Embedded Security Systems
Laboratory report for List 4

Grzegorz Zaborowski 236447, Mikołaj Grzegrzółka 241073

January 30, 2022

Task

Embedded systems credentials breaking and time side-channel analysis. You will be equipped with the program code for Arduino Uno with implementation of following features:

- Door keypad - PIN (for getting access)
- Login/Password (for getting privileges)
- Red herrings (for misleading adversary)

Upload the code to the Arduino and launch the device. Your task is to overcome each security means by using different analysis methods and prepare scripts / toolkits facilitating attacks. You should be able to establish connection between Arduino and your computer via serial port in order to perform the communication or use another Arduino. Your ultimate goal is changing the blinking frequency of the device.

Report

Software and equipment used:

- Arduino Uno
- Laptop
- Arduino IDE
- Hex Editor (hexdump or other)
- Node JS

Running a program

The first obvious step was to download the file/program attached to the task. It is a compiled program for Arduino Uno. We never sent each other compiled programs for the Arduino platform, so we had to find a way to upload it to our device. Analyzing the Arduino IDE, we did not find an option that would allow us to upload the program. We managed to find information on the Internet that the Arduino IDE uses the `avr-gcc` compiler and uses `avrdude` to upload the program to the board. We used the latter program to upload our file to the platform.

```
#!/bin/bash
avrdude -C avrdude.conf -F -V -c arduino -p ATMEGA328P -P COM3 -b 115200
    -U flash:w:code.ino.hex
```

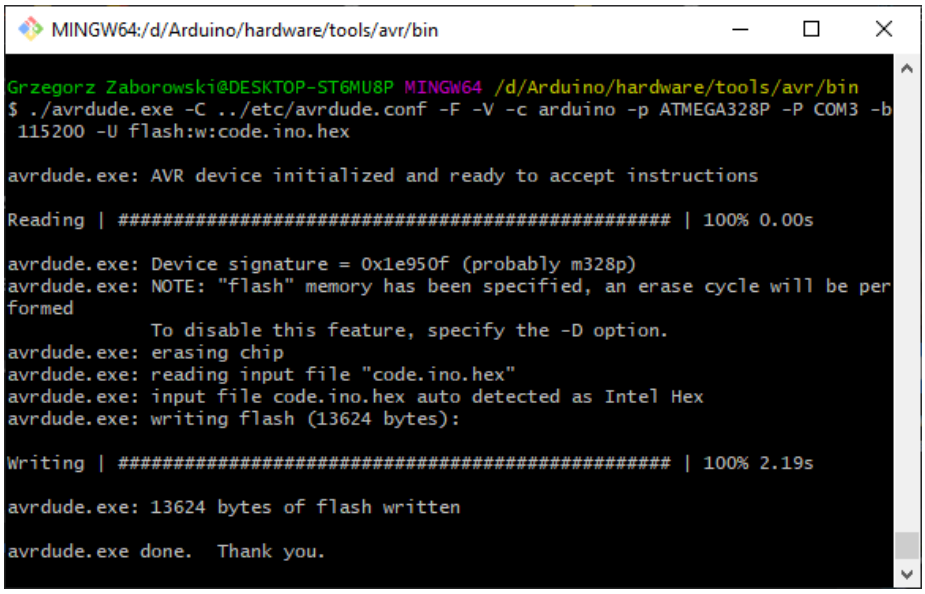

Figure 1: Uploading the program to the microcontroller

After starting the program, we used the Arduino IDE serial port monitor to establish connection using 9600 baud. We first message which we received was: 'CameraKey Driver - 2019'. Usually command line interface use some phrases to do some actions so we manually tried all single character and longer popular strings as: help, man, etc. As result we observed that avaliable options are: 's', 'd', 'l', 'c' and '¿. It is also worth noting that after typing 'help', the result was the same as for 'l', which means that the program reads the sent phrase character by character. Option 's' and 'c' returns 'Denied', 'l' list files, 'd' print some number, '?' show available options (see Figure 2). Due to the fact that

at this stage we did not know more what we can do here, we moved to the file analysis using hexdump.

