Universidade de Coimbra
Faculdade de Ciências e Tecnologias da Universidade de Coimbra
Mestrado em Segurança Informática

Trabalho 3 – Promiscuous Mode/MQTT

Introdução:

Este trabalho foi realizado âmbito da cadeira de Sistemas de Comunicação Móvel e foi dividido em duas etapas: uma pequena implementação de promiscuous mode para a captura de pacotes em vários canais e posteriormente uma integração de trackers usando MQTT.

Todo este trabalho foi realizado utilizando arduinos esp8266.

Meta 1 – *Promiscuous Mode*

O que é o *Promiscuos Mode*?

Antes de iniciar qualquer explicação de implementação ou responder a alguma pergunta é importante entender o que é o *Promiscuous Mode*. Este é nem mais nem menos do que um modo na interface de rede que permite a mesma visualizar todos os pacotes (identificados normalmente por um *MAC Address*) que passem pelo tráfego da rede. O *Promiscuous Mode* pode ser utilizado tanto para efetuar ações maliciosas como para auxiliar na execução de tarefas importantes.

Quando utilizado de forma sensata permite auxiliar a monitorizar o tráfego na rede, ajudando, por vezes, em questões que envolvam segurança, mas ainda assim pode ser usado inapropriadamente. Através deste modo, qualquer utilizador consegue aceder aos pacotes que são transmitidos, podendo eventualmente ocasionar ataques de *sniffing*. Ora, isto torna possível a interseção ou roubo de informação, caso os pacotes não estejam devidamente encriptados.

Podemos, assim, constatar que o seu uso é algo um pouco controverso, já que o *Promiscuous Mode* não permite garantir segurança suficiente nas redes.

Detalhes de implementação

A implementação realizada parte de código já existente, onde cada tipo de pacote é representado por uma *struct* distinta, visto conterem sempre informação diferente comparativamente com os restantes. O código base já atendia à exigência principal, ou seja, já conseguia detetar os pacotes na rede, mas para completar o mesmo com a informação pedida no enunciado foram efetuadas ligeiras alterações.

Inicialmente, só era detetado tráfico de rede num único canal definido, após efetuadas as alterações necessárias para o efeito pretendido passou-se a executar essa mesma tarefa só que em todos os canais. Para que isto fosse possível, na função *loop* foi adicionado código de modo a que o canal a analisar vá alterando de 10 em 10 segundos.

Era pedido também que, caso um dos dispositivos aparecesse na lista de pacotes, o *LED* do Arduino acendesse por 2 segundos. Neste sentido foram implementados dois *ifs* no código que refletissem esta funcionalidade perante a presença de um *MAC address* igual ao predefinido. Assim, teremos um if na função *wifi_sniffer_packet_handler*, onde, caso encontremos um pacote com destino ao *MAC address* predefinido, uma variável global é alterada. Na função loop há outro *if* para verificar qual é o valor desta mesma variável, se for igual 1 é sinal que foi encontrado o pacote que pretendiamos. Consequentemente, é dado print dessa informação no serial monitor e acendido o *LED* do arduino durante 2 segundos.

Possíveis soluções para não ser *tracked*

Como foi visto, métodos de monitorização, como o *Promiscuous mode*, podem trazer associados alguns riscos aos dispositivos na rede, sendo uma das principais preocupações a possibilidade de ser efetuada *tracking* aos mesmo. Focando a atenção em dispositivos móveis é possível pensar em algumas soluções que evitem que estes sejam *tracked*:

João Carlos Borges Silva nº2019216753
Inês Martins Marçal nº2019215917

- Desligar *wifi*: durante o dia quando os nossos dispositivos estão conectados à Internet, estes efetuam *broadcast* do *MAC address* dos mesmos para toda a gente na rede. Assim, ligar o *wifi* só quando é necessário não eliminará por completo a existência deste risco, mas pode reduzir a probabilidade de isso acontecer.
- *MAC spoofing:* esta técnica consiste basicamente na falsificação do *MAC address* do dispositivo, não permitindo que quem esteja a intercetar pacotes descubra a máquina destino. Apesar de parecer uma ideia promissora ao início, esta técnica poderá trazer algumas implicações, caso mais tarde nos queiramos conectar, por exemplo, a uma rede que tenha uma whitelist de *MAC addresses*.
- *MAC randomization:* é gerado um *MAC address random* de x em x tempo, dificultando assim o *tracking* do mesmo e assegurando, ao mesmo tempo, uma maior privacidade do dispositivo móvel. Assim que é efetuada uma ligação a uma rede *wifi,* este processo é interrompido para manter uma ligação estável.

