Embedded Security (Summer Term 2018)
Dr. Stefan Nürnberger
CISPA Helmholtz Center i.G. / Saarland University

Exercise #4 (24 Points)

Date due: June 25., 9:59 am

Submission Instructions

The solutions must be submitted via CMS until **June 25., 9:59 am** in PDF format. Make sure that you clearly indicate the names and matriculation numbers of all students who authored the solutions **on the sheet itself**. Please do not share info that is not publicly known (via the lecture or the exercise sheet), especially not in the forum. If you have questions that might leak deduced information, write a mail to the tutors or visit the office hours instead of creating a post in the forum. If you wrote any code to solve the exercises, please add it to your solution. **Undocumented code is not accepted**. If it is more than a few lines, please put it into a **dedicated appendix** and refer to it from your solution. We only accept solutions written in *C*, *C++* and *Python* if not stated otherwise. Be kind and format your code nicely.
Label your solutions corresponding to the exercises on the task sheet. Handwritten solutions are accepted, but they have to be readable; otherwise points might be deducted.

1 Designing secure Software (17 points)

Once again, *MetaCortex* needs your assistance. Unfortunately they did not pay attention to the IoT trend and now they are far behind with designing software for embedded systems (not speaking of security). They decided to make a compromise and ask you to design a secure over-the-air software update mechanism, so they can quickly sell the devices now and fix the bugs later.

For this exercise you will work with ESP-32 shields. The new image you should update to is located here[1], its signature here[2]. For creating the signature we hashed the file (SHA-256) and encrypted the hash with textbook RSA. The content of*image.bin.sig* represent the result in base64. You may want to download the public PEM key[3] which you can use to verify the signature.

You can assume that until the deadline these files do not get compromised. After submitting you can expect us to run your software in an environment where an attacker has full control over the network and the server (for sure he will not have access to the private key which can be used to sign other image files). Note that we might change the image file *image.bin* and the signature *image.bin.sig*, so do not hardcode the files signature into your program.

[1] `https://emsec.cispa.saarland/files/image.bin`
[2] `https://emsec.cispa.saarland/files/image.bin.sig`
[3] `https://emsec.cispa.saarland/files/pubcert.pem`

You need to setup your IDE first. We recommend the ESP-IDF toolchain[4]. We cannot assure you that the task can be successfully completed with the Arduino IDE.

(2 points) (a) In order to keep the update process simple, *MetaCortex*'s first idea is to not verify the image, but just establish a secure connection over HTTPS. Briefly explain why it is important to check the integrity of a download with a signature.

(3 points) (b) The *esp_ota_ops* library includes tools which simplify the update process. Given the header file[5] and an example[6], briefly explain how the update process works. In order to do so, give an example where you outline *which* parts of the flash is used for *what* during the update process. Furthermore state one drawback of this mechanism.

(10 points) (c) Implement your approach. Your program should...

(1) ...connect to the WiFi **SecureUpdate** with the password **EmsecExercise42018** and obtain an IP via DHCP.

(3) ...connect to the Server given above through HTTPS.

(5) ...download the flash file and its certificate, verify the image and boot from it if the signature is valid. It should fail downloading the image from her e[7].

(1) ...blink the onboard LED if there was an error during the update process.

Hint: If you somehow do not manage to perform a successful SSL handshake with *emsec.cispa.saarland*, you maybe want to check if you can do one with another website.

(2 points) (d) *MetaCortex* now created the software for a media control system. Unfortunately, the image size is fairly huge (about 3MB) in comparison to the 4MB flash size. They propose to create a small custom second stage bootloader (a few hundred KB) that can automatically connect to a network, download the image from the server in small chunks, feed these chunks to a hash function (like SHA256) and if the resulting hash is successfully checked against the certificate, the server is trusted and the image file seen as not compromised. Then the image file is downloaded again by the bootloader, overwriting the old firmware. What could possibly go wrong?

Important note: You should submit both code *and* the compiled binary. Make sure to submit the image for the whole flash, not only the flash image starting at vector 0x10000. Your submission must be a zip archive that contains these elements:

[4] `https://github.com/espressif/esp-idf`

[5] `https://github.com/espressif/esp-idf/blob/master/components/app_update/include/esp_ota_ops.h`

[6] `https://github.com/espressif/esp-idf/blob/master/examples/system/ota/main/ota_example_main.c`

[7] `https://emsec.cispa.saarland/files/image.bin.evil`

1. A directory that contains all the files necessary to build the project. This includes the Makefile. Make sure that calling *make* in this directory results in the submitted binary.
2. Your solution as a PDF.
3. The binary file.

2 Wireless problems (7 points)

Many IoT-devices use wireless interfaces for communication to be as flexible as possible.

(2 points) (a) Briefly explain why bluetooth is not susceptible to narrowband jamming. Explain how this technique works.

(2 points) (b) Briefly explain why the Blueborne attack collection is especially critical for IoT-devices. **Hint:** Think about which Android version runs today on a Samsung Galaxy S4 device and what that means for software security.

(2 points) (c) Describe how you can get access to the encrypted communication of a Zigbee Light Link device. Consider how the gateway and the Zigbee device agree on a common key.

(1 point) (d) Briefly explain a way for the gateway and the Zigbee device to securely agree on a common key.