003> ? @rcx
Evaluate expression: 556139011556 = 00000081`7c77dde4
0:003> gu
NgcCtnr!PolicyManager::GetProviderName+0x1cc:
00007ff8`372e0580 89442430        mov     dword ptr [rsp+30h],eax ss:00000081`7c77dde0=00000000
0:003> db 00000081`7c77dde4
00000081`7c77dde4  01 00 00 00 80 00 00 7c-02 00 00 00 28 e1 77 7c  .......|....(.w|
[...]
0:003> pc
NgcCtnr!PolicyManager::GetProviderName+0x1e3:
00007ff8`372e0597 e81cf9ffff      call    NgcCtnr!PolicyManager::IsInsecureTpmBlockedByTpmPolicy (00007ff8`372dfeb8)
0:003> pc
NgcCtnr!PolicyManager::GetProviderName+0x1ff:
00007ff8`372e05b3 e82cfaffff      call    NgcCtnr!PolicyManager::IsInsecureTpmBlockedByWHfBPolicy (00007ff8`372dffe4)
0:003> pc
NgcCtnr!PolicyManager::GetProviderName+0x50f:
00007ff8`372e08c3 e8b0e9fdff      call    NgcCtnr!std::basic_string<unsigned short,std::char_traits<unsigned short>,
std::allocator<unsigned short> >::assign (00007ff8`372bf278)
0:003> t
NgcCtnr!std::basic_string<unsigned short,std::char_traits<unsigned short>,std::allocator<unsigned short> >::assign:
00007ff8`372bf278 48895c2408      mov     qword ptr [rsp+8],rbx ss:00000081`7c77ddb0=0000000000000002
0:003> du @rdx
00007ff8`372f1e40  "Microsoft Software Key Storage P"
00007ff8`372f1e80  "rovider"
```

Figure 5: Workflow of PolicyManager::GetProviderName (vulnerable TPM chipset)

```
Breakpoint 0 hit
NgcCtnr!PolicyManager::IsInsecureTpm:
00007ff8`372dfd8c 48895c2410      mov     qword ptr [rsp+10h],rbx ss:00000081`7c77ddb8=0000000000000000
0:003> ? @rcx
Evaluate expression: 556139011556 = 00000081`7c77dde4
0:003> gu
NgcCtnr!PolicyManager::GetProviderName+0x1cc:
00007ff8`372e0580 89442430        mov     dword ptr [rsp+30h],eax ss:00000081`7c77dde0=00000000
0:003> eb 00000081`7c77dde4
00000081`7c77dde4 01 00
00
[...]
0:003> pc
NgcCtnr!PolicyManager::GetProviderName+0x2e4:
00007ff8`372e0698 e8dbebfdff      call    NgcCtnr!std::basic_string<unsigned short,std::char_traits<unsigned short>,
std::allocator<unsigned short> >::assign (00007ff8`372bf278)
0:003> t
NgcCtnr!std::basic_string<unsigned short,std::char_traits<unsigned short>,std::allocator<unsigned short> >::assign:
00007ff8`372bf278 48895c2408      mov     qword ptr [rsp+8],rbx ss:00000081`7c77ddb0=0000000000000002
0:003> du @rcx
00000081`7c77de60  ""
0:003> du @rdx
00007ff8`372f2020  "Microsoft Platform Crypto Provid"
00007ff8`372f2060  "er"
```

Figure 6: Workflow of PolicyManager::GetProviderName (non-vulnerable TPM chipset)

The *NgcCtnrContainer::InternalCreateProtectors* function, implemented in *NgcCtnr.dll*, invokes the *ProtectorManager::GenerateAuthKey* function. This function generates the authentication key (the sequence starting with *32 2d cb 1a ..* in Figure 7). Once the authentication key is generated, the *KeyManager::EncodeAuthDataForPassword* function encodes the user-provided PIN (*1111* in Figure 7) so that it can be used as a decryption passphrase. Then, the *ProtectorManager::_PackUserAuthBlob* function modifies the format of the authentication key to prepare it for sealing (the sequence starting with *20 00 00 00 32 2d cb ...* in Figure 8).