Como *Android* e *Apple iOS* lidam com esta situação

Tanto o *Android* como a *Apple* optaram por implementar a solução de *MAC randomization*. A *Apple* implementou este método em 2014 de modo a que a sua ativação já seja automática. Já no *Android*, esta opção já existe desde o *Android 5*, mas só a partir do *Android* 10 é que começou a vir ativa por *default*. Ainda assim, os *users* têm sempre a opção de desativar ou não esta opção.

Código do main.ino:

```
#include <Arduino.h>
#include <ESP8266WiFi.h>
#include "sdk_structs.h"
#include "ieee80211_structs.h"
#include "string_utils.h"

int chn = 1, prm = 0;

extern "C"
{
  #include "user_interface.h"
}

// According to the SDK documentation, the packet type can be inferred from the
// size of the buffer. We are ignoring this information and parsing the type-subtype
// from the packet header itself. Still, this is here for reference.
wifi_promiscuous_pkt_type_t packet_type_parser(uint16_t len)
{
    switch(len)
    {
      // If only rx_ctrl is returned, this is an unsupported packet
      case sizeof(wifi_pkt_rx_ctrl_t):
      return WIFI_PKT_MISC;

      // Management packet
      case sizeof(wifi_pkt_mgmt_t):
      return WIFI_PKT_MGMT;

      // Data packet
      default:
      return WIFI_PKT_DATA;
    }
}

// In this example, the packet handler function does all the parsing and output work.
// This is NOT ideal.
void wifi_sniffer_packet_handler(uint8_t *buff, uint16_t len)
{
  // First layer: type cast the received buffer into our generic SDK structure
  const wifi_promiscuous_pkt_t *ppkt = (wifi_promiscuous_pkt_t *)buff;
  // Second layer: define pointer to where the actual 802.11 packet is within the structure
  const wifi_ieee80211_packet_t *ipkt = (wifi_ieee80211_packet_t *)ppkt->payload;
  // Third layer: define pointers to the 802.11 packet header and payload
  const wifi_ieee80211_mac_hdr_t *hdr = &ipkt->hdr;
  const uint8_t *data = ipkt->payload;
```

```
  // Pointer to the frame control section within the packet header
  const wifi_header_frame_control_t *frame_ctrl = (wifi_header_frame_control_t *)&hdr->frame_ctrl;

  // Parse MAC addresses contained in packet header into human-readable strings
  char addr1[] = "00:00:00:00:00:00\0";
  char addr2[] = "00:00:00:00:00:00\0";
  char addr3[] = "00:00:00:00:00:00\0";

  mac2str(hdr->addr1, addr1);
  mac2str(hdr->addr2, addr2);
  mac2str(hdr->addr3, addr3);

  //mac address a encontrar
  char mymacadd[] = "36:b5:82:47:17:24\0";

  //se um dos mac addresses pertencentes ao pacote for o pretendido
  if(!strcmp(addr1, mymacadd) || !strcmp(addr2, mymacadd) || !strcmp(addr3, mymacadd)){
    prm = 1;
  }


  // Print ESSID if beacon
  if (frame_ctrl->type == WIFI_PKT_MGMT && frame_ctrl->subtype == BEACON)
  {
    const wifi_mgmt_beacon_t *beacon_frame = (wifi_mgmt_beacon_t*) ipkt->payload;
    char ssid[32] = {0};

    if (beacon_frame->tag_length >= 32)
    {
      strncpy(ssid, beacon_frame->ssid, 31);
    }
    else
    {
      strncpy(ssid, beacon_frame->ssid, beacon_frame->tag_length);
    }

    //Serial.printf("%s", ssid);
  }
}


void setup()
{
  pinMode(LED_BUILTIN, OUTPUT);
  digitalWrite(LED_BUILTIN, 1);
```

```
  // Serial setup
  Serial.begin(9600);
  delay(10000);
  wifi_set_channel(1);

  // Wifi setup
  wifi_set_opmode(STATION_MODE);
  wifi_promiscuous_enable(0);
  WiFi.disconnect();

  // Set sniffer callback
  wifi_set_promiscuous_rx_cb(wifi_sniffer_packet_handler);
  wifi_promiscuous_enable(1);
}

void loop()
{
  chn = chn % 13 + 1;
  wifi_set_channel(chn);

  //se a variável global estiver ativa (ou seja, encontrou-se o pacote pretendido)
  if(prm == 1){
    digitalWrite(LED_BUILTIN, 0);
    Serial.println("PACKET FOUND: LED ON");
    delay(2000);
    digitalWrite(LED_BUILTIN, 1);
  }

  prm = 0;

  delay(10000);
}
```

