Embedded Security (Summer Term 2019)
Dr. Stefan Nürnberger
CISPA Helmholtz Center i.G. / Saarland University

Exercise #2 (26 Points)

Date due: May 29, 23:55

Submission Instructions

- The solutions must be submitted via CMS as a PDF.
- Put names and matriculation numbers of all students who authored the solutions **on the PDF itself**.
- Explain how you solved each task (otherwise 0 points).
- If not stated otherwise, code must be written in *C*, *C++* or *Python*.
- Additional files (e.g. source code) should be added to a .zip (with the PDF solution)
- 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.
- 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.

1 General info (0 points)

For this exercise **you will work with shared hardware! Start working early!** The hardware is located in the student room 2.17 at CISPA. There you will find an instrument cluster from a *Renault Zoe* (task 2) and from a *Seat Ibiza* (task 3). For the third task you can communicate to the Arduino via the Ethernet HUB we provide – details on how to send/receive data in the tasks. Please keep in mind that if multiple teams send data at the same time, collisions might occur: **synchronise to avoid such scenarios**. Please, **do not overwrite the firmware on the Arduinos!**

Finally, the CISPA building is accessible only from 9am to 6pm (you can stay inside until later – but you won't be able to enter again if you go out) during weekdays. If you want to have access outside these hours and during weekends, send an email to **sohyeon.park@cispa.saarland** with your **UdS card number** which is located on the top-left corner of the card (above the picture).

2 Teach me how to drive (6 points)

Since the day you helped out your friend to extract the key from the compromised device, your name is on everyone's lips. You recently upgraded your office and are now relaxing in a hammock, as the telephone wakes you from your dreams. The news of your skills got around quickly, so the car manufacturer from the next city, *FolksCar*, asks you for your help. They want to upgrade their car fleet to drive autonomously and have already developed a perfect self-driving software. Unfortunately they lost the documentation of their CAN bus protocol (rumor has it that there was never one), so they are unable to send the command which displays the correct speed and state of car to the driver. You agree kindly to work for them, under the condition that they send you a dump such that you can work on the task at your office.

For this task, we provide you a file containing a snapshot of a Renault Zoe CAN traffic. Each row in the file encodes a can frame: the first 3 characters are the frame ID, followed by a #, followed by up to 16 HEX characters which build 8 bytes pairwise. The related Arduino receives a UDP packet with a payload that represents a CAN frame. The formatting is the same as the file (example: 2b9#deadbeefc0cac01a). You can send UDP packets to the Arduino at IP **192.168.1.176** and port **8888** (just connect with the provided ethernet cable).

If something goes wrong, the Arduino responds with a debug UDP packet with the error message back to your IP at port 5000. You need a static ip for this exercise. If you are unfamiliar with it, you may want to look up how to configure a static ip.

(2 points) (a) As a first step, you need to make the cluster work. If you do everything right the cluster should turn (and stay) ON (some errors might still be visible – ignore them). List the frequencies you choose for each ID and motivate your choice and reasoning. Reasonable frequencies are between 10 to 500 milliseconds.

(4 points) (b) You need to discover which ID the speedometer listens to and the frame that sets the speed to 136km/h. Similarly, you need to discover which bit of frame 0x652 makes the writing "ready" disappear from the cluster. Briefly and precisely explain how you got there and how the data is interpreted.

3 Baby you can drive my car! (12 points)

You successfully completed *FolksCar*'s job and as a reward they gifted you their newest car, the model *Croquet 7*. The self-driving works like a charm; the only drawback is that you can no longer drive on your own. You know, however, that the car would let you drive in case of malfunction. Luckily, the car has a built-in WiFi function to allow a smartphone enabling the cars seat heating function remotely. You now search for a "hidden function" to fake the engine's temperature such that it thinks it's malfunctioning.

You got your hands on some code driving the media system. It is included in the zip file. **The task involves buffer overflows: please reset (by pushing the reset button) the Arduino if it stops working.**

(1 point) (a) Briefly explain what buffer overflows are.

(1 point) (b) Briefly explain how one can find vulnerabilities that can be exploited using a buffer overflow.

(1 point) (c) Briefly explain two ways to prevent buffer overflows.

(2 points) (d) The Arduino is broadcasting each frame it transmits on IP address 255.255.255.255 and port 5000 (the frame format is the same as before). Your first task is to parse these packets and make a table listing (i) all transmitted IDs, (ii) whether the payload changes or not, and (iii) the (approximate) transmission frequency.

(1 point) (e) By just analysing the logs, which frame(s) might be transmitting the speed? Motivate your reasoning.

(1 point) (f) Again, by just analysing the logs, which frame(s) are most likely to contain the airbag status (ON/OFF)? Motivate your reasoning.

(5 points) (g) Now analyse the code which is part of the infotainment system.

- State at which line the program is vulnerable to buffer overflows.
- Sketch your attack and what your benefit would be.
- Finally, state the payload of the UDP packet which lets you send the payload **0x33, 0xed, 0x00, 0x00, 0x00, 0x58, 0x52, 0x00** with the id **0x288** as a CAN message.
- If sent in the right frequency for several times, the engine temperature should overheat.

Figure 1: Control light for engine temperature

We highly recommend you to actually mount the attack, as a not working approach results in no points. If your attack was successful, the engine overheat light should come on. **The Arduino can be contacted at IP 192.168.1.177 and port 8888 – CAN packets and error messages are encoded the same as before**. You can connect to the Arduino via the switch.

4 The bus CAN do. (8 points)

(1 point) (a) Briefly state and explain a reason why messages must be transmitted periodically, even though the state of the corresponding device did not change.

(1 point) (b) When designing the CAN architecture, the manufacturer has to ensure that bus utilisation is never too high when designing all messages. Given the frames and frequencies you utilised for exercise **2**: what is the CAN bus utilisation (in %) on a 500kbit/s bus (you may ignore bit stuffing).

(1 point) (c) In task **2** we learned how to manipulate CAN frames in order to show spoofed information to the driver. In most cars, our attack would not set the actual speed of a car, i.e. it would not accelerate or brake according to your spoofed speed. Briefly explain why.

(1 point) (d) *FolksCar* were impressed by your ability to spoof the speed of the car. However, they are now afraid that if someone finds a flaw in their quality software™, someone might be able to set the speed of the car while someone is driving. They instructed you again to find solutions for this problem. You learned that the CAN bus protocol is old and it is not possible to verify the integrity of the message. As cars are getting more and more connected, this topic becomes more and more interesting, hence researchers proposed different ways of tackling this problem. One such technique is explained here[1].
Briefly explain what is used to secure the integrity of the messages.

(4 points) (e) The authors decided to use symmetric cryptography with static keys. List one advantage and one disadvantage each for instead using asymmetric cryptography or using a key exchange to establish a common key.

[1] `https://link.springer.com/chapter/10.1007/978-3-662-53140-2_6`, only free in eduroam