```
Breakpoint 3 hit
NgcCtnr!NgcCtnrContainer::InternalCreateProtectors:
00007ff8`372bbe3c 4055            push    rbp
0:002> pc
[...]
00007ff8`372bc075 e86a620100      call    NgcCtnr!ProtectorManager::GenerateAuthKey (00007ff8`372d22e4)
0:002> t
NgcCtnr!ProtectorManager::GenerateAuthKey:
00007ff8`372d22e4 488bc4          mov     rax,rsp
0:002> ? @rcx
Evaluate expression: 556138486744 = 00000081`7c6fdbd8
0:002> gu
NgcCtnr!NgcCtnrContainer::InternalCreateProtectors+0x23e:
00007ff8`372bc07a 8bd8            mov     ebx,eax
0:002> db poi(00000081`7c6fdbd8)
000001bf`ebc16350  32 2d cb 1a cc 30 51 fa-76 d9 5a 2d 23 b4 68 f8  2-...0Q.v.Z-#.h.
000001bf`ebc16360  f3 53 5d e4 5f eb c9 05-bb 61 bf 6c 84 e3 f5 6c  .S]._....a.l...l
000001bf`ebc16370  01 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
[...]
0:002> pc
NgcCtnr!NgcCtnrContainer::InternalCreateProtectors+0x336:
00007ff8`372bc172 e84d330100      call    NgcCtnr!ProtectorManager::CreatePinProtector (00007ff8`372cf4c4)
0:002> t
Breakpoint 1 hit
NgcCtnr!ProtectorManager::CreatePinProtector:
00007ff8`372cf4c4 4055            push    rbp
0:002> pc
[...]
NgcCtnr!PropertyManager::CreateProtector (00007ff8`372cae1c)
0:002> pc
NgcCtnr!ProtectorManager::CreatePinProtector+0x14c:
00007ff8`372cf610 e87b96ffff      call    NgcCtnr!KeyManager::EncodeAuthDataForPassword (00007ff8`372c8c90)
0:002> t
NgcCtnr!KeyManager::EncodeAuthDataForPassword:
00007ff8`372c8c90 488bc4          mov     rax,rsp
0:002> db @rcx
000001bf`ebc6c380  31 31 31 31 f8 7f 00 00-10 6c c9 eb bf 01 00 00  1111.....l......
[...]
```

Figure 7: Binding a PIN to a device (part 1)

