Hackable ESP device

Generated by Doxygen 1.9.3

Chapter 1

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`

1.1 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.

1.1.1 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.

1.1.2 Dependencies

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

1.1.3 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.

1.1.3.1 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. Upload the program to the device.
8. Connect to the Configure Smartlight Wifi AP to configure the wifi.

   ESP8266 Sketch Data Upload

   (a) The Arduino IDE won't have the option 'ESP8266 Sketch Data Upload'.

   (b) You can download it from this link.

   (c) The file should be unpacked at <home_dir>/Arduino-<version>/tools/ESP8266↩ FS/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.

1.1.3.2 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. (Platoformio > 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.

1.1.4 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`.

1.1.5 Running

1.1.5.1 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.

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

1.2 Hardware

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

1.3 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.

1.4 Authors

- **Luke de Munk** - *Head author* - LinkedIn
- **Thijs Takken** - *Head author* - LinkedIn
- **Christina Kostine** - *Head author* - LinkedIn
- **Twenne Elffers** - *Head author* - LinkedIn

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Class Documentation

4.1 BufferOverflow Class Reference

`#include <BufferOverflow.h>`

Public Member Functions

- BufferOverflow ()
  Constructor.
- void ls ()
  Prints the fake list of files.
- void vi ()
  Prints the vulnerable testprogram.
- void objectDump ()
  Prints the disassembled code of the vulnerable testprogram.
- bool runCProgram (String arg)
  Simulates the vulnerable testprogram.

4.1.1 Detailed Description

Definition at line 20 of file BufferOverflow.h.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 BufferOverflow()

`BufferOverflow::BufferOverflow ( )`

Constructor.

Definition at line 15 of file BufferOverflow.cpp.

4.1.3 Member Function Documentation

4.1.3.1 ls()

```
void BufferOverflow::ls ( )
```

Prints the fake list of files.

Definition at line 24 of file BufferOverflow.cpp.

4.1.3.2 objectDump()

```
void BufferOverflow::objectDump ( )
```

Prints the disassembled code of the vulnerable testprogram.

Definition at line 70 of file BufferOverflow.cpp.

4.1.3.3 runCProgram()

```
bool BufferOverflow::runCProgram (
            String arg )
```

Simulates the vulnerable testprogram.

Parameters

arg	Given argument

Returns

bool True if the buffer overflow attack is done correctly

Definition at line 127 of file BufferOverflow.cpp.

4.1.3.4 vi()

```
void BufferOverflow::vi ( )
```

Prints the vulnerable testprogram.

Definition at line 34 of file BufferOverflow.cpp.

The documentation for this class was generated from the following files:

- hackableEspDevice/BufferOverflow.h
- hackableEspDevice/BufferOverflow.cpp

4.2 name Class Reference

`#include <classTemplate.h>`

Public Member Functions

- name ()
  Constructor.
- void test ()
  brief

4.2.1 Detailed Description

Definition at line 13 of file classTemplate.h.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 name()

`name::name ( )`

Constructor.

Parameters

var	desc

Returns

var desc

Definition at line 18 of file classTemplate.cpp.

4.2.3 Member Function Documentation

4.2.3.1 test()

```
void name::test ( )
```

brief

Parameters

var	desc

Returns

var desc

Definition at line 29 of file classTemplate.cpp.

The documentation for this class was generated from the following files:

- classTemplate.h
- classTemplate.cpp

4.3 SerialCommandExecuter Class Reference

`#include <SerialCommandExecuter.h>`

Public Member Functions

- SerialCommandExecuter ()
  Constructor.
- void executeCommand ()
  Reads the commands and sends them to the parser.
- void setUsers (String *users, uint8_t numUsers)
  Sets the users for user list.

4.3.1 Detailed Description

Definition at line 46 of file SerialCommandExecuter.h.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 SerialCommandExecuter()

`SerialCommandExecuter::SerialCommandExecuter ( )`

Constructor.

Definition at line 16 of file SerialCommandExecuter.cpp.

4.3.3 Member Function Documentation

4.3.3.1 executeCommand()

```
void SerialCommandExecuter::executeCommand ( )
```

Reads the commands and sends them to the parser.

Definition at line 40 of file SerialCommandExecuter.cpp.

4.3.3.2 setUsers()

```
void SerialCommandExecuter::setUsers (
            String * users,
            uint8_t numUsers )
```

Sets the users for user list.

Parameters

users	Array of the users
numUsers	Number of users

Definition at line 27 of file SerialCommandExecuter.cpp.

The documentation for this class was generated from the following files:

- hackableEspDevice/SerialCommandExecuter.h
- hackableEspDevice/SerialCommandExecuter.cpp

4.4 UserHandler Class Reference

```
#include <userHandler.h>
```

Public Member Functions

- UserHandler (ESP8266WebServer *server)
  Constructor.
- void updateUsers ()
  Updates the users from the config file in workmemory.
- String * getUsers ()
  Gets users.
- uint8_t getNumberOfUsers ()
  Gets number users.
- bool checkPermission (uint8_t permissionLevel, ESP8266WebServer *server)
  Checks if user has permission.

4.4.1 Detailed Description

Definition at line 16 of file userHandler.h.

4.4.2 Constructor & Destructor Documentation

4.4.2.1 UserHandler()

```
UserHandler::UserHandler (
            ESP8266WebServer * server )
```

Constructor.

Parameters

server	Webserver object

Definition at line 17 of file userHandler.cpp.

4.4.3 Member Function Documentation

4.4.3.1 checkPermission()

```
bool UserHandler::checkPermission (
            uint8_t permissionLevel,
            ESP8266WebServer * server )
```

Checks if user has permission.

Parameters

permissionLevel	0 = not logged in, 1 = user, 2 = admin
server	Webserver object

Returns

bool If user has permission

Definition at line 80 of file userHandler.cpp.

4.4.3.2 getNumberOfUsers()

```
uint8_t UserHandler::getNumberOfUsers ( )
```

Gets number users.

Definition at line 68 of file userHandler.cpp.

4.4.3.3 getUsers()

```
String * UserHandler::getUsers ( )
```

Gets users.

Definition at line 59 of file userHandler.cpp.

4.4.3.4 updateUsers()

```
void UserHandler::updateUsers ( )
```

Updates the users from the config file in workmemory.

Definition at line 26 of file userHandler.cpp.

The documentation for this class was generated from the following files:

- hackableEspDevice/userHandler.h
- hackableEspDevice/userHandler.cpp

Chapter 5

File Documentation

5.1 classTemplate.cpp File Reference

5.2 classTemplate.cpp

Go to the documentation of this file.

```
00001 /*
00002  * File:      name.cpp
00003  * Author:    Luke de Munk
00004  * Class:     name
00005  * Version:   0.1
00006  *
00007  * Description
00008  */
00009 //#include "name.h"
00010
00011 /*****************************************************************************/
00017 /*****************************************************************************/
00018 name::name() {
00019
00020 }
00021
00022 /*****************************************************************************/
00028 /*****************************************************************************/
00029 void name::test() {
00030
00031 }
```

5.3 classTemplate.h File Reference

Classes

- class name

5.4 classTemplate.h

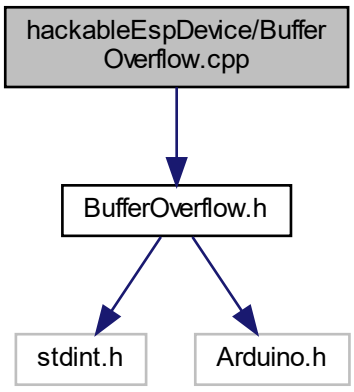
Go to the documentation of this file.

```
00001 /*
00002  * File:      name.h
00003  * Author:    Luke de Munk
00004  * Class:     name
00005  * Version:   0.1
```

```
00006  *
00007  * Description
00008  */
00009 #ifndef name_H
00010 #define name_H
00011 //#include <Arduino.h>
00012
00013 class name
00014 {
00015     public:
00016         name();
00017         void test();
00018
00019     private:
00020 };
00021 #endif
```

5.5 hackableEspDevice/BufferOverflow.cpp File Reference

`#include "BufferOverflow.h"`
Include dependency graph for BufferOverflow.cpp:



5.6 BufferOverflow.cpp

Go to the documentation of this file.

```
00001 /*
00002  * File:      BufferOverflow.h
00003  * Author:    Luke de Munk
00004  * Version:   0.1
00005  *
00006  * Buffer flow simulator. All elements of the bufferflow are in this class.
00007  */
00008 #include "BufferOverflow.h"
00009
00010 /*****************************************************************************/
00014 /*****************************************************************************/
00015 BufferOverflow::BufferOverflow() {
00016     _clearInput();                                                    //First time call is the
       declaration of the array.
00017 }
00018
00019 /*****************************************************************************/
00023 /*****************************************************************************/
00024 void BufferOverflow::ls() {
```

```
00025     Serial.println(F("testprogram.c"));
00026     Serial.println(F("testprogram"));
00027 }
00028
00029 /*****************************************************************************/
00033 /*****************************************************************************/
00034 void BufferOverflow::vi() {
00035
       Serial.println(F("|-----------FILENAME-----------|-----------TYPE-----------|-----------AUTOR-----------|"));
00036
       Serial.println(F("|---------testprogram.c--------|---------READONLY---------|-----------admin-----------|"));
00037
       Serial.println(F("|-------------------------------------------------------------------------------------|"));
00038     Serial.println(F("1     /*
            |"));
00039     Serial.println(F("2      * File:   testprogram.c
            |"));
00040     Serial.println(F("3      * Author: admin
            |"));
00041     Serial.println(F("4      *
            |"));
00042     Serial.println(F("5      * To test superuser login. DELETE WHEN FINISHING DEVELOPMENT!!!
            |"));
00043     Serial.println(F("6      */
            |"));
00044     Serial.println(F("7     #include <stdio.h>
            |"));
00045     Serial.println(F("8     #include <string.h>
            |"));
00046     Serial.println(F("9
            |"));
00047     Serial.println(F("10
            |"));
00048     Serial.println(F("11
       /*****************************************************************************/    |"));
00049     Serial.println(F("12    /*!
            |"));
00050     Serial.println(F("13      @brief    Logs given user name in as superuser and logs out again.
            |"));
00051     Serial.println(F("14    */
            |"));
00052     Serial.println(F("15
       /*****************************************************************************/    |"));
00053     Serial.println(F("16    int main(int argc, char** argv) {
            |"));
00054     Serial.println(F("17        char username[10];
            |"));
00055     Serial.println(F("18        strcpy(username, argv[1]);
            |"));
00056     Serial.println(F("19        login(*username);
            |"));
00057     Serial.println(F("20        logout();
            |"));
00058     Serial.println(F("21
            |"));
00059     Serial.println(F("22        return 0;
            |"));
00060     Serial.println(F("23    }
            |"));
00061
       Serial.println(F("|-------------------------------------------------------------------------------------|"));
00062     Serial.println(F(""));
00063 }
00064
00065 /*****************************************************************************/
00069 /*****************************************************************************/
00070 void BufferOverflow::objectDump() {
00071     Serial.println(F("testprogram:     file format elf32-littlearm"));
00072     Serial.println(F(""));
00073     Serial.println(F("Disassembly of section .init:"));
00074     Serial.println(F(""));
00075     Serial.println(F("00010438 <main>:"));
00076     Serial.println(F("   10438: e92d4800  push  {fp, lr}"));
00077     Serial.println(F("   1043c: e28db004  add fp, sp, #4"));
00078     Serial.println(F("   10440: e24dd018  sub sp, sp, #24"));
00079     Serial.println(F("   10444: e50b0018  str r0, [fp, #-24]  ; 0xffffffe8"));
00080     Serial.println(F("   10448: e50b101c  str r1, [fp, #-28]  ; 0xffffffe4"));
00081     Serial.println(F("   1044c: e51b301c  ldr r3, [fp, #-28]  ; 0xffffffe4"));
00082     Serial.println(F("   10450: e2833004  add r3, r3, #4"));
00083     Serial.println(F("   10454: e5932000  ldr r2, [r3]"));
00084     Serial.println(F("   10458: e24b3010  sub r3, fp, #16"));
00085     Serial.println(F("   1045c: e1a01002  mov r1, r2"));
00086     Serial.println(F("   10460: e1a00003  mov r0, r3"));
00087     Serial.println(F("   10464: ebffffab  bl  10318 <strcpy@plt>"));
00088     Serial.println(F("   10468: e55b3010  ldrb  r3, [fp, #-16]"));
00089     Serial.println(F("   1046c: e1a00003  mov r0, r3"));
00090     Serial.println(F("   10470: eb000004  bl  10488 <login>"));
```

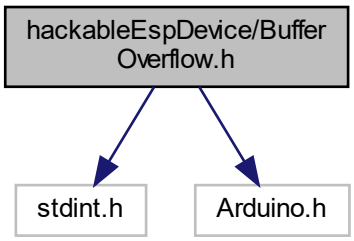
```
00091     Serial.println(F("   10474: eb00000d  bl  104b0 <logout>"));
00092     Serial.println(F("   10478: e3a03000  mov r3, #0"));
00093     Serial.println(F("   1047c: e1a00003  mov r0, r3"));
00094     Serial.println(F("   10480: e24bd004  sub sp, fp, #4"));
00095     Serial.println(F("   10484: e8bd8800  pop {fp, pc}"));
00096     Serial.println(F(""));
00097     Serial.println(F("00010488 <login>:"));
00098     Serial.println(F("   10488: e92d4800  push  {fp, lr}"));
00099     Serial.println(F("   1048c: e28db004  add fp, sp, #4"));
00100     Serial.println(F("   10490: e24dd008  sub sp, sp, #8"));
00101     Serial.println(F("   10494: e50b0008  str r0, [fp, #-8]"));
00102     Serial.println(F("   10498: e59f000c  ldr r0, [pc, #12] ; 104ac <login+0x24>"));
00103     Serial.println(F("   1049c: ebffff9a  bl  1030c <printf@plt>"));
00104     Serial.println(F("   104a0: e1a00000  nop     ; (mov r0, r0)"));
00105     Serial.println(F("   104a4: e24bd004  sub sp, fp, #4"));
00106     Serial.println(F("   104a8: e8bd8800  pop {fp, pc}"));
00107     Serial.println(F("   104ac: 0001053c  .word 0x0001053c"));
00108     Serial.println(F(""));
00109     Serial.println(F("000104b0 <logout>:"));
00110     Serial.println(F("   104b0: e92d4800  push  {fp, lr}"));
00111     Serial.println(F("   104b4: e28db004  add fp, sp, #4"));
00112     Serial.println(F("   104b8: e59f0008  ldr r0, [pc, #8]  ; 104c8 <logout+0x18>"));
00113     Serial.println(F("   104bc: ebffff92  bl  1030c <printf@plt>"));
00114     Serial.println(F("   104c0: e1a00000  nop     ; (mov r0, r0)"));
00115     Serial.println(F("   104c4: e8bd8800  pop {fp, pc}"));
00116     Serial.println(F("   104c8: 00010548  .word 0x00010548"));
00117     Serial.println(F(""));
00118 }
00119
00120 /*****************************************************************************/
00126 /*****************************************************************************/
00127 bool BufferOverflow::runCProgram(String arg) {
00128     _formatInput(arg);
00129
00130     if (_numChars < OVERFLOW_BEGIN) {
00131         Serial.println("You are now super user.");
00132         Serial.print("Hello ");
00133         Serial.println(arg);
00134         Serial.println("You are not longer super user.");
00135     } else {
00136         if (_checkBufferOverflow()) {
00137             return true;
00138         }
00139     }
00140     return false;
00141 }
00142
00143 /*****************************************************************************/
00148 /*****************************************************************************/
00149 bool BufferOverflow::_checkBufferOverflow() {
00150     if(_getOverflowPortion() == RETURN_ADDRESS) {
00151         return true;
00152     }
00153
00154     _printOverflowError();                                          //If the overflow is not
      correctly, print value of the return address pointer
00155     return false;
00156 }
00157
00158 /*****************************************************************************/
00162 /*****************************************************************************/
00163 void BufferOverflow::_printOverflowError() {
00164     Serial.println("Program received signal SIGSEGV, Segmentation fault.");
00165     Serial.print("0x");
00166     _getOverflowPortion(true);
00167     Serial.println(" in ?? ()");
00168 }
00169
00170 /*****************************************************************************/
00175 /*****************************************************************************/
00176 void BufferOverflow::_formatInput(String input) {
00177     String tmp = "";
00178
00179     _clearInput();
00180
00181     /* Set every character in an element */
00182     for (uint16_t i = 0; i < input.length(); i++) {
00183         if (input[i] == '\\') {
00184             _formattedInput[_numChars] = "\\x";                     //Move all hex chars in
      one element (for ex.: '\x90')
00185             _formattedInput[_numChars] += input[i+2];
00186             _formattedInput[_numChars] += input[i+3];
00187             i += 3;                                                 //Increase with 3, because
      the number of chars taken for a hex is 4 ('\x90')
00188         } else {
00189             _formattedInput[_numChars] = input[i];
00190         }
```

```
00191         _numChars++;
00192     }
00193
00194     /* Turn the whole array, to simulate little endian systems */
00195     for (uint8_t i = 0; i < _numChars/2-1; i++) {
00196         tmp = _formattedInput[i];
00197         _formattedInput[i] = _formattedInput[_numChars-i-1];
00198         _formattedInput[_numChars-i-1] = tmp;
00199         tmp = "";
00200     }
00201 }
00202
00203 /*****************************************************************************/
00209 /*****************************************************************************/
00210 String BufferOverflow::_getOverflowPortion(bool print) {
00211     String overflowPortion = "";
00212
00213     if (_numChars < OVERFLOW_LENGTH) {
00214         uint8_t numMissingBytes = OVERFLOW_LENGTH - _numChars;
00215         overflowPortion += _generateRandomBytes(numMissingBytes);
00216
00217         if (print) {
00218             Serial.print(overflowPortion);
00219         }
00220
00221         /* To determine and print the overflow portion */
00222         for (uint8_t i = 0; i < ADDRESS_LENGTH - numMissingBytes; i++) {
00223             /* Check if is hex number, else print as hex */
00224             if (_formattedInput[i][0] == '\\') {
00225                 overflowPortion += _formattedInput[i][2];
00226                 overflowPortion += _formattedInput[i][3];
00227                 if (print) {
00228                     Serial.print(_formattedInput[i][2]);
00229                     Serial.print(_formattedInput[i][3]);
00230                 }
00231             } else {
00232                 overflowPortion += _formattedInput[i];
00233                 if (print) {
00234                     Serial.print(char(_formattedInput[i][0]), HEX);
00235                 }
00236             }
00237         }
00238     } else {
00239         /* To print the overflow portion */
00240         uint8_t delta = abs(_numChars - OVERFLOW_LENGTH);
00241         for (uint8_t i = delta; i < delta + ADDRESS_LENGTH; i++) {
00242             /* Check if is hex number, else print as hex */
00243             if (_formattedInput[i][0] == '\\') {
00244                 overflowPortion += _formattedInput[i][2];
00245                 overflowPortion += _formattedInput[i][3];
00246                 if (print) {
00247                     Serial.print(_formattedInput[i][2]);
00248                     Serial.print(_formattedInput[i][3]);
00249                 }
00250             } else {
00251                 if (print) {
00252                     Serial.print(char(_formattedInput[i][0]), HEX);
00253                 }
00254                 overflowPortion += _formattedInput[i];
00255             }
00256         }
00257     }
00258     return overflowPortion;
00259 }
00260
00261 /*****************************************************************************/
00265 /*****************************************************************************/
00266 void BufferOverflow::_clearInput() {
00267     for (uint16_t i = 0; i < MAX_NUM_CHARS; i++) {
00268         _formattedInput[i] = "";
00269     }
00270     _numChars = 0;
00271 }
00272
00273 /*****************************************************************************/
00280 /*****************************************************************************/
00281 String BufferOverflow::_generateRandomBytes(uint8_t numBytes) {
00282     String bytes = "";
00283     randomSeed(numBytes);
00284
00285     for (uint8_t i = 0; i < numBytes; i++) {
00286         bytes += String(random(127), HEX);
00287     }
00288     return bytes;
00289 }
```