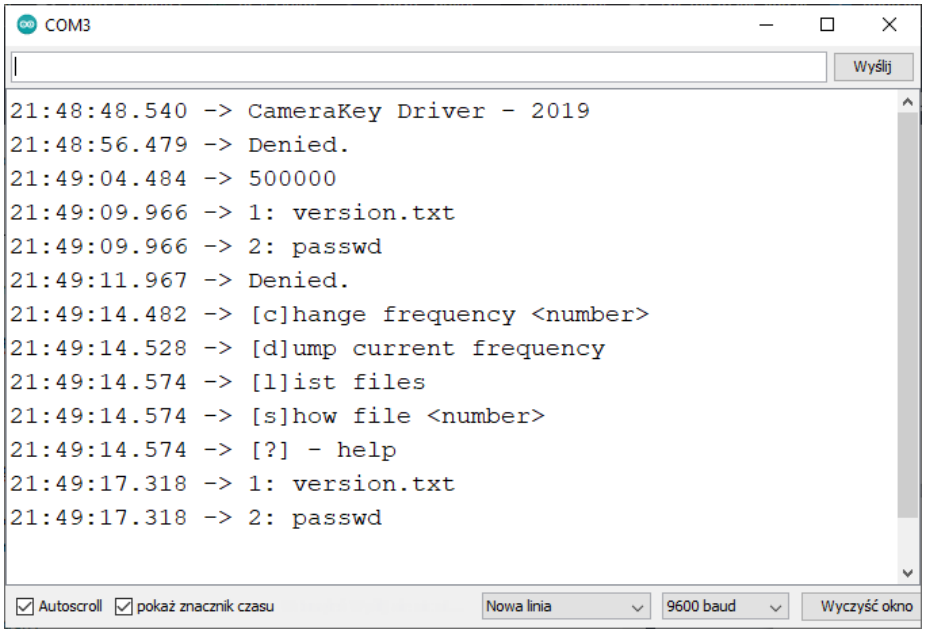

Figure 2: Avaliable options

Hexdump

Hexdump is a utility that displays the contents of binary files in hexadecimal, decimal, octal, or ASCII. It's a utility for inspection and can be used for data recovery, reverse engineering, and programming.

A typical Hexdump representation divides the binary data into 8-bit bytes and displays the value of each byte as a two-digit, zero-padded hexadecimal number (ranging from 00 to FF.) In case of Hexdump we are dumping hex values directly from the Ram of a binary file. This process is usually used for debbuging of a software. We can consider standard binary file as a compiled file which can run on a particular CPU. The data stream studied this way could be e.g.:

- a file read from a disk device
- raw bytes read from a disk device (displaying the partitioning or filesystem structures)
- something captured from a serial line

- something captured from a network device (IP packets, Ethernet frames) a memory dump
- data captured from some communications bus (USB, I2C etc.)

Tools which allow viewing data this way are useful when debugging file formats or communication protocols as they allow seeing the raw values of each byte in the stream unambiguously, without any interpretation or processing.

Hexdumps and hex editors are also useful for reverse engineering and hacking since an unobstructed view to the raw data often allows deducing and reasoning many things about the structure and contents of a file or a communications protocol even if its format is not publicly documented.

Decoding the file we got some information which we already known (see Figure 3). New information for as was: 'Admin Mode' and one string which looks like hash.