```
0:002> gu
NgcCtnr!ProtectorManager::CreatePinProtector+0x151:
00007ff8`372cf615 44396b14        cmp     dword ptr [rbx+14h],r13d ds:000001bf`ebc9d544=00000002
0:002> pc
NgcCtnr!ProtectorManager::CreatePinProtector+0x221:
00007ff8`372cf6e5 e812310000      call    NgcCtnr!ProtectorManager::_PackUserAuthBlob (00007ff8`372d27fc)
0:002> t
NgcCtnr!ProtectorManager::_PackUserAuthBlob:
00007ff8`372d27fc 488bc4          mov     rax,rsp
0:002> ? @rdx
Evaluate expression: 556138486264 = 00000081`7c6fd9f8
0:002> gu
NgcCtnr!ProtectorManager::CreatePinProtector+0x226:
00007ff8`372cf6ea 4c8d442468      lea     r8,[rsp+68h]
0:002> db poi(00000081`7c6fd9f8)
000001bf`ebc45690  20 00 00 00 32 2d cb 1a-cc 30 51 fa 76 d9 5a 2d   ...2-...0Q.v.Z-
000001bf`ebc456a0  23 b4 68 f8 f3 53 5d e4-5f eb c9 05 bb 61 bf 6c  #.h..S]._....a.l
000001bf`ebc456b0  84 e3 f5 6c 00 00 00 00-07 00 00 00 00 00 00 00  ...l............
[...]
NgcCtnr!HardwareKeyFactory::SealData:
00007ff8`372c7370 488bc4          mov     rax,rsp
0:002> pc
[...]
NgcCtnr!HardwareKeyFactory::SealData+0x144:
00007ff8`372c74b4 ff155eec0400    call    qword ptr [NgcCtnr!_imp_NCryptEncrypt (00007ff8`37316118)]
0:002> t
ncrypt!NCryptEncrypt:
00007ff8`496e17f0 48895c2408      mov     qword ptr [rsp+8],rbx ss:00000081`7c6fd720=000000817c6fd788
0:002> dp @rcx
000001bf`ebca41e0  00000000`44440002 000001bf`ebc05be0
000001bf`ebca41f0  00000000`fffffff1 000001bf`ebca4200
[...]
0:002> du 000001bf`ebca4200
000001bf`ebca4200  "MICROSOFT_PCP_KSP_RSA_SEAL_KEY_3"
000001bf`ebca4240  "BD1C4BF-004E-4E2F-8A4D-0BF633DCB"
000001bf`ebca4280  "074"
0:002> db @rdx L(@r8)
000001bf`ebc45690  20 00 00 00 32 2d cb 1a-cc 30 51 fa 76 d9 5a 2d   ...2-...0Q.v.Z-
000001bf`ebc456a0  23 b4 68 f8 f3 53 5d e4-5f eb c9 05 bb 61 bf 6c  #.h..S]._....a.l
000001bf`ebc456b0  84 e3 f5 6c                                      ...l
0:002> pc
[...]
```

Figure 8: Binding a PIN to a device (part 2)

Once the authentication key is ready for sealing, the *HardwareKeyFactory::SealData* function invokes the *NCryptEncrypt* function of the *ncrypt* library, which conducts the sealing. The first parameter of this function specifies a key named *MICROSOFT_PCP_KSP_RSA_SEAL_KEY_3BD1C4BF-004E-4E2F-8A4D-0BF633DCB074*. This key refers to the storage root key (SRK) key, whose private part is stored in, and cannot leave, the TPM. The second parameter of *NCryptEncrypt* stores the data to be sealed, that is the modified authentication key (the sequence starting with *20 00 00 00 32 2d cb ...* in Figure 8). Finally, the *PLATFORM_20_SEAL_PROVIDER::Seal* function sends TPM commands to the TPM for sealing the authentication key.

First, the TPM command *TPM_CC_ReadPublic*, identified by the byte sequence *00 00 01 73* ([Tru16], Section 12.4) obtains the public key of the SRK from the TPM (the byte sequence starting with *b9 b2 95 ...* in Figure 9). In order to verify that the public key obtained in *PLATFORM_20_SEAL_PROVIDER::Seal* is that of the SRK, we extracted the public key of the SRK using the *PCPTool* utility, which displayed the same byte sequence (the byte sequence starting with *b9 b2 95 ...* in Figure 10). Using the SRK's public key, *PLATFORM_20_SEAL_PROVIDER::Seal* encrypts, that is, seals, the data by issuing the *TPM_CC_Create* TPM command, identified by the byte sequence *00 00 01 53* (see Figure 9, [Tru16], Section 12.1).