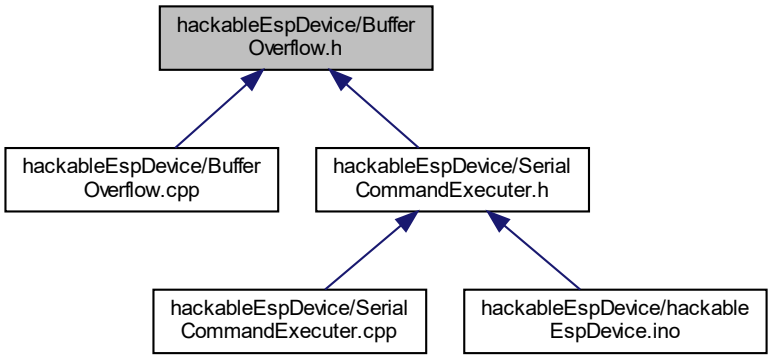
5.7 hackableEspDevice/BufferOverflow.h File Reference

```
#include <stdint.h>
#include "Arduino.h"
```

Include dependency graph for BufferOverflow.h:



This graph shows which files directly or indirectly include this file:



Classes

- class BufferOverflow

Macros

- #define OVERFLOW_BEGIN 16
- #define ADDRESS_LENGTH 4
- #define OVERFLOW_LENGTH 20
- #define RETURN_ADDRESS "00010488"
- #define MAX_NUM_CHARS 256

5.7.1 Macro Definition Documentation

5.7.1.1 ADDRESS_LENGTH

```
#define ADDRESS_LENGTH 4
```

Definition at line 15 of file BufferOverflow.h.

5.7.1.2 MAX_NUM_CHARS

```
#define MAX_NUM_CHARS 256
```

Definition at line 18 of file BufferOverflow.h.

5.7.1.3 OVERFLOW_BEGIN

```
#define OVERFLOW_BEGIN 16
```

Definition at line 14 of file BufferOverflow.h.

5.7.1.4 OVERFLOW_LENGTH

```
#define OVERFLOW_LENGTH 20
```

Definition at line 16 of file BufferOverflow.h.

5.7.1.5 RETURN_ADDRESS

```
#define RETURN_ADDRESS "00010488"
```

Definition at line 17 of file BufferOverflow.h.

5.8 BufferOverflow.h

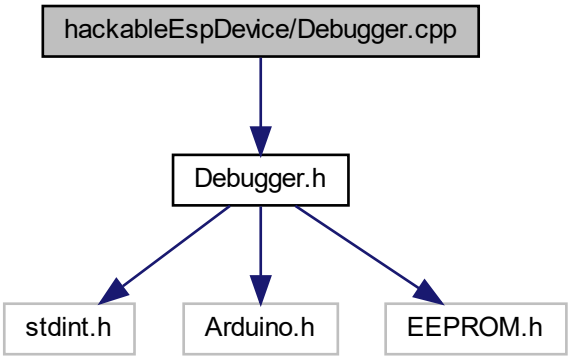
Go to the documentation of this file.

```
00001 /*
00002  * File:      BufferOverflow.h
00003  * Author:    Luke de Munk
00004  * Class:     BufferOverflow
00005  * Version:   0.1
00006  *
00007  * Buffer flow simulator. All elements of the bufferflow are in this class.
00008  */
00009 #ifndef BUFFER_OVERFLOW_H
00010 #define BUFFER_OVERFLOW_H
00011 #include <stdint.h>                                                    //For defining bits per
       integer
00012 #include "Arduino.h"
00013
00014 #define OVERFLOW_BEGIN    16                                           //Because array is in byte
       resolution, 10 becomes 16. Then the return address pointer starts
00015 #define ADDRESS_LENGTH    4                                            //Address is 32 bits long,
       so 4 bytes
00016 #define OVERFLOW_LENGTH   20                                           //OVERFLOW_BEGIN +
       ADDRESS_LENGTH
00017 #define RETURN_ADDRESS    "00010488"                                   //0x00010488 == address of
       login function.
00018 #define MAX_NUM_CHARS     256
00019
00020 class BufferOverflow
00021 {
00022     public:
00023         BufferOverflow();
00024         void ls();
00025         void vi();
00026         void objectDump();
00027         bool runCProgram(String arg);
00028
00029     private:
00030         bool _checkBufferOverflow();
00031         void _printOverflowError();
00032         void _formatInput(String input);
00033         String _getOverflowPortion(bool print = false);
00034         void _clearInput();
00035         String _generateRandomBytes(uint8_t numberOfBytes);
00036
00037         String _formattedInput[256];
00038         uint8_t _numChars;
00039 };
00040 #endif
```

5.9 hackableEspDevice/Debugger.cpp File Reference

```
#include "Debugger.h"
```

Include dependency graph for Debugger.cpp:



Functions

- void debug (String text)

  Prints text if debug is enabled.
- void debugln (String text)

  *Prints text (+'
  ') if debug is enabled.*
- bool getDebugEnabled ()

  Gets if debug is enabled.
- void setDebugEnabled (bool isEnabled)

  Sets if debug is enabled.

5.9.1 Function Documentation

5.9.1.1 debug()

```
void debug (
            String text )
```

Prints text if debug is enabled.

Parameters

text	String of text that needs to be printed

Definition at line 16 of file Debugger.cpp.

5.9.1.2 debugln()

```
void debugln (
            String text )
```

Prints text (+'
') if debug is enabled.

Parameters

text	String of text that needs to be printed

Definition at line 32 of file Debugger.cpp.

5.9.1.3 getDebugEnabled()

```
bool getDebugEnabled ( )
```

Gets if debug is enabled.

Returns

isEnabled If debug is enabled (true == enabled)

Definition at line 47 of file Debugger.cpp.

5.9.1.4 setDebugEnabled()

```
void setDebugEnabled (
            bool isEnabled )
```

Sets if debug is enabled.

Parameters

isEnabled	If debug is enabled (true == enabled)

Definition at line 60 of file Debugger.cpp.

5.10 Debugger.cpp

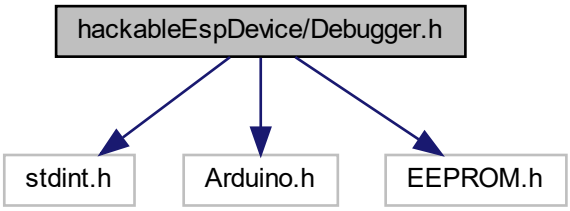
Go to the documentation of this file.

```
00001 /*
00002  * File:      Debugger.h
00003  * Author:    Luke de Munk
00004  * Version:   0.1
00005  *
00006  * Class for handling the debug prints.
00007  */
00008 #include "Debugger.h"
00009
00010 /*****************************************************************************/
00015 /*****************************************************************************/
00016 void debug(String text) {
00017     EEPROM.begin(1);
00018     bool isEnabled = EEPROM.read(ENABLE_DEBUG_FLAG_ADDRESS);
00019
00020     if(isEnabled) {
00021         Serial.print(text);
00022     }
00023     EEPROM.end();
00024 }
00025
00026 /*****************************************************************************/
00031 /*****************************************************************************/
00032 void debugln(String text) {
00033     EEPROM.begin(1);
00034     bool isEnabled = EEPROM.read(ENABLE_DEBUG_FLAG_ADDRESS);
00035     if(isEnabled) {
00036         Serial.println(text);
00037     }
00038     EEPROM.end();
00039 }
00040
00041 /*****************************************************************************/
00046 /*****************************************************************************/
00047 bool getDebugEnabled() {
00048     EEPROM.begin(1);
00049     bool isEnabled = EEPROM.read(ENABLE_DEBUG_FLAG_ADDRESS);
00050     EEPROM.end();
00051     return isEnabled;
00052 }
00053
00054 /*****************************************************************************/
00059 /*****************************************************************************/
00060 void setDebugEnabled(bool isEnabled) {
00061     EEPROM.begin(1);
00062     EEPROM.write(ENABLE_DEBUG_FLAG_ADDRESS, (uint8_t) isEnabled);           //Set the debug flag
00063     EEPROM.commit();                                                        //Write to EEPROM
00064     EEPROM.end();
00065 }
```

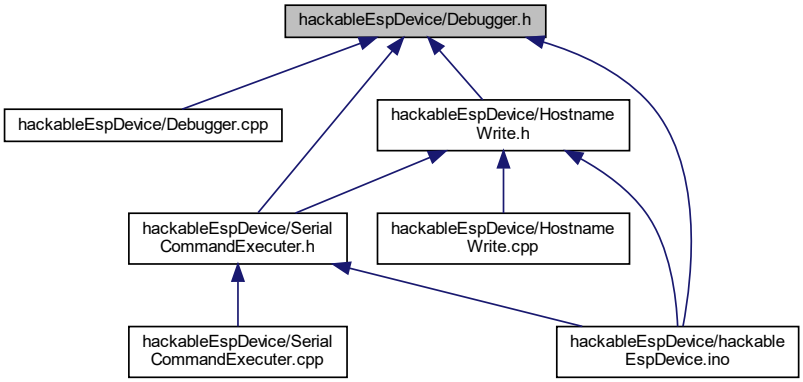
5.11 hackableEspDevice/Debugger.h File Reference

```
#include <stdint.h>
#include "Arduino.h"
#include <EEPROM.h>
```

Include dependency graph for Debugger.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define ENABLE_DEBUG_FLAG_ADDRESS 0

Functions

- void debug (String text)

  Prints text if debug is enabled.
- void debugln (String text)

  *Prints text (+'
  ') if debug is enabled.*
- bool getDebugEnabled ()

  Gets if debug is enabled.
- void setDebugEnabled (bool isEnabled)

  Sets if debug is enabled.

5.11.1 Macro Definition Documentation

5.11.1.1 ENABLE_DEBUG_FLAG_ADDRESS

```
#define ENABLE_DEBUG_FLAG_ADDRESS 0
```

Definition at line 21 of file Debugger.h.

5.11.2 Function Documentation

5.11.2.1 debug()

```
void debug (
            String text )
```

Prints text if debug is enabled.

Parameters

text	String of text that needs to be printed

Definition at line 16 of file Debugger.cpp.

5.11.2.2 debugln()

```
void debugln (
            String text )
```

Prints text (+'
') if debug is enabled.

Parameters

text	String of text that needs to be printed

Definition at line 32 of file Debugger.cpp.

5.11.2.3 getDebugEnabled()

```
bool getDebugEnabled ( )
```

Gets if debug is enabled.

Returns

isEnabled If debug is enabled (true == enabled)

Definition at line 47 of file Debugger.cpp.

5.11.2.4 setDebugEnabled()

```
void setDebugEnabled (
            bool isEnabled )
```

Sets if debug is enabled.

Parameters

isEnabled	If debug is enabled (true == enabled)

Definition at line 60 of file Debugger.cpp.

5.12 Debugger.h

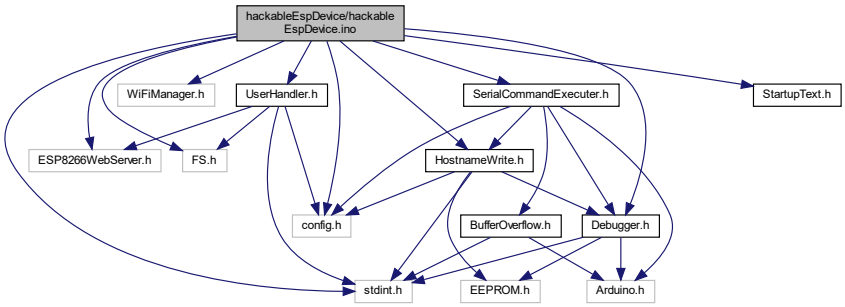
Go to the documentation of this file.

```
00001 /*
00002  * File:      Debugger.h
00003  * Author:    Luke de Munk
00004  * Version:   0.1
00005  *
00006  * Class for handling the debug prints.
00007  */
00008 #ifndef DEBUGGER_H
00009 #define DEBUGGER_H
00010 #include <stdint.h>                                                    //For defining bits per
       integer
00011 #include "Arduino.h"
00012 #include <EEPROM.h>                                                    //For reading from and
       writing to flash memory, used for resetting wifi
00013
00014 /*
00015  * 1 byte to store the enable debug flag.
00016  * Is done in EEPROM, because the
00017  * flag is then non-volatile and can
00018  * be used by multiple classes. Also
00019  * is saved during restart.
00020  */
00021 #define ENABLE_DEBUG_FLAG_ADDRESS   0
00022
00023 void debug(String text);
00024 void debugln(String text);
00025 bool getDebugEnabled();
00026 void setDebugEnabled(bool isEnabled);
00027
00028 #endif
```

5.13 hackableEspDevice/hackableEspDevice.ino File Reference

```
#include <ESP8266WebServer.h>
#include <FS.h>
#include <stdint.h>
#include <WiFiManager.h>
#include "config.h"
#include "UserHandler.h"
#include "SerialCommandExecuter.h"
#include "Debugger.h"
#include "HostnameWrite.h"
#include "StartupText.h"
```

Include dependency graph for hackableEspDevice.ino:



Macros

- #define ON HIGH
- #define OFF LOW
- #define MIN_BRIGHTNESS 1022

Functions

- ESP8266WebServer server (80)
- void setup ()

  Setup microchip.
- void initializeHostname ()

  Initializes hostname.
- void setupWifi ()

  Connects to WiFi if it can, otherwise starts as AP to configure WiFI.
- void initializeServer ()

  Takes care of the webservices like pageloading.
- void sendToFrontend (String var)

  Replaces placeholders with actual data in HTML page.
- void loop ()

  Mainloop.
- String getContentType (String filename)

  Converts the file extension to the MIME type.
- void handleFileRequest (String path, uint8_t permissionLevel)

Sends the requested file if the user has permission.

