

Teacher Manual

Hackable ESP Device

IoT Cyber Security Course

5 Dollar ESP Controller

ESPinoza - 24/01/2022

Version 0.1



Hogeschool van Amsterdam

Team:

Thijs Takken

Luke de Munk

Christina Kostine

Twenne Elffers

Table of contents

Version history	4
1. Introduction	5
2. Setup	6
3. Vulnerabilities	8
3.1 Weak, Guessable or Hardcoded Passwords	8
3.1.1 Serial leak - Easy	8
3.1.2 Web server users leak - Medium	9
3.2 Insecure Network Services	10
3.2.1 Cross-site request forgery - Medium	10
3.2.2 BotLED - Medium	11
3.3 Insecure Ecosystem Interfaces	12
3.3.1 Encryption leak - Medium	12
3.3.2 Buffer overflow - Hard	13
3.4 Lack of Secure Update Mechanism	14
3.4.1 Exploitable configuration upload & download - Medium	14
3.5 Use of Insecure or Outdated Components	16
3.6 Insufficient Privacy Protection	16
3.7 Insecure Data Transfer and Storage	17
3.8 Lack of Device Management	17
3.9 Insecure Default Settings	17
3.10 Lack of Physical Hardening	17
Appendix A: README	18
Appendix B: Graphical steps	22

Version history

Version	Changes	Date	Author
0.1	Initial version	10-9-2021	Team 1
1.0	Final Draft	31/01/2022	Team 1

1. Introduction

Eurofins has given the project to create a small hackable esp device which can emulate real world IoT vulnerabilities. The first finished version of this device has been made. Because Eurofins wishes to use it as a teaching aid in IoT security training, this teachers guide has been made. It Contains information on how to prepare the device for the trainees and a detailed explanation on how to exploit the vulnerabilities on it.

2. Setup

For the most up to date version of the setup refer to the readme.md file

All platform:

1. Install the CH340-Drivers for the esp8266
(<https://github.com/HobbyComponents/CH340-Drivers>)
2. Clone the repository

Possible configuration changes:

Default user and passwd

It is possible to change the default username and password for the users on the webpage.

The file for this is /hackableEspDevice/data/config.conf.

Here the username and password is stored and can be changed to increase or decrease the challenge. the format is "<Username>:<Password>:<authorization level (1|2)>".

To create the new encrypted string you need to encrypt it using the `gpg --encrypt` command in the serial interface of the device.

Level 1 is a normal user, level 2 is an admin user.

config.h

/hackableEspDevice/config.h also contains a few interesting possible settings.

The settings that can be changed are the:

- The default **hostname** for the device (DEFAULT_HOSTNAME)
- **Access Point name** for the configuration phase (WIFI_CONF_AP_NAME)
- The **root password** for the Serial interface (ROOT_PASSWORD)

Installing on the esp8266:

There are multiple ways to upload the program files to the board. The two ways listed here are using Arduino IDE and Platformio on visual studio code

Arduino IDE

1. Install the Arduino IDE (<https://www.arduino.cc/en/software>)
2. Add the esp8266 libraries to Arduino IDE
(<https://www.nonscio.com/blog/installing-esp8266-libraries-to-the-arduino-ide>)
3. Follow this tutorial about the SPIFFS.
(<https://randomnerdtutorials.com/install-esp8266-filesystem-uploader-arduino-ide>)
4. Navigate to the "hackableEspDevice" folder.
5. Open "hackableEspDevice.ino".
6. Select the LOLIN(WeMos) D1 R1 (or WeMos D1 R1).
7. Upload the files in the "data" folder (see the tutorial).
8. Upload the program to the device.

Visual Studio Code + Platformio

1. Install the Platformio plugin.
(<https://platformio.org/install/ide?install=vscode>)
2. Prepare files for platformio
 - Run the "toPlatformio.ps1" script and select the copy or symbolic option
 - Symbolic changes the original ideal for editing the files
 - Copy simply copies the files to a new location for platformio files
 - Run the "toPlatformio.ps1" script and select fix
 - Or prepare the files manually see manual prep platformio (See the readme.md)
3. Open visual studio code in the HackableEspDevicePlatformio directory
4. In visual studio code open the project in the platformio addon. (Platformio > Projects > open HackableEspDevicePlatformio)
5. Upload the program (project tasks > General> Upload)
6. Upload the filesystem Image (Project tasks > Platform > Upload filesystem Image)

First start

1. Start up the device
2. Connect to the "Configure Smartlight Wifi" Access Point
3. Go to the IP address listed in the serial monitor most of the time this is
`http://192.168.4.1/`
4. Follow the steps on the website to configure a wifi connection
5. The device should now restart, connect to the selected wifi network and be ready for use

3. Vulnerabilities

In this chapter all the implemented vulnerabilities are listed, accompanied with a description and how to reproduce them. The order is based on the OWASP IoT top ten. This way there is a standardized way of ordering and structuring the vulnerabilities.