```
0:002>
PCPTPM12!PLATFORM_20_SEAL_PROVIDER::Seal+0x169:
00007ff8`4102ff29 e8325d0100      call    PCPTPM12!tpm12class::TPMW8_COMMAND::Execute (00007ff8`41045c60)
0:002> t
[...]
0:002> pc
PCPTPM12!WinPlatformW8TbsSubmit+0xa3:
00007ff8`41038a63 ff15bf150500    call    qword ptr [PCPTPM12!_imp_Tbsip_Submit_Command (00007ff8`4108a028)]
0:002> t
tbs!Tbsip_Submit_Command:
00007ff8`40fe1bf0 4883ec48        sub     rsp,48h
0:002> db @r9
000001bf`ebc6ca70  80 01 00 00 00 0e 00 00-01 73 81 00 00 01 54 00  .........s....T.
[...]
0:002> dps @rsp
00000081`7c6fc808  00007ff8`41038a69 PCPTPM12!WinPlatformW8TbsSubmit+0xa9
[...]
00000081`7c6fc830  000001bf`0000000e
00000081`7c6fc838  000001bf`ebc76450
[...]
0:002> gu
PCPTPM12!WinPlatformW8TbsSubmit+0xa9:
00007ff8`41038a69 488b5c2468      mov     rbx,qword ptr [rsp+68h] ss:00000081`7c6fc878=000000817c6fcdb0
0:002> db 000001bf`ebc76450
000001bf`ebc76450  80 01 00 00 01 6e 00 00-00 00 01 1a 00 01 00 0b  .....n..........
000001bf`ebc76460  00 03 04 72 00 00 00 06-00 80 00 43 00 10 08 00  ...r.......C....
000001bf`ebc76470  00 00 00 00 01 00 b9 b2-95 0c ee 30 a4 62 c6 9b  ...........0.b..
000001bf`ebc76480  54 f7 8b df 42 87 63 e5-7c 3c b5 58 73 99 4d b5  T...B.c.|<.Xs.M.
000001bf`ebc76490  d4 34 01 1e 35 7a c6 7c-4c 40 3e 6f 22 9a 42 ea  .4..5z.|L@>o".B.
000001bf`ebc764a0  d5 3b 53 5a a5 46 50 74-ac 7c 55 8c 98 8e 84 60  .;SZ.FPt.|U....`
000001bf`ebc764b0  66 67 35 e8 f2 bd 11 7c-2e 5b 90 eb f9 f5 46 7b  fg5....|.[....F{
000001bf`ebc764c0  69 74 1b 8b 05 8f 08 7d-e2 7a f1 d0 40 1e e8 09  it.....}.z..@...
0:002> gu
[...]
0:002> pc
PCPTPM12!WinPlatformW8TbsSubmit+0xa3:
00007ff8`41038a63 ff15bf150500    call    qword ptr [PCPTPM12!_imp_Tbsip_Submit_Command (00007ff8`4108a028)]
0:002> t
tbs!Tbsip_Submit_Command:
00007ff8`40fe1bf0 4883ec48        sub     rsp,48h
0:002> db @r9
000001bf`ebc2ad40  80 02 00 00 00 db 00 00-01 53 81 00 00 01 00 00  .........S......
[...]
```

Figure 9: Binding a PIN to a device (part 3)