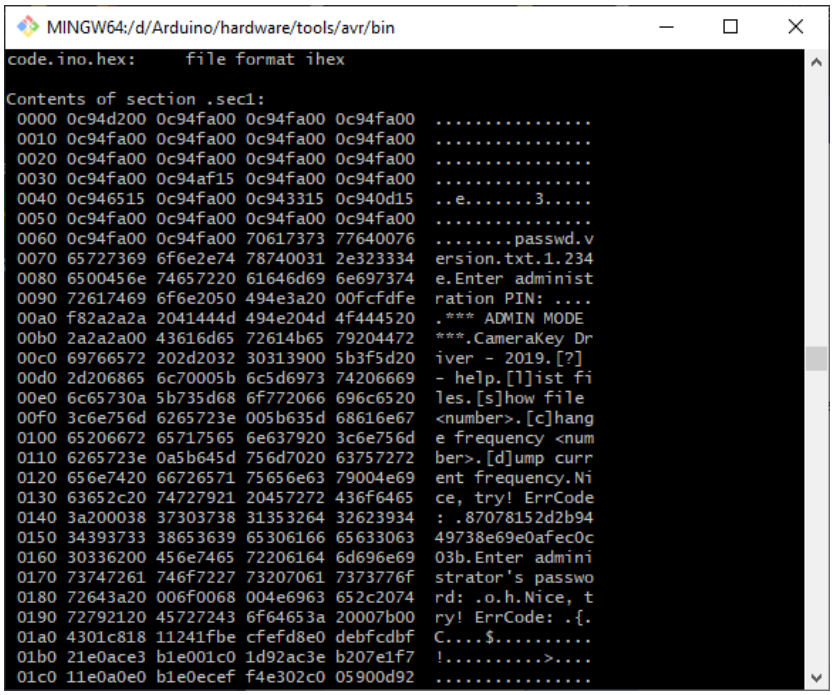

Figure 3: Hexdump

Admin mode

While analyzing the file, we discovered that there is some string "*** ADMIN MODE ***" there, which made us believe that there was some way to get ad-

ministrator privileges. Based on our knowledge that manufacturers of certain chips / controllers block various options on devices (see curiosity below) e.g. by desoldering or replacing a resistor, we searched for various information on the internet how you can get something like this with an Arduino. Following one answer (which we found very logical), we decided that it is possible for the program to read the state of a pin. We found that connecting pin 12 to the GDN puts the program in admin mode.

Curiosity: the same models of graphics cards use GDDR memory from different manufacturers, which cannot be freely changed. In the NVIDIA 2XXX or 3XXX series graphics cards, it was enough to change the place of one resistor to switch BIOS between Samsung memories and Micron memories (and probably two soldered resistors for Hynix memory) – if someone is interested in the repair / electronics of graphics cards, I recommend the Polish Youtube channel Forest.

Attack on PIN

To get access to the admin mode it was necessary to enter the PIN. While analyzing the file, we were unable to find anything that could resemble a PIN, so we decided to conduct an attack brute force. Knowing that the PIN consists of 4 digits, we had only 10,000 combinations, so breaking the PIN was very quick (see Figure 5). For this purpose, we wrote a program in Node JS that tries all possible combinations (it is worth mentioning that the program had no lock after entering the wrong pin N times).

```
const SerialPort = require('serialport');
const Readline = require('@serialport/parser-readline');
const port = new SerialPort('COM3', { baudRate: 9600 });
const parser = port.pipe(new Readline({ delimiter: '\n' }));

port.on('open', () => {
  console.log('Serial port open');
  main();
});

const serialReadLine = () => {
  return new Promise(resolve => {
    parser.once('data', data => {
      resolve(data);
    });
  });
};

const serialWrite = message => {
  return new Promise((resolve, reject) => {
    port.write(`${message}\n`, err => {
      if (err) reject(err);
      else resolve();
    });
  });
};
```

```
27
28 const main = async () => {
29   // skip first line
30   await serialReadLine();
31
32   // start timer
33   console.time('brute_force');
34
35   const characterArray = ['0','1','2','3','4','5','6','7','8','9'];
36
37   // generate all possible PINS
38   const allPINS = [];
39   for (const c1 of characterArray)
40     for (const c2 of characterArray)
41       for (const c3 of characterArray)
42         for (const c4 of characterArray)
43             allPINS.push(`${c1}${c2}${c3}${c4}`);
44
45   // brute force
46   for (const pin of allPINS) {
47     serialWrite(pin);
48
49     if ((await serialReadLine()) != 'Enter administration PIN: \r') {
50       console.log(`PIN is: ${pin}`);
51       break;
52     }
53   }
54
55   // stop timer
56   console.timeEnd('brute_force');
57   process.exit(0);
58 };
```

