RUHR-UNIVERSITY BOCHUM

Chair for
System Security

Lecture Trusted Computing
Exercise No. 2, SS 2009

Exercise 2: Simple Programming - Extending a PCR Register

1 Introduction

In this exercise we will introduce the Trusted Computing Group (TCG) Software Stack and will learn how to develop (simple) trusted computing-enabled applications in the C programming language. The TCG Software Stack (TSS) is generally an entry point for any programmer writing such a trusted computing-enabled application. Therefore, the TSS provides an Application Programming Interface (API) that allows applications and the operating system to use the TPM.

1.1 TSS Architecture

Figure 1 depicts the general architecture of the TSS. In the following, we will explain the several layers and interfaces of the TSS architecture.

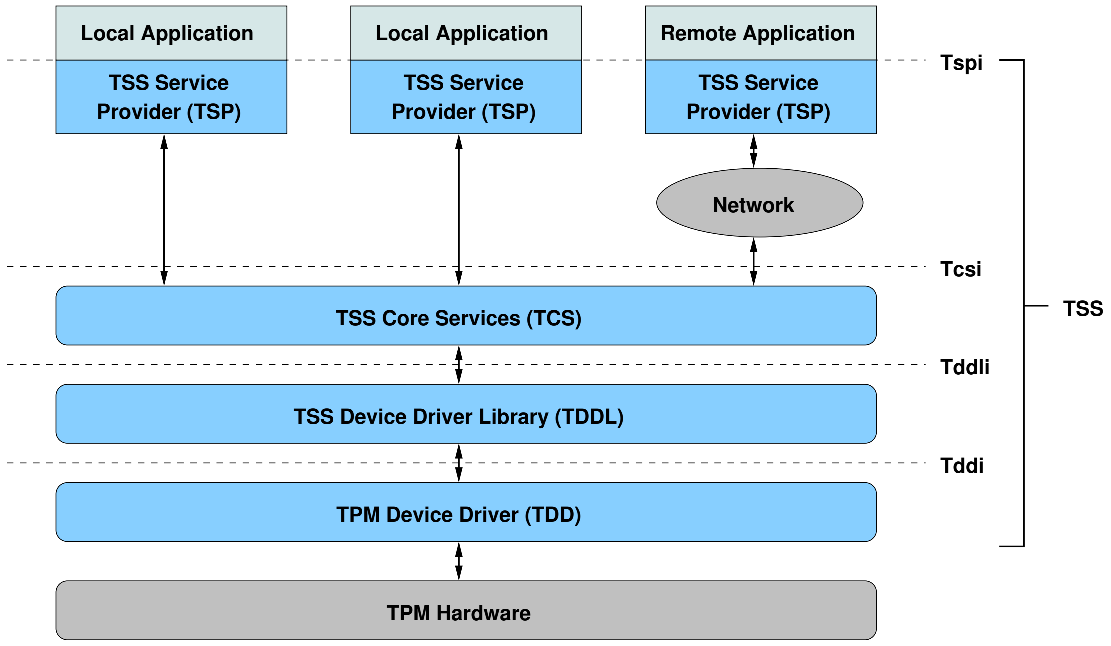

Figure 1: Architecture of the TSS

TSP At the top of the picture are the applications, local or remote, which have to use a TSS Service Provider (TSP) to employ TPM functionality. The TSP provides high-level TCG functions through a TSP interface (Tspi). Every application has its own TSP layer implemented as a shared object or a dynamically linked library.

TCS Generally, the TCG Core Service (TCS) layer provides management of the TPM resources and offers a common set of operations to the several TSPs. Moreover, the TCS layer handles and parses TPM commands.

TDDL The TSS Device Driver Library (TDDL) provides an interface to the TPM device driver. Through the TDDL it is possible to open or close the device driver, send and receive data blobs and get or set the attributes of the TPM.

TDD The TPM Device Driver (TDD) runs in kernel mode[1] and is the only component of the TSS that directly interacts with the TPM. The TDD forwards the commands received from the TDDL and returns the responses of the TPM back to the TDDL.

Each Tspi API is associated with an object type. The current TSS 1.2 specification [1] defines 13 object types, which are listed in table 1. In this exercise we especially need the context object and the TPM object, that we introduce in the remainder of this section.

