HITB+CyberWeek 2021
Brought to you by DisruptAD

WIFI Security - From 0 To 1

sili luo | @hac425

Huawei RO0T Lab

About Me

- Security Researcher at Huawei RO0T Lab
- Focus on software vulnerability research
- Github: https://github.com/hac425xxx
- Twitter: https://twitter.com/hac425/
- Blog: https://www.cnblogs.com/hac425
- Speaker at HITB2021Sin, BlackHat USA Arsenal 2021

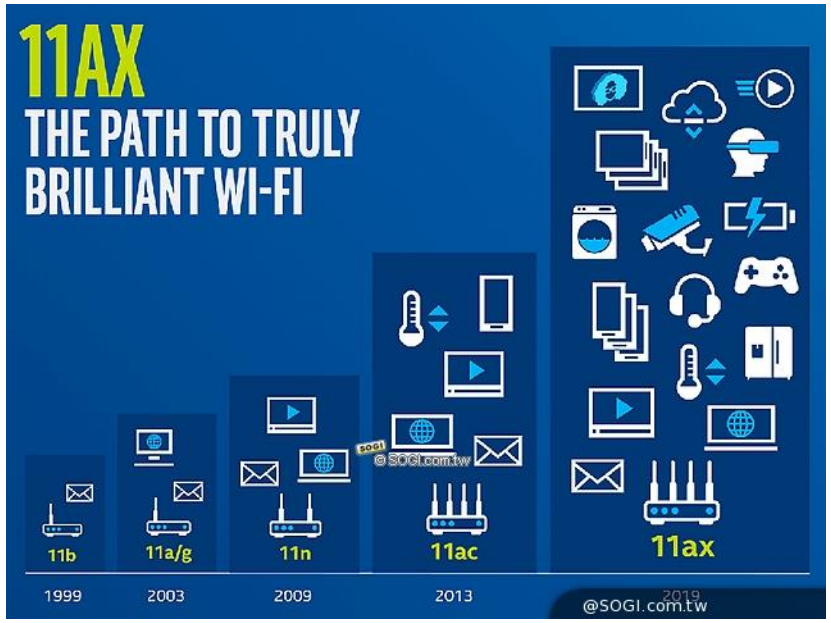
Outline

1. Background
2. WIFI Code Review
3. The Implementation of wifi-hunter
4. Local Attack Surface of Linux WIFI Driver
5. Conclusion

Background

Why Attack WIFI

1. WIFI is more and more widely used, such as mobile phone, IOT, smart car
2. The WIFI protocol is more complicated and prone to vulnerability
3. 0click attack, once the attack is successful, user data can be touch



Related Research

Over The Air: Exploiting Broadcom's Wi-Fi Stack	2017	Gal Beniamini	RSN IE overflow	Exploit
Broadpwn	2017	Nitay Artenstein	WMM IE overflow	
An iOS zero-click radio proximity exploit odyssey	2020	Ian Beer	AWDL TLV overflow	
Remote and Local Exploitation of Network Drivers	2007	Yuriy Bulygin	Fuzz Beacon Frame	Fuzz
WIFI-Important Remote Attack Surface	2020	Xie Haikuo	Introduce the implementation of WIFI Fuzz	
GREYHOUND: Directed Greybox Wi-Fi Fuzzing	2020	Matheus E. Garbelini	Stateful WIFI Fuzz	
OWFuzz: WiFi Protocol Fuzzing Tool Based on OpenWiFi	**2021**	**Hongjian Cao**	**fuzzer based on openwifi, open source!**	

WIFI Attack Surface

Remote Attack Surface

1. Various TLV structure, such as IE, EAPOL, P2P attribute, vendor customized structure.
2. Fragmentation and Aggregation of data frame

Local Attack Surface of Linux WIFI Driver

1. ioctl interface/**wext**
2. nl80211 interface/**cfg80211**

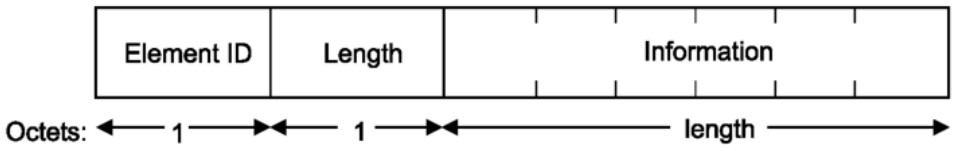


Figure 7-37—Element format

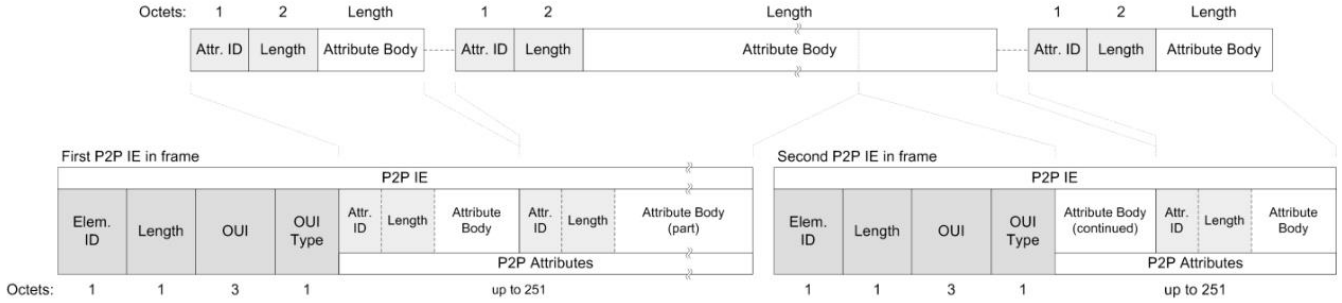
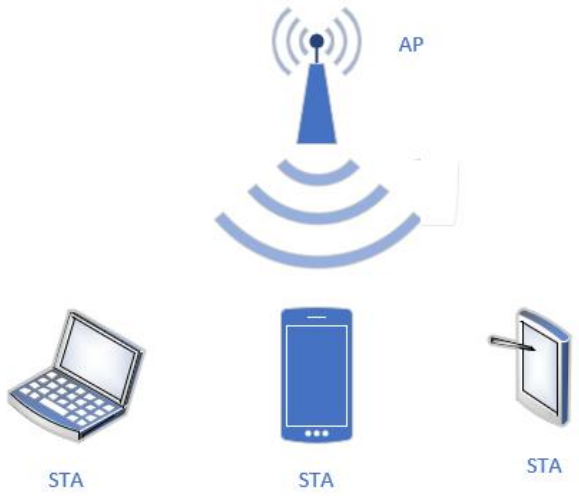


