

Embedded Security Systems

Laboratory report for List 4

Grzegorz Zaborowski 236447, Mikołaj Grzegorzówka 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
```

```
MINGW64:/d/Arduino/hardware/tools/avr/bin
Grzegorz Zaborowski@DESKTOP-ST6MU8P MINGW64 /d/Arduino/hardware/tools/avr/bin
$ ./avrdude.exe -C ../etc/avrdude.conf -F -V -c arduino -p ATMEGA328P -P COM3 -b 115200 -U flash:w:code.ino.hex

avrdude.exe: AVR device initialized and ready to accept instructions

Reading | ##### | 100% 0.00s

avrdude.exe: Device signature = 0x1e950f (probably m328p)
avrdude.exe: NOTE: "flash" memory has been specified, an erase cycle will be performed
        To disable this feature, specify the -D option.
avrdude.exe: erasing chip
avrdude.exe: reading input file "code.ino.hex"
avrdude.exe: input file code.ino.hex auto detected as Intel Hex
avrdude.exe: writing flash (13624 bytes):

Writing | ##### | 100% 2.19s

avrdude.exe: 13624 bytes of flash written

avrdude.exe done.  Thank you.
```

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 available options are: `'s'`, `'d'`, `'l'`, `'c'` and `'i'`. 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.

```
COM3
21:48:48.540 -> CameraKey Driver - 2019
21:48:56.479 -> Denied.
21:49:04.484 -> 500000
21:49:09.966 -> 1: version.txt
21:49:09.966 -> 2: passwd
21:49:11.967 -> Denied.
21:49:14.482 -> [c]hange frequency <number>
21:49:14.528 -> [d]ump current frequency
21:49:14.574 -> [l]ist files
21:49:14.574 -> [s]how file <number>
21:49:14.574 -> [?] - help
21:49:17.318 -> 1: version.txt
21:49:17.318 -> 2: passwd

Autoscroll [x] pokaż znacznik czasu Nowa linia 9600 baud Wyczyść okno
```

Figure 2: Available 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 debugging 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.

```

MINGW64:/d/Arduino/hardware/tools/avr/bin
code.ino.hex:      file format ihex

Contents of section .sec1:
0000 0c94d200 0c94fa00 0c94fa00 0c94fa00 .....
0010 0c94fa00 0c94fa00 0c94fa00 0c94fa00 .....
0020 0c94fa00 0c94fa00 0c94fa00 0c94fa00 .....
0030 0c94fa00 0c94af15 0c94fa00 0c94fa00 .....
0040 0c946515 0c94fa00 0c943315 0c940d15 ..e.....3....
0050 0c94fa00 0c94fa00 0c94fa00 0c94fa00 .....
0060 0c94fa00 0c94fa00 70617373 77640076 .....passwd.v
0070 65727369 6f6e2e74 78740031 2e323334 .....ersion.txt.1.234
0080 6500456e 74657220 61646d69 6e697374 e.Enter administ
0090 72617469 6f6e2050 494e3a20 00fcfdfe ration PIN: ....
00a0 f82a2a2a 2041444d 494e204d 4f444520 .*** ADMIN MODE
00b0 2a2a2a00 43616d65 72614b65 79204472 ***.CameraKey Dr
00c0 69766572 202d2032 30313900 5b3f5d20 iver - 2019.[?]
00d0 2d206865 6c70005b 6c5d6973 74206669 - help.[l]ist fi
00e0 6c65730a 5b735d68 6f772066 696c6520 les.[s]how file
00f0 3c6e756d 6265723e 005b635d 68616e67 <number>.[c]hang
0100 65206672 65717565 6e637920 3c6e756d e frequency <num
0110 6265723e 0a5b645d 756d7020 63757272 ber>.[d]ump curr
0120 656e7420 66726571 75656e63 79004e69 ent frequency.Ni
0130 63652c20 74727921 20457272 436f6465 ce, try! ErrCode
0140 3a200038 37303738 31353264 32623934 : .87078152d2b94
0150 34393733 38653639 65306166 65633063 49738e69e0afecOc
0160 30336200 456e7465 72206164 6d696e69 03b.Enter admini
0170 73747261 746f7227 73207061 7373776f strator's passwo
0180 72643a20 006f0068 004e6963 652c2074 rd: .o.h.Nice, t
0190 72792120 45727243 6f64653a 20007b00 ry! ErrCode: .{.
01a0 4301c818 11241fbc cfe4d8e0 debfcd8f C....$.
01b0 21e0ace3 b1e001c0 1d92ac3e b207e1f7 !.....>....
01c0 11e0a0e0 b1e0ecef f4e302c0 05900d92 .....

```

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-



administrator 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).

```
1  const SerialPort = require('serialport');
2  const Readline = require('@serialport/parser-readline');
3  const port = new SerialPort('COM3', { baudRate: 9600 });
4  const parser = port.pipe(new Readline({ delimiter: '\n' }));
5
6  port.on('open', () => {
7    console.log('Serial port open');
8    main();
9  });
10
11 const serialReadLine = () => {
12   return new Promise(resolve => {
13     parser.once('data', data => {
14       resolve(data);
15     });
16   });
17 };
18
19 const serialWrite = message => {
20   return new Promise((resolve, reject) => {
21     port.write(`${message}\n`, err => {
22       if (err) reject(err);
23       else resolve();
24     });
25   });
26 };
```



```

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 };

```

```

MINGW64:/c/Users/Grzegorz Zaborowski/Desktop/Embedde...
Grzegorz Zaborowski@DESKTOP-ST6MU8P MINGW64 ~/Desktop/Embedded/lab5
$ node index.js
Serial port open
PIN is: 0124
brute_force: 9301.486ms

```

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 previous 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.