3.1 Weak, Guessable or Hardcoded Passwords

Use of easily brute forced, publicly available, or unchangeable credentials, including backdoors in firmware or client software that grants unauthorized access to deployed systems.

3.1.1 Serial leak - Easy

Preparation

To be able to exploit the serial vulnerabilities, a serial connection is needed. The best way to monitor the device is by connecting it through USB and opening the COM port in the Arduino IDE. In the list below, the steps are given

1. Connect the USB cable to the computer.
2. Open COM port with Arduino IDE
3. Set baud rate at 115200
4. Push the reset button of the device

Description

The serial console is often used by testing software on microchips. The debug messages can contain sensitive data. This device has the capability to interpret several commands as well. The serial is in this case not turned off before the deployment, which can lead to serious security risks. Especially when commands can be executed, the data on the device is not secure.

Steps to reproduce

To get the **root password** out of the serial data (in this case the startup debug data), do the following:

1. Send the "debug -on" command
2. Send the "reboot" command
3. Convert the bytes that are printed at startup
4. Discover the **root password**

3.1.2 Web server users leak - Medium

Steps to reproduce

This vulnerability can only be exploited when the user is a superuser.

1. Send the “su [password]” command
2. Send the “users” command
3. Get the **usernames and passwords** of non admin users

The ‘Exploitable configuration upload & download’ vulnerability is also related to this category.

3.2 Insecure Network Services

Unneeded or insecure network services running on the device itself, especially those exposed to the internet, that compromise the confidentiality, integrity/authenticity, or availability of information or allow unauthorized remote control.

3.2.1 Cross-site request forgery - Medium

Description

The application which turns the LED on the ESP On and Off is vulnerable to Cross-site request forgery (CSRF). It is possible to make calls to the device 's IP and control it. This way, a hacker could execute an attack, where, if the victim opens a webpage created by the hacker. It will execute a command that controls the ESP.

For more information on CSRF see <https://owasp.org/www-community/attacks/csrf>.

Steps to reproduce

In order to leverage the CSRF vulnerability, a malicious html page has to be made. The webpage should call the ESP on the correct IP and the attacker can choose whether to turn the LED on or off. When a victim opens that HTML page and has an LED controller in their network the LED will turn on or off depending on the settings.

Code snippet of a working example

```
<!DOCTYPE html>
<html>
<body>

<h1>This is an malicious page that will execute an CSRF attack</h1>

<p>Through the request of an image, it will make the call to the LED
controller and turn on the LED</p>

<!-- Only a valid IP is needed, and a "state", 0 = LED Off and 1 =
LED On

<!-- This will ask to receive that data, thereby making the call to
the LED controller-->


</body>
</html>
```

3.2.2 BotLED - Medium

Description

The LED controller has a callable function with which you can control the LED. In this challenge it is the goal to control a whole group of ESPs on a network and manipulate their LED state.

Steps to reproduce

For this assignment the following files are needed:

1. **BotLED.py**

Complete and working, for teachers only. This contains finished code which can be run to control the LEDs on the ESPs. Only make sure to set the right subnet inside of the script before running.

2. **BotLED_skel.py**

This is the stripped file, the control function is missing. The trainees will have to program this function. And they should be able to when they have found the CSRF vulnerability already or basically that they can control the powerstate with calling:

IP/set_power?state=0

or

IP/set_power?state=1

3.3 Insecure Ecosystem Interfaces

Insecure web, backend API, cloud, or mobile interfaces in the ecosystem outside of the device that allows compromise of the device or its related components. Common issues include a lack of authentication/authorization, lacking or weak encryption, and a lack of input and output filtering.

3.3.1 Encryption leak - Medium

Description

CBC mode is an AES block cipher mode that is XOR-ing the first plaintext block with an initialization vector before encrypting it. The decryption works in the same way with ciphered text.