Figure 19—Example of P2P attributes carried in two P2P IEs

WIFI Protocol Basics



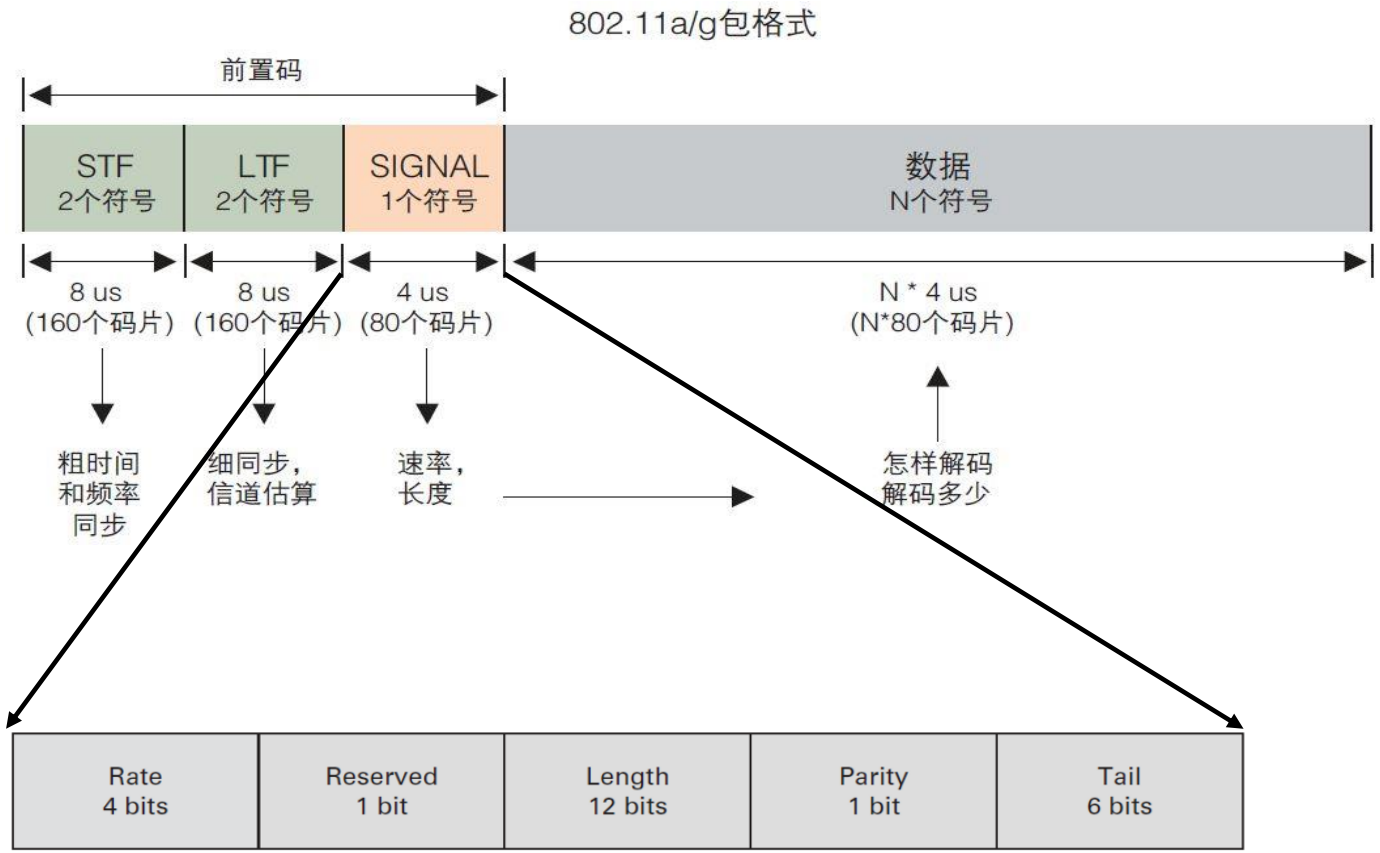
WIFI network architecture

datalink layer (OSI layer 2)	logical link control (LLC)	IEEE 802.2
	medium access control (MAC)	IEEE 802.11 (MAC & PLCP)
physical layer (OSI layer 1)	physical layer convergence protocol (PLCP)	
	physical protocol layer (PHY)	**IEEE 802.11** FH-PHY / **IEEE 802.11** DS-PHY / **IEEE 802.11** IR-PHY

WIFI Protocol Stack Layering

WIFI protocol basics - the physical frame format

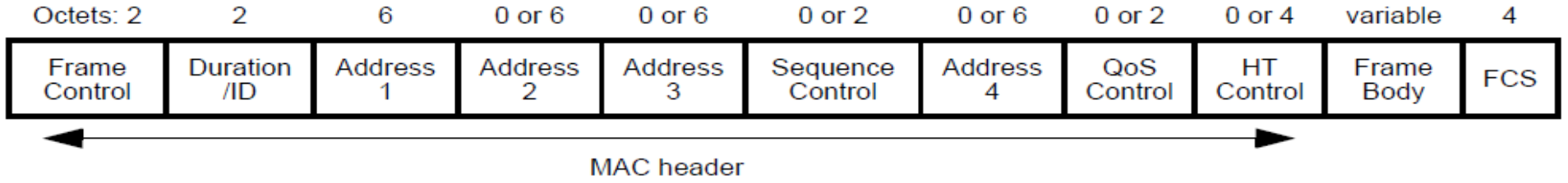
PLCP header



the Length field is to identify the length of the data (MAC layer data)

WIFI protocol basics - MAC frame format

A MAC frame is composed of a frame header and a frame body, and the type of frame is determined according to the information in the frame header

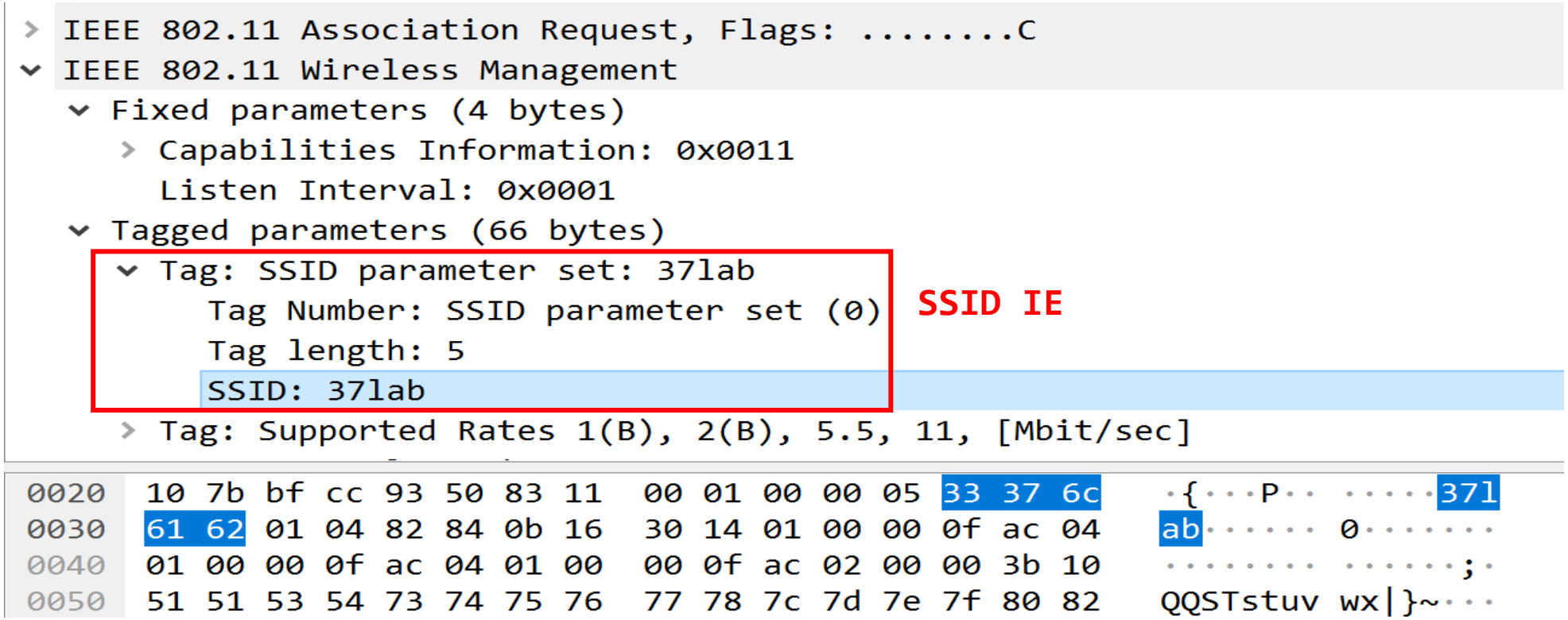



```
IEEE 802.11 Beacon frame, Flags: ........
    Type/Subtype: Beacon frame (0x0008)
  > Frame Control Field: 0x8000
    .000 0000 0000 0000 = Duration: 0 microseconds
    Receiver address: Broadcast (ff:ff:ff:ff:ff:ff)
    Destination address: Broadcast (ff:ff:ff:ff:ff:ff)
    Transmitter address: 08:10:7b:bf:cc:93 (08:10:7b:bf:cc:93)
    Source address: 08:10:7b:bf:cc:93 (08:10:7b:bf:cc:93)
    BSS Id: 08:10:7b:bf:cc:93 (08:10:7b:bf:cc:93)
    .... .... .... 0000 = Fragment number: 0
    0000 1100 1100 .... = Sequence number: 204
```