Código de ieee80211_structs.h

```
#ifndef _IEE80211_STRUCTS_H_
#define _IEE80211_STRUCTS_H_

#include <ESP8266WiFi.h>

// IEEE802.11 data structures ----------------------

typedef enum
{
    WIFI_PKT_MGMT,
    WIFI_PKT_CTRL,
    WIFI_PKT_DATA,
    WIFI_PKT_MISC,
} wifi_promiscuous_pkt_type_t;

typedef enum
{
    ASSOCIATION_REQ,
    ASSOCIATION_RES,
    REASSOCIATION_REQ,
    REASSOCIATION_RES,
    PROBE_REQ,
    PROBE_RES,
    NU1,  /* ......................*/
    NU2,  /* 0110, 0111 not used */
    BEACON,
    ATIM,
    DISASSOCIATION,
    AUTHENTICATION,
    DEAUTHENTICATION,
    ACTION,
    ACTION_NACK,
} wifi_mgmt_subtypes_t;

typedef struct
{
  unsigned interval:16;
  unsigned capability:16;
  unsigned tag_number:8;
  unsigned tag_length:8;
  char ssid[0];
  uint8 rates[1];
} wifi_mgmt_beacon_t;
```

```
typedef struct
{
    unsigned protocol:2;
    unsigned type:2;
    unsigned subtype:4;
    unsigned to_ds:1;
    unsigned from_ds:1;
    unsigned more_frag:1;
    unsigned retry:1;
    unsigned pwr_mgmt:1;
    unsigned more_data:1;
    unsigned wep:1;
    unsigned strict:1;
} wifi_header_frame_control_t;

/**
 * Ref: https://github.com/lpodkalicki/blog/blob/master/esp32/016_wifi_sniffer/main/main.c
 */
typedef struct
{
    wifi_header_frame_control_t frame_ctrl;
    //unsigned duration_id:16; /* !!!! ugly hack */
    uint8_t addr1[6]; /* receiver address */
    uint8_t addr2[6]; /* sender address */
    uint8_t addr3[6]; /* filtering address */
    unsigned sequence_ctrl:16;
    uint8_t addr4[6]; /* optional */
} wifi_ieee80211_mac_hdr_t;

typedef struct
{
    wifi_ieee80211_mac_hdr_t hdr;
    uint8_t payload[2]; /* network data ended with 4 bytes csum (CRC32) */
} wifi_ieee80211_packet_t;

#endif
```