Even if Cbc mode is more secure than EBC one as it hides the patterns of plaintext. This vulnerability is about finding the key and decrypting the file with it.

Steps to reproduce

This vulnerability can only be exploited when the user is a superuser.

1. Send the “su [password]” command
2. Send the “keys” command
3. Get the **AES keys**

3.3.2 Buffer overflow - Hard

Description

In the past, buffer overflows were common. Today, most of the software is written in a way that a buffer overflow is not possible anymore. In microchips such as the ESP device, buffer overflows can still exist. A buffer overflow (or buffer overrun) occurs when the volume of data exceeds the storage capacity of the memory buffer. As a result, the program attempting to write the data to the buffer overwrites adjacent memory locations.

For example, a buffer for log-in credentials may be designed to expect username and password inputs of 8 bytes, so if a transaction involves an input of 10 bytes (that is, 2 bytes more than expected), the program may write the excess data past the buffer boundary.

If the transaction overwrites executable code, it can cause the program to behave unpredictably and generate incorrect results, memory access errors, or crashes. If attackers know the memory layout of a program, they can intentionally feed input that the buffer cannot store, and overwrite areas that hold executable code, replacing it with their own code. For example, an attacker can overwrite the program-pointer (an object that points to another area in memory) and point it to an exploit payload, to gain control over the program.

Steps to reproduce

To **become a superuser without password**, the login function needs to be executed and after that, the program needs to stop. This is one of the hardest vulnerabilities. It can be exploited by doing this:

1. Send the "ls" command and look at the programs
2. Send the "vi ./testprogram.c" command to see the buffer size at line 17 (10, but in reality 16)
3. Send the "objdump -d ./testprogram" command to see the address of the login function (00010488)
4. Calculate needed input string by testing different parameters while executing the test program (for example: ./testprogram Luke).
5. Notice that this device is little endian (can also be seen on the first line of the disassembled program)
6. String could be: "AAAAAAAAAAAAAAAAAA\x88\x04\x01\x00"
7. Send the "./testprogram AAAAAAAAAAAAAAAAAAA\x88\x04\x01\x00" command to **become a superuser**.

3.4 Lack of Secure Update Mechanism

Lack of ability to securely update the device. This includes lack of firmware validation on devices, lack of secure delivery (un-encrypted in transit), lack of anti-rollback mechanisms, and lack of notifications of security changes due to updates.

3.4.1 Exploitable configuration upload & download - Medium

Description

There is an upload and download function which would allow a user to backup the configuration. This function has not been properly protected and allows a normal user to obtain the backup file.

Variables

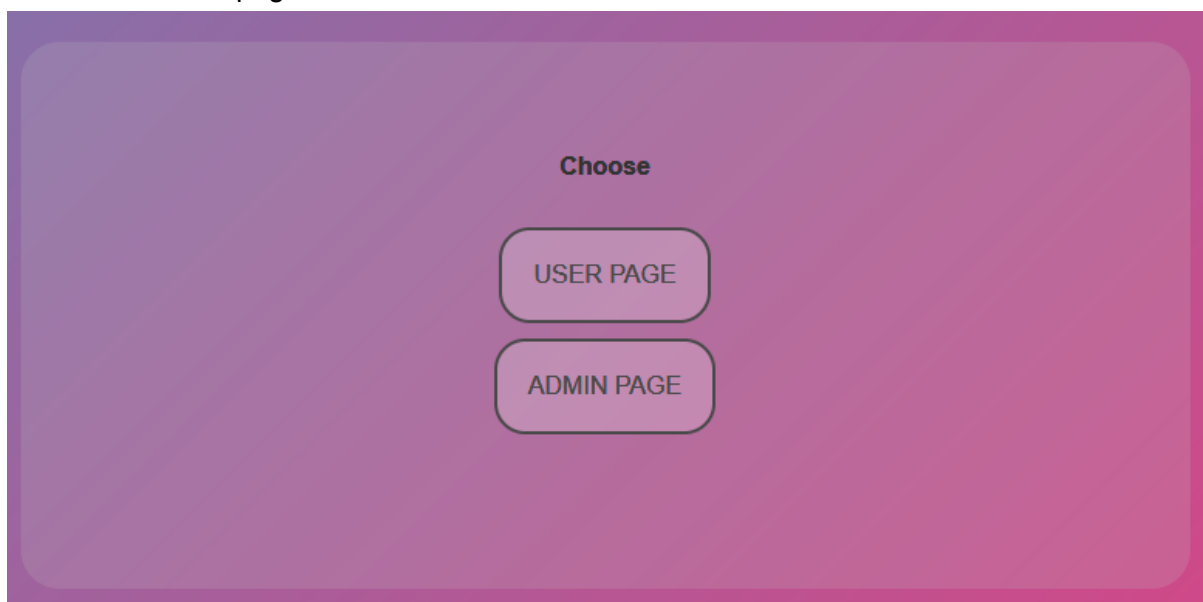
Admin and user password and username are set in the `hackableEspDevice/data/config.conf` file