- void handleFileUpload ()

  Handles the file upload to the SPIFFS.

- void handleFileDownload ()

  Handles the file download to the SPIFFS.

Variables

- uint8_t ledState = OFF
- uint16_t brightness = 1023
- UserHandler userHandler & server
- SerialCommandExecuter cliExecuter
- File fsUploadFile

5.13.1 Macro Definition Documentation

5.13.1.1 MIN_BRIGHTNESS

```
#define MIN_BRIGHTNESS 1022
```

Definition at line 25 of file hackableEspDevice.ino.

5.13.1.2 OFF

```
#define OFF LOW
```

Definition at line 23 of file hackableEspDevice.ino.

5.13.1.3 ON

```
#define ON HIGH
```

Definition at line 22 of file hackableEspDevice.ino.

5.13.2 Function Documentation

5.13.2.1 getContentType()

```
String getContentType (
            String filename )
```

Converts the file extension to the MIME type.

Parameters

filename	Name of the file

Returns

String MIME type of the file

Definition at line 302 of file hackableEspDevice.ino.

5.13.2.2 handleFileDownload()

```
void handleFileDownload ( )
```

Handles the file download to the SPIFFS.

Definition at line 386 of file hackableEspDevice.ino.

5.13.2.3 handleFileRequest()

```
void handleFileRequest (
            String path,
            uint8_t permissionLevel )
```

Sends the requested file if the user has permission.

Parameters

path	Path to the file
permissionLevel	0 = not logged in, 1 = user, 2 = admin

Definition at line 319 of file hackableEspDevice.ino.

5.13.2.4 handleFileUpload()

```
void handleFileUpload ( )
```

Handles the file upload to the SPIFFS.

Definition at line 351 of file hackableEspDevice.ino.

5.13.2.5 initializeHostname()

```
void initializeHostname ( )
```

Initializes hostname.

Definition at line 75 of file hackableEspDevice.ino.

5.13.2.6 initializeServer()

```
void initializeServer ( )
```

Takes care of the webservices like pageloading.

Definition at line 133 of file hackableEspDevice.ino.

5.13.2.7 loop()

```
void loop ( )
```

Mainloop.

Definition at line 287 of file hackableEspDevice.ino.

5.13.2.8 sendToFrontend()

```
void sendToFrontend (
            String var )
```

Replaces placeholders with actual data in HTML page.

Definition at line 274 of file hackableEspDevice.ino.

5.13.2.9 server()

```
ESP8266WebServer server (
            80  )
```

5.13.2.10 setup()

```
void setup ( )
```

Setup microchip.

Definition at line 41 of file hackableEspDevice.ino.

5.13.2.11 setupWifi()

```
void setupWifi ( )
```

Connects to WiFi if it can, otherwise starts as AP to configure WiFI.

Definition at line 106 of file hackableEspDevice.ino.

5.13.3 Variable Documentation

5.13.3.1 brightness

```
uint16_t brightness = 1023
```

Definition at line 29 of file hackableEspDevice.ino.

5.13.3.2 cliExecuter

```
SerialCommandExecuter cliExecuter
```

Definition at line 32 of file hackableEspDevice.ino.

5.13.3.3 fsUploadFile

```
File fsUploadFile
```

Definition at line 34 of file hackableEspDevice.ino.

5.13.3.4 ledState

```
uint8_t ledState = OFF
```

Definition at line 28 of file hackableEspDevice.ino.

5.13.3.5 server

```
UserHandler userHandler& server
```

Definition at line 31 of file hackableEspDevice.ino.

5.14 hackableEspDevice.ino

Go to the documentation of this file.

```
00001 /*
00002  * File:      hackableEspDevice.ino
00003  * Authors:   ESPinoza (Team 1)
00004  * Version:   0.1
00005  *
00006  * The main file of the firmware of a vunerable-by-design ESP8266 controller.
00007  * For more information, go to: https://gitlab.fdmci.hva.nl/munkl/hackable_esp_device
00008  *
00009  */
00010 #include <ESP8266WebServer.h>                                    //For running the
       webserver
00011 #include <FS.h>                                                  //For SPIFFS
00012 #include <stdint.h>                                              //For defining bits per
       integer
00013 #include <WiFiManager.h>                                         //For web-based wifi
       configuration
00014 #include "config.h"                                              //For the configuration
00015 #include "UserHandler.h"                                         //For handling the users
       from the config.conf
00016 #include "SerialCommandExecuter.h"                               //For handling serial
       commands
00017 #include "Debugger.h"                                            //For handling debug
       messages
00018 #include "HostnameWrite.h"                                       //For handling the
       hostname changes
00019 #include "StartupText.h"                                         //For printing startup log
       files
00020
00021 /* On and off are inverted because the built-in led is active low */
00022 #define ON                    HIGH
00023 #define OFF                   LOW
00024
00025 #define MIN_BRIGHTNESS        1022                               //analogWrite() on ESP8266
       D1 Mini board is inverted
00026
00027 ESP8266WebServer server(80);                                     //Object that listens for
       HTTP requests on port 80
00028 uint8_t ledState = OFF;                                          //Declare led state
       variable
00029 uint16_t brightness = 1023;                                      //For LED brightnesss
00030
00031 UserHandler userHandler(&server);                                //For handling the
       authentication
00032 SerialCommandExecuter cliExecuter;
00033
00034 File fsUploadFile;                                               //A File object to
       temporarily store the received file
00035
00036 /*********************************************************************/
00040 /*********************************************************************/
00041 void setup() {
00042     Serial.begin(115200);                                        //Serial port for
       debugging purposes
00043
00044     /* Initialize SPIFFS */
```

```
00045     if (!SPIFFS.begin()) {
00046         Serial.println("An Error has occurred while mounting SPIFFS");
00047         return;
00048     }
00049 
00050     debugln("Debug is enabled");
00051 
00052     /* If debug is enabled, the root password is printed in a big string of text */
00053     if (getDebugEnabled()) {
00054       String mess = "ROOT: " + String(ROOT_PASSWORD);
00055       printStartupText(mess);
00056     }
00057 
00058     pinMode(LED_BUILTIN, OUTPUT);
00059     analogWrite(LED_BUILTIN, 1023);
00060 
00061     initializeHostname();
00062     setupWifi();
00063     initializeServer();
00064     userHandler.updateUsers();
00065     cliExecuter.setUsers(userHandler.getUsers(), userHandler.getNumberOfUsers());
00066 
00067     debugln("Serial commands available. Typ 'help' for help.");
00068 }
00069 
00070 /*****************************************************************************/
00074 /*****************************************************************************/
00075 void initializeHostname() {
00076     String customHostname = getHostname();
00077     /* Check if custom hostname is set, otherwise use default */
00078     if (customHostname != "") {
00079         /* Check if hostname can be set */
00080         if (WiFi.hostname(customHostname)){
00081             debug(customHostname);
00082             debugln(" is the hostname.");
00083         } else {
00084             debug("Could not set '");
00085             debug(customHostname);
00086             debugln("' as hostname.");
00087         }
00088     } else {
00089         if (WiFi.hostname(DEFAULT_HOSTNAME)) {
00090             debug(DEFAULT_HOSTNAME);
00091             debugln(" is the hostname.");
00092         } else {
00093             debug("Could not set '");
00094             debug(DEFAULT_HOSTNAME);
00095             debugln("' as hostname.");
00096         }
00097     }
00098 }
00099 
00100 /*****************************************************************************/
00105 /*****************************************************************************/
00106 void setupWifi() {
00107     WiFiManager wifiManager;
00108 
00109     if (wifiManager.autoConnect(WIFI_CONF_AP_NAME)) {
00110         Serial.print("Connected to: ");
00111         Serial.println(WiFi.SSID());
00112         Serial.print("IP: ");
00113         Serial.println(WiFi.localIP());
00114     } else {
00115         Serial.println("Failed to connect, connect with AP");
00116         ESP.restart();
00117     }
00118 
00119     debug("Copy and paste the following URL: http://");
00120 
00121     if (WiFi.hostname(DEFAULT_HOSTNAME)) {
00122         debugln(DEFAULT_HOSTNAME);
00123     } else {
00124         debugln(WiFi.hostname().c_str());
00125     }
00126 }
00127 
00128 /*****************************************************************************/
00132 /*****************************************************************************/
00133 void initializeServer() {
00134     /*
00135     *  Routes for loading all the necessary files
00136     */
00137     /* Route for home page */
00138     server.on("/", HTTP_GET, []() {
00139         handleFileRequest("/index.html", PERMISSION_LVL_ALL);
00140     });
00141 
```

```
00142     server.on("/state", HTTP_GET, []() {
00143         sendToFrontend("ledState");
00144     });
00145
00146     server.on("/brightness", HTTP_GET, []() {
00147         sendToFrontend("brightness");
00148     });
00149
00150     /* Route for admin controls */
00151     server.on("/admin", HTTP_GET, []() {
00152         handleFileRequest("/admin.html", PERMISSION_LVL_ADMIN);
00153     });
00154
00155     /* Route for user controls */
00156     server.on("/user", HTTP_GET, []() {
00157         handleFileRequest("/user.html", PERMISSION_LVL_USER);
00158     });
00159
00160     /* Route for file upload page */
00161     server.on("/upload", HTTP_GET, []() {
00162         handleFileRequest("/upload.html", PERMISSION_LVL_ADMIN);
00163     });
00164
00165     /* Route for file download page */
00166     server.on("/download", HTTP_GET, []() {
00167         handleFileRequest("/download.html", PERMISSION_LVL_USER);
00168     });
00169
00170     /* Load style_desktop.css file, styling for desktop version */
00171     server.on("/styles.css", HTTP_GET, []() {
00172         handleFileRequest("/styles.css", PERMISSION_LVL_ALL);
00173     });
00174
00175     /* Load style_mobile.css file, styling for mobile version */
00176     server.on("/styles_mobile.css", HTTP_GET, []() {
00177         handleFileRequest("/styles_mobile.css", PERMISSION_LVL_ALL);
00178     });
00179
00180     /* Load style_switch.css file, styling for the on/off switch */
00181     server.on("/style_switch.css", HTTP_GET, []() {
00182         handleFileRequest("/style_switch.css", PERMISSION_LVL_ALL);
00183     });
00184
00185     /* Load favicon.ico file, site icon */
00186     server.on("/favicon.ico", HTTP_GET, []() {
00187         handleFileRequest("/favicon.ico", PERMISSION_LVL_ALL);
00188     });
00189
00190     /* Load jquery.min.js file, for ajax */
00191     server.on("/jquery.min.js", HTTP_GET, []() {
00192         handleFileRequest("/jquery.min.js", PERMISSION_LVL_ALL);
00193     });
00194
00195     /* Load base.js file, JavaScript for site */
00196     server.on("/base.js", HTTP_GET, []() {
00197         handleFileRequest("/base.js", PERMISSION_LVL_ALL);
00198     });
00199
00200     /* Load switch.js file, JavaScript for on/off switch */
00201     server.on("/switch.js", HTTP_GET, []() {
00202         handleFileRequest("/switch.js", PERMISSION_LVL_ALL);
00203     });
00204     /*
00205     * End of file loading
00206     */
00207
00208     /*
00209     * Routes for JavaScript data receiving
00210     */
00211     /* Route for setting power */
00212     server.on("/set_power", HTTP_GET, []() {
00213         if (server.arg("state")) {
00214             ledState = atoi(server.arg("state").c_str());
00215             if(ledState == ON) {
00216                 analogWrite(LED_BUILTIN, MIN_BRIGHTNESS-brightness);
00217             } else {
00218                 analogWrite(LED_BUILTIN, 1023);
00219             }
00220         }
00221         handleFileRequest("/index.html", PERMISSION_LVL_ALL);
00222     });
00223
00224     /* Route for brightness */
00225     server.on("/update_brightness", HTTP_GET, []() {
00226         if (server.arg("brightness")) {
00227             brightness = atoi(server.arg("brightness").c_str());
00228             if(ledState == ON) {
```

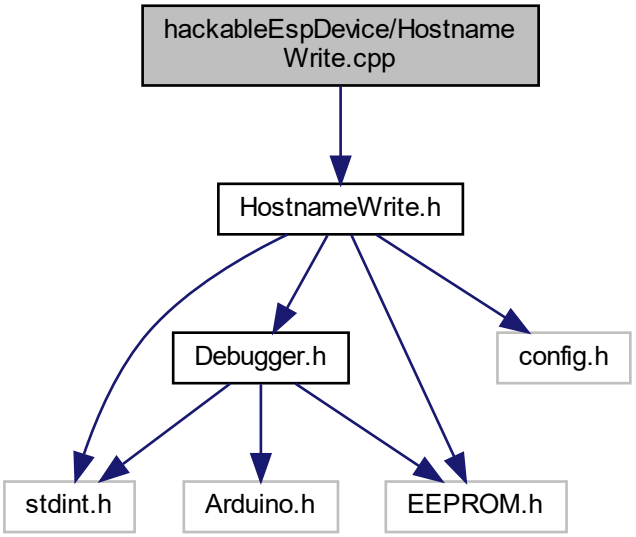
```
00229                 analogWrite(LED_BUILTIN, MIN_BRIGHTNESS-brightness);
00230             }
00231         }
00232         handleFileRequest("/index.html", PERMISSION_LVL_ALL);
00233     });
00234
00235
00236     /* Route for restarting the server */
00237     server.on("/restart", HTTP_GET, [](){
00238         handleFileRequest("/", PERMISSION_LVL_ALL);
00239         ESP.restart();
00240     });
00241     /*
00242     * End of JavaScript data receiving
00243     */
00244     /*
00245     * Routes for file management
00246     */
00247     /* Route for file upload request */
00248     server.on("/upload", HTTP_POST, [](){
00249         server.send(200);                                                   //HTTP code 200 == OK
00250         debugln("Wait, something got uploaded");
00251       }, handleFileUpload                                                   //Receive and save the
    file
00252     );
00253     /* Route for file upload request */
00254     server.on("/download", HTTP_POST, [](){
00255         debugln("File download request");
00256       }, handleFileDownload                                                 //Receive and save the
    file
00257     );
00258     /*
00259     * End of routes for file management
00260     */
00261     /* Not found */
00262     server.onNotFound([](){                                                 //If the client requests
    any URI
00263         handleFileRequest(server.uri(), PERMISSION_LVL_ALL);                //send it if it exists
00264         debugln("NOT_FOUND?");
00265     });
00266     server.begin();                                                         //Start server
00267 }
00268
00269 /*********************************************************************/
00273 /*********************************************************************/
00274 void sendToFrontend(String var){
00275     if (var == "ledState") {
00276         server.send(200, "text/plain", String (ledState));
00277     } else if (var == "brightness") {
00278         server.send(200, "text/plain", String (brightness));
00279     }
00280 }
00281
00282 /*********************************************************************/
00286 /*********************************************************************/
00287 void loop() {
00288   server.handleClient();
00289
00290   if (Serial.available()) {
00291     cliExecuter.executeCommand();
00292   }
00293 }
00294
00295 /*********************************************************************/
00301 /*********************************************************************/
00302 String getContentType(String filename) {
00303   if (filename.endsWith(".html")) return "text/html";
00304   else if (filename.endsWith(".css")) return "text/css";
00305   else if (filename.endsWith(".js")) return "application/javascript";
00306   else if (filename.endsWith(".ico")) return "image/x-icon";
00307   else if (filename.endsWith(".gz")) return "application/x-gzip";
00308   else if (filename.endsWith(".txt")) return "text/plain";
00309   return "text/plain";
00310 }
00311
00312 /*********************************************************************/
00318 /*********************************************************************/
00319 void handleFileRequest(String path, uint8_t permissionLevel) {
00320     if(!userHandler.checkPermission(permissionLevel, &server)) {
00321         server.requestAuthentication();
00322         return;
00323     }
00324
00325     debugln(String("Requested file: ") + path);
00326
00327     String contentType = getContentType(path);                              //Get the MIME type
00328     String pathWithGz = path + ".gz";
```

```
00329
00330     if (SPIFFS.exists(pathWithGz)) {                                    //If there's a compressed
      version available
00331         path += ".gz";                                                  //Use the compressed
      verion
00332     }
00333
00334     if (SPIFFS.exists(path)) {
00335         File file = SPIFFS.open(path, "r");                             //Open the file
00336         size_t sent = server.streamFile(file, contentType);             //Send it to the client
00337         file.close();                                                   //Close the file again
00338         debugln(String("Sent file: ") + path);
00339         return;
00340     }
00341
00342     debugln(String("File Not Found: ") + path);                         //If the file doesn't
      exist, return false
00343     server.send(404, "text/plain", "404: Not Found");                   //otherwise, respond with
      a 404 (Not Found) error
00344 }
00345
00346 /*********************************************************************/
00350 /*********************************************************************/
00351 void handleFileUpload() {
00352     HTTPUpload& upload = server.upload();
00353
00354     if (upload.status == UPLOAD_FILE_START) {
00355         String filename = upload.filename;
00356
00357         if (!filename.startsWith("/")) {
00358             filename = "/" + filename;
00359         }
00360
00361         debugln(String("Upload file named: ") + filename);
00362
00363         fsUploadFile = SPIFFS.open(filename, "w");                      //Open the file for
      writing in SPIFFS (create if it doesn't exist)
00364
00365     } else if (upload.status == UPLOAD_FILE_WRITE && fsUploadFile ) {
00366         fsUploadFile.write(upload.buf, upload.currentSize);             //Write the received bytes
      to the file
00367     } else if (upload.status == UPLOAD_FILE_END) {
00368         if (fsUploadFile) {                                             //If the file was
      successfully created
00369             fsUploadFile.close();                                       //Close the file again
00370             debugln(String("handleFileUpload Size: ") + upload.totalSize);
00371             server.sendHeader("Location","/success.html");              //Redirect the client to
      the success page
00372             server.send(303);
00373             userHandler.updateUsers();
00374             cliExecuter.setUsers(userHandler.getUsers(), userHandler.getNumberOfUsers()); //Update
      users for cli as well
00375         } else {
00376             server.send(500, "text/plain", "500: couldn't create file");
00377         }
00378     }
00379 }
00380
00381 /*********************************************************************/
00385 /*********************************************************************/
00386 void handleFileDownload() {
00387     String filename = server.arg("filekey");                            //Get user input for
      filename
00388
00389     if (!filename.startsWith("/")) {
00390         filename = "/" + filename;
00391     }
00392
00393     if (!SPIFFS.exists(filename)) {
00394         server.send(404, "text/plain", "404: file not found!");
00395         return;
00396     }
00397
00398     File download = SPIFFS.open(filename, "r");
00399
00400     debugln("Start sending file");
00401
00402     server.sendHeader("Content-Type", "text/text");
00403     server.sendHeader("Content-Disposition", "attachment; filename="+filename);
00404     server.sendHeader("Connection", "close");
00405     server.streamFile(download, "application/octet-stream");
00406     download.close();
00407     server.send(200);                                                   //HTTP code 200 == OK
00408 }
```