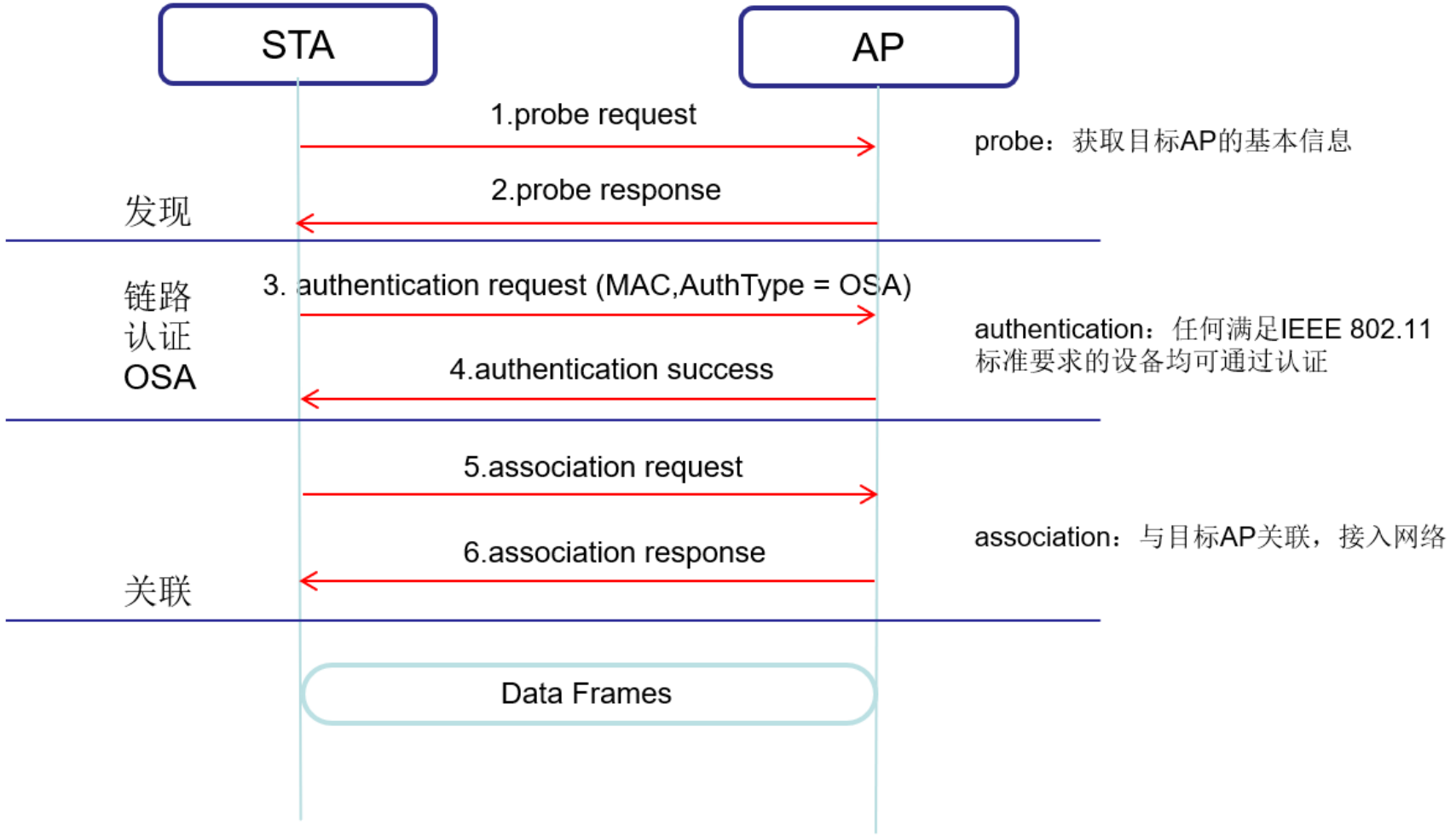
```
0000  80 00 00 00 ff ff ff ff  ff ff 08 10 7b bf cc 93   ·....... ....{...
0010  08 10 7b bf cc 93 c0 0c  32 00 96 00 00 00 00 00   ··{····· 2·......
0020  64 00 31 14 00 0a 38 38  31 32 61 75 2d 77 69 6e   d·1···88 12au-win
```

WIFI protocol basics - MAC frame format

The frame body of the management frame consists of some fixed-length fields and **variable-length IE (Information Element)**