File is noted as follows “<Username>:<Password>:<authorization level (1|2)>” but encrypted. To decrypt use the `gpg` command in the serial interface on the device.

Level 1 is a normal user, level 2 is an admin user

Exploitation

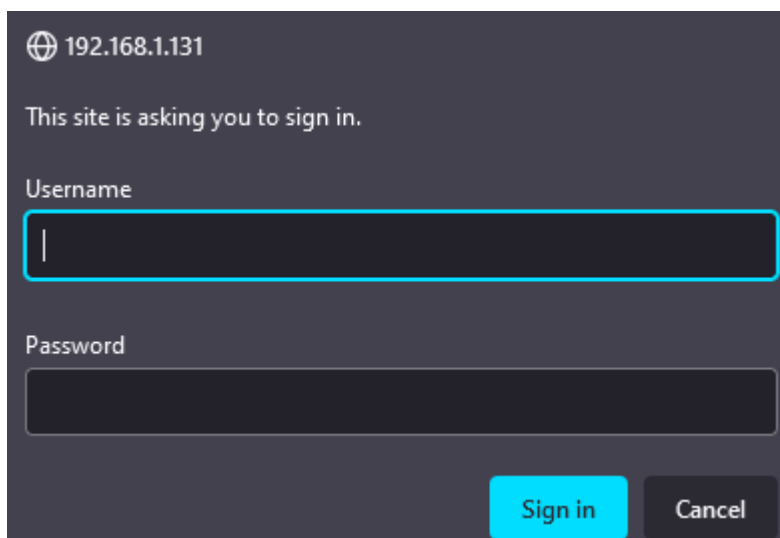
1. Go to the homepage of the device
2. Choose the “user page” at the bottom



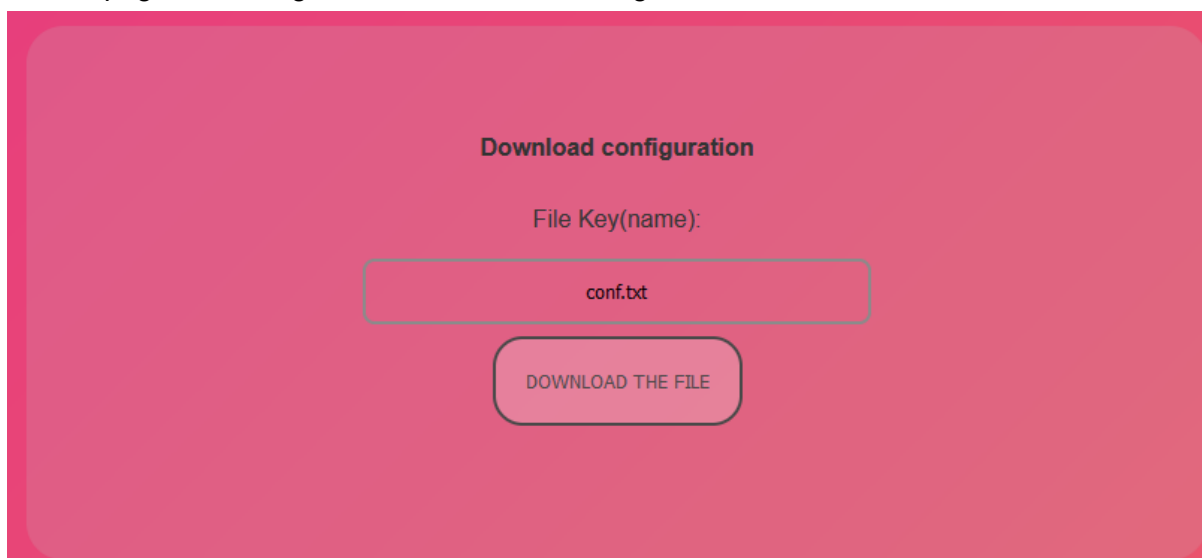
3. This shows a login form. Where the credentials for the user or admin need to be filled in. These are stored in the second field of the `config.conf` file.
The default credentials are:
Username: **user**