5.15 hackableEspDevice/HostnameWrite.cpp File Reference

```
#include "HostnameWrite.h"
```

Include dependency graph for HostnameWrite.cpp:



Functions

- String getHostname ()
  Gets the hostname from the EEPROM.
- void writeHostname (char hostname[MAX_HOSTNAME_LENGTH])
  Writes the new hostname to the EEPROM.
- void setEEPROMToNULL (int writeLength, int startAdress)
  Resets the EEPROM at the startAdress.
- void checkEepromCommit ()
  Checks if the eeprom was actually committed.

5.15.1 Function Documentation

5.15.1.1 checkEepromCommit()

```
void checkEepromCommit ( )
```

Checks if the eeprom was actually committed.

Definition at line 80 of file HostnameWrite.cpp.

5.15.1.2 getHostname()

```
String getHostname ( )
```

Gets the hostname from the EEPROM.

Returns

hostname contains the hostname from eeprom (true == enabled)

Definition at line 17 of file HostnameWrite.cpp.

5.15.1.3 setEEPROMToNULL()

```
void setEEPROMToNULL (
            int writeLength,
            int startAdress )
```

Resets the EEPROM at the startAdress.

Parameters

writeLength	int of total lenth to be writen
startAdress	int of start adress

Definition at line 57 of file HostnameWrite.cpp.

5.15.1.4 writeHostname()

```
void writeHostname (
            char hostname[MAX_HOSTNAME_LENGTH] )
```

Writes the new hostname to the EEPROM.

Parameters

hostname	char[32] that contains the hostname to be written

Definition at line 38 of file HostnameWrite.cpp.

5.16 HostnameWrite.cpp

Go to the documentation of this file.

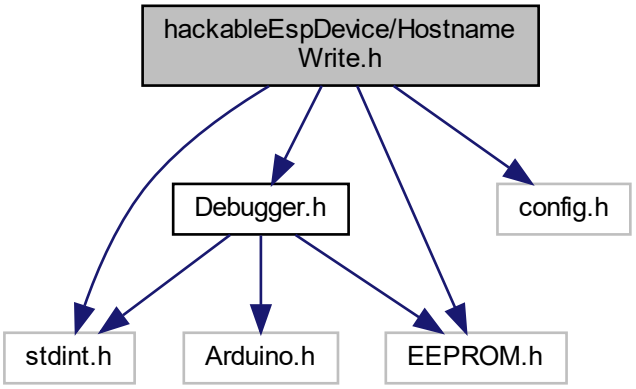
```
00001 /*
00002  * File:      HostnameWrite.cpp
00003  * Author:    Twenne Elffers
00004  * Class:     HostnameWrite
00005  * Version:   0.1
00006  *
00007  * Writes hostname to the EEPROM.
00008  */
00009 #include "HostnameWrite.h"
00010
00011 /*****************************************************************************/
00016 /*****************************************************************************/
00017 String getHostname() {
00018     char hostname[MAX_HOSTNAME_LENGTH];
00019     EEPROM.begin(MAX_HOSTNAME_LENGTH);
00020
00021     for (uint8_t i = 0; i < MAX_HOSTNAME_LENGTH; i++) {
00022         EEPROM.get(HOSTNAME_ADRESS+i, hostname[i]);
00023         if (hostname[i] == 0xFF) {
00024             break;                                                    //Skips the unreadable
      chars
00025             }
00026       }
00027
00028     EEPROM.end();
00029     return String(hostname);
00030 }
00031
00032 /*****************************************************************************/
00037 /*****************************************************************************/
00038 void writeHostname(char hostname[MAX_HOSTNAME_LENGTH]) {
00039     EEPROM.begin(MAX_HOSTNAME_LENGTH);
00040
00041     for (int i = 0; i< MAX_HOSTNAME_LENGTH; i++){
00042         EEPROM.write(HOSTNAME_ADRESS+i, hostname[i]);
00043         yield();
00044     }
00045
00046     checkEepromCommit();
00047     EEPROM.end();
00048 }
00049
00050 /*****************************************************************************/
00056 /*****************************************************************************/
00057 void setEEPROMToNULL(int writeLength, int startAdress){
00058     EEPROM.begin(writeLength);
00059
00060     for (int i = 0; i< writeLength; i++){
00061         EEPROM.write(startAdress+i, 0);
00062         yield();
00063     }
00064
00065     checkEepromCommit();
00066
00067     debug("Reset Value at: ");
00068     debug(String(startAdress));
00069     debug(" till ");
00070     debugln(String(startAdress+writeLength));
00071
00072     EEPROM.end();
00073 }
00074
00075 /*****************************************************************************/
00079 /*****************************************************************************/
00080 void checkEepromCommit() {
00081     if (EEPROM.commit()) {
00082         Serial.println("Data written!");
00083     } else {
00084         Serial.println("ERROR! Data not written!");
00085     }
00086 }
```

5.17 hackableEspDevice/HostnameWrite.h File Reference

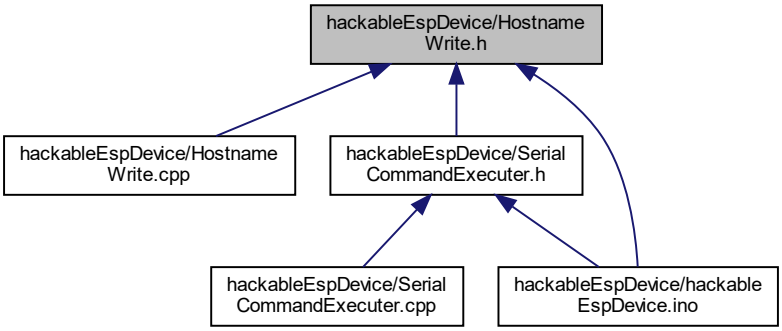
```
#include <stdint.h>
#include <EEPROM.h>
#include "Debugger.h"
```

```
#include "config.h"
```

Include dependency graph for HostnameWrite.h:



This graph shows which files directly or indirectly include this file:



Functions

- String getHostname ()

  Gets the hostname from the EEPROM.
- void writeHostname (char hostname[32])
- void setEEPROMToNULL (int writeLength, int startAdress)

  Resets the EEPROM at the startAdress.
- void checkEepromCommit ()

  Checks if the eeprom was actually committed.

5.17.1 Function Documentation

5.17.1.1 checkEepromCommit()

```
void checkEepromCommit ( )
```

Checks if the eeprom was actually committed.

Definition at line 80 of file HostnameWrite.cpp.

5.17.1.2 getHostname()

```
String getHostname ( )
```

Gets the hostname from the EEPROM.

Returns

hostname contains the hostname from eeprom (true == enabled)

Definition at line 17 of file HostnameWrite.cpp.

5.17.1.3 setEEPROMToNULL()

```
void setEEPROMToNULL (
            int writeLength,
            int startAdress )
```

Resets the EEPROM at the startAdress.

Parameters

writeLength	int of total lenth to be writen
startAdress	int of start adress

Definition at line 57 of file HostnameWrite.cpp.

5.17.1.4 writeHostname()

```
void writeHostname (
            char hostname[32] )
```

5.18 HostnameWrite.h

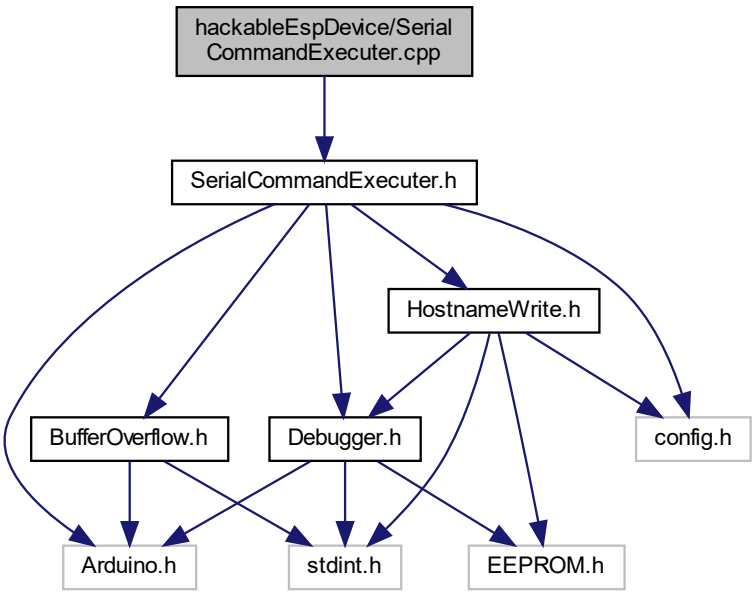
Go to the documentation of this file.

```
00001 /*
00002  * File:      HostnameWrite.h
00003  * Author:    Twenne Elffers
00004  * Class:     HostnameWrite
00005  * Version:   0.1
00006  *
00007  * Writes hostname to the EEPROM.
00008  */
00009 #ifndef HOSTNAME_WRITE_H
00010 #define HOSTNAME_WRITE_H
00011 #include <stdint.h>                                          //For defining bits per
       integer
00012 #include <EEPROM.h>                                          //For reading from and
       writing to EEPROM
00013 #include "Debugger.h"                                        //For handling debug
       messages
00014 #include "config.h"                                         //For the configuration
00015
00016 String getHostname();
00017 void writeHostname(char hostname[32]);
00018 void setEEPROMToNULL(int writeLength, int startAdress);
00019 void checkEepromCommit();
00020 #endif
```

5.19 hackableEspDevice/SerialCommandExecuter.cpp File Reference

`#include "SerialCommandExecuter.h"`

Include dependency graph for SerialCommandExecuter.cpp:



5.20 SerialCommandExecuter.cpp

Go to the documentation of this file.

```
00001 /*
00002  * File:      SerialCommandExecuter.cpp
00003  * Author:    Luke de Munk & Twenne Elffers
00004  * Class:     SerialCommandExecuter
00005  * Version:   0.1
00006  *
00007  * Parses and executes serial terminal commands.
00008  */
00009 #include "SerialCommandExecuter.h"
00010
00011 /*****************************************************************************/
00015 /*****************************************************************************/
00016 SerialCommandExecuter::SerialCommandExecuter() {
00017     _isLoggedIn = false;
00018 }
00019
00020 /*****************************************************************************/
00026 /*****************************************************************************/
00027 void SerialCommandExecuter::setUsers(String* users, uint8_t numUsers) {
00028     /* Copy users */
00029     for (uint8_t i = 0; i < MAX_NUMBER_USERS*USER_INFO_LENGTH; i++) {
00030         _users[i] = users[i];
00031     }
00032     _numberUsers = numUsers;
00033 }
00034
00035 /*****************************************************************************/
00039 /*****************************************************************************/
00040 void SerialCommandExecuter::executeCommand() {
00041     String command = Serial.readString();
00042
00043     if (command != "") {
00044         if (_isLoggedIn) {
00045             Serial.print("~# ");                                    //For the Linux feeling,
      superuser
00046         } else {
00047             Serial.print("~$ ");                                    //For the Linux feeling,
      no superuser
00048         }
00049         Serial.print(command);                                     //Echo command (command
      ends with \n)
00050
00051         _parseCommand(command);
00052     }
00053 }
00054
00055 /*****************************************************************************/
00061 /*****************************************************************************/
00062 bool SerialCommandExecuter::_parseCommand(String commandString) {
00063     String* trimmedCmdLine = _trimCommand(commandString);
00064     String command = trimmedCmdLine[0].c_str();
00065     String params[MAX_NUMBER_PARAMS] = {""};
00066     uint8_t numParams = 0;
00067
00068     while (numParams < MAX_NUMBER_PARAMS) {
00069         if (trimmedCmdLine[numParams+1] == "") {                   //+1, because the command
      is in the first cell
00070             break;
00071         }
00072         numParams++;
00073     }
00074
00075     for (uint8_t i = 1; i-1 < numParams; i++){                     //+1, because the command
      is in the first cell
00076         params[i-1] = trimmedCmdLine[i].c_str();
00077     }
00078
00079     /* Check which command is given */
00080     if (command == COMMAND_HELP) {
00081         _printHelp(COMMAND_HELP);
00082         return true;
00083     } else {
00084         /* If help needs to be printed, print it and return */
00085         if (_checkHelp(params[0], command)) {
00086             return true;
00087         }
00088     }
00089
00090     if (command == COMMAND_DEBUG) {
00091         if (!_checkParams(numParams, 1, 1) || !_enableDebug(params[0])) {
00092             return false;
00093         }
00094     } else if (command == COMMAND_SU) {
00095         if (!_checkParams(numParams, 1, 1) || !_superUserLogin(params[0])) {
00096             return false;
00097         }
00098     } else if (command == COMMAND_KEYS) {
```

```
00099         if (!_viewKey()) {
00100             return false;
00101         }
00102     } else if ((command == COMMAND_RESTART)) {
00103         _restart();
00104     } else if (command == COMMAND_USERS) {
00105         if (!_viewUsers()) {
00106             return false;
00107         }
00108     } else if (command == COMMAND_HOSTNAME) {
00109         if (!_checkParams(numParams, 0, 2) || !_hostname(params)) {
00110             return false;
00111         }
00112     } else if (command == COMMAND_WHOAMI) {
00113         if (_isLoggedIn) {
00114             Serial.println("superuser");
00115         } else {
00116             Serial.println("user");
00117         }
00118         return true;
00119     } else if (command == COMMAND_LS) {
00120         buffOverflow.ls();
00121     } else if (command == COMMAND_VI) {
00122         if (_checkParams(numParams, 1, 1)) {
00123             if (params[0] == "./testprogram.c" || params[0] == "testprogram.c") {
00124                 buffOverflow.vi();
00125             } else {
00126                 Serial.println(ERROR_NO_FILE);
00127                 return false;
00128             }
00129         }
00130     } else if (command.substring(0, 2) == COMMAND_RUN) {                    //Substring == "./" the
      rest is filename
00131         if (_checkParams(numParams, 0, 1)) {
00132             String filename = command.substring(2);                          //The rest of the
      command is filename
00133
00134             if (filename == "testprogram.c") {
00135                 Serial.println(ERROR_PERM_DENIED);
00136                 return false;
00137             }
00138
00139             if (filename != "testprogram") {
00140                 Serial.println(ERROR_NO_FILE_DIR);
00141                 return false;
00142             }
00143
00144             if (numParams == 1) {
00145                 /* If buffer overflow is done correctly,
00146                  * user is logged in.
00147                  */
00148                 if (buffOverflow.runCProgram(params[0])) {
00149                     _isLoggedIn = true;
00150                     Serial.println(MESS_SUPER_USER);
00151                 }
00152             } else {
00153                 buffOverflow.runCProgram("");
00154             }
00155         }
00156     } else if (command == COMMAND_OBJDUMP) {
00157         if (_checkParams(numParams, 2, 2)) {
00158             if (params[0] != "-d") {
00159                 Serial.println(ERROR_WRONG_ARGS);
00160                 return false;
00161             }
00162             if (params[1] == "./testprogram" || params[1] == "testprogram") {
00163                 buffOverflow.objectDump();
00164             } else {
00165                 Serial.println(ERROR_NO_FILE);
00166                 return false;
00167             }
00168         }
00169     } else {
00170         Serial.println(ERROR_CMD_NOT_FOUND);
00171         return false;
00172     }
00173     return true;
00174 }
00175
00176 /*****************************************************************************/
00182 /*****************************************************************************/
00183 String* SerialCommandExecuter::_trimCommand(String commandString) {
00184     static String commandItems[1+MAX_NUMBER_PARAMS] = {""};                  //To save command and
      parameters, each in own cell
00185     String item = "";                                                        //Can be a command or
      parameter
00186     uint8_t paramCounter = 0;
```

```
00187
00188     /* Reset static array */
00189     for (uint16_t x = 0; x < 1+MAX_NUMBER_PARAMS; x++) {
00190         commandItems[x] = "";
00191     }
00192
00193     /* Count number of parameters by adding to temp variable if not a whitespace or end of line*/
00194     for (uint16_t c = 0; c < commandString.length(); c++) {
00195         if (commandString[c] == ' ' || commandString[c] == '\n') {
00196             /* If item is not empty: add to item array */
00197             if (item != ""){
00198                 commandItems[paramCounter] = item;                          //Save param to items list
00199                 item = "";                                                 //Reset item value
00200                 paramCounter++;
00201             }
00202         } else { // if not a whitepace add to item
00203             item += commandString[c];
00204         }
00205     }
00206     return commandItems;
00207 }
00208
00209 /*****************************************************************************/
00216 /*****************************************************************************/
00217 bool SerialCommandExecuter::_checkParams(uint8_t numParams, uint8_t minNumberParams, uint8_t
     maxNumberParams) {
00218     if (numParams < minNumberParams) {
00219         Serial.println(ERROR_TOO_FEW_ARGS);
00220         return false;
00221     } else if (numParams > maxNumberParams) {
00222         Serial.println(ERROR_TOO_MANY_ARGS);
00223         return false;
00224     }
00225     return true;
00226 }
00227
00228 /*****************************************************************************/
00233 /*****************************************************************************/
00234 void SerialCommandExecuter::_printHelp(String command) {
00235     /* Print help lines according to command */
00236     if (command == "" || command == COMMAND_HELP) {                        //Default help
00237         Serial.println("|--------------------------HELP--------------------------|");
00238         Serial.println("This is a commandline interface that allows access to the smartlight config");
00239         _printCommands();
00240     } else if (command == COMMAND_DEBUG) {
00241         Serial.println("Usage: debug [--off]                    Turns the debug off");
00242         Serial.println("       debug [--on]                     Turns the debug on");
00243     } else if (command == COMMAND_SU) {
00244         Serial.println("Usage: su {passwd}                      Login as superuser");
00245     } else if (command == COMMAND_KEYS) {
00246         Serial.println("Usage: privatekeys                      Shows private encryption keys");
00247     } else if (command == COMMAND_RESTART) {
00248         Serial.println("Usage: reboot                           Reboots the device");
00249     } else if (command == COMMAND_USERS) {
00250         Serial.println("Usage: users                            Shows usertable of website");
00251     } else if (command == COMMAND_HOSTNAME) {
00252         Serial.println("Usage: hostname                         Gives the current hostname");
00253         Serial.println("       hostname [--set] {newhostname}   Set new hostname. (needs reboot)");
00254         Serial.println("       hostname [--default]             Sets the hostname to the default
     hostname");
00255     } else if (command == COMMAND_LS) {
00256         Serial.println("Usage: ls                               Shows files in current folder");
00257     } else if (command == COMMAND_VI) {
00258         Serial.println("Usage: vi {filename}                    Opens file in text editor");
00259     }else if (command == COMMAND_RUN) {
00260         Serial.println("Usage: ./{filename}                     Runs an executable file");
00261     } else if (command == COMMAND_OBJDUMP) {
00262         Serial.println("Usage: objdump -d {filename}            Prints disassembled code of an
     executable file");
00263     } else {
00264         Serial.println(ERROR_CMD_NOT_FOUND);
00265     }
00266 }
00267
00268 /*****************************************************************************/
00272 /*****************************************************************************/
00273 void SerialCommandExecuter::_printCommands() {
00274     Serial.println("Available commands:");
00275     Serial.println(COMMAND_HELP);
00276     Serial.println(COMMAND_DEBUG);
00277     Serial.println(COMMAND_SU);
00278     Serial.println(COMMAND_KEYS);
00279     Serial.println(COMMAND_RESTART);
00280     Serial.println(COMMAND_USERS);
00281     Serial.println(COMMAND_HOSTNAME);
00282     Serial.println(COMMAND_LS);
00283     Serial.println(COMMAND_VI);
```

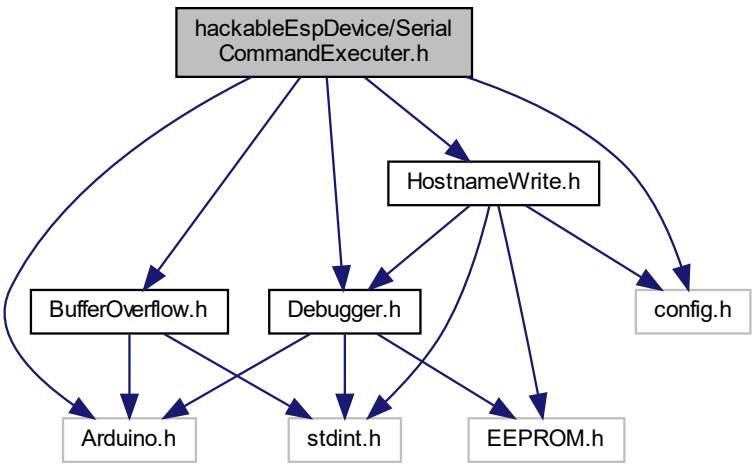
```
00284     Serial.println(COMMAND_OBJDUMP);
00285     Serial.println(COMMAND_WHOAMI);
00286 }
00287
00288 /*****************************************************************************/
00295 /*****************************************************************************/
00296 bool SerialCommandExecuter::_enableDebug(String enable) {
00297     if (enable == "--on") {
00298         setDebugEnabled(true);
00299         Serial.println("debug = true");
00300     } else if (enable == "--off") {
00301         setDebugEnabled(false);
00302         Serial.println("debug = false");
00303     } else {
00304         Serial.println(ERROR_WRONG_ARGS);
00305         return false;
00306     }
00307     return true;
00308 }
00309
00310 /*****************************************************************************/
00316 /*****************************************************************************/
00317 bool SerialCommandExecuter::_superUserLogin(String password) {
00318     if (password == ROOT_PASSWORD) {
00319         _isLoggedIn = true;
00320         Serial.println(MESS_SUPER_USER);
00321     } else {
00322         Serial.println(ERROR_WRONG_PWD);
00323         return false;
00324     }
00325     return true;
00326 }
00327
00328 /*****************************************************************************/
00333 /*****************************************************************************/
00334 bool SerialCommandExecuter::_viewKey() {
00335     if (!_isLoggedIn) {
00336         Serial.println(ERROR_NO_PERMISSION);
00337         return false;
00338     }
00339     Serial.println("Private encryption keys. Don't share!!!");
00340     Serial.println("");
00341     return true;
00342 }
00343
00344 /*****************************************************************************/
00348 /*****************************************************************************/
00349 void SerialCommandExecuter::_restart() {
00350     Serial.print("Restarting in ");
00351
00352     /* Wait 3 seconds */
00353     for (uint8_t s = 3; s > 0; s--) {
00354         Serial.print(s);
00355         Serial.print(" ");
00356         delay(1000);
00357     }
00358     ESP.restart();
00359 }
00360
00361 /*****************************************************************************/
00366 /*****************************************************************************/
00367 bool SerialCommandExecuter::_viewUsers() {
00368     String userPrints[USER_INFO_LENGTH] = {""};
00369
00370     if (!_isLoggedIn) {
00371         Serial.println(ERROR_NO_PERMISSION);
00372         return false;
00373     }
00374
00375     Serial.println("|-USERNAME------|-PASSWORD------|-ROLE--|");
00376
00377     for (uint8_t i = 0; i < _numberUsers; i += 3) {
00378         userPrints[0] = _users[i].c_str();                                  //Username
00379         if(atoi(_users[i+2].c_str()) == PERMISSION_LVL_USER) {
00380             userPrints[1] = _users[i+1].c_str();                           //Password
00381             userPrints[2] = "User";                                        //Permission level/role
00382         } else if (atoi(_users[i+2].c_str()) == PERMISSION_LVL_ADMIN) {
00383             userPrints[1] = "******";                                      //Password, not printed
00384             userPrints[2] = "Admin";                                       //Permission level/role
00385         }
00386         Serial.printf("| %s\t| %s\t| %s\t|\n", userPrints[0].c_str(), userPrints[1].c_str(),
      userPrints[2].c_str());
00387     }
00388     return true;
00389 }
00390
00391 /*****************************************************************************/
```

```
00397 /*****************************************************************************/
00398 bool SerialCommandExecuter::_hostname(String* params) {
00399     uint8_t numParams = params->length();
00400     if (numParams == 0) {                                          //If empty: show hostname
00401         Serial.print("Hostname is: ");
00402         Serial.println(String(getHostname()));
00403         return true;
00404     }
00405
00406     if (params[0] == "--set" && params[1] != "") {                 //If parameter == "--set"
      check if next value is not empty
00407         char newHostname[MAX_HOSTNAME_LENGTH];
00408         params[1].toCharArray(newHostname, MAX_HOSTNAME_LENGTH);
00409         writeHostname(newHostname);
00410     } else if (params[0] == "--default"){
00411         writeHostname(DEFAULT_HOSTNAME);
00412     } else {
00413         Serial.println(ERROR_WRONG_ARGS);                          //If it can't find
      suitable params: give error
00414         return false;
00415     }
00416     return true;
00417 }
00418
00419 /*****************************************************************************/
00427 /*****************************************************************************/
00428 bool SerialCommandExecuter::_checkHelp(String param, String command) {
00429     if (param == "-h" || param == "--help") {
00430         _printHelp(command);
00431         return true;
00432     }
00433     return false;
00434 }
```

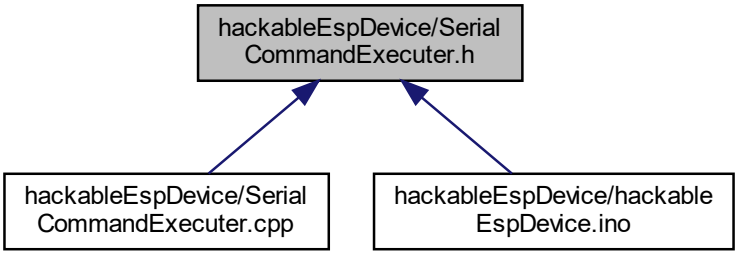
5.21 hackableEspDevice/SerialCommandExecuter.h File Reference

```
#include "Arduino.h"
#include "config.h"
#include "Debugger.h"
#include "HostnameWrite.h"
#include "BufferOverflow.h"
```

Include dependency graph for SerialCommandExecuter.h:



This graph shows which files directly or indirectly include this file:



Classes

- class SerialCommandExecuter

Macros

- #define MAX_NUMBER_PARAMS 2
- #define COMMAND_HELP "help"
- #define COMMAND_DEBUG "debug"
- #define COMMAND_SU "su"
- #define COMMAND_KEYS "privatekeys"
- #define COMMAND_RESTART "reboot"
- #define COMMAND_USERS "users"
- #define COMMAND_HOSTNAME "hostname"
- #define COMMAND_WHOAMI "whoami"
- #define COMMAND_LS "ls"
- #define COMMAND_VI "vi"
- #define COMMAND_RUN "./"
- #define COMMAND_OBJDUMP "objdump"
- #define MESS_SUPER_USER "You are now super user."
- #define ERROR_TOO_MANY_ARGS "Too many arguments. Typ 'help' for help."
- #define ERROR_CMD_NOT_FOUND "Bash: command not found. Typ 'help' for help."
- #define ERROR_PERM_DENIED "Bash: Permission denied"
- #define ERROR_WRONG_ARGS "Wrong argument(s). Add '-h' or '--help' to the command for help."
- #define ERROR_TOO_FEW_ARGS "Too few arguments. Typ 'help' for help."
- #define ERROR_WRONG_PWD "Wrong password."
- #define ERROR_NO_PERMISSION "You are no super user. Use 'su {password}' to log in."
- #define ERROR_NO_FILE "No such file."
- #define ERROR_NO_FILE_DIR "No such file or directory."

5.21.1 Macro Definition Documentation

5.21.1.1 COMMAND_DEBUG

```
#define COMMAND_DEBUG "debug"
```

Definition at line 20 of file SerialCommandExecuter.h.

5.21.1.2 COMMAND_HELP

```
#define COMMAND_HELP "help"
```

Definition at line 19 of file SerialCommandExecuter.h.

5.21.1.3 COMMAND_HOSTNAME

```
#define COMMAND_HOSTNAME "hostname"
```

Definition at line 25 of file SerialCommandExecuter.h.

5.21.1.4 COMMAND_KEYS

```
#define COMMAND_KEYS "privatekeys"
```

Definition at line 22 of file SerialCommandExecuter.h.

5.21.1.5 COMMAND_LS

```
#define COMMAND_LS "ls"
```

Definition at line 29 of file SerialCommandExecuter.h.

5.21.1.6 COMMAND_OBJDUMP

```
#define COMMAND_OBJDUMP "objdump"
```

Definition at line 32 of file SerialCommandExecuter.h.

5.21.1.7 COMMAND_RESTART

```
#define COMMAND_RESTART "reboot"
```

Definition at line 23 of file SerialCommandExecuter.h.

5.21.1.8 COMMAND_RUN

```
#define COMMAND_RUN "./"
```

Definition at line 31 of file SerialCommandExecuter.h.

5.21.1.9 COMMAND_SU

```
#define COMMAND_SU "su"
```

Definition at line 21 of file SerialCommandExecuter.h.

5.21.1.10 COMMAND_USERS

```
#define COMMAND_USERS "users"
```

Definition at line 24 of file SerialCommandExecuter.h.

5.21.1.11 COMMAND_VI

```
#define COMMAND_VI "vi"
```

Definition at line 30 of file SerialCommandExecuter.h.

5.21.1.12 COMMAND_WHOAMI

```
#define COMMAND_WHOAMI "whoami"
```

Definition at line 26 of file SerialCommandExecuter.h.

5.21.1.13 ERROR_CMD_NOT_FOUND

```
#define ERROR_CMD_NOT_FOUND "Bash:  command not found.  Typ 'help' for help."
```

Definition at line 37 of file SerialCommandExecuter.h.

5.21.1.14 ERROR_NO_FILE

```
#define ERROR_NO_FILE "No such file."
```

Definition at line 43 of file SerialCommandExecuter.h.

5.21.1.15 ERROR_NO_FILE_DIR

```
#define ERROR_NO_FILE_DIR "No such file or directory."
```

Definition at line 44 of file SerialCommandExecuter.h.

5.21.1.16 ERROR_NO_PERMISSION

```
#define ERROR_NO_PERMISSION "You are no super user.  Use 'su {password}' to log in."
```

Definition at line 42 of file SerialCommandExecuter.h.