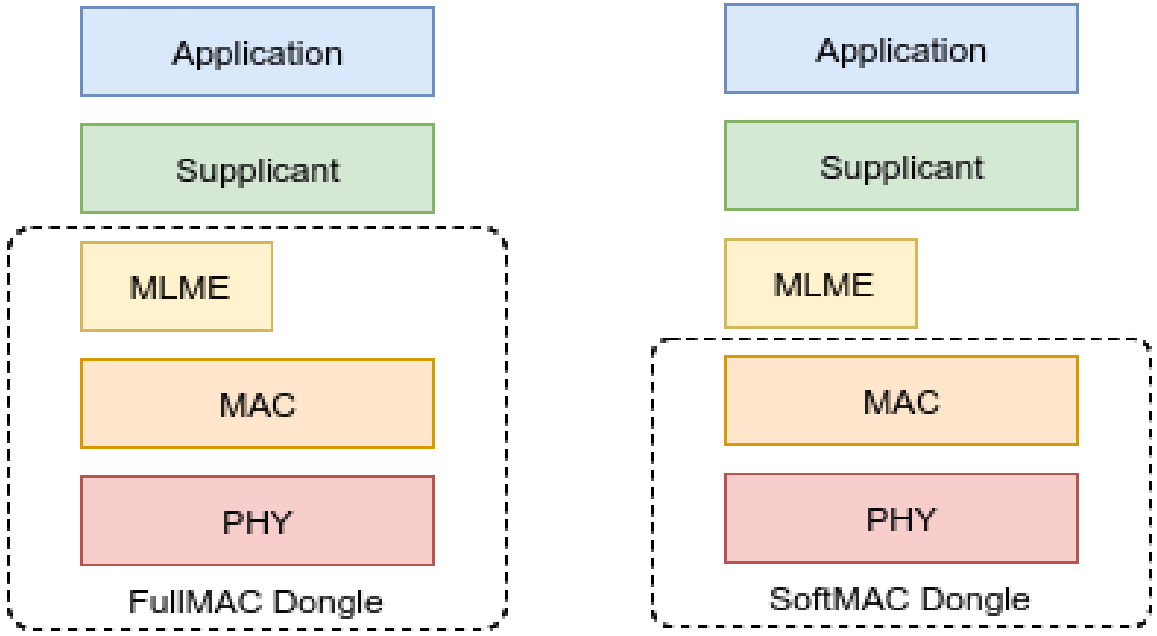


WIFI protocol basics - STA/AP interaction

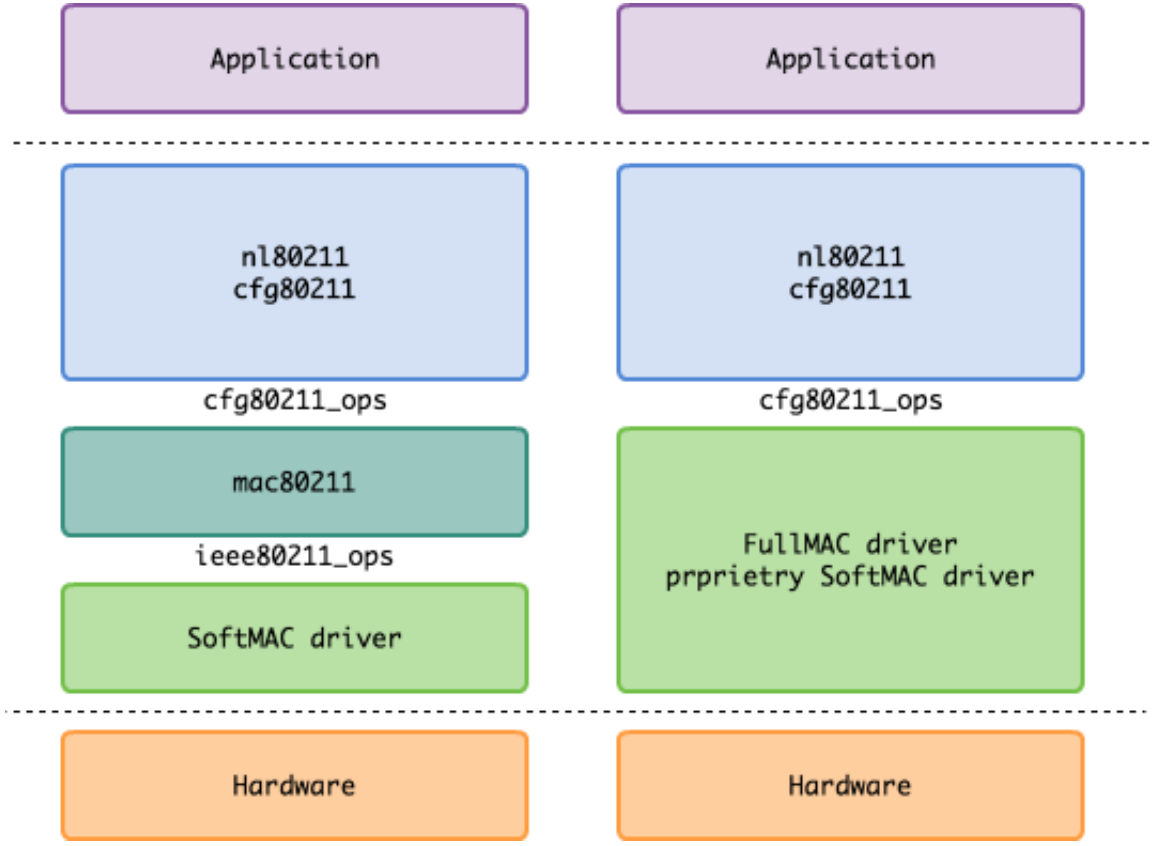


WIFI Code Review

Architecture of WIFI Protocol Stack



Two types of WIFI Card



Two types of Linux SoftMAC driver

WIFI Code Review

"Manual Data Flow Tracking"

1. Locate data parsing logic and data source
    1. Search Keyword, such as probe, beacon, parse, etc.
    2. WIFI Protocol Specification and the data processing logic in the code.
    3. History vulnerabilities.
2. Tracking data sources and discovering vulnerabilities
    1. Focus on the processing of variable-length data, such as IE, EAPOL structure etc.
    2. Numerical operations, integer overflow, etc.

WIFI Code Review - RTL8195A SDK

In February 2021, VDOO disclosed several RTL8195A's vulnerabilities, most vulnerabilities are caused by parsing various TLV structures.

Security Vulnerabilities

Major Vulnerabilities discovered and patched in Realtek RTL8195A Wi-Fi Module

Uriya Yavniely February 3, 2021

The above functions both call the CheckMIC() function, which is responsible for checking the integrity of the MIC part in the EAP packet.

In **CheckMIC()** an unsafe copy can be triggered:

```
rtl_memcpy(tmpbuf, EAPOLMsgRecvd.Octet, EAPOLMsgRecvd.Length);
```

from: decompilation of **CheckMIC()** function in lib_wlan.a

WIFI Code Review - RTL8195A SDK

RTL8195A is a FullMAC wireless network card. The WIFI protocol stack is located in the static library of the SDK. Users develop upper-level applications based on the SDK, such as HTTP services.

E:\WIFI\rtl8195a-reverse-2.0.10-v3\ameba_1-2.0.10-v3.tar.gz\ameba_1-2.0.10-v3.tar\hardware\variants\rtl8195a\

名称	压缩后大小	修改时间	模式	大小	用户	组	链接
linker_scripts	52 224	2021-07-1...	drwx------	50 930	zhangzhe...	RTSGDom...	
lib_ameba.a	7 524 864	2021-07-1...	-rwx------	7 524 616	zhangzhe...	RTSGDom...	
lib_usbh.a	2 722 816	2021-07-1...	-rwx------	2 722 554	zhangzhe...	RTSGDom...	
lib_usbd.a	896 000	2021-07-1...	-rwx------	895 898	zhangzhe...	RTSGDom...	
lib_p2p.a	732 160	2021-07-1...	-rwx------	731 696	zhangzhe...	RTSGDom...	
lib_wlan.a	667 136	2021-07-1...	-rwx------	666 934	zhangzhe...	RTSGDom...	
lib_arduino_alexa.a	542 720	2021-07-1...	-rwx------	542 264	zhangzhe...	RTSGDom...	
lib_mdns.a	511 488	2021-07-1...	-rwx------	511 294	zhangzhe...	RTSGDom...	
lib_wps.a	419 840	2021-07-1...	-rwx------	419 578	zhangzhe...	RTSGDom...	
lib_platform.a	381 440	2021-07-1...	-rwx------	381 236	zhangzhe...	RTSGDom...	
lib_codec.a	294 400	2021-07-1...	-rwx------	294 178	zhangzhe...	RTSGDom...	