Password: Password1

These should be insecure that is the first vulnerability



4. This takes you to a user page with the option “Download files”
5. On this page the user gets to download the config.conf file



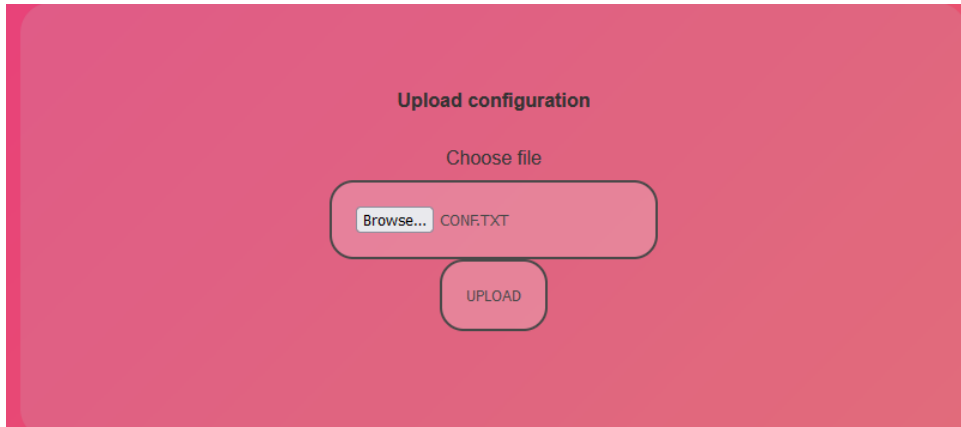
6. This file contains the credentials of all other users including the admin account. Unfortunately these credentials are encrypted.
7. To decrypt the credentials the tool in the serial interface of the d1 mini needs to be used.
8. First the user must get the key with the command “privatekeys”
9. This command requires the user to be a super user the credentials for this can be found in the “serial leak”
10. The key from the privatekeys can then be used in combination with the “gpg –decrypt <key> <text from file>” command
The notation of output should be “<Username>:<Password>:<authorization level (1|2)>”

11. The user can then use this to obtain the admin password and login to the admin page
12. The admin page in addition to the download option also contains the option for the user to upload a file.

The user can then choose to upload their own files.

This can be any file of their choosing. They can choose to upload a new config.conf but also a new index.html page with malicious code.

The files that are uploaded are **persistent** and require a reinstall to reset them..



No further vulnerabilities present in this category.

3.5 Use of Insecure or Outdated Components

Use of deprecated or insecure software components/libraries that could allow the device to be compromised. This includes insecure customization of operating system platforms, and the use of third-party software or hardware components from a compromised supply chain

We bet that there are insecure components that can be hardware hacked, but this was not part of the current project.

3.6 Insufficient Privacy Protection

User's personal information stored on the device or in the ecosystem that is used insecurely, improperly, or without permission.

The following vulnerabilities have implications on the privacy category.

1. Serial leak
2. Web server users leak
3. Exploitable configuration upload & download

3.7 Insecure Data Transfer and Storage

Lack of encryption or access control of sensitive data anywhere within the ecosystem, including at rest, in transit, or during processing.

The 'Exploitable configuration upload & download' vulnerability is also related to this category.

3.8 Lack of Device Management

Lack of security support on devices deployed in production, including asset management, update management, secure decommissioning, systems monitoring, and response capabilities.

There is no remote device management present, therefore no vulnerabilities in this category are present.

3.9 Insecure Default Settings

Devices or systems shipped with insecure default settings or lack the ability to make the system more secure by restricting operators from modifying configurations.

The following vulnerabilities are using some default and hard coded variables, therefore they are shipped with the same hard coded variables every time.

1. Serial leak
2. Web server users leak
3. Encryption leak
4. Exploitable configuration upload & download

3.10 Lack of Physical Hardening

Lack of physical hardening measures, allowing potential attackers to gain sensitive information that can help in a future remote attack or take local control of the device.

This category is broken by design, every trainee receives a device and physical hardening is therefore always broken by design.

