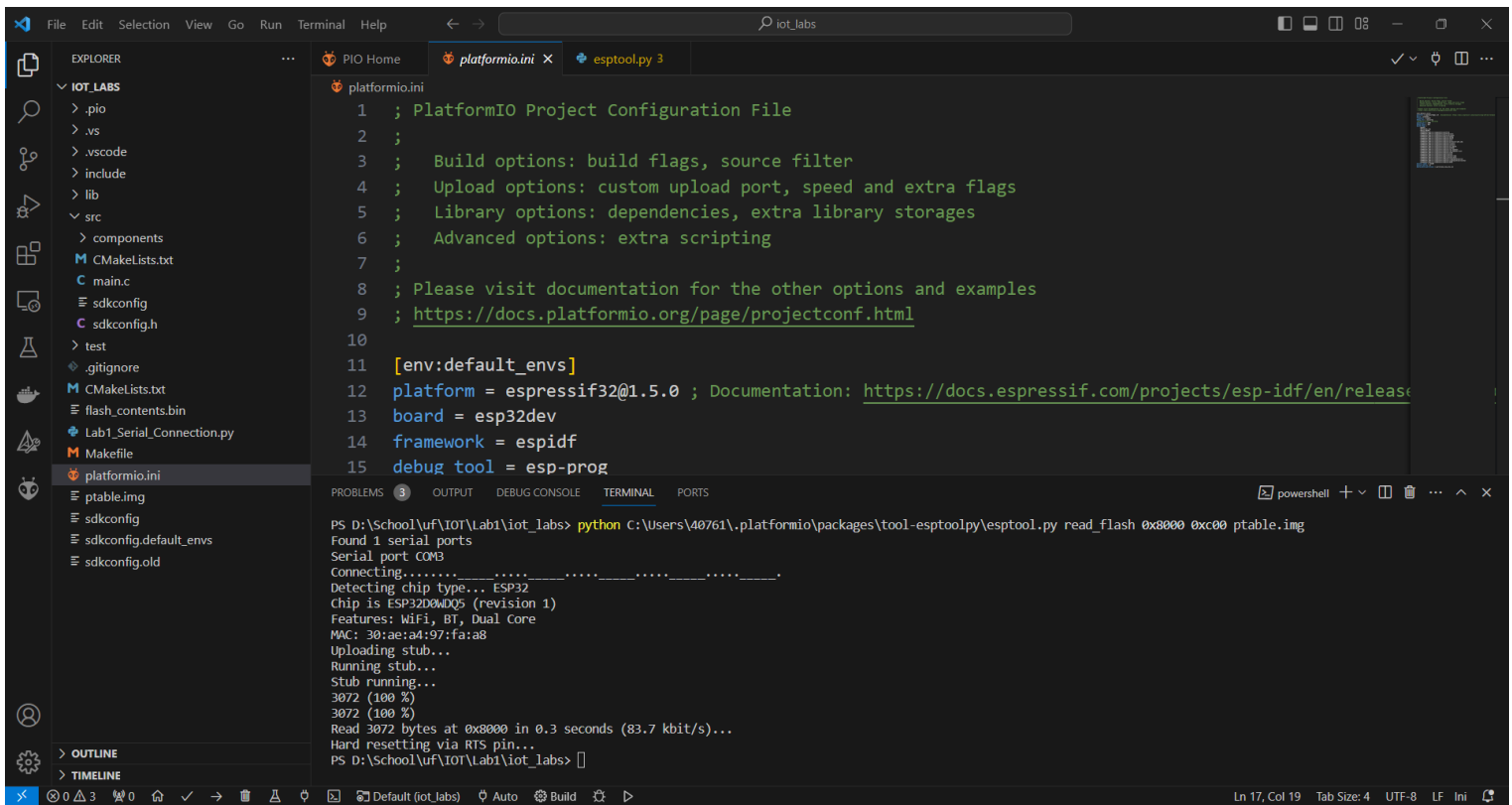


IoT Security and Privacy

Lab2 – ESP32 Flash Hack

Questions

- a. `python C:\Users\40761\.platformio\packages\tool-esptoolpy\esptool.py read_flash 0x8000 0xc00 ptable.img`
where 0x8000 is the start address of the partition table and 0xc00 is the length of the partition table. \$USER is the username of the user that installed platformio. The binary partition table is saved in `ptable.img`. Please provide a screenshot of the command running. [10 point]