Vulnerability & Patch Analysis

```
ar -x lib_ameba.a
ar -x lib_codec.a
ar -x lib_hs_uart_redirect.a
ar -x lib_mdns.a
ar -x lib_p2p.a
ar -x lib_rtlstd.a
ar -x lib_sdcard.a
ar -x lib_usbh.a
ar -x lib_wlan.a
ar -x lib_xmodem.a
ar -x lib_arduino_alexa.a
ar -x lib_google_cloud_iot.a
ar -x lib_i2c_redirect.a
ar -x lib_mmf.a
ar -x lib_platform.a
ar -x lib_rtsp.a
ar -x lib_usbd.a
ar -x lib_websocket.a
ar -x lib_wps.a
rm console_i2c.o
rm alexa_mem.o
arm-none-eabi-gcc -w  -shared *.o -o  liball.so
```

convert .a to .so

CVE-2020-9395

```
bool __fastcall CheckMIC_constprop_14(int data, unsigned int len, int out)
{
  char v6; // r8
  unsigned __int8 *v8; // r5
  int v9; // r1
  char v10[20]; // [sp+14h] [bp-22Ch] BYREF
  unsigned __int8 text[512]; // [sp+28h] [bp-218h] BYREF

  v6 = *(data + 20);
  if ( len > 0x200 )
    return 0;
  v8 = &text[95];
  _rtl_memcpy_veneer(text, data, len);
  _rtl_memset_veneer(&text[95], 0, 16);
  v9 = (__ntohs_veneer(*&text[16]) + 4);
  if ( (len - 13) <= v9 )
    return 0;
  if ( (v6 & 7) != 1 )
  {
    if ( (v6 & 7) == 2 )
    {
      v8 = v10;
      _rt_hmac_sha1_veneer(&text[14], v9, out, 16, v10);
      return _rtl_memcmp_veneer(v8, data + 95, 16) == 0;
    }
    return 0;
  }
  _rt_md5_hmac_veneer(&text[14], v9, out, 16, &text[95]);
  return _rtl_memcmp_veneer(v8, data + 95, 16) == 0;
00063C04 CheckMIC.constprop.14:1 (30023C04)
```

Added length check

Find the target function and analyze it

CVE-2020-9395

```
bool __fastcall CheckMIC_constprop_14(int data, unsigned int len, int out)
{
  char v6; // r8
  unsigned __int8 *v8; // r5
  int v9; // r1
  char v10[20]; // [sp+14h] [bp-22Ch] BYREF
  unsigned __int8 text[512]; // [sp+28h] [bp-218h] BYREF

  v6 = *(data + 20);
  if ( len > 0x200 )
    return 0;
  v8 = &text[95];
  _rtl_memcpy_veneer(text, data, len);
  _rtl_memset_veneer(&text[95], 0, 16);
  v9 = (__ntohs_veneer(*&text[16]) + 4);
  if ( (len - 13) <= v9 )
    return 0;
  if ( (v6 & 7) != 1 )
  {
    if ( (v6 & 7) == 2 )
    {
      v8 = v10;
      _rt_hmac_sha1_veneer(&text[14], v9, out, 16, v10);
      return _rtl_memcmp_veneer(v8, data + 95, 16) == 0;
    }
    return 0;
  }
  _rt_md5_hmac_veneer(&text[14], v9, out, 16, &text[95]);
  return _rtl_memcmp_veneer(v8, data + 95, 16) == 0;
00063C04 CheckMIC.constprop.14:1 (30023C04)
```

added length check

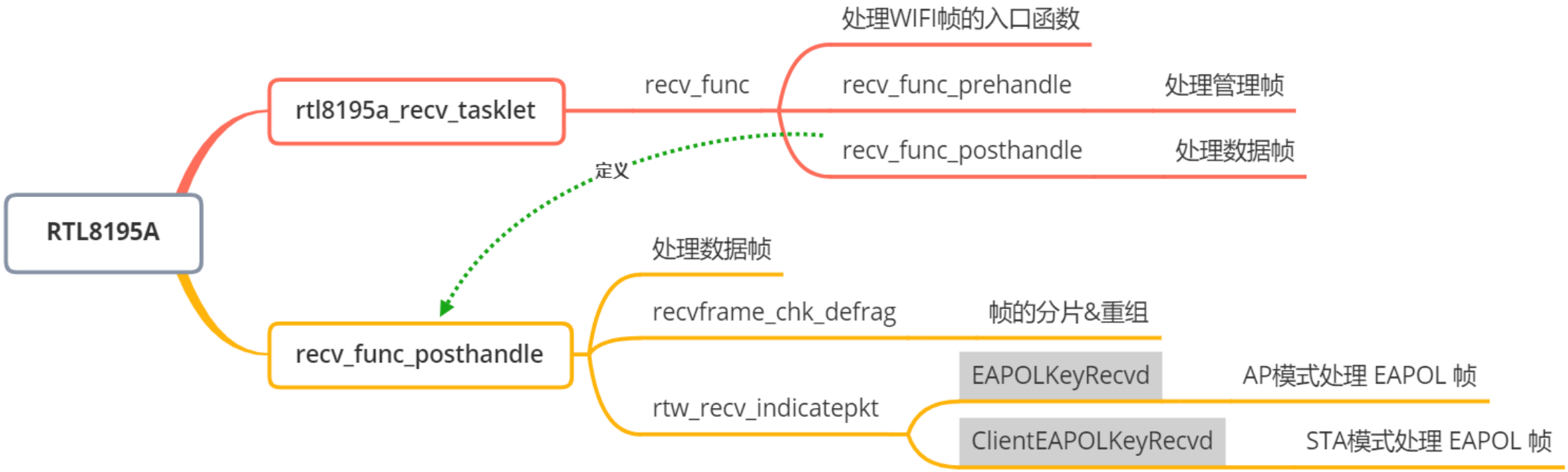
```
> Frame 7: 191 bytes on wire (1528 bits), 191 bytes captured (1528 bits) on interface
> Radiotap Header v0, Length 32
> 802.11 radio information
> IEEE 802.11 QoS Data, Flags: .......TC
> Logical-Link Control
v 802.1X Authentication
    Version: 802.1X-2001 (1)
    Type: Key (3)
    Length: 117
    Key Descriptor Type: EAPOL RSN Key (2)
    [Message number: 2]
  > Key Information: 0x010a
    Key Length: 0
    Replay Counter: 1
    WPA Key Nonce: 7a265b04401c6b167d5abbcc466c7430ae67a03d025d326b264ac02f7763e5a1
    Key IV: 00000000000000000000000000000000
    WPA Key RSC: 0000000000000000
    WPA Key ID: 0000000000000000
    WPA Key MIC: bb09f908a6eef7c8afa965bd37980e2d
    WPA Key Data Length: 22
  > WPA Key Data: 30140100000fac040100000fac040100000fac020000
```

New Bug

1. backtracking upward from CheckMIC
2. locate controllable data
3. found other vulnerabilities

```
1  unsigned int __fastcall ClientEAPOLKeyRecvd(_adapter *padapter, sta_info *psta)
2  {
3    info = padapter->sta_pkt_info_array[1];
4    data = info->data;  // info->data 外部不可信数据
5
6    info->EapolKeyMsgRecvd = (data + 18);
7    if ( (data[20] & 8) == 0 )
8    {
9
10     if (!CheckMIC_constprop_14(info->data, info->len, &info->field_94) )
11     {
12       return _rtw_exit_critical_bh_veneer(v2, v35);
13     }
14
15     keylen = padapter->MulticastCipher == 2 ? 32 : 16;
16     if ( !DecGTK(
17             info->data,
18             info->len,
19             info->kek,
20             0x10u,
21             keylen,
22             &padapter->GTK[8 * ((*(info->EapolKeyMsgRecvd + 2) >> 4) & 3) + 58]) )
23       return _rtw_exit_critical_bh_veneer(v2, v35);
24
25     if ( info->EapolKeyMsgRecvd[0] == 2 )
26     {
27       _rtl_memcpy_veneer(&out, &v26[8 * v29 + 58], info->EapolKeyMsgRecvd[94] +
         (info->EapolKeyMsgRecvd[93] << 8));// 溢出
```