Object Type	C Data Type	Available in TSS version
Context	TSS_HCONTEXT	1.1 and 1.2
Data	TSS_HENCDATA	1.1 and 1.2
Key	TSS_HKEY	1.1 and 1.2
Hash	TSS_HHASH	1.1 and 1.2
PCR Composite	TSS_HPCRS	1.1 and 1.2
Policy	TSS_HPOLICY	1.1 and 1.2
TPM	TSS_HTPM	1.1 and 1.2
Non-Volatile Data	TSS_HNVSTORE	1.2
Migratable Data Object	TSS_HMIGDATA	1.2
Delegation Family	TSS_HDELFAMILY	1.2
DAA Credential	TSS_HDAA_CREDENTIAL	1.2
DAA Issuer Key	TSS_HDAA_ISSUER_KEY	1.2
DAA Anonymity Revocation Authority Key	TSS_DAA_ARA_KEY	1.2

Table 1: TSP Object Types

1.2 Context Object and TPM Object

A context object is used to maintain a handle to the current TSP-library and to connect to a local or remote TCS provider. Additionally, the context object can be used to load keys and to store or to retrieve key objects. Each of the other TSP objects are created through a context object and are associated with the context object. Consequently, none of the remaining 12 TSP objects can be created if a context object was not created before. To create and close a context object, we can use the TSS-functions `Tspi_Context_Create` and `Tspi_Context_Close` in the following way:

[1]The different modes of operation are described profoundly in the book "Building a Secure Computer System" by Morrie Gasser

```
TSS_HCONTEXT hContext;

Tspi_Context_Create(&hContext);
Tspi_Context_Close(hContext);
```

To send commands to the TPM, it is still necessary to connect the TSP's context to a TCS provider. This can be done by using the TSS-function `Tspi_Context_Connect`. This function expects the destination address (IP address or hostname) of the TCS provider as the second argument. To connect to the local TCS provider a NULL pointer should be used. Furthermore, if a context object is connected to a TCS provider, then a TPM object is created implicitly. To retrieve a handle to the implicitly created TPM object, the TSS-function `Tspi_Context_GetTPMObject` can be used.

2 Theoretical Assignments (5 Points)

Please prepare the theoretical assignments **at home**. You have to hand in your answers at the beginning of each practical assignment!

1. Point out the differences between the TSP, TCS, and TDDL layers in the TSS!

2. What is integrity measurement? How is integrity measurement accomplished in the TPM?

3. A PCR register is extended according to the following formula, where SHA-1 is a hash algorithm and *data* the file or value that should be measured:
   $PCR_i = SHA-1(PCR_i||data)$
   Explain the reasons why the PCR registers are extended in that way!

   Explain why the designers of the TSS have not used the following formulas:

   1. $PCR_i = SHA-1(data)$

   2. $PCR_i = SHA-1(PCR_i)$

   3. $PCR_i = SHA-1(PCR_i \text{ xor } data)$

4. Assume that the PCR registers could be reset by a command while the system *is running*. Which security problem could arise in this case?

5. Describe the notions of *Secure Boot* and *Authenticated Boot*! What are the differences between them?

3 Practical Assignments

The goal of this exercise is extending an existing implementation of a process starter with TSS functions. Currently, the process starter computes a SHA-1 value of the first received argument, prints the hash to `stdout` and finally starts the process with the received arguments.

For this exercise we assume that we use a Linux operating system. Furthermore, the process starter is written in C and uses the libraries *libgcrypt* and *libtspi* which have to be linked against with `-l`. The process starter can be executed in the following ways:

1. `$ ./startproc /bin/bash`
   Prints the SHA-1 sum of */bin/bash* and starts a new shell.

2. `$ ./startproc /bin/ls /home`
   Prints the SHA-1 sum of */bin/ls* and lists the files of the home directory.

3.1 Get familiar with the process starter (1 Point)

1. Compile the program `startproc.c` (located in the directory */Exercise Data*) by using the `gcc` compiler! For successful compilation you have to tell the compiler that you want to include the libraries *libgcrypt* and *libtspi*. Write down the command you used to compile the program! Hint: If you are not familiar with `gcc`, read Chapter 2 from [3].