The screenshot shows the VS Code editor with the PlatformIO configuration file `platformio.ini` open. The file contains the following content:

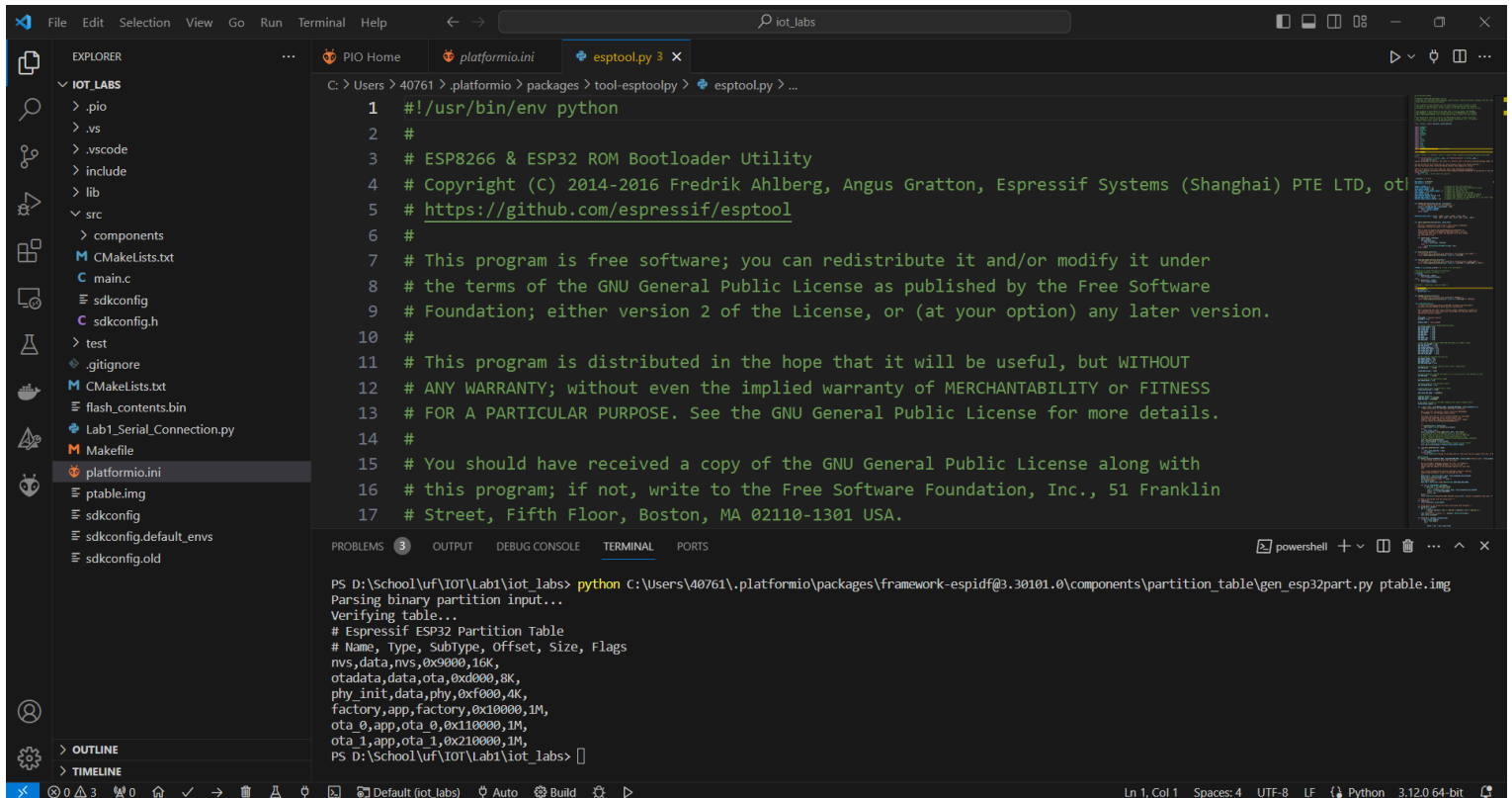
```
1 ; PlatformIO Project Configuration File
2 ;
3 ; Build options: build flags, source filter
4 ; Upload options: custom upload port, speed and extra flags
5 ; Library options: dependencies, extra library storages
6 ; Advanced options: extra scripting
7 ;
8 ; Please visit documentation for the other options and examples
9 ; https://docs.platformio.org/page/projectconf.html
10
11 [env:default_envs]
12 platform = espressif32@1.5.0 ; Documentation: https://docs.espressif.com/projects/esp-idf/en/release
13 board = esp32dev
14 framework = espidf
15 debug tool = esp-prog
```

