Embedded Security Systems - Report 4

Bartosz Drzazga, Mateusz Jachniak

August 31, 2021

Contents

1 List 4

1.1 Task

Embedded systems credentials breaking and time side-channel analysis. You will be equipped with the program code for Arduino Uno with a few 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.

Remember about everything you have learned so far!

2 Uploading code to the board (Arduino Uno)

Our first task was to upload the new code to the device. The file we were given is a `.hex` compiled file ready to flash. It is an Intel HEX file that conveys binary information in ASCII text form. Usually, when working with Arduino, we would just press "Upload" button in Arduino IDE and our code would get compiled and uploaded to the device. This time we do not have the source code, just the compiled program. It is still possible to upload the code manually.

Understanding steps executed by Arduino IDE when uploading a new code could be helpful. The first step is compiling the source code. This step is complicated, involves running different compilers on our sources, libraries and Arduino libraries. We are not interested in this procedures, we only care about the final product of linking all of the pieces witch is a single `.hex` file. Just like the one we were given.

The final step Arduino IDE executes is flashing the device. This is achieved with only one command:

```
avrdude -C//etc/avrdude.conf -v -patmega328p -carduino -P<path to the device>
    -b<baud rate> -D -Uflash:w:<path to the .hex file>:i
```

At this point we knew everything we needed to upload the given file. The command we used was:

```
avrdude -C//etc/avrdude.conf -v -patmega328p -carduino -P/dev/ttyUSB0 -b115200
     -D -Uflash:w:/home/dell/Desktop/studies/ESS/lab5/code.hex:i
```

In order to confirm the device now runs the freshly flashed program we connected to the Arduino Uno board with a serial connection. The device greeted us with a message:

```
CameraKey Driver - 2019
```

3 Getting basic information about the program

After flashing the given code and establishing a connection with the device we started to play around with it in order to collect information. We start by typing in special characters, like `>`, `$` and `?`. We thought we were lucky (spoiler alert: it is a red herring) because we got something interesting basically in the first try. After sending `?` to the device, Arduino responds with:

```
[c]hange frequency <number>
[d]ump current frequency
[l]ist files
[s]how file <number>
[?] - help
```

Of course, we decide to try each of this options displayed in the menu. The dumped frequency is:

```
500000
```

Listing files returns:

```
1: version.txt
2: passwd
```

However, some of the options are accessible only for privileged users. We were unable to change frequency nor see contents of both files. Each time we got:

```
Denied.
```

We spent way too much time trying to manually type in different characters, looking for secret commands and so on. We observed that "unsupported" characters are ignored, i.e. `login` and `l` produce the same result which is the above list of files; `pwd` dumps the current frequency just as `d` does. We decide to construct a simple Python script that sends different combinations of inputs and let it run for a while.

```
import serial
import time
import string

port = '/dev/ttyUSB0'

arduino = serial.Serial(port,9600,timeout=0.1)

def getAllAvaibleInput():
    results = {}
    for character in string.printable:
        start = time.perf_counter()
        arduino.write(character.encode())
        reply = b''
        # To get the whole response
        while message :=arduino.read(8):
            if type(message) == bytes:
                reply += message
        end = time.perf_counter()
        if(reply != b''):
            print(f'Input: {character}, reply: {reply}, time:{end-start}')

getAllAvaibleInput()
```

Result:

```
Input: c, reply: b'Denied.\r\n', time:0.12317325699405046
Input: d, reply: b'00000\r\n', time:0.20058297899959143
Input: l, reply: b'1: versin.txt\r\n2: passwd\r\n', time:0.23222749800333986
Input: s, reply: b'Denied.\r', time:1.1118301909955335
Input: ?, reply: b'[c]hangefrequenc <number\n[d]ump urrent fequency\r\n[l]ist
    files\n[show file <number>\r\n[?] -help\r\n', time:0.30177813199406955
```

It did not succeed - there were no other character than those provided by question mark command. After some modifications in the script, we also tried complicated inputs. In the meantime we decided to analyze the `.hex` file similarly to our previous tasks.
The most revealing method was `objdump` which gave us some interesting string constants. A fragment of `objdump`'s output:

```
$ objdump -s code.hex | head -30

code.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  49738e69e0afec0c
 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:␣.{.
```

