System Security
Constructor University
Dr. Jürgen Schönwälder

Module: MCSSE-CYB-02
Date: 2026-04-24
Due: 2026-05-01

Problem Sheet #7

Problem 7.1: *measured boot* (1+1+1+1 = 4 points)

Lets assume we have a TPM using a hash function producing 8-bit digests. We use a trivial hash function so that calculations using pen and paper are feasible. The hash function is implemented by the functions `hash_init()`, `hash_update()`, and `hash_digest()` as shown below (use the comments if you are not familiar with the details of the C language).

```
typedef uint8_t hash_t;

void hash_init(hash_t *hash)
{
    *hash = 0;                          // set the stored hash to zero
}

void hash_update(hash_t *hash, const uint8_t *data, size_t len)
{
    uint8_t *p = (uint8_t *) data;
    uint8_t h = *hash;                  // obtain the stored hash

    while (len--) {                     // while there are data bytes left
        h = h * 7 + (*p++);             // multiply hash by a prime number and
    }                                   // add the next byte to the product

    *hash = h;                          // update the stored hash
}

uint8_t hash_digest(hash_t *hash)
{
    return *hash;                       // return the stored hash
}
```

A platform configuration register (PCR) is an unsigned 8-bit number supporting the operations `pcr_reset()` and `pcr_extend()`.

```
typedef uint8_t pcr_t;

void pcr_reset(pcr_t *pcr)
{
    *pcr = 0;                           // reset the pcr to zero
}

void pcr_extend(pcr_t *pcr, const uint8_t *data, size_t len)
{
    hash_t hash;
    hash_init(&hash);                   // calculate a hash over
    hash_update(&hash, pcr, 1);         // the current pcr value
    hash_update(&hash, data, len);      // and the data bytes
    *pcr = hash_digest(&hash);
}
```

A certain PCR is used for measured boot. We assume that the measured boot covers an initial hardware configuration check, a first stage boot loader, a second stage boot loader, and finally the loading of the operating system kernel.

a) The initial measurement of the hardware components yields the hexadecimal value 0xCA, which is extended into the PCR. What is the value of the PCR after the extend operation?

b) The measurement of the first stage boot loader yields the hexadecimal value 0xFE, which is extended into the PCR. What is the value of the PCR after the extend operation?

c) The measurement of the second stage boot loader yields the hexadecimal value 0xFA. What is the value of the PCR after the extend operation?

d) The measurement of the kernel yields the hexadecimal value 0xCE. What is the value of the PCR after the extend operation?

Problem 7.2: *hardened linux kernel* (1+1+1+1+1+1 = 6 points)

The Linux kernel coming with your Linux distribution is designed to run on many different hardware configurations. Hence, it comes with a large set of loadable kernel modules. The goal of this exercise is to build your own kernel targted to your hardware, ideally with all modules built into the kernel and additional kernel hardening features enabled.

a) Install a fresh virtual machine running a default minimal installation of Debian or Ubuntu.

b) Install the development tools necessary to compile your own kernel.

c) Build a kernel matching the original configuration of your distribution.

d) Build a minimal kernel with all necessary modules built statically into the kernel.

e) Enable additional kernel hardening options as you see fit.

f) Compare the size of the initial kernel you have built against the size of the hardened kernel.