The terminal window shows the execution of the command `python C:\Users\40761\.platformio\packages\tool-esptoolpy\esptool.py read_flash 0x8000 0xc00 ptable.img`. The output is as follows:

```
PS D:\School\uf\IoT\Lab1\iot_labs> python C:\Users\40761\.platformio\packages\tool-esptoolpy\esptool.py read_flash 0x8000 0xc00 ptable.img
Found 1 serial ports
Serial port COM3
Connecting.....
Detecting chip type... ESP32
Chip is ESP32D0Q05 (revision 1)
Features: WiFi, BT, Dual Core
MAC: 30:ae:a4:97:fa:a8
Uploading stub...
Running stub...
Stub running...
3072 (100 %)
3072 (100 %)
Read 3072 bytes at 0x8000 in 0.3 seconds (83.7 kbit/s)...
Hard resetting via RTS pin...
PS D:\School\uf\IoT\Lab1\iot_labs>
```

- b. following command will print out the partition table of our IoT kit in the CSV (comma-separated values) format. The partition table shows how the flash is partitioned. `python C:\Users\40761\.platformio\packages\framework-esp-idf@3.30101.0\components\partition_table\gen_esp32part.py ptable.img`

Please provide a screenshot of the command running. [10 point]



The screenshot shows a VS Code editor with the file explorer on the left displaying the 'IOT_LABS' project structure. The main editor window shows the 'esptool.py' script, which is a bootloader utility for ESP8266 and ESP32. The terminal at the bottom shows the command being executed: `python C:\Users\40761\.platformio\packages\framework-espidf@3.30101.0\components\partition_table\gen_esp32part.py ptable.img`. The output of the command is a partition table for the ESP32 device, listing partitions such as 'nvs', 'otadata', 'phy_init', 'factory', 'ota_0', and 'ota_1' with their respective types, subtypes, addresses, and sizes.

```
1 #!/usr/bin/env python
2 #
3 # ESP8266 & ESP32 ROM Bootloader Utility
4 # Copyright (C) 2014-2016 Fredrik Ahlberg, Angus Gratton, Espressif Systems (Shanghai) PTE LTD, ot
5 # https://github.com/espressif/esptool
6 #
7 # This program is free software; you can redistribute it and/or modify it under
8 # the terms of the GNU General Public License as published by the Free Software
9 # Foundation; either version 2 of the License, or (at your option) any later version.
10 #
11 # This program is distributed in the hope that it will be useful, but WITHOUT
12 # ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
13 # FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
14 #
15 # You should have received a copy of the GNU General Public License along with
16 # this program; if not, write to the Free Software Foundation, Inc., 51 Franklin
17 # Street, Fifth Floor, Boston, MA 02110-1301 USA.
```

```
PS D:\School\uf\IOT\Lab1\iot_labs> python C:\Users\40761\.platformio\packages\framework-espidf@3.30101.0\components\partition_table\gen_esp32part.py ptable.img
Parsing binary partition input...
Verifying table...
# Espressif ESP32 Partition Table
# Name, Type, SubType, Offset, Size, Flags
nvs,data,nvs,0x9000,16K,
otadata,data,ota,0xd000,8K,
phy_init,data,phy,0xf000,4K,
factory,app,factory,0x10000,1M,
ota_0,app,ota_0,0x110000,1M,
ota_1,app,ota_1,0x210000,1M,
PS D:\School\uf\IOT\Lab1\iot_labs>
```

c. Explain the partition table that is printed out. [10 point]