5.21.1.17 ERROR_PERM_DENIED

```
#define ERROR_PERM_DENIED "Bash:  Permission denied"
```

Definition at line 38 of file SerialCommandExecuter.h.

5.21.1.18 ERROR_TOO_FEW_ARGS

```
#define ERROR_TOO_FEW_ARGS "Too few arguments.  Typ 'help' for help."
```

Definition at line 40 of file SerialCommandExecuter.h.

5.21.1.19 ERROR_TOO_MANY_ARGS

```
#define ERROR_TOO_MANY_ARGS "Too many arguments.  Typ 'help' for help."
```

Definition at line 36 of file SerialCommandExecuter.h.

5.21.1.20 ERROR_WRONG_ARGS

```
#define ERROR_WRONG_ARGS "Wrong argument(s).  Add '-h' or '--help' to the command for help."
```

Definition at line 39 of file SerialCommandExecuter.h.

5.21.1.21 ERROR_WRONG_PWD

```
#define ERROR_WRONG_PWD "Wrong password."
```

Definition at line 41 of file SerialCommandExecuter.h.

5.21.1.22 MAX_NUMBER_PARAMS

```
#define MAX_NUMBER_PARAMS 2
```

Definition at line 17 of file SerialCommandExecuter.h.

5.21.1.23 MESS_SUPER_USER

```
#define MESS_SUPER_USER "You are now super user."
```

Definition at line 34 of file SerialCommandExecuter.h.

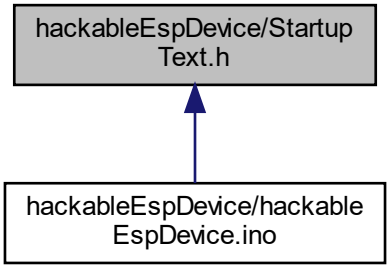
5.22 SerialCommandExecuter.h

Go to the documentation of this file.

```
00001 /*
00002  * File:      SerialCommandExecuter.h
00003  * Author:    Luke de Munk & Twenne Elffers
00004  * Class:     SerialCommandExecuter
00005  * Version:   0.1
00006  *
00007  * Parses and executes serial terminal commands.
00008  */
00009 #ifndef SERIAL_COMMAND_EXECUTER_H
00010 #define SERIAL_COMMAND_EXECUTER_H
00011 #include "Arduino.h"
00012 #include "config.h"                                                    //For the configuration
00013 #include "Debugger.h"                                                  //For handling debug
     messages
00014 #include "HostnameWrite.h"
00015 #include "BufferOverflow.h"
00016
00017 #define MAX_NUMBER_PARAMS  2
00018
00019 #define COMMAND_HELP            "help"
00020 #define COMMAND_DEBUG           "debug"
00021 #define COMMAND_SU              "su"
00022 #define COMMAND_KEYS            "privatekeys"
00023 #define COMMAND_RESTART         "reboot"
00024 #define COMMAND_USERS           "users"
00025 #define COMMAND_HOSTNAME        "hostname"
00026 #define COMMAND_WHOAMI          "whoami"
00027
00028 /* Used for buffer overflow */
00029 #define COMMAND_LS              "ls"
00030 #define COMMAND_VI              "vi"
00031 #define COMMAND_RUN             "./"
00032 #define COMMAND_OBJDUMP         "objdump"
00033
00034 #define MESS_SUPER_USER         "You are now super user."
00035
00036 #define ERROR_TOO_MANY_ARGS     "Too many arguments. Typ 'help' for help."
00037 #define ERROR_CMD_NOT_FOUND     "Bash: command not found. Typ 'help' for help."
00038 #define ERROR_PERM_DENIED       "Bash: Permission denied"
00039 #define ERROR_WRONG_ARGS        "Wrong argument(s). Add '-h' or '--help' to the command for help."
00040 #define ERROR_TOO_FEW_ARGS      "Too few arguments. Typ 'help' for help."
00041 #define ERROR_WRONG_PWD         "Wrong password."
00042 #define ERROR_NO_PERMISSION     "You are no super user. Use 'su {password}' to log in."
00043 #define ERROR_NO_FILE           "No such file."
00044 #define ERROR_NO_FILE_DIR       "No such file or directory."
00045
00046 class SerialCommandExecuter
00047 {
00048     public:
00049         SerialCommandExecuter();
00050         void executeCommand();
00051         void setUsers(String* users, uint8_t numUsers);
00052
00053     private:
00054         bool _parseCommand(String command);
00055         String* _trimCommand(String commandString);
00056         bool _checkParams(uint8_t numParams, uint8_t minNumberParams, uint8_t maxNumberParams);
00057
00058         void _printHelp(String command);
00059         void _printCommands();
00060         bool _enableDebug(String enable);
00061         bool _superUserLogin(String password);
00062         bool _viewKey();
00063         void _restart();
00064         bool _viewUsers();
00065         bool _hostname(String* params);
00066         bool _checkHelp(String param, String command);
00067
00068         bool _isLoggedIn;
00069         String _users[MAX_NUMBER_USERS*USER_INFO_LENGTH];
00070         uint8_t _numberUsers;
00071         BufferOverflow buffOverflow;
00072 };
00073 #endif
```

5.23 hackableEspDevice/StartupText.h File Reference

This graph shows which files directly or indirectly include this file:



Macros

- #define LENGTH 20

Functions

- bool printStartupText (String hiddenMess)

  Prints bytes of information with a message wrapped in it.
- bool printStringInBytes (String str)

  Converts message in bytes and prints it.

5.23.1 Macro Definition Documentation

5.23.1.1 LENGTH

```
#define LENGTH 20
```

Definition at line 12 of file StartupText.h.

5.23.2 Function Documentation

5.23.2.1 printStartupText()

```
bool printStartupText (
            String hiddenMess )
```

Prints bytes of information with a message wrapped in it.

Parameters

hiddenMess	String of text that needs to be printed

Returns

bool If conversion is successfull

Definition at line 25 of file StartupText.h.

5.23.2.2 printStringInBytes()

```
bool printStringInBytes (
             String str )
```

Converts message in bytes and prints it.

Parameters

str	String of text that needs to be printed

Returns

bool If conversion is successfull

Definition at line 303 of file StartupText.h.