We can see strings we already saw, like the menu and 2 file names. We notice new strings, namely:

- 1.234e
- Enter administration PIN:

- *** ADMIN MODE ***
- Nice, try! ErrCode:
- 87078152d2b9449738e69e0afec0c03b
- Enter administrator's password:

It seems there is an admin mode protected with a PIN, there is also a password protection. Unfortunately, we were unable to enter this admin mode. However, we guessed "1.234e" is stored in "version.txt" file, while "87078152d2b9449738e69e0afec0c03b" is saved as "passwd". The password is in fact a md5 hash of the password. Unfortunately, online hash lookup tools did not show any results.

We note that for all of our interactions with the device we were measuring times of response. For manual interactions we used timestamps, for a Python script we also measured time between writing to the serial connection and the response. The only thing we noticed was a slightly longer response for command `s`. We spent more time only for this command but with no positive outcome. We suspect this behavior was artificial and not a result of operations performed by the code, thus being another red herring.

4 Dead ends

4.1 Code disassembly

We also tried to disassembly the code, but it did not reveal anything new. The only useful information we got, was that which we have already known. We briefly went over assembly code but did not notice anything important. In fact, we were unable to get any information from this. Looking from the positive site - we found a very useful tool to analyze the code, here we provide a link to the online disassembler: https://onlinedisassembler.com/odaweb/InOC413h

4.2 Watching movies

We thought, that maybe some of the riddles were reused, so we once again watched all the movies from the lecturer website and also used other resources. Those were really interesting movies (especially this playlist https://www.youtube.com/playlist?list=PLhixgUqwRTjwNaT40TqIIagv3b4_bfB7M), but unfortunately they do not help us much.

5 Searching online. Google - our best friend

After a lot of unsuccessful attempts, we finally decided to search online for guidance. We looked up phrases seen in the menu but did not find anything connected with our task. As the last resort, we looked up a phrase we saw first: "CameraKey Driver - 2019". The first result was: https://cs.pwr.edu.pl/blaskiewicz/dydaktyka/embedded-security/CameraKey.html. We felt really dumb that only then we got the "macufactures" website while we probably were supposed to go there as the first thing to do. The website reveals a lot of useful informations. The most important for us are:

- To boot in administrator's mode, the control pin must be held LOW during boot.
- In case of lost password, please provide the contents of version.txt (firmware version).

We do not know which pin is "the control pin" but fortunately, we do not have many to try. We also notice that the password and the firmware version are somehow connected.

6 Switch to the admin mode

The information provided at the site said, that: "To boot in administrator's mode, the control pin must be held LOW during boot." But we did not know, at witch pin we should provide low voltage. Here we used an old, proven method - check all pins of the Arduino. The control pin is pin number 12. This time we worked separately so we used different methods of setting that pin in the LOW voltage level. In the end, both methods were working. After searching online we found out that for 5V boards, LOW is under $1.5V$. First method, the simpler one, was to connect Arduino with a $1.2V$ battery.