Código de sdk_structs.h

```
#ifndef _SDK_STRUCTS_H_
#define _SDK_STRUCTS_H_

#include <ESP8266WiFi.h>

// SDK structures ----------------------------------

typedef struct
{
    signed rssi:8;            /**< signal intensity of packet */
    unsigned rate:4;          /**< data rate */
    unsigned is_group:1;
    unsigned :1;              /**< reserve */
    unsigned sig_mode:2;      /**< 0:is not 11n packet; 1:is 11n packet */
    unsigned legacy_length:12;
    unsigned damatch0:1;
    unsigned damatch1:1;
    unsigned bssidmatch0:1;
    unsigned bssidmatch1:1;
    unsigned mcs:7;           /**< if is 11n packet, shows the modulation(range from 0 to 76) */
    unsigned cwb:1;           /**< if is 11n packet, shows if is HT40 packet or not */
    unsigned HT_length:16;          /**< reserve */
    unsigned smoothing:1;     /**< reserve */
    unsigned not_sounding:1;  /**< reserve */
    unsigned :1;              /**< reserve */
    unsigned aggregation:1;   /**< Aggregation */
    unsigned stbc:2;          /**< STBC */
    unsigned fec_coding:1;    /**< Flag is set for 11n packets which are LDPC */
    unsigned sgi:1;           /**< SGI */
    unsigned rxend_state:8;
    unsigned ampdu_cnt:8;     /**< ampdu cnt */
    unsigned channel:4;       /**< which channel this packet in */
    unsigned :4;              /**< reserve */
    signed noise_floor:8;
} wifi_pkt_rx_ctrl_t;

typedef struct {
    wifi_pkt_rx_ctrl_t rx_ctrl;
    uint8 buf[112];
    uint16 cnt;
    uint16 len; //length of packet
} wifi_pkt_mgmt_t;

 typedef struct {
   uint16 length;
   uint16 seq;
   uint8  address3[6];
 } wifi_pkt_lenseq_t;

 typedef struct {
     wifi_pkt_rx_ctrl_t rx_ctrl;
     uint8_t  buf[36];
     uint16_t cnt;
     wifi_pkt_lenseq_t lenseq[1];
 } wifi_pkt_data_t;

 typedef struct {
     wifi_pkt_rx_ctrl_t rx_ctrl; /**< metadata header */
     uint8_t payload[0];       /**< Data or management payload. Length of payload is described by rx_ctrl.sig_len.
                                    Type of content determined by packet type argument of callback. */
 } wifi_promiscuous_pkt_t;
```

Código de string_utils.cpp

```
#include "sdk_structs.h"
#include "ieee80211_structs.h"

// Uncomment to enable MAC address masking
//#define MASKED

//Returns a human-readable string from a binary MAC address.
//If MASKED is defined, it masks the output with XX
void mac2str(const uint8_t* ptr, char* string)
{
  #ifdef MASKED
  sprintf(string, "XX:XX:XX:%02x:%02x:XX", ptr[0], ptr[1], ptr[2], ptr[3], ptr[4], ptr[5]);
  #else
  sprintf(string, "%02x:%02x:%02x:%02x:%02x:%02x", ptr[0], ptr[1], ptr[2], ptr[3], ptr[4], ptr[5]);
  #endif
  return;
}

//Parses 802.11 packet type-subtype pair into a human-readable string
const char* wifi_pkt_type2str(wifi_promiscuous_pkt_type_t type, wifi_mgmt_subtypes_t subtype)
{
  switch(type)
  {
    case WIFI_PKT_MGMT:
      switch(subtype)
      {
         case ASSOCIATION_REQ:
         return "Mgmt: Association request";
         case ASSOCIATION_RES:
         return "Mgmt: Association response";
         case REASSOCIATION_REQ:
         return "Mgmt: Reassociation request";
         case REASSOCIATION_RES:
         return "Mgmt: Reassociation response";
         case PROBE_REQ:
         return "Mgmt: Probe request";
         case PROBE_RES:
         return "Mgmt: Probe response";
         case BEACON:
         return "Mgmt: Beacon frame";
         case ATIM:
         return "Mgmt: ATIM";
         case DISASSOCIATION:
         return "Mgmt: Dissasociation";
         case AUTHENTICATION:
         return "Mgmt: Authentication";
         case DEAUTHENTICATION:
         return "Mgmt: Deauthentication";
         case ACTION:
         return "Mgmt: Action";
         case ACTION_NACK:
         return "Mgmt: Action no ack";
      default:
        return "Mgmt: Unsupported/error";
      }

    case WIFI_PKT_CTRL:
    return "Control";

    case WIFI_PKT_DATA:
    return "Data";

    default:
      return "Unsupported/error";
  }
}
```