WIFI Code Review - RTL8195A SDK



WIFI Frame Processing Flow

Vulnerability Example: Parsing IE integer Overflow

when **pIE->Length is 3**, it will only alloc **1** byte for newie, but will **copy 0xff bytes**

```
struct _NDIS_802_11_VARIABLE_IEs
{
  u8 ElementID;
  u8 Length;
  u8 data[1];
};
```

```
1  unsigned int OnAssocRsp(_adapter *padapter, recv_frame *precv_frame)
2  {
3      NDIS_802_11_VARIABLE_IEs* pIE;
4      data = precv_frame->rx_data;
5      length = precv_frame->len;
6      offset = 30;
7      while (offset < length)
8      {
9          pIE = &data[offset];
10         if (pIe->ElementID == 221)
11         {
12             if (memcmp(pIE->data, 0x10008A2B, 4))
13             {
14                 NDIS_802_11_VARIABLE_IEs* newie = _pvPortMalloc_veneer(pIE->Length - 2);
15                 if (newie)
16                 {
17                     newie->ElementID = 45;
18                     newie->Length = pIE->Length - 4;  // 整数溢出
19                     memcpy(newie->data, &pIE[2], newie->Length);
20                     HT_caps_handler(padapter, newie);
21                     _vPortFree_veneer(newie);
22                 }
23             }
24         }
25         offset += pIE->Length + 2;
26     }
27 }
28
```

The Tortuous Road of BUG Repair

- On February 3, 2021, the VDOO security team disclosed some vulnerabilities in RTL8195A, and stated that the vulnerabilities were completely fixed in April 20.
- On February 4, 2021, after analyzing the vulnerabilities and patches, I discovered some additional vulnerabilities and submitted an issue to Realtek via GitHub.
- On February 26, 2021, Realtek say that the newly submitted vulnerability was fixed in SDK 2.0.10-build20210226.
- .....
- .....
- .....
- On August 16, 2021, I analyzed the latest version of the SDK again and found that some of the vulnerabilities I submitted before, most of which **have not been fixed!**
- On September 7, 2021, after repeated communication with Realtek, it was found that the builders **did not merge the patches** for the vulnerability!
- On September 7, 2021, Realtek released the latest SDK V2.0.10-v5.

CVE-2020-25857		Stack-based buffer overflow	Patched in wlan library built after 21/04/2020 For RTL8195AM Arduino SDK, the patch version is V2.0.10-v5, built on 07/09/2021

WIFI Code Review - Realtek SoftMAC driver

Due to the need for the Fuzz WIFI protocol in the 5G frequency band, I finally found the RTL8812au NIC and studied the driver code of it.

1. The best WiFi adapter for Kali Linux
2. Dual band (2.4GHz & 5.0GHz) WIFI adapters
3. Single band 2.4GHz WiFi adapters
4. Kali Linux compatible Internal WIFI laptop adapters

IMAGE	NAME	FEATURES	PRICE
	**Alfa AWUS1900**	• Chipset: Realtek RTL8814AU • Antennas: 4 antennas	CHECK PRICE
	**Alfa AWUS036ACH**	• Chipset:Realtek RTL8812AU • Antennas: 2 antennas	CHECK PRICE

RTL8812AU

802.11AC/ABGN USB WLAN NETWORK CONTROLLER

General Description

The Realtek RTL8812AU-CG is a highly integrated single-chip that supports 2-stream 802.11ac solutions
and RF in a single chip. The RTL8812AU-CG provides a complete solution for a high-performance integra