2. Run the program with the arguments used in the two examples above!

3.2 TSS Setup (6 Points)

As you surely observed, the process starter computes the hash value of a program that has to be executed and prints this value to `stdout`. Additionally, we want to write the computed hash value in a PCR register of the TPM using the TSS-function `Tspi_TPM_PcrExtend`.

As mentioned in the introduction, some TSS-functions can only be used if other TSS-functions has been invoked before. In the source code you will find an empty function `TSS_initialize()`. In this function block your task is to implement all the setup functions mentioned below, to run the `Tspi_TPM_PcrExtend` command successfully afterwards.

1. Create a context object! (refer to [4])

2. Connect the context object to the local TCS provider! (refer to [5])

3. Get a TPM Object! (refer to [6])

Hint: Refer to the manpages to find out which header files you have to include and how the particular TSS-function has to be used.

3.3 Specifying a PCR register (1 Point)

As you know from the lecture, the TPM holds 24 PCR registers, namely PCR_0 to PCR_{23}. In this section your task is to use the unix-tool `sha1sum` to determine a particular register index that is valid for your implementation only. Execute the following commands:

1. Create a file named *myname*: `$ touch myname`
2. Open the file and write your full name into the file (e.g., `$ nano myname`)
3. Save the file and compute the SHA-1 value of the file: `$ sha1sum myname`
4. Write down the last two hex digits of the calculated SHA-1 value and compute the appropriate decimal value modulo 17! The result is the index of the PCR register you should use for your implementation in the following section.

3.4 Extending a PCR register (4 Points)

Before you start to add the TSS-function `Tspi_TPM_PcrExtend` it is necessary that the setup functions are correctly implemented. If you still receive error messages during compilation contact your tutor.

1. In the source code you will find an empty function named `PCR_extend()`. Your task is to call the TSS-function `Tspi_TPM_PcrExtend` (refer to [7]) here! Use the PCR register you determined in the preceding task and set the value for `pPcrEvent` to NULL!
   Before you run the program for the first time, you should check the current value of your PCR register. If you enter the command `tpmmanager`, a new GUI appears where you can find the actual status of the TPM and the values of the PCR registers. Write down the value of your PCR register to be sure that your program really extends the PCR register.
   $PCR_{i=...}$:
2. After you have successfully implemented all of the TSS-functions mentioned above, you should finally extend the function block `TSS_close()` with TSS-functions that free the memory assigned to the context and finally closes the context.
3. Verify your implementation by running the program as in section 3.1.2! Check if the value of your PCR register has been changed.

3.5 Testing (3 Points)

1. Open the `tpmmanager` and write down the current value of your PCR register!
   $PCR_{i=...}$:

2. Execute your modified process starter with the two arguments */bin/ls* and */home*. Write down the computed hash value of */bin/ls* and the new value of your PCR register!
   /bin/ls:
   $PCR_{i=...}$:

3. Explain why the value of your PCR register is not equal to the computed SHA-1 sum of */bin/ls*? How was the value of your PCR register computed?

4. Prove that the value of your PCR register is correct regarding your implementation!
   Hint: The program `hexedit` allows you to store hex digits in a file. You can get the SHA-1 value of the file created using `hexedit` using the unix-tool `sha1sum`.

Appendix/Bibliography

References

[1] TCG Software Stack (TSS) Specification Version 1.2, `https://www.trustedcomputinggroup.org/specs/TSS/TSS_Version_1.2_Level_1_FINAL.pdf`

[2] A Practical Guide to Trusted Computing, D. Challener, K. Yoder, R. Catherman, D. Safford and L. Van Doorn, 2008, ISBN-10: 0132398427

[3] An Introduction to GCC, `http://www.network-theory.co.uk/docs/gccintro/`

[4] manpage of `Tspi_Context_Create`, `http://linux.die.net/man/3/tspi_context_create`

[5] manpage of `Tspi_Context_Connect`, `http://linux.die.net/man/3/tspi_context_connect`

[6] manpage of `Tspi_Context_GetTPMObject`, `http://linux.die.net/man/3/tspi_context_gettpmobject`

[7] manpage of `Tspi_TPM_PcrExtend`, `http://linux.die.net/man/3/tspi_tpm_pcrextend`