Código de string_utils.h

```
#ifndef _STRING_UTILS_H_
#define _STRING_UTILS_H_

#include "sdk_structs.h"
#include "ieee80211_structs.h"

void mac2str(const uint8_t* ptr, char* string);
const char* wifi_pkt_type2str(wifi_promiscuous_pkt_type_t type, wifi_mgmt_subtypes_t subtype);

#endif
```

Meta 2 – *MQTT Client*

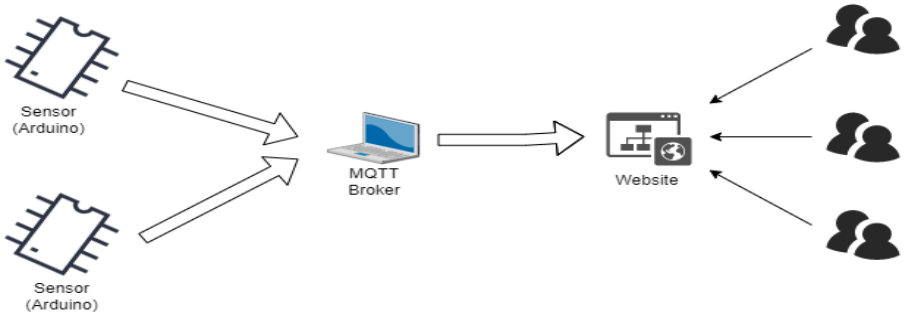
Modo de funcionamento do sistema

Tendo em conta o que foi implementado na meta 1, o desafio neste momento passaria por utilizar um broker MQTT com o intuito de transmitir a informação dos pacotes encontrados para uma aplicação web. Deste modo, teremos dois clientes MQTT (ou mais, dependendo do número de arduinos que analisam pacotes na rede):

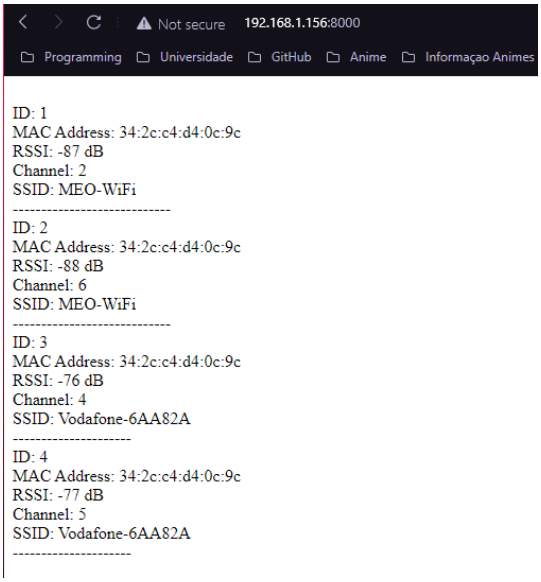
- Arduino: será responsável por receber a lista de MAC addresses e analisar pacotes referentes aos mesmos. A informação recolhida é mais tarde enviada para a aplicação web.
- Aplicação web: é a partir dela que serão enviados os MAC addresses para os arduinos e, posteriormente, recebida a informação proveniente dos mesmos. Os dados recebidos serão apresentados numa página no browser.

Para que todos os clientes possam comunicar foram definidos tópicos. Assim, criou-se um tópico específico para que possam ser transmitidos os MAC addresses, designado o mesmo como “macadd/to_found_devices”. Todos os arduinos subscrevem no início da sua execução a este tópico e aguardam para ler o conteúdo proveniente do mesmo.