Appendix A: README

Hackable ESP8266 device

Firmware for ESP8266 based device (D1 Mini board) with designed vulnerabilities to practice ethical hacking. The software is tested on the following boards:

- [D1 Mini](#)

Getting Started

These instructions will get you a copy of the project up and running on your D1 Mini (or other ESP8266 based boards) for development or hacking purposes.

Prerequisites

The software is written, compiled and uploaded using the [Arduino IDE](#). Platform.io and Visual Studio Code can be used as well. Use the script to convert the project to a Platform.io.

Dependencies

- ESP for Arduino IDE
- ESP Async WebServer V1.2.3
- Wifimanager V0.16.0

Installing

General Install

1. Install the [driver](#) for the esp8266.
2. Clone the repository.

There are multiple ways to upload the program files to the board. The two ways listed here are using Arduino IDE and Platformio on Visual Studio Code.

Option 1: Arduino IDE

1. Install the [Arduino IDE](#)
2. [Add the esp8266 libraries to Arduino IDE.](#)

3. Follow [this](#) tutorial about the SPIFFS.
4. Navigate to the hackableEspDevice folder.
5. Open hackableEspDevice.ino.
6. Upload the files in the data folder (see the tutorial).
7. Select the LOLIN(WeMos) D1 R1 (or WeMos D1 R1).
8. Upload the program to the device.
9. Connect to the Configure Smartlight Wifi AP to configure the wifi.

ESP8266 Sketch Data Upload

1. The Arduino IDE won't have the option 'ESP8266 Sketch Data Upload'.
2. You can download it from this [link](#).
3. The file should be unpacked at
 <home_dir>/Arduino-<version>/tools/ESP8266FS/tools/.
 - If the directory tools does not exist you should create it. You have to create a new file named "tools" if it doesn't exist already inside of [Arduino file](#).

Option 2: Visual Studio Code + Platformio

1. Install the [Platformio](#) plugin.
2. Prepare files for platformio.
 - Run the toPlatformio.ps1 script and select the copy or symbolic option.
 - Symbolic changes the original ideal for editing the files.
 - Copy simply copies the files to a new location for platformio files.
 - Run the toPlatformio.ps1 script and select fix.
 - Or prepare the files manually see manual prep platformio.
3. Open visual studio code in the HackableEspDevicePlatformio directory.
4. In visual studio code open the project in the platformio addon. (Platformio > Projects > open HackableEspDevicePlatformio).
5. Upload the program (project tasks > General> Upload).
6. Upload the filesystem Image (Project tasks > Platform > Upload filesystem Image).
7. Done. The device should now be ready for use.

Manual Platformio Prep

1. Create the correct hierarchy.

|HackableEspDevicePlatformio\

|--- platformio.ini

|--- src\

|--- src\main.cpp\

|--- data\

1. The src dir needs to contain all the files from the hackableEspDevice directory except the data directory.
2. Rename the hackableEspDevice.ino to main.cpp.
3. In main.cpp add a reference to all functions in main e.g.

void setup();

void setup();

void initializeHostname();

void connectWifi();

void initializeServer();

void loop();

String getContentType(String filename);

void handleFileRequest(String path, uint8_t permissionLevel);

void handleFileUpload();

void handleFileDownload();

4. Move the platformio.ini file from the root dir to the hackableEspDevicePlatformio dir.
5. Copy all files from hackableEspDevice\data to hackableEspDevicePlatformio\data.

Running

Wifi Manager First Boot

1. Start up the device.
2. Connect to the Configure Smartlight Wifi via a mobile device.
3. Go to the IP address listed in the serial monitor. Most of the time this is <http://192.168.4.1>.
4. Follow the steps on the website to configure a wifi connection.
5. The device should now restart, connect to the selected wifi network and be ready for use.

Customization of Hackable ESP (Contains spoilers) (Look in raw version of readme.md)

Hardware

- 1x D1 Mini Board
- 1x USB to USB-mini cable
- 1x ESP8266 casing

Questions or Feedback?

There is technical documentation available if you want to contribute to this project. There is a user manual as well, contact us for information. You can open an issue if you have questions or feedback for this repository.

Authors

- **Luke de Munk** - *Head author* - [LinkedIn](#)
- **Thijs Takken** - *Head author* - [LinkedIn](#)
- **Christina Kostine** - *Head author* - [LinkedIn](#)
- **Twenne Elffers** - *Head author* - [LinkedIn](#)

Appendix B: Graphical steps