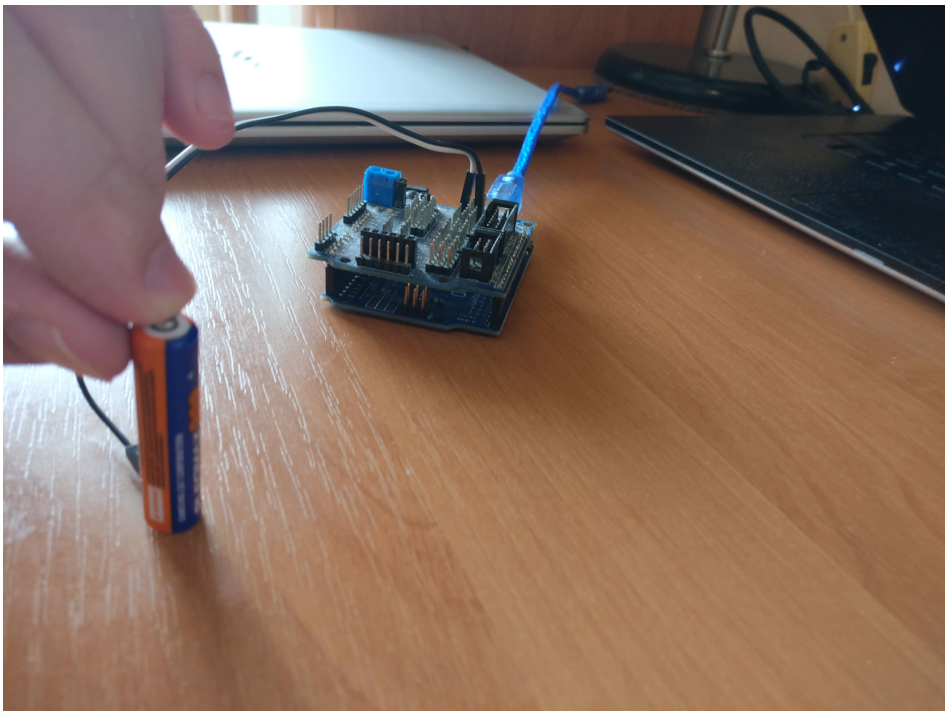

Figure 1: Arduino connected to the battery (this Arduino features an extension board)

The second idea, was to use some resistors and diodes. We used two 220 Ohm resistors and one IR diode, to reduce the voltage. Here is a visualisation of the successful attempt:

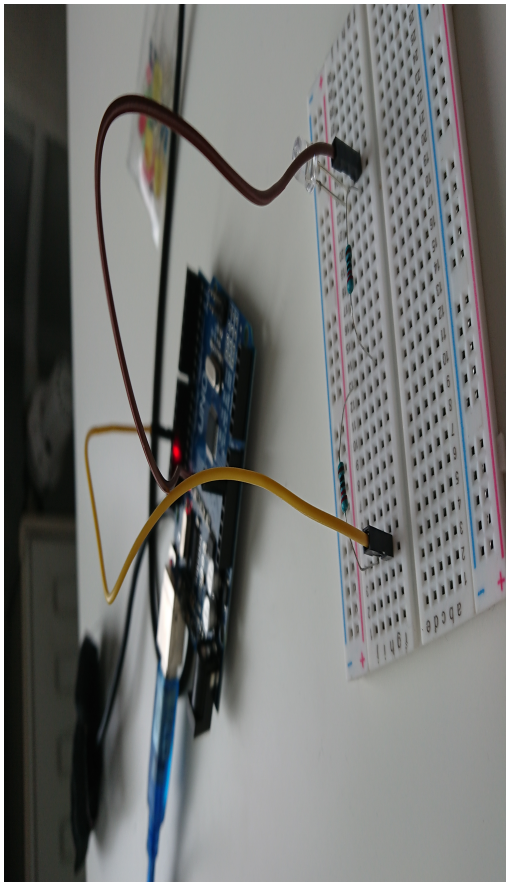

Figure 2: Reducing voltage using resistors and IR diode

After the whole task was done, we realized it suffices to just connect pin 12 and GND, but the methods above also work.
Now, we are able to boot in administrator's mode - unfortunately it was secured by a PIN.

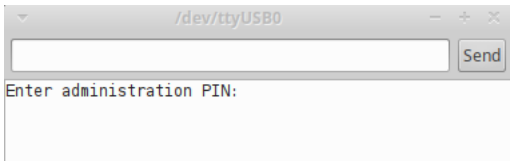

Figure 3: Admin mode panel view

7 Getting PIN - brute-force

We did not think much, we unsuccessfully tried few simple guesses and went on to bruteforcing. We modified the previous script to send a PIN:

```
def getPin():
    results = {}
```

```
    for pin in range(10000):
        start = time.perf_counter()
        arduino.write(pin)
        reply = b''
        # To get the whole response
        while message :=arduino.read(8):
            if type(message) == bytes:
                reply += message
        end = time.perf_counter()
        if(reply != b'Enter administration PIN: \r\n'):
            print(f'Pin: {pin}, reply: {reply}, time:{end-start}')

getPin()
```