Após este processo inicial, quando o arduino encontrar um pacote que tenha como referência um dos MAC addresses na lista, será ligado o LED durante alguns segundos e publicada essa informação no tópico macadd/found_devices/[número a definir no código]. É de notar que durante este processo o modo promíscuo deverá se encontrar desativado, sendo necessário reativar o wifi momentaneamente e efetuar uma conexão com o broker, através do qual se irá transmitir a informação. Posteriormente, interrompe-se a ligação com broker, desativando o wifi e ligando novamente o modo promíscuo. É necessário efetuar este procedimento, pois o modo promíscuo só funciona se o wifi estiver desativado, mas o MQTT só funciona se o arduino estiver conectado à mesma rede wifi do broker, tendo em conta que o arduino não precisa de estar 100% do tempo ligado ao broker. A informação a ser transmitida será: o id do sensor, o id do pacote (servirá para distinguir cada mensagem), MAC address encontrado, RSSI, o canal onde se encontra este pacote e finalmente o SSID (se se aplicável).



Do lado da aplicação web vai ser lida toda a informação que esteja nos tópicos dos arduino utilizando a *wildcard* asterisco. A informação será, posteriormente, atualizada no website (que estará alocado num ip e porta especificados pelo utilizador no código).



Detalhes de implementação

Arduino: Partindo do código já utilizado na meta 1 foram feitas algumas alterações de modo a permitir comunicações MQTT. Como já referenciado no modo de funcionamento, é necessário que todos os arduínos começem por receber os MAC addresses aos quais vão estar em alerta. Com vista a que isto fosse possível foi implementado um while(true) que só termina caso os arduínos consigam ler algo no tópico “macadd/to_found_devices”.

Foi criada também uma função chamada MQTT_connect() que, por sua vez, permitirá iniciar uma ligação ao wifi e ao broker MQTT, sendo chamada em todos os pontos em que os arduinos necessitam de comunicar com o broker, seja para subscrever e ler o tópico anteriormente referenciado ou para publicar informação.

Finalmente, foi implementada uma função que visa tratar o input recebido no tópico “macadd/to_found_devices”, já que a forma como são recebidos os MAC addresses é a seguinte: “[número de MAC addresses a ler];[MAC address 1];[MAC address 2];...”.

Aplicação web: Como já referido, procedeu-se à criação de uma aplicação web cujo o código efetuado para implementar a mesma foi desenvolvido em Python. Nesta temos uma parte relativa ao cliente MQTT e outra que permite dar host ao website, mostrando, assim, toda a informação recebida nos tópicos através dos dispositivos/pacotes encontrados.

A aplicação começa por pedir um input ao cliente que, por sua vez, contem os MAC addresses que os arduinos deverão analisar. O pedido de input irá terminar caso o utilizador insira a string “STOP”. Por outro lado, quaisquer MAC addresses inválidos inseridos serão detetados e não serão aceites.

Visto que é preciso estar constantemente subscrito aos tópicos (usando os comandos client.subscribe(“tópico”) e client.loop_forever()) e ao mesmo tempo ter uma parte do programa a dar host ao website, foram criadas 2 threads para que fosse possível a realização destes processos distintos. Enquanto isto, no processo principal do programa teremos um while(true) que terá como função passar as mensagens provenientes da classe do cliente MQTT para a classe do website, tornando possível que a informação seja atualizada no mesmo.

Com vista a encerrar o programa de forma correta é feito o *handle* do ctrl+c, que, por sua vez, ao ser pressionado irá desconectar o cliente MQTT do broker (terminando, assim, o seu loop interno) e encerrar o website, concluindo, por fim, a execução do programa.

Bibliotecas e código utilizado:

- Arduino: Arduino, ESP8266WiFI, Adafruit_MQTT, Adafruit_MQTT_Client, além do código indicado nas referências e do que era fornecido como exemplo na biblioteca Adafruit_MQTT.
- Python: signal, time, sys, random, threading, regex e http.server

Referências

- https://carvesystems.com/news/writing-a-simple-esp8266-based-sniffer/~
- https://github.com/n0w/esp8266-simple-sniffer
- https://github.com/adafruit/Adafruit_MQTT_Library
- https://en.wikipedia.org/wiki/Promiscuous_mode
- https://nordvpn.com/blog/mac-address/
- https://source.android.com/docs/core/connect/wifi-mac-randomization
- https://www.extremenetworks.com/extreme-networks-blog/wi-fi-mac-randomization-privacy-and-collateral-damage/