Features

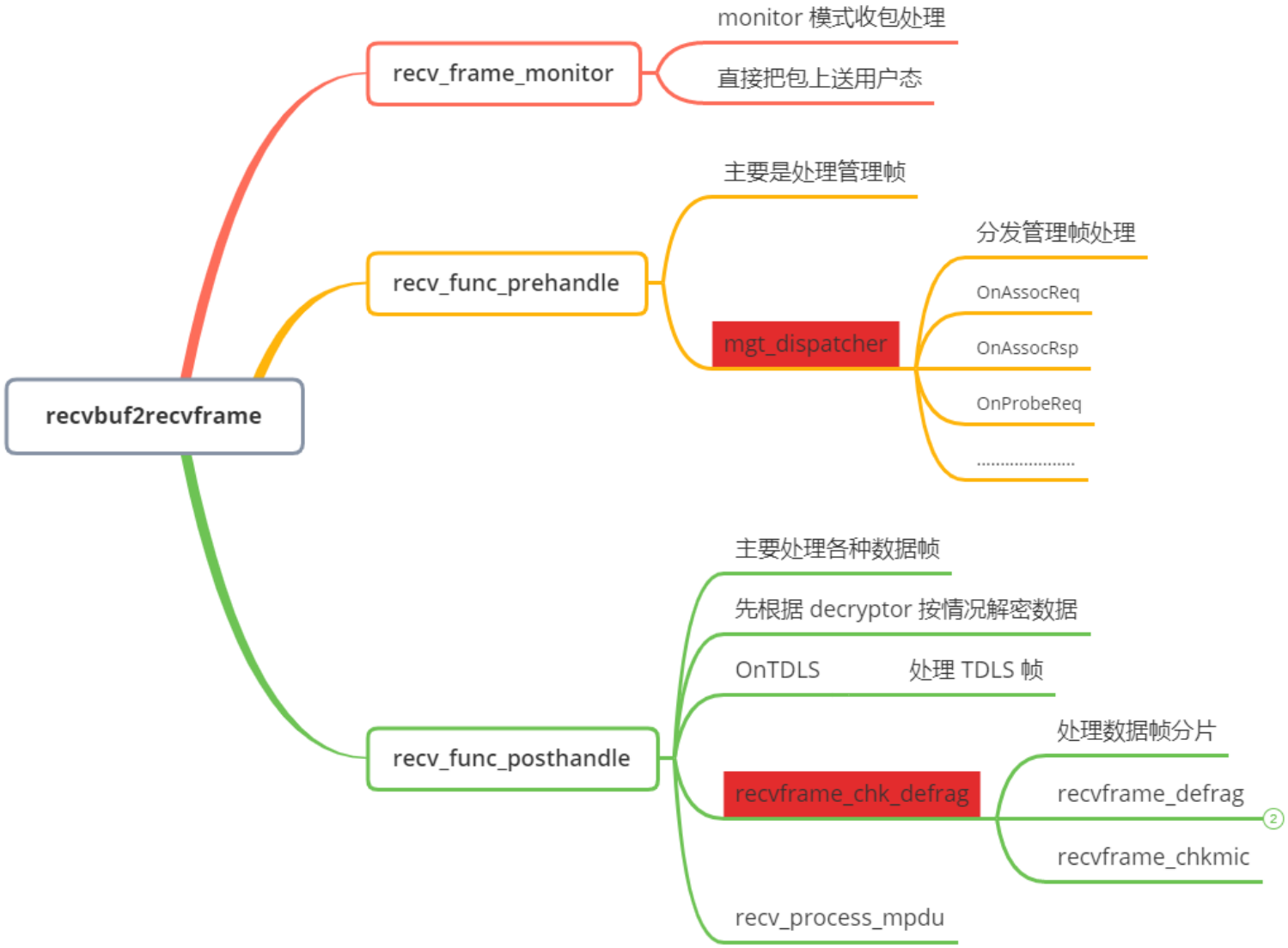
General

- QFN-76 package
- 802.11ac/abgn
- 802.11ac 2x2

Host interface

- USB3.0 for WLAN controller

WIFI Frame Processing Flow

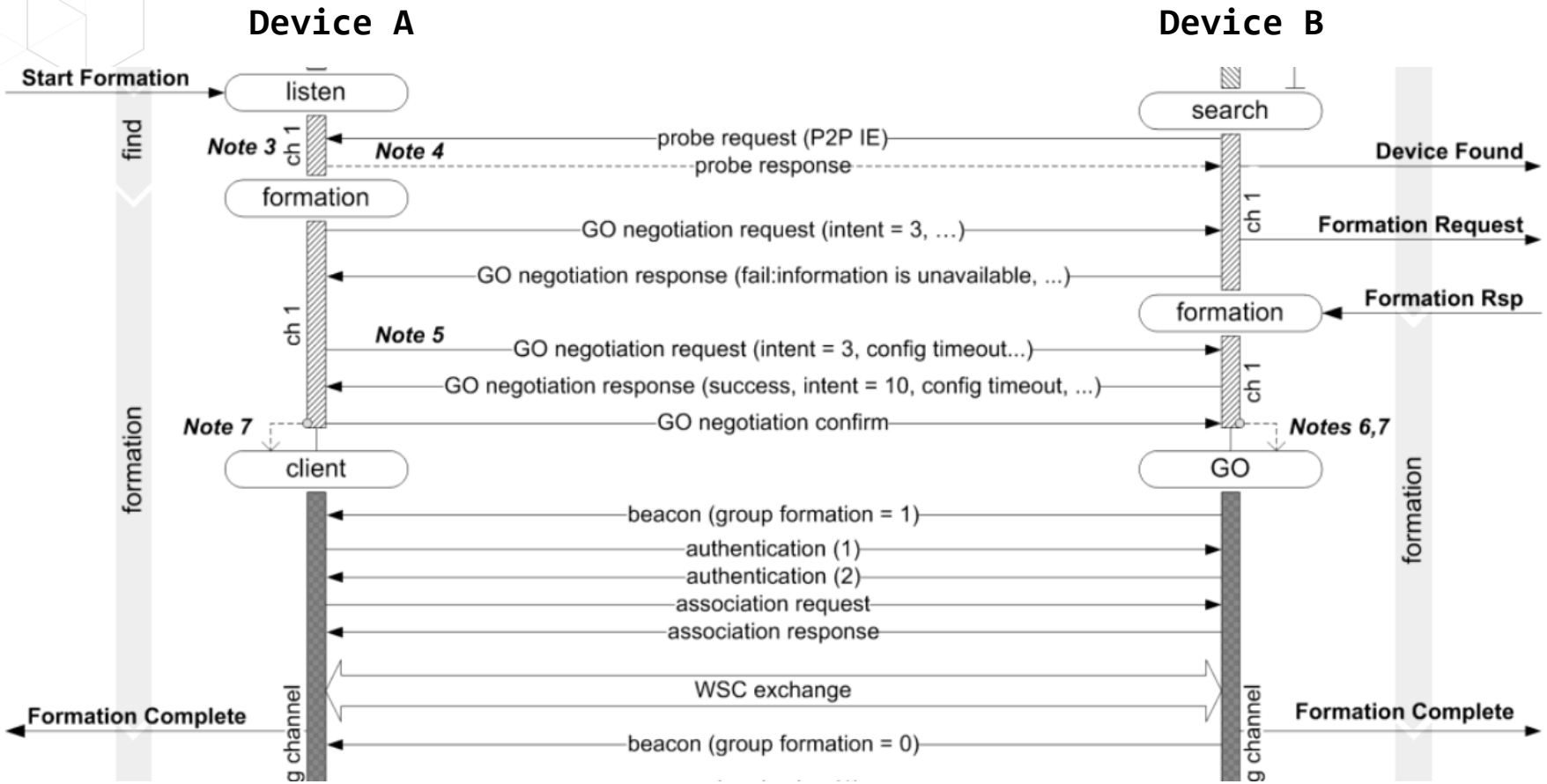


Attack Surface

1. Commonly used WIFI frames such as Beacon, Probe, Auth frames, etc. will be processed in the driver.
2. Action frames such as **TDLS and P2P** are also processed in the driver.

```
void Supported_rate_infra_ap(_adapter *padapter, PNDIS_802_11_VARIABLE_IEs pIE)
{
    unsigned int    i;
    struct mlme_ext_priv        *pmlmeext = &padapter->mlmeextpriv;
    struct mlme_ext_info        *pmlmeinfo = &(pmlmeext->mlmext_info);

    if (pIE == NULL)
        return;

    for (i = 0 ; i < pIE->Length; i++)
        pmlmeinfo->SupportedRates_infra_ap[i] = (pIE->data[i]);

}
```

P2P Discovery Diagrams



4.1.1 P2P IE format

The Vendor Specific information element format (as defined in IEEE Std 802.11-2012 [1]) is used to define the P2P information element (P2P IE) in this specification. The format of the P2P IE is shown in Table 4.

Table 4—P2P IE format

Field	Size (octets)	Value (Hexadecimal)	Description
Element ID	1	0xDD	IEEE 802.11 vendor specific usage.
Length	1	variable	Length of the following fields in the IE in octets. The Length field is a variable, and set to 4 plus the total length of P2P attributes.
OUI	3	50 6F 9A	WFA specific OUI.
OUI Type	1	0x09 (to be assigned)	Identifying the type or version of P2P IE. Setting to 0x09 indicates WFA P2P v1.0.
P2P Attributes	variable		One of more P2P attributes appear in the P2P IE.

Table 5—General format of P2P attribute

Field	Size (octets)	Value (Hexadecimal)	Description
Attribute ID	1	variable	Identifying the type of P2P attribute. The specific value is defined in Table 6.
Length	2	variable	Length of the following fields in the attribute.
Attributes body field	variable		Attribute-specific information fields.

```
Tag: Vendor Specific: Wi-Fi Alliance: P2P
   Tag Number: Vendor Specific (221)
   Tag length: 55
   OUI: 50:6f:9a (Wi-Fi Alliance)
   Vendor Specific OUI Type: 9
   P2P Capability: Device 0x27  Group 0x0
      Attribute Type: P2P Capability (2)
      Attribute Length: 9
      Device Capability Bitmap: 0x27
      .... ...1 = Service Discovery: 0x1
      .... ..1. = P2P Client Discoverability: 0x1
      .... .1.. = Concurrent Operation: 0x1
      .... 0... = P2P Infrastructure Managed: 0x0
      ...0 .... = P2P Device Limit: 0x0
      ..1. .... = P2P Invitation Procedure: 0x1
```

Parse Attribute In P2P IE

```
1  u32 process_assoc_req_p2p_ie()
2  {
3      p2p_ie = rtw_get_p2p_ie(ies , ies_len , NULL, &p2p_ielen);
4      /* Check P2P Capability ATTR */
5      if (rtw_get_p2p_attr_content(p2p_ie, p2p_ielen, P2P_ATTR_CAPABILITY, (u8 *)&
       cap_attr, (uint *) &attr_contentlen)) {
6          RTW_INFO("[%s] Got P2P Capability Attr!!\n", __FUNCTION__);
7          cap_attr = le16_to_cpu(cap_attr);
8          psta->dev_cap = cap_attr & 0xff;
9      }
```