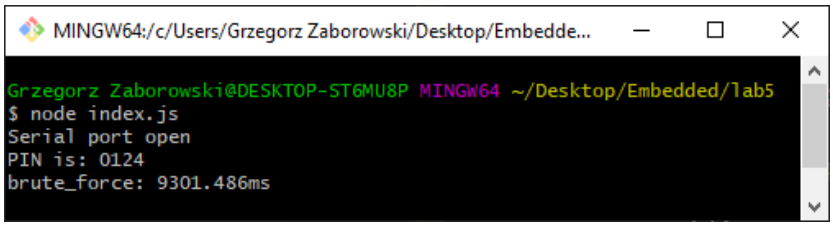

Figure 4: Brute force attack on PIN

Administrator password

When we tried to change the frequency (in admin mode) we got this message 'Enter administrator's password', so we need to find some way to crack password. In admin mode we could see what was in previouse listed files:

- passwd – 87078152d2b9449738e69e0afec0c03b
- version.txt – 1.234e

The string in passwd file looks like hash. Based on the hash length it looks like 128 bits, and the most popular hash of this length is md5. We also couldn't find any matching password in various hash databases on the internet. Our first thought was brute force attack (again!) but during test it was really slow so we abandoned it. After that we tried to use hashcat to decode that hash.

Hashcat

We used the standard Hashcat on Kali Linux password recovery tool. The first attempt at hashing by itself failed due to an attempt to find using an english wordlist. Using the polish wordlist, hashcat gave his potential assumptions about the password, but the program process itself did not crack the password, but was identified as Exhausted. Most likely the cracked password is not in the wordlist.

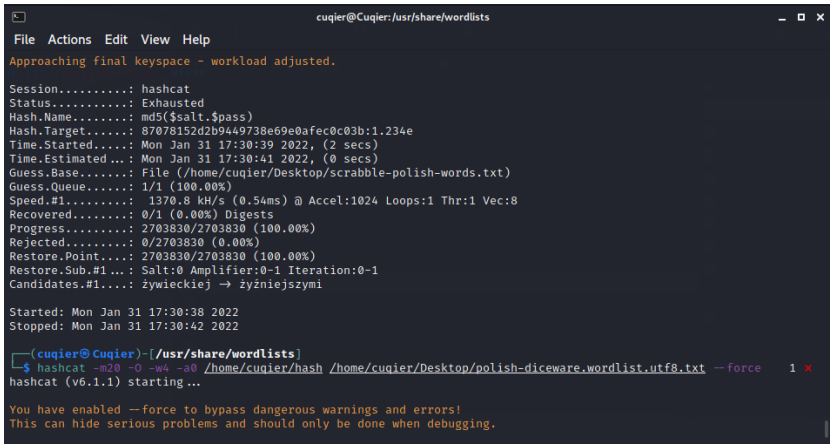

Figure 5: Hashcat

Summary

This list was not an easy one, but it was definitely the most interesting of all. First of all, it is a task to which you can use various techniques that we learned during classes. We managed to discover most of the things in the list. We analyzed the entire file and got all the useful things out of it. We managed to get into admin mode using a brute force attack to get a PIN. The only thing we have failed is to obtain the administrator password. The attack brute force for the administrator password seemed very slow (Arduino is a bottleneck), which may also be the conclusion that a lot of calculations are performed there (hash is counted). Despite trying to use the potential hash we found in the passwd file, we were unable to find a password for it. We tried to find it in the hash databases first, then using the hashcat and finally using the value from version

as the salt for the hash. Unfortunately, all our attempts to find the password for the hash have failed. We still had the idea of changing the hash directly in the program file, but we didn't know if it would work or if it was allowed, and we ran out of time.