UEFI Shell Utility to Display TPM TrEE Capabilities

Finnbarr P. Murphy
(fpm@fpmurphy.com)

With the drive towards hardening platform firmware, for example Microsoft's Secure Boot initiative, I have decided to explore what forensic artifacts concerning TCG Trusted Platform Module (TPM) can be retrieved from the UEFI shell command line.

The EFI Trusted Execution Environment (TrEE) protocol implements a subset of the TPM 2.0 library specification. Microsoft pushed the TrEE protocol due to the delay in finalizing the TCG EFI Protocol Specification Family "2.0". As of the date of this post, this TCG specification is currently at the public review stage.

This post provides the source code for a small UEFI shell utility that retrieves and displays the members of the TrEE Capabilities structure.
Here is the source code for the utility:

```
//
//  Copyright (c) 2015  Finnbarr P. Murphy.   All rights reserved.
//
//  Display TCG TPM TrEE Protocol capabilities
//
//  License: BSD License
//
#include <Uefi.h>
#include <Library/UefiLib.h>
#include <Library/ShellCEntryLib.h>
#include <Library/ShellLib.h>
#include <Library/BaseMemoryLib.h>
#include <Library/UefiBootServicesTableLib.h>
#include <Protocol/EfiShell.h>
#include <Protocol/LoadedImage.h>
#include <Protocol/TrEEProtocol.h>
#include <Protocol/Tcg2Protocol.h>
#include <IndustryStandard/UefiTcgPlatform.h>
#define EFI_TREE_SERVICE_BINDING_PROTOCOL_GUID \
   {0x4cf01d0a, 0xc48c, 0x4271, {0xa2, 0x2a, 0xad, 0x8e, 0x55, 0x97, 0x81, 0x88}}
VOID
PrintEventDetail(UINT8 *Detail, UINT32 Size)
{
    UINT8 *d = Detail;
    int Offset = 0;
    int Row = 1;
    Print(L"               Event Detail: %08x: ", Offset);
    for (int i = 0; i < Size; i++) {
        Print(L"%02x", *d++);
        Offset++; Row++;
        if (Row == 17 || Row == 33) {
            Print(L" ");
        }
        if (Row > 48) {
            Row = 1;
            Print(L"\n                                %08x: ", Offset);
        }
    }
    Print(L"\n");
```



```
}
VOID
PrintEventType(UINT32 EventType, BOOLEAN Verbose)
{
    Print(L"                Event Type: ");
    if (Verbose) {
        Print(L"%08x ", EventType);
    }
    switch (EventType) {
        case  EV_POST_CODE:                     Print(L"Post Code");
                                                break;
        case  EV_NO_ACTION:                     Print(L"No Action");
                                                break;
        case  EV_SEPARATOR:                     Print(L"Separator");
                                                break;
        case  EV_S_CRTM_CONTENTS:               Print(L"CTRM Contents");
                                                break;
        case  EV_S_CRTM_VERSION:                Print(L"CRTM Version");
                                                break;
        case  EV_CPU_MICROCODE:                 Print(L"CPU Microcode");
                                                break;
        case  EV_TABLE_OF_DEVICES:              Print(L"Table of Devices");
                                                break;
        case  EV_EFI_VARIABLE_DRIVER_CONFIG:    Print(L"Variable Driver Config");
                                                break;
        case  EV_EFI_VARIABLE_BOOT:             Print(L"Variable Boot");
                                                break;
        case  EV_EFI_BOOT_SERVICES_APPLICATION: Print(L"Boot Services Application");
                                                break;
        case  EV_EFI_BOOT_SERVICES_DRIVER:      Print(L"Boot Services Driver");
                                                break;
        case  EV_EFI_RUNTIME_SERVICES_DRIVER:   Print(L"Runtime Services Driver");
                                                break;
        case  EV_EFI_GPT_EVENT:                 Print(L"GPT Event");
                                                break;
        case  EV_EFI_ACTION:                    Print(L"Action");
                                                break;
        case  EV_EFI_PLATFORM_FIRMWARE_BLOB:    Print(L"Platform Fireware Blob");
                                                break;
        case  EV_EFI_HANDOFF_TABLES:            Print(L"Handoff Tables");
                                                break;
        case  EV_EFI_VARIABLE_AUTHORITY:        Print(L"Variable Authority");
                                                break;
        default:                                Print(L"Unknown Type");
                                                break;
    }
    Print(L"\n");
}
VOID
PrintSHA1(TCG_DIGEST Digest)
{
    Print(L"               SHA1 Digest: " );
    for (int j = 0; j < SHA1_DIGEST_SIZE; j++ ) {
         Print(L"%02x", Digest.digest[j]);
    }
    Print(L"\n");
}
CHAR16 *
ManufacturerStr(UINT32 ManufacturerID)