This partition table is for the flash memory layout of the ESP32 device. Each entry represents a partition in the flash memory.

nvs: type data, subtype nvs. It starts at address 0x9000 and has a size of 16K. This partition is typically used to store Non-Volatile Storage data.

otadata: type data, subtype ota, which starts at address 0xd000 and has a size of 8K. This partition is used to store OTA (Over-The-Air) update data.

phy_init: type data, subtype phy, starts at address 0xf000 and is 4K in size. this partition is used to store ESP32 PHY initialization data.

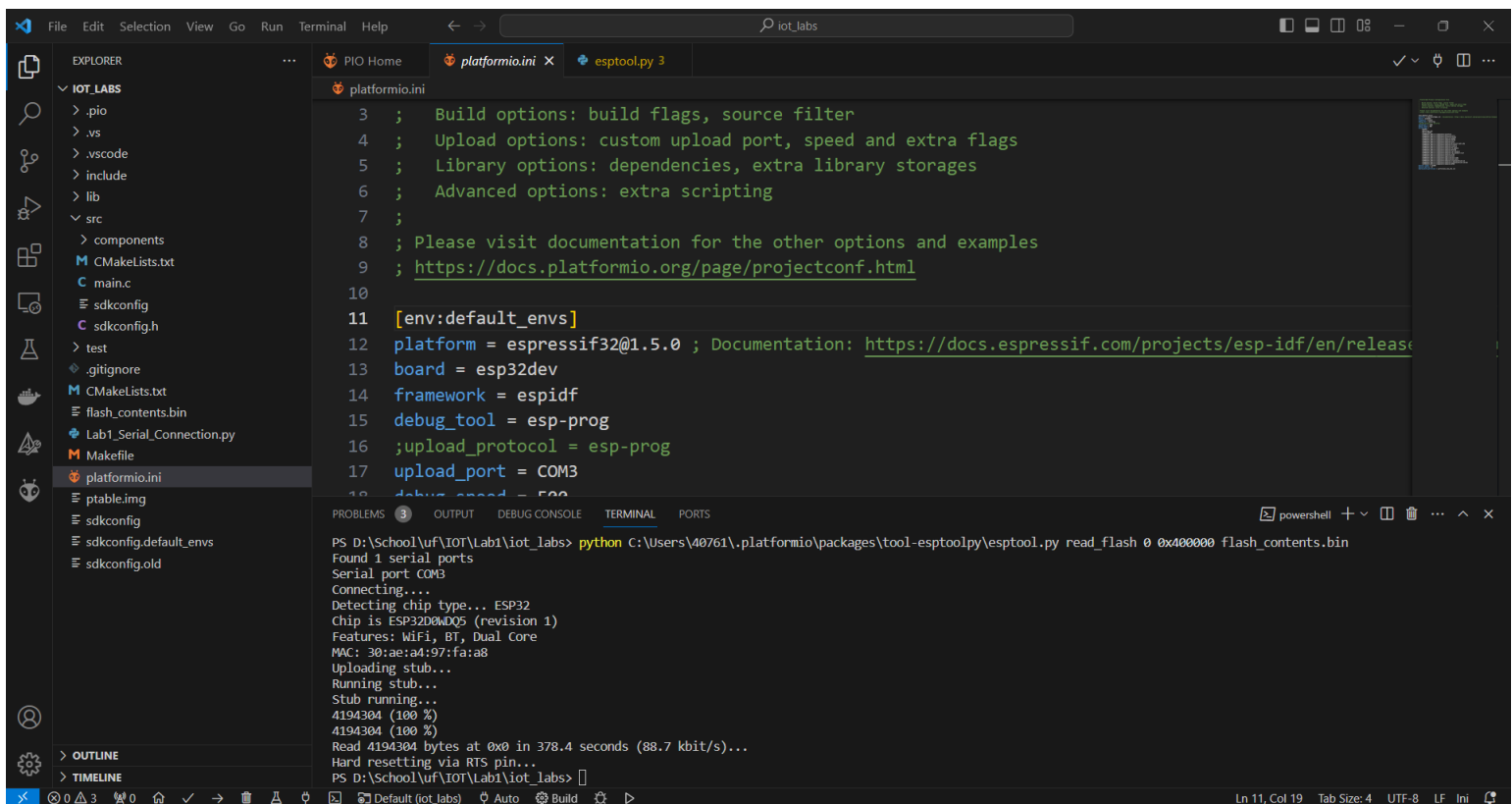
factory: type app, subtype factory, it starts at address 0x10000 and is 1M in size. this is the factory firmware partition, which usually contains the original firmware of the device when it was shipped.