```
C:\Users\ernw\Desktop\Release>PCPTool GetSrk
<RSAKey size="283" keyName="StorageRootKey">
  <Magic>RSA1<!-- 0x31415352 --></Magic>
  <BitLength>2048</BitLength>
  <PublicExp size="3">
    010001
  </PublicExp>
  <Modulus size="256" digest="bbae4527355dc4c70beca4f67a2987d3ef028021">
    b9b2950cee30a462c69b54f78bdf428763e57c3cb55873994db5d434011e357ac67c4c403e6f229a42ead53b535aa5465074ac7c558
    c988e8460666735e8f2bd117c2e5b90ebf9f5467b69741b8b058f087de27af1d0401ee80990ad1e85e6db7a77d6577adda2f9deb4b4
    8c722a244d428804c16d712809dfe57b512249c5fa7ecf9eb0ef7a953abe454cf91c59a12909dad74b054d5d80fc9d02fc9c53a05b0
    76d0652ec2a4a5792c32cc4c2581aff5b8f0c82aa9275e3c6abf9c34f618bd58a90223735b1ca85e0b5558ab83ae90f23bc3eb2fe67
    dd69b95351a8a12b0efac2bcc9ee490c45daa92f8fcf105c258dd223a7f43f3d7bf4e4922c2db90a3ea5
  </Modulus>
  <Prime1/>
  <Prime2/>
</RSAKey>
```

Figure 10: The public key of the SRK as displayed by the PCPTool utility

3 Generation of EventLog Log Entries

The KB4041691 update modifies Windows 10 such that it generates an EventLog log entry with the ID 1794 in the *Windows Logs/System* event logging channel.[12] The entry indicates that a vulnerable TPM chipset is deployed on the device where the entry has been generated. This helps system administrators to detect the presence of a vulnerable TPM chipset in a straightforward manner.

The log entry with ID 1794 is generated by the *TpmCheckIFXRSAKeyGenVulnerability* function, implemented in the *TpmCoreProvisioning.dll* library file. *TpmCheckIFXRSAKeyGenVulnerability* is invoked by the *CTpmTasksHandler::Worker* function, implemented in the *TpmTasks.dll* library file. *CTpmTasksHandler::Worker* is invoked when the *Tpm-Maintenance* scheduled task is executed. This task is executed as part of the TPM auto-provisioning process. If TPM auto-provisioning is enabled, the process is conducted at every system startup.

Figure 11 depicts relevant aspects of the implementation of *TpmCheckIFXRSAKeyGenVulnerability*. *TpmCheckIFXRSAKeyGenVulnerability* invokes the *NCryptGetProperty* function in order to obtain the value of the *PCP_TPM_IFX_RSA_KEYGEN_VULNERABILITY* property (see Section 2). If this property is set, *TpmCheckIFXRSAKeyGenVulnerability* generates an EventLog log entry with an ID of 1794. The metadata describing this log entry is stored in the variable *TPMCOREEVENT_TPM_VULNERABLE_FIRMWARE_DETECTED*, implemented in *TpmCoreProvisioning.dll* (see Figure 11). Figure 12 depicts this metadata. The metadata is stored in an *_EVENT_DESCRIPTOR* structure that is defined in the context of the *ntdll.dll* library file.[13] The hexadecimal value of the field *Id* of the *_EVENT_DESCRIPTOR* structure is the ID of the log entry (*1794* in decimal form).

```
0:010> uf TpmCoreProvisioning!TpmCheckIFXRSAKeyGenVulnerability
[...]

00007fff`e3897a3a 488d15bf0e0400  lea     rdx,[TpmCoreProvisioning!`string' (00007fff`e38d8900)]
00007fff`e3897a41 ff15a1dc0300    call    qword ptr [TpmCoreProvisioning!_imp_NCryptGetProperty (00007fff`e38d56e8)]
00007fff`e3897a47 85c0            test    eax,eax

[...]

TpmCoreProvisioning!TpmCheckIFXRSAKeyGenVulnerability+0x6a:
00007fff`e3897a5a 488d1547ea0300  lea     rdx,[TpmCoreProvisioning!TPMCOREEVENT_TPM_VULNERABLE_FIRMWARE_DETECTED (00007fff`e38d64a8)]
00007fff`e3897a61 e8e299fdff      call    TpmCoreProvisioning!TemplateEventDescriptor (00007fff`e3871448)

[...]

0:010> du 00007fff`e38d8900
00007fff`e38d8900  "PCP_TPM_IFX_RSA_KEYGEN_VULNERABI"
00007fff`e38d8940  "LITY"
```

Figure 11: Workflow of TpmCheckIFXRSAKeyGenVulnerability

```
0:010> ? TpmCoreProvisioning!TPMCOREEVENT_TPM_VULNERABLE_FIRMWARE_DETECTED
Evaluate expression: 140737011082408 = 00007fff`e38d64a8
0:010> dt ntdll!_EVENT_DESCRIPTOR 00007fff`e38d64a8
   +0x000 Id               : 0x702
   +0x002 Version          : 0 ''
   +0x003 Channel          : 0x8 ''
   +0x004 Level            : 0x2 ''
   +0x005 Opcode           : 0 ''
   +0x006 Task             : 0
   +0x008 Keyword          : 0x80000000`00000000
```

Figure 12: The content of TPMCOREEVENT_TPM_VULNERABLE_FIRMWARE_DETECTED

[12] https://portal.msrc.microsoft.com/en-us/security-guidance/advisory/ADV170012 [Retrieved: 17/12/2019]

[13] https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-dtyp/a6110d36-28c1-4290-b79e-26aa95a0b1a0 [Retrieved: 17/12/2019]

References

[Tru16] 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.