{
    static CHAR16 Mcode[5];
    Mcode[0] = (CHAR16) ((ManufacturerID &amp;amp; 0xff000000) >> 24);
    Mcode[1] = (CHAR16) ((ManufacturerID &amp;amp; 0x00ff0000) >> 16);
    Mcode[2] = (CHAR16) ((ManufacturerID &amp;amp; 0x0000ff00) >> 8);
    Mcode[3] = (CHAR16)  (ManufacturerID &amp;amp; 0x000000ff);
    Mcode[4] = (CHAR16) '&amp;#92;&amp;#48;';
    return Mcode;
}
```



```
VOID
Usage(CHAR16 *Str)
{
    Print(L"Usage: %s [-v|--verbose]\n", Str);
}
INTN
EFIAPI
ShellAppMain(UINTN Argc, CHAR16 **Argv)
{
    EFI_STATUS Status = EFI_SUCCESS;
    EFI_HANDLE *HandleBuffer;
    UINTN HandleCount = 0;
    EFI_TREE_PROTOCOL *TreeProtocol;
    EFI_GUID gEfiGuid = EFI_TREE_SERVICE_BINDING_PROTOCOL_GUID;
    TREE_BOOT_SERVICE_CAPABILITY CapabilityData;
    EFI_PHYSICAL_ADDRESS EventLogLocation;
    EFI_PHYSICAL_ADDRESS EventLogLastEntry;
    BOOLEAN EventLogTruncated;
    TCG_PCR_EVENT *Event = NULL;
    BOOLEAN Verbose = FALSE;
    if (Argc == 2) {
        if (!StrCmp(Argv[1], L"--verbose") ||
            !StrCmp(Argv[1], L"-v")) {
            Verbose = TRUE;
        }
        if (!StrCmp(Argv[1], L"--help") ||
            !StrCmp(Argv[1], L"-h") ||
            !StrCmp(Argv[1], L"-?")) {
            Usage(Argv[0]);
            return Status;
        }
    }
    // Try locating EFI_TREE_SERVICE_BINDING handles
    Status = gBS->LocateHandleBuffer( ByProtocol,
                                      &amp;gEfiGuid,
                                      NULL,
                                      &amp;HandleCount,
                                      &amp;HandleBuffer);
    if (EFI_ERROR (Status)) {
        Print(L"No EFI_TREE_SERVICE_BINDING_PROTOCOL handles found.\n\n");
    }
    Status = gBS->LocateProtocol( &amp;gEfiTrEEProtocolGuid,
                                  NULL,
                                  (VOID **) &amp;TreeProtocol);
    if (EFI_ERROR (Status)) {
        Print(L"Failed to locate EFI_TREE_PROTOCOL [%d]\n", Status);
        return Status;
    }
    CapabilityData.Size = (UINT8)sizeof(CapabilityData);
    Status = TreeProtocol->GetCapability(TreeProtocol, &amp;CapabilityData);
    if (EFI_ERROR (Status)) {
        Print(L"ERROR: TrEEProtocol GetCapacity [%d]\n", Status);
        return Status;
    }
    // check TrEE Protocol present flag and exit if false
    if (CapabilityData.TrEEPresentFlag == FALSE) {
        Print(L"ERROR: TrEEProtocol TrEEPresentFlag is false.\n");
        return Status;
    }
    Print(L"          Structure version: %d.%d\n",
           CapabilityData.StructureVersion.Major,
           CapabilityData.StructureVersion.Minor);
    Print(L"           Protocol version: %d.%d\n",
           CapabilityData.ProtocolVersion.Major,
           CapabilityData.ProtocolVersion.Minor);
    Print(L"  Supported Hash Algorithms: ");
    if ((CapabilityData.HashAlgorithmBitmap &amp; EFI_TCG2_BOOT_HASH_ALG_SHA1) != 0) {
        Print(L"SHA1 ");
```



```
    }
    if ((CapabilityData.HashAlgorithmBitmap &amp;amp; EFI_TCG2_BOOT_HASH_ALG_SHA256) != 0) {
        Print(L"SHA256 ");
    }
    if ((CapabilityData.HashAlgorithmBitmap &amp;amp; EFI_TCG2_BOOT_HASH_ALG_SHA384) != 0) {
        Print(L"SHA384 ");
    }
    if ((CapabilityData.HashAlgorithmBitmap &amp;amp; EFI_TCG2_BOOT_HASH_ALG_SHA512) != 0) {
        Print(L"SHA512 ");
    }
    Print(L"\n");
    Print(L"          TrEE Present Flag: ");
    if (CapabilityData.TrEEPresentFlag) {
        Print(L"True\n");
    } else {
        Print(L"False\n");
    }
    Print(L"Supported Event Log Formats: ");
    if ((CapabilityData.SupportedEventLogs &amp;amp; TREE_EVENT_LOG_FORMAT_TCG_1_2) != 0) {
        Print(L"TCG_1.2 ");
    }
    Print(L"\n");
    Print(L"       Maximum Command Size: %d\n", CapabilityData.MaxCommandSize);
    Print(L"      Maximum Response Size: %d\n", CapabilityData.MaxResponseSize);
    Print(L"             Manufactuer ID: %s\n", ManufacturerStr(CapabilityData.Manufacture
rID));
    Status = TreeProtocol->GetEventLog( TreeProtocol,
                                        TREE_EVENT_LOG_FORMAT_TCG_1_2,
                                        &amp;amp;EventLogLocation,
                                        &amp;amp;EventLogLastEntry,
                                        &amp;amp;EventLogTruncated );
    if (EFI_ERROR (Status)) {
        Print(L"ERROR: TreeProtocol GetEventLog [%d]\n", Status);
        return Status;
    }
    Event = (TCG_PCR_EVENT *) EventLogLastEntry;
    Print(L"       Last Event PCR Index: %u\n", Event->PCRIndex);
    PrintEventType(Event->EventType, Verbose);
    PrintSHA1(Event->Digest);
    Print(L"                 Event Size: %d\n", Event->EventSize);
    if (Verbose) {
        PrintEventDetail(Event->Event, Event->EventSize);
    }
    Print(L"\n");
    return Status;
}
```