ota_0 and ota_1: These partitions are of type app and subtype ota_0 and ota_1, which start at address 0x110000 and 0x210000 respectively, and have a size of 1 M. These partitions are used to store OTA updated firmware. During the OTA update process, the new firmware is written to the unused OTA partition, and then the device switches to this new firmware at the next boot.

2. The following command retrieves the whole flash content although it is also possible to refer to the partition table and select only the occupied part of the flash.

a. `python C:\Users\40761\.platformio\packages\tool-esptoolpy\esptool.py read_flash 0 0x400000 flash_contents.bin`

where 0 is the starting address and 0x400000 is the length of the flash of the ESP32-WROOM-32 surface-mount module board that our IoT kit uses. The whole flash in the binary format is saved in `flash_contents.bin`. Please provide a screenshot of the command. [10 point]



The screenshot shows the Visual Studio Code interface with the PlatformIO extension. The Explorer pane on the left shows the project structure for 'IOT_LABS', including files like `platformio.ini`, `ptable.img`, `sdkconfig`, `flash_contents.bin`, and `Lab1_Serial_Connection.py`. The main editor displays the `platformio.ini` file with the following content:

```
3 ; Build options: build flags, source filter
4 ; Upload options: custom upload port, speed and extra flags
5 ; Library options: dependencies, extra library storages
6 ; Advanced options: extra scripting
7 ;
8 ; Please visit documentation for the other options and examples
9 ; https://docs.platformio.org/page/projectconf.html
10
11 [env:default_envs]
12 platform = espressif32@1.5.0 ; Documentation: https://docs.espressif.com/projects/esp-idf/en/release
13 board = esp32dev
14 framework = espidf
15 debug_tool = esp-prog
16 ;upload_protocol = esp-prog
17 upload_port = COM3
18 debug_speed = 115200
```

The bottom panel shows the TERMINAL window with the following output:

```
PS D:\School\uf\IOT\Lab1\iot_labs> python C:\Users\40761\.platformio\packages\tool-esptoolpy\esptool.py read_flash 0 0x400000 flash_contents.bin
Found 1 serial ports
Serial port COM3
Connecting...
Detecting chip type... ESP32
Chip is ESP32D0MDQ5 (revision 1)
Features: WiFi, BT, Dual Core
MAC: 30:ae:a4:97:fa:a8
Uploading stub...
Running stub...
Stub running...
4194304 (100 %)
4194304 (100 %)
Read 4194304 bytes at 0x0 in 378.4 seconds (88.7 kbit/s)...
Hard resetting via RTS pin...
PS D:\School\uf\IOT\Lab1\iot_labs>
```

- b. Students can use a hex editor (e.g. wxhexeditor, imhex) to search the WiFi credentials in the flash dump. Please provide a screenshot of found WiFi credentials (e.g. password or key) using a hex editor. [10 point]

the same length as the original. Students must then try to write the changed flash dump to the IoT kit. Please provide a screenshot of the command. Describe the results and discuss potential reasons for what was observed. [20 points]