But it did not work! Here comes an idea: what if the PIN starts with one or more 0. So we have modified our code a little, and then we got the following response:

```
Input: 0124, reply: b'CameraKey Driver- 2019\r\n*** ADMIN MODE ***\r\n', time
    :0.23347015299805207
```

The PIN was: 0124

Admin mode enables us to look at the files. Our guess about their content was correct. “version.txt” contains “1.234e” and “passwd” contains “87078152d2b9449738e69e0afec0c03b”. We still cannot change the frequency, the program asks for a password:

```
Enter administrator's password:
```

8 Last step - getting admin password

In the last step we have to provide admin’s password in order to be able to change the frequency of the diode. So our goal was to get a password from the hash. The string “87078152d2b9449738e69e0afec0c03b” has 32 characters so it is a result from the md5 hashing function (the result from the function has 128 bits which is equal to 32 hexadecimal digits). The last useful information was the message on the site: "In case of lost password, please provide the contents of version.txt (firmware version)." So we suppose the version number is a part of the password. Now, we are ready to try our brute force attack. Here is a simple python code, which we used to solve our problem:

```
import hashlib
from itertools import permutations
from string import digits, ascii_uppercase, ascii_lowercase

hash = '87078152d2b9449738e69e0afec0c03b'
version = '1.234e'
```

```
def get_hash0(word):
    enc_input = word.encode()
    return   hashlib.md5(enc_input).hexdigest()

def get_hash1(word):
    word = version + word
    enc_input = word.encode()
    return   hashlib.md5(enc_input).hexdigest()

def get_hash2(word):
    word = word + version
    enc_input = word.encode()
    return   hashlib.md5(enc_input).hexdigest()

for i in range(16):
    charset = permutations(ascii_lowercase+ascii_uppercase, i)
    for password in charset:
        pos_hash = get_hash0(''.join(password))
        if(pos_hash == hash):
            print(password)
        pos_hash = get_hash1(''.join(password))
        if(pos_hash == hash):
            print(version + password)
        pos_hash = get_hash2(''.join(password))
        if(pos_hash == hash):
            print(password + version)
```

We started breaking the hash from every possible combination of capital and small letter. Also, we tried to concatenate version number before and after every string. Fortunately, it does not take a lot of time - the password has only 3 characters (without the version part, which in the end was really necessary to compute password).

The password is: **1.234ePWR**.

So now, knowing the admin's password we could finally change the frequency of the diode:

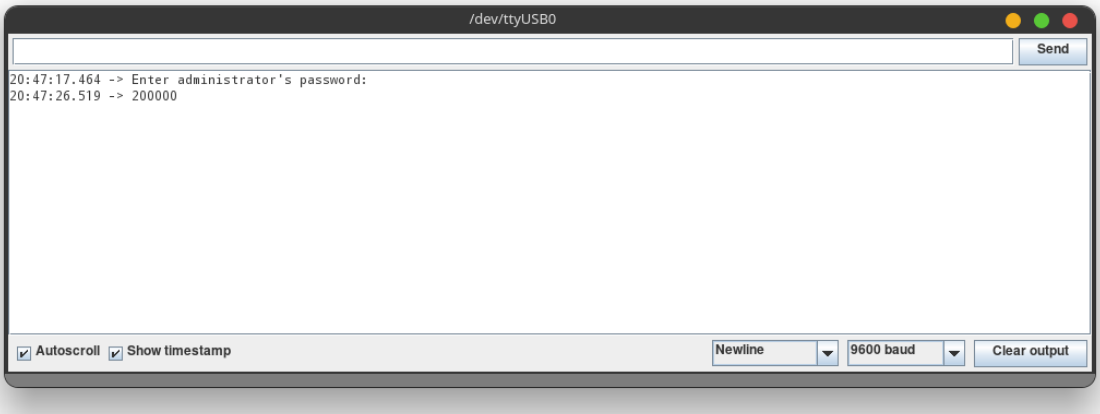

Figure 4: A new frequency of 200000

9 Conclusions

Last exercise was really hard to finish, but the satisfactions which comes after succeeding it is even bigger. Sometimes we did very deep research and the solution was at our fingertips. But we think, that those failures gave us a loot of knowledge which who knows, maybe we could use in similar challenges in the future.