The code was built using the UDK2015 development environment on a X64 platform but should be easily portable to other UEFI development environments. As usual, it has not been extensively tested; I leave that up to you the reader.

Interestingly, the *TREE_BOOT_SERVICE_CAPABILITY* structure is not a packed structure – unlike most of the structures one encounters when programming at the firmware level.

Here is the build .INF:

```
[Defines]
  INF_VERSION                    = 0x00010006
  BASE_NAME                      = ShowTrEE
  FILE_GUID                      = 4ea87c51-7395-4ccd-0355-747010f3ce51
  MODULE_TYPE                    = UEFI_APPLICATION
  VERSION_STRING                 = 0.1
  ENTRY_POINT                    = ShellCEntryLib
  VALID_ARCHITECTURES            = X64
```



```
[Sources]
  ShowTrEE.c

[Packages]
  MdePkg/MdePkg.dec
  ShellPkg/ShellPkg.dec

[LibraryClasses]
  ShellCEntryLib
  ShellLib
  BaseLib
  BaseMemoryLib
  UefiLib

[Protocols]
  gEfiTrEEProtocolGuid         ## CONSUMES

[BuildOptions]

[Pcd]
```

I tested the utility on a Lenovo T450 laptop. This particular laptop comes with two TPMs – the default is a discrete TPM 1.2 chip and the second is the CPU-based Intel Platform Trust Technology (PTT). The discrete TPM does not support the EFI TrEE protocol. Most of the latest Intel processors come with PTT and support the TrEE protocol.

Here is the output from the utility when interacting with the Intel PTT TPM:

```
fs0> ShowTrEE.efi -h
Usage: ShowTrEE.efi [-v|--verbose]

fs0> ShowTrEE.efi
No EFI_TREE_SERVICE_BINDING_PROTOCOL handles found.

          Structure version: 1.0
           Protocol version: 1.0
  Supported Hash Algorithms: SHA1
          TrEE Present Flag: True
Supported Event Log Formats: TCG_1.2
       Maximum Command Size: 1280
      Maximum Response Size: 1280
             Manufactuer ID: INTC
       Last Event PCR Index: 4
                 Event Type: Boot Services Application
                 SHA1 Digest: 3B6803805866A6ECE02896437ECC78397D9338EE
                 Event Size: 158

fs0> ShowTrEE.efi -v
No EFI_TREE_SERVICE_BINDING_PROTOCOL handles found.

          Structure version: 1.0
           Protocol version: 1.0
  Supported Hash Algorithms: SHA1
          TrEE Present Flag: True
Supported Event Log Formats: TCG_1.2
       Maximum Command Size: 1280
      Maximum Response Size: 1280
             Manufactuer ID: INTC
       Last Event PCR Index: 4
                 Event Type: 80000003 Boot Services Application
                 SHA1 Digest: 3B6803805866A6ECE02896437ECC78397D9338EE
                 Event Size: 158
               Event Detail: 00000000: 1810DAB300000000007E000000000000 00000000000000007E
00000000000000 02010C00D041030A0000000001010600
                             00000030: 001D030506000000030506000100040l 2A0001000000000800
```



```
00000000000065 770000000000554546490000000000000
                              00000060: 0000000000000101040432005C006500 660069005C00740065
00730074005C00 530068006F00770054007200450045004500
                              00000090: 2E00650066006900000007FFF0400
```

Working in the TCG TCM space from the UEFI shell sometimes requires a considerable amount of investigation. There are few working code examples to experiment with. There is a lot of technical jargon to master. Good luck experimenting in this area!