Hint: Here is a command writing the flash_contents_modified.bin to the IoT kit

python C:\Users\40761\.platformio\packages\tool-esptoolpy\esptool.py write_flash 0 flash_contents_modified.bin

change all the passwords:DummyPass into Li&&Zhang using ImHex

The screenshot shows the ImHex hex editor interface. The main window displays a hex dump of a file named 'flash_contents_modified.bin'. The hex dump shows addresses from 00000000 to 000000FF. The ASCII column shows the corresponding text values. The 'Data Inspector' panel on the right shows a list of data types and their values. The 'Pattern Data' panel at the bottom shows a table of patterns.

Name	Color	Start	End	Size	Type	Value

The screenshot shows a VS Code editor with a file explorer on the left, a code editor in the center, and a terminal at the bottom. The file explorer shows a project named 'IOT_LABS' with various files and folders. The code editor displays a Python script named 'esptool.py' with line numbers 403 to 416. The script contains comments and code for setting RTS, sleeping, and setting DTR. The terminal shows the command 'python C:\Users\40761\.platformio\packages\tool-esptoolpy\esptool.py write_flash 0 flash_contents_modified.bin' and its output, which includes finding a serial port (COM3), connecting, identifying the chip as ESP32D0WDQ5 (revision 1), uploading a stub, configuring flash size, writing the data, and verifying the hash.

```
403 self._setRTS(True) # EN=LOW, chip in reset
404 time.sleep(0.1)
405 if esp32r0_delay:
406     # Some chips are more likely to trigger the esp32r0
407     # watchdog reset silicon bug if they're held with EN=LOW
408     # for a longer period
409     time.sleep(1.2)
410 self._setDTR(True) # IO0=LOW
411 self._setRTS(False) # EN=HIGH, chip out of reset
412 if esp32r0_delay:
413     # Sleep longer after reset.
414     # This workaround only works on revision 0 ESP32 chips,
415     # it exploits a silicon bug spurious watchdog reset.
416     time.sleep(0.4) # allow watchdog reset to occur
```

```
PS D:\School\uf\IOT\Lab1\iot_labs> python C:\Users\40761\.platformio\packages\tool-esptoolpy\esptool.py write_flash 0 flash_contents_modified.bin
Found 1 serial ports
Serial port COM3
Connecting....
Chip is ESP32D0WDQ5 (revision 1)
Features: WiFi, BT, Dual Core
MAC: 30:ae:a4:97:fa:a8
Uploading stub...
Running stub...
Stub running...
Configuring flash size...
Auto-detected Flash size: 4MB
Compressed 4194304 bytes to 1238935...
Wrote 4194304 bytes (1238935 compressed) at 0x00000000 in 110.4 seconds (effective 304.0 kbit/s)...
Hash of data verified.

Leaving...
Hard resetting via RTS pin...
PS D:\School\uf\IOT\Lab1\iot_labs>
```

The system found a serial port (COM3) and successfully connected to the ESP32 chip. This indicates that the device is properly connected to the computer and that serial communication is working.

The system recognizes the chip as an ESP32D0WDQ5 (revision 1) with WiFi, Bluetooth, and Dual Core. This is a normal device recognition process.

The system uploads a small program (stub) to the ESP32 device to assist in the write process. This is part of the normal write process.

The system automatically detects a flash size of 4MB, which is one of the standard configurations for ESP32 devices.

The modified flash file is compressed and written to the device's flash memory. A total of 4,194,304 bytes were written in approximately 110.4 seconds. This indicates that the write operation completed successfully.

After the write was completed, the system performed data verification to ensure that the written data matched the original file. "Hash of data verified" indicates that the data was verified successfully and no errors were found.

Finally, a hard reset was performed via the RTS pin, which is the normal procedure for rebooting the device. Then, we observe that the board will lose its original function and will no longer carry out its work and we won't be able to test with Lab1's scripts.