5.24 StartupText.h

Go to the documentation of this file.

```
00001 /*
00002  * File:      startupText.h
00003  * Author:    Luke de Munk
00004  * Version:   0.1
00005  *
00006  * Static text in bytes that is printed when debug is on, to show
00007  * vulnerable information packed in it.
00008  */
00009 #ifndef STARTUP_TEXT_H
00010 #define STARTUP_TEXT_H
00011
00012 #define LENGTH    20                                          //Number of bytes per line
00013
00014 /* Declare functions, because it is not a class */
00015 bool printStartupText(String hiddenMess);
00016 bool printStringInBytes(String str);
00017
00018 /*****************************************************************************/
00024 /*****************************************************************************/
00025 bool printStartupText(String hiddenMess) {
00026     /* Serial.println(F(x));, because then the strings are stored in FLASH */
00027     Serial.println(F("Bootlog file print: "));
00028     Serial.println(F("53 74 61 72 74 75 70 20 62 75 73 79 2E 2E 2E 0A 65 74 73"));
00029     Serial.println(F("20 4A 61 6E 20 20 38 20 32 30 31 33 2C 72 73 74 20 63 61"));
00030     Serial.println(F("75 73 65 3A 32 2C 20 62 6F 6F 74 20 6D 6F 64 65 3A 28 33"));
00031     Serial.println(F("2C 36 29 0A 6C 6F 61 64 20 30 78 34 30 31 30 66 30 30 30"));
```

```
00032     Serial.println(F("2C 20 6C 65 6E 20 33 35 38 34 2C 20 72 6F 6F 6D 20 31 36"));
00033     Serial.println(F("20 0A 74 61 69 6C 20 30 0A 63 68 6B 73 75 6D 20 30 78 62"));
00034     Serial.println(F("30 0A 63 73 75 6D 20 30 78 62 30 0A 76 32 38 34 33 61 35"));
00035     Serial.println(F("61 63 0A 7E 6C 64 0A 45 78 65 63 75 74 61 62 6C 65 20 73"));
00036     Serial.println(F("65 67 6D 65 6E 74 20 73 69 7A 65 73 3A 0A 49 52 4F 4D 20"));
00037     Serial.println(F("20 20 3A 20 33 30 38 31 35 36 20 20 20 20 20 20 20 20 20"));
00038     Serial.println(F("20 2D 20 63 6F 64 65 20 69 6E 20 66 6C 61 73 68 20 20 20"));
00039     Serial.println(F("20 20 20 20 20 20 28 64 65 66 61 75 6C 74 20 6F 72 20 49"));
00040     Serial.println(F("43 41 43 48 45 5F 46 4C 41 53 48 5F 41 54 54 52 29 20 0A"));
00041     Serial.println(F("49 52 41 4D 20 20 20 3A 20 32 37 32 39 32 20 20 20 2F 20"));
00042     Serial.println(F("33 32 37 36 38 20 2D 20 63 6F 64 65 20 69 6E 20 49 52 41"));
00043     Serial.println(F("4D 20 20 20 20 20 20 20 20 20 20 28 49 43 41 43 48 45 5F"));
00044     Serial.println(F("52 41 4D 5F 41 54 54 52 2C 20 49 53 52 73 2E 2E 2E 29 20"));
00045     Serial.println(F("0A 44 41 54 41 20 20 20 3A 20 31 32 35 32 20 20 29 20 20"));
00046     Serial.println(F("20 20 20 20 20 20 2D 20 69 6E 69 74 69 61 6C 69 7A 65"));
00047     Serial.println(F("64 20 76 61 72 69 61 62 6C 65 73 20 28 67 6C 6F 62 61 6C"));
00048     Serial.println(F("2C 20 73 74 61 74 69 63 29 20 69 6E 20 52 41 4D 2F 48 45"));
00049     Serial.println(F("41 50 20 0A 52 4F 44 41 54 41 20 3A 20 33 30 35 36 20 20"));
00050     Serial.println(F("29 20 2F 20 38 31 39 32 30 20 2D 20 63 6F 6E 73 74 61 6E"));
00051     Serial.println(F("74 73 20 20 20 20 20 20 20 20 20 20 20 20 20 28 67 6C 6F"));
00052     Serial.println(F("62 61 6C 2C 20 73 74 61 74 69 63 29 20 69 6E 20 52 41 4D"));
00053     Serial.println(F("2F 48 45 41 50 20 0A 42 53 53 20 20 20 20 3A 20 32 36 33"));
00054     Serial.println(F("36 38 20 29 20 20 20 20 20 20 20 20 20 20 2D 20 7A 65 72 6F"));
00055     Serial.println(F("65 64 20 76 61 72 69 61 62 6C 65 73 20 20 20 20 20 20 28"));
00056     Serial.println(F("67 6C 6F 62 61 6C 2C 20 73 74 61 74 69 63 29 20 69 6E 20"));
00057     Serial.println(F("52 41 4D 2F 48 45 41 50 20 0A 42 6F 61 72 64 20 20 3A 20"));
00058     Serial.println(F("22 57 65 4D 6F 73 20 44 31 20 4D 69 6E 69 22 0A 44 65 62"));
00059     Serial.println(F("75 67 20 20 3A 20 54 72 75 65 0A 43 50 55 20 66 72 65 71"));
00060     Serial.println(F("75 65 6E 63 79 20 3A 20 38 30 4D 48 7A 0A 56 75 6C 6E 65"));
00061     Serial.println(F("72 61 62 69 6C 69 74 79 20 41 73 73 65 73 73 6D 65 6E 74"));
00062     Serial.println(F("20 53 63 61 6E 20 53 74 61 74 75 73 0A 53 69 6E 67 6C 65"));
00063     Serial.println(F("20 6D 61 74 63 68 69 6E 67 20 61 63 63 6F 75 6E 74 20 66"));
00064     Serial.println(F("6F 75 6E 64 20 69 6E 20 64 6F 6D 61 69 6E 0A 55 73 65 72"));
00065     Serial.println(F("20 61 75 74 68 65 6E 74 69 63 61 74 69 6F 6E 20 61 67 61"));
00066     Serial.println(F("69 6E 73 74 20 41 63 74 69 76 65 20 44 69 72 65 63 74 6F"));
00067     Serial.println(F("72 79 20 66 61 69 6C 65 64 20 73 69 6E 63 65 20 75 73 65"));
00068     Serial.println(F("72 20 69 73 20 63 6F 6E 73 69 64 65 72 65 64 20 74 6F 20"));
00069     Serial.println(F("62 65 20 69 6E 20 72 65 73 74 72 69 63 74 65 64 20 6C 6F"));
00070     Serial.println(F("67 6F 6E 20 68 6F 75 72 73 0A 54 72 75 73 74 73 65 63 20"));
00071     Serial.println(F("65 67 72 65 73 73 20 70 6F 6C 69 63 79 20 77 61 73 20 73"));
00072     Serial.println(F("75 63 63 65 73 73 66 75 6C 6C 79 20 64 6F 77 6E 6C 6F 61"));
00073     Serial.println(F("64 65 64 0A 52 41 44 49 55 53 20 44 54 4C 53 3A 20 72 65"));
00074     Serial.println(F("63 65 69 76 65 64 20 63 6C 69 65 6E 74 20 68 65 6C 6C 6F"));
00075     Serial.println(F("20 76 65 72 69 66 79 20 72 65 71 75 65 73 74 0A 54 68 65"));
00076     Serial.println(F("20 75 73 65 72 27 73 20 6F 72 20 68 6F 73 74 27 73 20 61"));
00077     Serial.println(F("63 63 6F 75 6E 74 20 69 73 20 69 6E 20 72 65 73 74 72 69"));
00078     Serial.println(F("63 74 65 64 20 6C 6F 67 6F 6E 20 68 6F 75 72 73 3B 20 73"));
00079     Serial.println(F("65 74 74 69 6E 67 20 74 68 65 20 49 64 65 6E 74 69 74 79"));
00080     Serial.println(F("41 63 63 65 73 73 52 65 73 74 72 69 63 74 65 64 20 66 6C"));
00081     Serial.println(F("61 67 20 74 6F 20 74 72 75 65 2E 20 74 72 75 65 0A 53 65"));
00082     Serial.println(F("6E 74 20 54 45 41 50 20 52 65 73 75 6C 74 20 54 4C 56 20"));
00083     Serial.println(F("69 6E 64 69 63 61 74 69 6E 67 20 73 75 63 63 65 73 73 0A"));
00084     Serial.println(F("47 75 65 73 74 20 73 65 73 73 69 6F 6E 20 6C 69 6D 69 74"));
00085     Serial.println(F("20 69 73 20 61 63 74 69 76 65 3B 20 72 65 6D 6F 76 69 6E"));
00086     Serial.println(F("67 20 6F 6C 64 65 72 20 67 75 65 73 74 20 73 65 73 73 69"));
00087     Serial.println(F("6F 6E 73 0A 53 65 76 65 72 61 6C 20 63 65 72 74 69 66 69"));
00088     Serial.println(F("63 61 74 65 73 20 61 72 65 20 63 6F 6E 66 69 67 75 72 65"));
00089     Serial.println(F("64 20 6F 6E 20 49 64 50 2C 68 6F 77 65 76 65 72 20 63 61"));
00090     Serial.println(F("6E 20 6E 6F 74 20 64 65 74 65 72 6D 69 6E 65 20 63 65 72"));
00091     Serial.println(F("74 69 66 69 63 61 74 65 20 66 6F 72 20 73 69 67 6E 61 74"));
00092     Serial.println(F("75 72 65 0A 53 75 73 70 65 6E 64 20 6C 6F 67 20 63 6F 6C"));
00093     Serial.println(F("6C 65 63 74 6F 72 0A 46 61 69 6C 65 64 20 74 6F 20 6A 6F"));
00094     Serial.println(F("69 6E 20 74 6F 20 41 44 0A 49 67 6E 6F 72 65 20 4D 61 63"));
00095     Serial.println(F("68 69 6E 65 20 41 75 74 68 6F 72 69 7A 61 74 69 6F 6E 20"));
00096     Serial.println(F("50 41 43 20 72 65 71 75 65 73 74 20 62 65 63 61 75 73 65"));
00097     Serial.println(F("20 6F 66 20 63 75 72 72 65 6E 74 20 50 41 43 20 6F 66 20"));
00098     Serial.println(F("74 68 65 20 73 61 6D 65 20 74 79 70 65 20 77 61 73 20 75"));
00099     Serial.println(F("73 65 64 20 74 6F 20 73 6B 69 70 20 69 6E 6E 65 72 20 6D"));
00100     Serial.println(F("65 74 68 6F 64 0A 4E 54 50 20 53 65 72 76 65 72 20 73 65"));
00101     Serial.println(F("74 0A 43 68 69 70 20 69 73 20 45 53 50 38 32 36 36 45 58"));
00102     Serial.println(F("0A 46 65 61 74 75 72 65 73 3A 20 57 69 46 69 0A 43 72 79"));
00103     Serial.println(F("73 74 61 6C 20 69 73 20 32 36 4D 48 7A 0A 4D 41 43 3A 20"));
00104     Serial.println(F("38 63 3A 61 61 3A 62 35 3A 37 62 3A 65 30 3A 61 38 0A 43"));
00105     Serial.println(F("6F 6D 70 72 65 73 73 65 64 20 33 34 34 36 30 38 20 62 79"));
00106     Serial.println(F("74 65 73 20 74 6F 20 32 34 38 38 32 36 2E 2E 2E 0A 48 61"));
00107     Serial.println(F("73 68 20 6F 66 20 64 61 74 61 20 76 65 72 69 66 69 65 64"));
00108     Serial.println(F("2E 0A 43 6C 69 65 6E 74 20 63 65 72 74 69 66 69 63 61 74"));
00109     Serial.println(F("65 20 77 61 73 20 72 65 71 75 65 73 74 65 64 20 62 75 74"));
00110     Serial.println(F("20 6E 6F 74 20 72 65 63 65 69 76 65 64 20 69 6E 73 69 64"));
00111     Serial.println(F("65 20 74 68 65 20 74 75 6E 6E 65 6C 2E 20 57 69 6C 6C 20"));
00112     Serial.println(F("63 6F 6E 74 69 6E 75 65 20 77 69 74 68 20 69 6E 6E 65 72"));
00113     Serial.println(F("20 6D 65 74 68 6F 64 2E 0A 54 65 6C 65 6D 65 74 72 79 20"));
00114     Serial.println(F("6D 65 73 73 61 67 65 73 20 77 65 72 65 20 73 65 6E 74 20"));
00115     Serial.println(F("73 75 63 63 65 73 73 66 75 6C 6C 79 0A 44 65 6C 65 74 65"));
00116     Serial.println(F("20 6E 6F 64 65 20 66 61 69 6C 65 64 0A 50 72 6F 66 69 6C"));
00117     Serial.println(F("65 72 20 45 6E 64 50 6F 69 6E 74 20 63 6F 6C 6C 65 63 74"));
00118     Serial.println(F("69 6F 6E 20 65 76 65 6E 74 20 6F 63 63 75 72 72 65 64 0A"));
```

```
00119     Serial.println(F("52 41 44 49 55 53 20 44 54 4C 53 20 43 6F 41 20 68 61 6E"));
00120     Serial.println(F("64 73 68 61 6B 65 20 73 74 61 72 74 65 64 0A 52 75 6E 6E"));
00121     Serial.println(F("69 6E 67 20 73 74 75 62 2E 2E 2E 0A 53 74 75 62 20 72 75"));
00122     Serial.println(F("6E 6E 69 6E 67 2E 2E 2E 0A 53 74 6F 70 70 65 64 20 54 41"));
00123     Serial.println(F("43 41 43 53 2B 20 6C 69 73 74 65 6E 65 72 0A 53 65 6C 65"));
00124     Serial.println(F("63 74 65 64 20 41 63 63 65 73 73 20 53 65 72 76 69 63 65"));
00125     Serial.println(F("20 74 79 70 65 20 69 73 20 6E 6F 74 20 44 65 76 69 63 65"));
00126     Serial.println(F("20 41 64 6D 69 6E 69 73 74 72 61 74 69 6F 6E 0A 4C 6F 63"));
00127     Serial.println(F("61 6C 20 6D 6F 64 65 0A 55 73 65 72 20 61 75 74 68 65 6E"));
00128     Serial.println(F("74 69 63 61 74 69 6F 6E 20 61 67 61 69 6E 73 74 20 41 63"));
00129     Serial.println(F("74 69 76 65 20 44 69 72 65 63 74 6F 72 79 20 66 61 69 6C"));
00130     Serial.println(F("65 64 20 73 69 6E 63 65 20 75 73 65 72 20 68 61 73 20 69"));
00131     Serial.println(F("6E 76 61 6C 69 64 20 63 72 65 64 65 6E 74 69 61 6C 73 0A"));
00132     Serial.println(F("43 41 20 73 65 72 76 69 63 65 20 64 69 73 61 62 6C 65 64"));
00133     Serial.println(F("0A 43 68 61 6E 67 69 6E 67 20 62 61 75 64 20 72 61 74 65"));
00134     Serial.println(F("20 74 6F 20 34 36 30 38 30 30 0A 43 6F 6E 66 69 67 75 72"));
00135     Serial.println(F("69 6E 67 20 66 6C 61 73 68 20 73 69 7A 65 2E 2E 2E 0A 41"));
00136     Serial.println(F("75 74 6F 2D 64 65 74 65 63 74 65 64 20 46 6C 61 73 68 20"));
00137     Serial.println(F("73 69 7A 65 3A 20 34 4D 42 0A 49 6E 76 61 6C 69 64 20 6E"));
00138     Serial.println(F("65 77 20 70 61 73 73 77 6F 72 64 2E 20 43 6F 6E 74 61 69"));
00139     Serial.println(F("6E 73 20 72 65 73 65 72 76 65 64 20 77 6F 72 64 0A 52 53"));
00140     Serial.println(F("41 20 61 67 65 6E 74 20 63 6F 6E 66 69 67 75 72 61 74 69"));
00141     Serial.println(F("6F 6E 20 75 70 64 61 74 65 64 2C 20 52 53 41 20 61 67 65"));
00142     Serial.println(F("6E 74 20 72 65 73 74 61 72 74 65 64 0A 4C 6F 6F 6B 75 70"));
00143     Serial.println(F("20 53 49 44 20 42 79 20 4E 61 6D 65 20 72 65 71 75 65 73"));
00144     Serial.println(F("74 20 66 61 69 6C 65 64 0A 53 74 61 72 74 20 6C 69 73 74"));
00145     Serial.println(F("65 6E 69 6E 67 20 74 6F 20 74 63 70 20 70 6F 72 74 0A 49"));
00146     Serial.println(F("67 6E 6F 72 65 20 4D 61 63 68 69 6E 65 20 41 75 74 68 6F"));
00147     Serial.println(F("72 69 7A 61 74 69 6F 6E 20 50 41 43 20 72 65 71 75 65 73"));
00148     Serial.println(F("74 20 62 65 63 61 75 73 65 20 6F 66 20 63 75 72 72 65 6E"));
00149     Serial.println(F("74 20 50 41 43 20 6F 66 20 74 68 65 20 73 61 6D 65 20 74"));
00150     Serial.println(F("79 70 65 20 77 61 73 20 75 73 65 64 20 74 6F 20 73 6B 69"));
00151     Serial.println(F("70 20 69 6E 6E 65 72 20 6D 65 74 68 6F 64 0A 49 53 45 20"));
00152     Serial.println(F("75 70 67 72 61 64 65 20 2D 20 4D 6E 54 0A 49 53 45 20 42"));
00153     Serial.println(F("61 63 6B 75 70 20 68 61 73 20 73 74 61 72 74 65 64 0A 54"));
00154     Serial.println(F("72 75 73 74 73 65 63 20 65 67 72 65 73 73 20 70 6F 6C 69"));
00155     Serial.println(F("63 79 20 77 61 73 20 73 75 63 63 65 73 73 66 75 6C 6C 79"));
00156     Serial.println(F("20 64 6F 77 6E 6C 6F 61 64 65 64 0A 52 41 44 49 55 53 20"));
00157     Serial.println(F("44 54 4C 53 3A 20 72 65 63 65 69 76 65 64 20 63 6C 69 65"));
00158     Serial.println(F("6E 74 20 68 65 6C 6C 6F 20 76 65 72 69 66 79 20 72 65 0A"));
00159     /* Print the message, return false if is not successfull */
00160     if (!printStringInBytes(hiddenMess)) {
00161       return false;
00162     }
00163     Serial.println(F("75 65 73 74 0A 47 75 65 73 74 20 73 65 73 73 69 6F 6E 20"));
00164     Serial.println(F("6C 69 6D 69 74 20 69 73 20 61 63 74 69 76 65 3B 20 72 65"));
00165     Serial.println(F("6D 6F 76 69 6E 67 20 6F 6C 64 65 72 20 67 75 65 73 74 20"));
00166     Serial.println(F("73 65 73 73 69 6F 6E 73 0A 49 67 6E 6F 72 65 20 4D 61 63"));
00167     Serial.println(F("68 69 6E 65 20 41 75 74 68 6F 72 69 7A 61 74 69 6F 6E 20"));
00168     Serial.println(F("50 41 43 20 72 65 71 75 65 73 74 20 62 65 63 61 75 73 65"));
00169     Serial.println(F("20 6F 66 20 63 75 72 72 65 6E 74 20 50 41 43 20 6F 66 20"));
00170     Serial.println(F("74 68 65 20 73 61 6D 65 20 74 79 70 65 20 77 61 73 20 75"));
00171     Serial.println(F("73 65 64 20 74 6F 20 73 6B 69 70 20 69 6E 6E 65 72 20 6D"));
00172     Serial.println(F("65 74 68 6F 64 0A 43 6C 69 65 6E 74 20 63 65 72 74 69 66"));
00173     Serial.println(F("69 63 61 74 65 20 77 61 73 20 72 65 71 75 65 73 74 65 64"));
00174     Serial.println(F("20 62 75 74 20 6E 6F 74 20 72 65 63 65 69 76 65 64 20 69"));
00175     Serial.println(F("6E 73 69 64 65 20 74 68 65 20 74 75 6E 6E 65 6C 2E 20 57"));
00176     Serial.println(F("69 6C 6C 20 63 6F 6E 74 69 6E 75 65 20 77 69 74 68 20 69"));
00177     Serial.println(F("6E 6E 65 72 20 6D 65 74 68 6F 64 2E 0A 54 65 6C 65 6D 65"));
00178     Serial.println(F("74 72 79 20 6D 65 73 73 61 67 65 73 20 77 65 72 65 20 73"));
00179     Serial.println(F("65 6E 74 20 73 75 63 63 65 73 73 66 75 6C 6C 79 0A 50 72"));
00180     Serial.println(F("6F 66 69 6C 65 72 20 45 6E 64 50 6F 69 6E 74 20 63 6F 6C"));
00181     Serial.println(F("6C 65 63 74 69 6F 6E 20 65 76 65 6E 74 20 6F 63 63 75 72"));
00182     Serial.println(F("72 65 64 0A 52 41 44 49 55 53 20 44 54 4C 53 20 43 6F 41"));
00183     Serial.println(F("20 68 61 6E 64 73 68 61 6B 65 20 73 74 61 72 74 65 64 0A"));
00184     Serial.println(F("53 74 6F 70 70 65 64 20 54 41 43 41 43 53 2B 20 6C 69 73"));
00185     Serial.println(F("74 65 6E 65 72 0A 53 65 6C 65 63 74 65 64 20 41 63 63 65"));
00186     Serial.println(F("73 73 20 53 65 72 76 69 63 65 20 74 79 70 65 20 69 73 20"));
00187     Serial.println(F("6E 6F 74 20 44 65 76 69 63 65 20 41 64 6D 69 6E 69 73 74"));
00188     Serial.println(F("72 61 74 69 6F 6E 0A 43 41 20 73 65 72 76 69 63 65 20 64"));
00189     Serial.println(F("69 73 61 62 6C 65 64 0A 52 53 41 20 61 67 65 6E 74 20 63"));
00190     Serial.println(F("6F 6E 66 69 67 75 72 61 74 69 6F 6E 20 75 70 64 61 74 65"));
00191     Serial.println(F("64 2C 20 52 53 41 20 61 67 65 6E 74 20 72 65 73 74 61 72"));
00192     Serial.println(F("74 65 64 0A 53 74 61 72 74 20 6C 69 73 74 65 6E 69 6E 67"));
00193     Serial.println(F("20 74 6F 20 74 63 70 20 70 6F 72 74 0A 49 67 6E 6F 72 65"));
00194     Serial.println(F("20 4D 61 63 68 69 6E 65 20 41 75 74 68 6F 72 69 7A 61 74"));
00195     Serial.println(F("69 6F 6E 20 50 41 43 20 72 65 71 75 65 73 74 20 62 65 63"));
00196     Serial.println(F("61 75 73 65 20 6F 66 20 63 75 72 72 65 6E 74 20 50 41 43"));
00197     Serial.println(F("20 6F 66 20 74 68 65 20 73 61 6D 65 20 74 79 70 65 20 77"));
00198     Serial.println(F("61 73 20 75 73 65 64 20 74 6F 20 73 6B 69 70 20 69 6E 6E"));
00199     Serial.println(F("65 72 20 6D 65 74 68 6F 64 0A 49 53 45 20 42 61 63 6B 75"));
00200     Serial.println(F("70 20 68 61 73 20 73 74 61 72 74 65 64 0A 53 6D 61 72 74"));
00201     Serial.println(F("20 4C 69 63 65 6E 73 69 6E 67 20 61 75 74 68 6F 72 69 7A"));
00202     Serial.println(F("61 74 69 6F 6E 20 72 65 6E 65 77 61 6C 20 73 75 63 63 65"));
00203     Serial.println(F("73 73 0A 52 65 6D 69 6E 64 65 72 3A 20 41 73 73 69 67 6E"));
00204     Serial.println(F("20 4E 41 44 20 50 72 6F 66 69 6C 65 73 2E 0A 52 41 44 49"));
00205     Serial.println(F("55 53 20 44 54 4C 53 3A 20 73 65 6E 74 20 66 69 6E 69 73"));
```

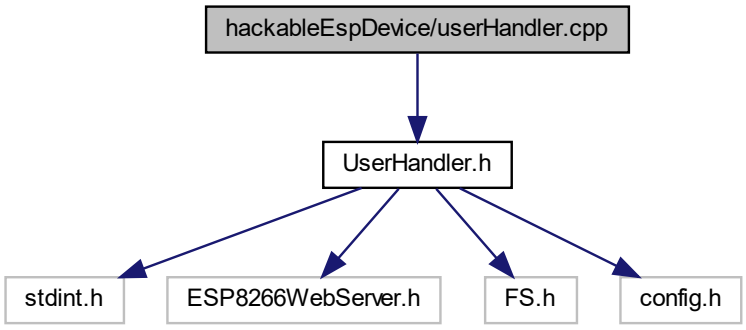
```
00206     Serial.println(F("68 65 64 20 6D 65 73 73 61 67 65 0A 50 72 65 70 61 72 65"));
00207     Serial.println(F("64 20 54 4C 53 20 53 65 72 76 65 72 4B 65 79 45 78 63 68"));
00208     Serial.println(F("61 6E 67 65 20 6D 65 73 73 61 67 65 0A 54 68 65 20 73 65"));
00209     Serial.println(F("63 75 72 69 64 20 66 69 6C 65 20 68 61 73 20 62 65 65 6E"));
00210     Serial.println(F("20 72 65 6D 6F 76 65 64 0A 55 70 64 61 74 65 64 20 45 41"));
00211     Serial.println(F("50 2D 54 4C 53 20 4D 61 73 74 65 72 20 4B 65 79 20 47 65"));
00212     Serial.println(F("6E 65 72 61 74 69 6F 6E 20 70 65 72 69 6F 64 0A 50 65 72"));
00213     Serial.println(F("66 6F 72 6D 65 64 20 66 61 6C 6C 62 61 63 6B 20 74 6F 20"));
00214     Serial.println(F("73 65 63 6F 6E 64 61 72 79 20 4F 43 53 50 20 73 65 72 76"));
00215     Serial.println(F("65 72 0A 49 53 45 20 68 61 73 20 72 65 66 72 65 73 68 65"));
00216     Serial.println(F("64 20 61 75 74 68 65 6E 74 69 63 61 74 69 6F 6E 20 61 67"));
00217     Serial.println(F("61 69 6E 73 74 20 41 50 49 43 20 73 75 63 63 65 73 73 66"));
00218     Serial.println(F("75 6C 6C 79 0A 52 41 44 49 55 53 20 44 54 4C 53 3A 20 53"));
00219     Serial.println(F("65 6E 74 20 61 6E 20 4F 43 53 50 20 72 65 71 75 65 73 74"));
00220     Serial.println(F("20 74 6F 20 74 68 65 20 70 72 69 6D 61 72 79 20 4F 43 53"));
00221     Serial.println(F("50 20 73 65 72 76 65 72 20 66 6F 72 20 74 68 65 20 43 41"));
00222     Serial.println(F("0A 55 73 65 72 20 6F 72 20 68 6F 73 74 20 64 69 73 61 62"));
00223     Serial.println(F("6C 65 64 20 69 6E 20 63 75 72 72 65 6E 74 20 49 44 53 74"));
00224     Serial.println(F("6F 72 65 20 69 6E 20 61 74 74 72 69 62 75 74 65 20 72 65"));
00225     Serial.println(F("74 72 69 65 76 61 6C 20 6D 6F 64 65 0A 53 6B 69 70 70 69"));
00226     Serial.println(F("6E 67 20 75 6E 75 73 61 62 6C 65 20 64 6F 6D 61 69 6E 0A"));
00227     Serial.println(F("50 72 65 70 61 72 65 64 20 45 41 50 2D 52 65 71 75 65 73"));
00228     Serial.println(F("74 20 77 69 74 68 20 61 6E 6F 74 68 65 72 20 45 41 50 2D"));
00229     Serial.println(F("4D 53 43 48 41 50 20 63 68 61 6C 6C 65 6E 67 65 0A 49 64"));
00230     Serial.println(F("65 6E 74 69 74 79 20 70 6F 6C 69 63 79 20 72 65 73 75 6C"));
00231     Serial.println(F("74 20 69 73 20 63 6F 6E 66 69 67 75 72 65 64 20 66 6F 72"));
00232     Serial.println(F("20 70 61 73 73 77 6F 72 64 20 62 61 73 65 64 20 61 75 74"));
00233     Serial.println(F("68 65 6E 74 69 63 61 74 69 6F 6E 20 6D 65 74 68 6F 64 73"));
00234     Serial.println(F("20 62 75 74 20 72 65 63 65 69 76 65 64 20 63 65 72 74 69"));
00235     Serial.println(F("66 69 63 61 74 65 20 62 61 73 65 64 20 61 75 74 68 65 6E"));
00236     Serial.println(F("74 69 63 61 74 69 6F 6E 20 72 65 71 75 65 73 74 0A 46 61"));
00237     Serial.println(F("69 6C 65 64 20 74 6F 20 66 6F 72 77 61 72 64 20 72 65 71"));
00238     Serial.println(F("75 65 73 74 20 74 6F 20 63 75 72 72 65 6E 74 20 72 65 6D"));
00239     Serial.println(F("6F 74 65 20 52 41 44 49 55 53 20 73 65 72 76 65 72 3B 20"));
00240     Serial.println(F("61 6E 20 69 6E 76 61 6C 69 64 20 72 65 73 70 6F 6E 73 65"));
00241     Serial.println(F("20 77 61 73 20 72 65 63 65 69 76 65 64 0A 55 73 65 72 20"));
00242     Serial.println(F("6C 6F 67 69 6E 20 74 6F 20 49 53 45 20 63 6F 6E 66 69 67"));
00243     Serial.println(F("75 72 61 74 69 6F 6E 20 6D 6F 64 65 20 66 61 69 6C 65 64"));
00244     Serial.println(F("0A 55 6E 61 62 6C 65 20 74 6F 20 66 69 6E 64 20 27 75 73"));
00245     Serial.println(F("65 72 6E 61 6D 65 27 20 61 74 74 72 69 62 75 74 65 20 61"));
00246     Serial.println(F("73 73 65 72 74 69 6F 6E 0A 56 61 6C 69 64 20 69 6E 63 6F"));
00247     Serial.println(F("6D 69 6E 67 20 61 63 63 6F 75 6E 74 69 6E 67 20 72 65 71"));
00248     Serial.println(F("75 65 73 74 0A 41 75 74 68 65 6E 74 69 63 61 74 69 6F 6E"));
00249     Serial.println(F("20 66 61 69 6C 65 64 20 62 65 63 61 75 73 65 20 4E 54 4C"));
00250     Serial.println(F("4D 20 77 61 73 20 62 6C 6F 63 6B 65 64 0A 53 6B 69 70 70"));
00251     Serial.println(F("69 6E 67 20 75 6E 6A 6F 69 6E 65 64 20 64 6F 6D 61 69 6E"));
00252     Serial.println(F("0A 54 68 65 20 75 73 65 72 20 69 73 20 6E 6F 74 20 66 6F"));
00253     Serial.println(F("75 6E 64 20 69 6E 20 74 68 65 20 69 6E 74 65 72 6E 61 6C"));
00254     Serial.println(F("20 67 75 65 73 74 73 20 69 64 65 6E 74 69 74 79 20 73 74"));
00255     Serial.println(F("6F 72 65 0A 43 68 61 6E 67 65 20 70 61 73 73 77 6F 72 64"));
00256     Serial.println(F("20 61 67 61 69 6E 73 74 20 41 63 74 69 76 65 20 44 69 72"));
00257     Serial.println(F("65 63 74 6F 72 79 20 66 61 69 6C 65 64 20 73 69 6E 63 65"));
00258     Serial.println(F("20 75 73 65 72 20 68 61 73 20 61 20 6E 6F 6E 2D 63 6F 6D"));
00259     Serial.println(F("70 6C 69 61 6E 74 20 70 61 73 73 77 6F 72 64 0A 41 70 70"));
00260     Serial.println(F("61 72 65 6E 74 20 6D 69 73 63 6F 6E 66 69 67 75 72 61 74"));
00261     Serial.println(F("69 6F 6E 20 6F 66 20 45 78 74 65 72 6E 61 6C 20 50 6F 6C"));
00262     Serial.println(F("69 63 79 20 53 65 72 76 65 72 0A 41 75 74 68 6F 72 69 7A"));
00263     Serial.println(F("61 74 69 6F 6E 20 70 72 6F 66 69 6C 65 2F 73 20 73 70 65"));
00264     Serial.println(F("63 69 66 69 65 64 20 61 72 65 20 6E 6F 74 20 73 75 69 74"));
00265     Serial.println(F("65 64 20 66 6F 72 20 74 68 69 73 20 4E 65 74 77 6F 72 6B"));
00266     Serial.println(F("20 41 63 63 65 73 73 20 44 65 76 69 63 65 0A 52 65 63 65"));
00267     Serial.println(F("69 76 65 64 20 61 20 72 65 61 75 74 68 65 6E 74 69 63 61"));
00268     Serial.println(F("74 65 20 72 65 73 70 6F 6E 73 65 0A 4C 6F 67 67 69 6E 67"));
00269     Serial.println(F("20 63 6F 6D 70 6F 6E 65 6E 74 20 6E 6F 77 20 72 65 61 64"));
00270     Serial.println(F("79 20 74 6F 20 72 65 63 65 69 76 65 20 63 6F 6E 66 69 67"));
00271     Serial.println(F("75 72 61 74 69 6F 6E 20 63 68 61 6E 67 65 73 0A 52 65 74"));
00272     Serial.println(F("75 72 6E 65 64 20 54 41 43 41 43 53 2B 20 41 75 74 68 65"));
00273     Serial.println(F("6E 74 69 63 61 74 69 6F 6E 20 52 65 70 6C 79 0A 45 76 61"));
00274     Serial.println(F("6C 75 61 74 69 6E 67 20 47 72 6F 75 70 20 4D 61 70 70 69"));
00275     Serial.println(F("6E 67 20 50 6F 6C 69 63 79 0A 4C 44 41 50 20 66 65 74 63"));
00276     Serial.println(F("68 20 66 6F 75 6E 64 20 6E 6F 20 6D 61 74 63 68 69 6E 67"));
00277     Serial.println(F("20 61 63 63 6F 75 6E 74 20 69 6E 20 64 6F 6D 61 69 6E 0A"));
00278     Serial.println(F("4D 61 63 68 69 6E 65 20 61 75 74 68 65 6E 74 69 63 61 74"));
00279     Serial.println(F("69 6F 6E 20 61 67 61 69 6E 73 74 20 41 63 74 69 76 65 20"));
00280     Serial.println(F("44 69 72 65 63 74 6F 72 79 20 66 61 69 6C 65 64 20 73 69"));
00281     Serial.println(F("6E 63 65 20 6D 61 63 68 69 6E 65 20 69 73 20 63 6F 6E 73"));
00282     Serial.println(F("69 64 65 72 65 64 20 74 6F 20 62 65 20 69 6E 20 72 65 73"));
00283     Serial.println(F("74 72 69 63 74 65 64 20 6C 6F 67 6F 6E 20 68 6F 75 72 73"));
00284     Serial.println(F("0A 41 73 73 65 72 74 69 6F 6E 20 64 6F 65 73 20 6E 6F 74"));
00285     Serial.println(F("20 63 6F 6E 74 61 69 6E 20 73 75 62 6A 65 63 74 20 63 6F"));
00286     Serial.println(F("6E 66 69 72 6D 61 74 69 6F 6E 0A 55 73 65 72 20 72 65 63"));
00287     Serial.println(F("6F 72 64 20 77 61 73 20 63 61 63 68 65 64 20 69 6E 20 50"));
00288     Serial.println(F("61 73 73 63 6F 64 65 20 63 61 63 68 65 0A 49 64 65 6E 74"));
00289     Serial.println(F("69 74 79 20 72 65 73 6F 6C 75 74 69 6F 6E 20 62 79 20 63"));
00290     Serial.println(F("65 72 74 69 66 69 63 61 74 65 20 66 6F 75 6E 64 20 61 6D"));
00291     Serial.println(F("62 69 67 75 6F 75 73 20 61 63 63 6F 75 6E 74 73 0A 53 74"));
00292     Serial.println(F("61 72 74 75 70 20 43 6F 6D 70 6C 65 74 65 21 2E 2E 2E 2E"));
```

```
00293     return true;
00294 }
00295
00296 /*************************************************************************/
00302 /*************************************************************************/
00303 bool printStringInBytes(String str) {
00304     uint8_t messLength = str.length() + 1;
00305     /* Check if string is not too long */
00306     if (messLength > LENGTH) {
00307       return false;
00308     }
00309
00310     unsigned char messBytes[messLength];
00311     str.getBytes(messBytes, messLength);
00312     uint8_t i;
00313
00314     for (i = 0; i < messLength; i++) {
00315         if (messBytes[i] != 0) {
00316             Serial.print(messBytes[i], HEX);
00317             Serial.print(" ");
00318         }
00319     }
00320     Serial.print("0A ");
00321     i++;
00322     /* Print . (2E) until end of line, to match random data */
00323     while (i < LENGTH-1) {
00324         Serial.print("2E ");
00325         i++;
00326     }
00327     Serial.println("2E");
00328     return true;
00329 }
00330 #endif
```

5.25 hackableEspDevice/userHandler.cpp File Reference

`#include "UserHandler.h"`

Include dependency graph for userHandler.cpp:



5.26 userHandler.cpp

Go to the documentation of this file.

```
00001 /*
00002  * File:      UserHandler.cpp
00003  * Author:    Luke de Munk & Twenne Elffers
00004  * Class:     UserHandler
00005  * Version:   0.1
```

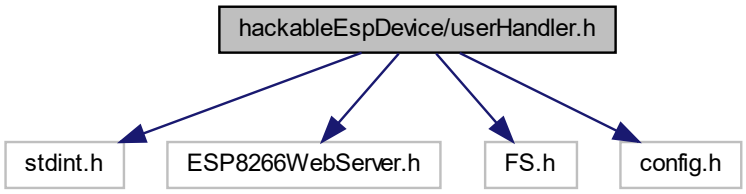
```
00006  *
00007  * Class for the http authentication process.
00008  */
00009 #include "UserHandler.h"
00010
00011 /*****************************************************************************/
00016 /*****************************************************************************/
00017 UserHandler::UserHandler(ESP8266WebServer *server) {
00018     _numberUsers = 0;
00019 }
00020
00021 /*****************************************************************************/
00025 /*****************************************************************************/
00026 void UserHandler::updateUsers() {
00027     /* If there is no file, return 0 users */
00028     if (!SPIFFS.exists(HTTP_CONFIG_LOCATION)) {
00029         _numberUsers = 0;
00030         return;
00031     }
00032
00033     File configFile = SPIFFS.open(HTTP_CONFIG_LOCATION, "r");
00034     String line;
00035     String* user;
00036
00037     /* Extract user information line by line */
00038     for(uint8_t i = 0; i < MAX_NUMBER_USERS*USER_INFO_LENGTH; i+=USER_INFO_LENGTH) {
00039         line = configFile.readStringUntil('\n');                              //Read a line from the
 file
00040         if (line != "" && line.indexOf(":") != -1) {
00041             user = _parseLine(line);
00042             _users[i] = user[0].c_str();
00043             _users[i+1] = user[1].c_str();
00044             _users[i+2] = user[2].c_str();
00045         } else {
00046             _numberUsers = i-1;
00047             break;
00048         }
00049         _numberUsers = i-1;
00050     }
00051     configFile.close();
00052 }
00053
00054 /*****************************************************************************/
00058 /*****************************************************************************/
00059 String* UserHandler::getUsers() {
00060     return _users;
00061 }
00062
00063 /*****************************************************************************/
00067 /*****************************************************************************/
00068 uint8_t UserHandler::getNumberOfUsers() {
00069     return _numberUsers;
00070 }
00071
00072 /*****************************************************************************/
00079 /*****************************************************************************/
00080 bool UserHandler::checkPermission(uint8_t permissionLevel, ESP8266WebServer *server) {
00081     bool isLoggedIn = false;
00082     bool hasPermission = false;
00083     uint8_t userIndex = 0;
00084
00085     if (permissionLevel == PERMISSION_LVL_ALL) {
00086         return true;
00087     } else {
00088         for (uint8_t i = 0; i < _numberUsers; i += 3) {
00089             if (server->authenticate(_users[i].c_str(), _users[i+1].c_str())) {
00090                 userIndex = i;
00091                 isLoggedIn = true;
00092                 break;
00093             }
00094         }
00095
00096         if (isLoggedIn && atoi(_users[userIndex+2].c_str()) >= permissionLevel) {
00097             return true;
00098         }
00099     }
00100     return false;
00101 }
00102
00103 /*****************************************************************************/
00109 /*****************************************************************************/
00110 String* UserHandler::_parseLine(String line) {
00111     static String userInfo[3];
00112
00113     uint8_t indexForUsername = line.indexOf(":");                             //gets loc of first ":"
00114     uint8_t indexForPassword = line.indexOf(":", indexForUsername+1);         //gets loc of second ":"
00115
```

```
00116     userInfo[0] = line.substring(0, indexForUsername);                    //Selects xxxx from
      xxxx:yyyy:zzzz, username
00117     userInfo[1] = line.substring(indexForUsername+1, indexForPassword);   //Selects yyyy from
      xxxx:yyyy:zzzz, password
00118     userInfo[2] = line.substring(indexForPassword+1);                     //Selects zzzz from
      xxxx:yyyy:zzzz, usertype
00119     return userInfo;
00120 }
```

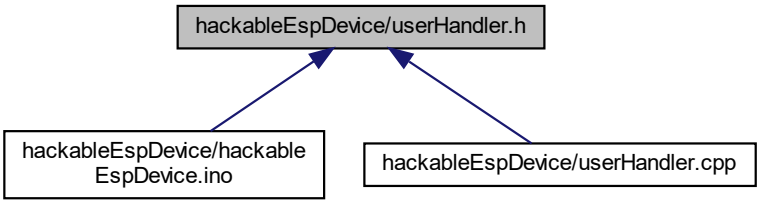
5.27 hackableEspDevice/userHandler.h File Reference

```
#include <stdint.h>
#include <ESP8266WebServer.h>
#include <FS.h>
#include "config.h"
```

Include dependency graph for userHandler.h:



This graph shows which files directly or indirectly include this file:



Classes

- class UserHandler

5.28 userHandler.h

Go to the documentation of this file.

```
00001 /*
00002  * File:      UserHandler.h
00003  * Author:    Luke de Munk
00004  * Class:     UserHandler
00005  * Version:   0.1
00006  *
00007  * Class for the http authentication process.
00008  */
00009 #ifndef USER_HANDLER_H
00010 #define USER_HANDLER_H
00011 #include <stdint.h>                                          //For defining bits per
       integer
00012 #include <ESP8266WebServer.h>                                //For running the
       webserver
00013 #include <FS.h>                                              //For SPIFFS
00014 #include "config.h"                                          //For the configuration
00015
00016 class UserHandler
00017 {
00018     public:
00019         UserHandler(ESP8266WebServer *server);
00020         void updateUsers();
00021         String* getUsers();
00022         uint8_t getNumberOfUsers();
00023         bool checkPermission(uint8_t permissionLevel, ESP8266WebServer *server);
00024
00025     private:
00026         String* _parseLine(String line);
00027         String _users[MAX_NUMBER_USERS*USER_INFO_LENGTH];
00028         uint8_t _numberUsers;
00029 };
00030 #endif
```

5.29 README.md File Reference

Index