Parse Attribute in P2P IE

```
1   u8 *rtw_get_p2p_attr_content(u8 *p2p_ie, uint p2p_ielen, u8 attr_id , u8 *buf, uint *len)
2   {
3       u8 *attr_ptr;
4       u32 attr_len;
5
6       attr_ptr = rtw_get_p2p_attr(p2p_ie, p2p_ielen, attr_id, NULL, &attr_len);
7       if (attr_ptr && attr_len) {
8           if (buf_content)
9               _rtw_memcpy(buf_content, attr_ptr + 3, attr_len - 3);
10
11          if (len)
12              *len = attr_len - 3;
13          return attr_ptr + 3;
14      }
15      return NULL;
16  }
```

attr_len is not checked

Fragmentation of WIFI frame

Time	No.	Source	Destination	Protocol	Length	Info
25.412096	473	e2:34:76:61:e5:ed	Broadcast	802.11	96	Beacon frame, SN=1031, FN=0, Flags=........, BI=100, SSID=vuln_hostapd
25.516156	474	e2:34:76:61:e5:ed	Broadcast	802.11	96	Beacon frame, SN=1033, FN=0, Flags=........, BI=100, SSID=vuln_hostapd
25.620453	475	e2:34:76:61:e5:ed	Broadcast	802.11	96	Beacon frame, SN=1035, FN=0, Flags=........, BI=100, SSID=vuln_hostapd
25.620632	476	e2:34:76:61:e5:ed	SGMTechn_77:09:cf	EAPOL	131	Key (Message 1 of 4)
25.724033	477	SGMTechn_77:09:cf	e2:34:76:61:e5:ed	802.11	1224	Fragmented IEEE 802.11 frame
25.724306	478	SGMTechn_77:09:cf	e2:34:76:61:e5:ed	802.11	1224	Fragmented IEEE 802.11 frame
25.724502	479	SGMTechn_77:09:cf	e2:34:76:61:e5:ed	802.11	1224	Fragmented IEEE 802.11 frame
25.724722	480	SGMTechn_77:09:cf	e2:34:76:61:e5:ed	802.11	1224	Fragmented IEEE 802.11 frame
25.725167	481	SGMTechn_77:09:cf	e2:34:76:61:e5:ed	802.11	1224	Fragmented IEEE 802.11 frame
25.725390	482	SGMTechn_77:09:cf	e2:34:76:61:e5:ed	802.11	1224	Fragmented IEEE 802.11 frame
25.725621	483	SGMTechn_77:09:cf	e2:34:76:61:e5:ed	LLC	1224	S F, func=RR, N(R)=48; DSAP 0x60 Group, SSAP 0x60 Response

```
    0000 .... = Subtype: 0
  Flags: 0x05
        01 = DS status: Frame from STA to DS via an AP (To DS: 1 From DS: 0) (0x1)
      .... .1.. = More Fragments: More fragments follow
      .... 0... = Retry: Frame is not being retransmitted
      ...0 .... = PWR MGT: STA will stay up
      ..0. .... = More Data: No data buffered
      .0.. .... = Protected flag: Data is not protected
      0... .... = +HTC/Order flag: Not strictly ordered
  .000 0001 0011 1010 = Duration: 314 microseconds
  Receiver address: e2:34:76:61:e5:ed (e2:34:76:61:e5:ed)
  Transmitter address: SGMTechn_77:09:cf (00:22:95:77:09:cf)
  Destination address: e2:34:76:61:e5:ed (e2:34:76:61:e5:ed)
  Source address: SGMTechn_77:09:cf (00:22:95:77:09:cf)
  BSS Id: e2:34:76:61:e5:ed (e2:34:76:61:e5:ed)
  STA address: SGMTechn_77:09:cf (00:22:95:77:09:cf)
  .... .... .... 0000 = Fragment number: 0
  0001 0011 1110 .... = Sequence number: 318
  [Reassembled 802.11 in frame: 483]
Data (1200 bytes)
```

Indicates that there are fragments in the follow-up

Fragment Number

```
0000  08 05 3a 01 e2 34 76 61  e5 ed 00 22 95 77 09 cf   ··:··4va ···"·w··
0010  e2 34 76 61 e5 ed e0 13  61 61 61 61 62 61 61 61   ·4va···· aaaabaaa
0020  63 61 61 61 64 61 61 61  65 61 61 61 66 61 61 61   caaadaaa eaaafaaa
0030  67 61 61 61 68 61 61 61  69 61 61 61 6a 61 61 61   gaaahaaa iaaajaaa
```

Fragmentation of WIFI frame

25.516156	474	e2:34:76:61:e5:ed	Broadcast	802.11	96	Beacon frame, SN=1033, FN=0, Flags=........, BI=100, SSID=vuln_hostapd
25.620453	475	e2:34:76:61:e5:ed	Broadcast	802.11	96	Beacon frame, SN=1035, FN=0, Flags=........, BI=100, SSID=vuln_hostapd
25.620632	476	e2:34:76:61:e5:ed	SGMTechn_77:09:cf	EAPOL	131	Key (Message 1 of 4)
25.724033	477	SGMTechn_77:09:cf	e2:34:76:61:e5:ed	802.11	1224	Fragmented IEEE 802.11 frame
25.724306	478	SGMTechn_77:09:cf	e2:34:76:61:e5:ed	802.11	1224	Fragmented IEEE 802.11 frame
25.724502	479	SGMTechn_77:09:cf	e2:34:76:61:e5:ed	802.11	1224	Fragmented IEEE 802.11 frame
25.724722	480	SGMTechn_77:09:cf	e2:34:76:61:e5:ed	802.11	1224	Fragmented IEEE 802.11 frame
25.725167	481	SGMTechn_77:09:cf	e2:34:76:61:e5:ed	802.11	1224	Fragmented IEEE 802.11 frame
25.725390	482	SGMTechn_77:09:cf	e2:34:76:61:e5:ed	802.11	1224	Fragmented IEEE 802.11 frame
25.725621	483	SGMTechn_77:09:cf	e2:34:76:61:e5:ed	LLC	1224	S F, func=RR, N(R)=48; DSAP 0x60 Group, SSAP 0x60 Response

```
    0000 .... = Subtype: 0
  Flags: 0x01
    .... ..01 = DS status: Frame from STA to DS via an AP (To DS: 1 From DS: 0) (0x1)
    .... .0.. = More Fragments: This is the last fragment
    .... 0... = Retry: Frame is not being retransmitted
    ...0 .... = PWR MGT: STA will stay up
    ..0. .... = More Data: No data buffered
    .0.. .... = Protected flag: Data is not protected
    0... .... = +HTC/Order flag: Not strictly ordered
  .000 0001 0011 1010 = Duration: 314 microseconds
  Receiver address: e2:34:76:61:e5:ed (e2:34:76:61:e5:ed)
  Transmitter address: SGMTechn_77:09:cf (00:22:95:77:09:cf)
  Destination address: e2:34:76:61:e5:ed (e2:34:76:61:e5:ed)
  Source address: SGMTechn_77:09:cf (00:22:95:77:09:cf)
  BSS Id: e2:34:76:61:e5:ed (e2:34:76:61:e5:ed)
  STA address: SGMTechn_77:09:cf (00:22:95:77:09:cf)
  .... .... .... 0110 = Fragment number: 6
  0001 0011 1110 .... = Sequence number: 318
> [7 802.11 Fragments (8400 bytes): #477(1200), #478(1200), #479(1200), #480(1200), #481(1200), #482(1200), #483(1200)]
> Logical-Link Control
```

Fragmentation of WIFI frame

```
1   union recv_frame *recvframe_defrag(_adapter *adapter, _queue *defrag_q)
2   {
3       // 把数据聚合到 链表头部
4       curfragnum = 0;
5
6       phead = get_list_head(defrag_q);
7       plist = get_next(phead);
8       prframe = LIST_CONTAINOR(plist, union recv_frame, u);
9       pfhdr = &prframe->u.hdr;
10      rtw_list_delete(&(prframe->u.list));
11
12      while (rtw_end_of_queue_search(phead, plist) == _FALSE) {
13          pnextrframe = LIST_CONTAINOR(plist, union recv_frame , u);
14          pnfhdr = &pnextrframe->u.hdr;
15
16          /* copy the 2nd~n fragment frame's payload to the first fragment */
17          _rtw_memcpy(pfhdr->rx_tail, pnfhdr->rx_data, pnfhdr->len); // 溢出位置
18          recvframe_put(prframe, pnfhdr->len);
19
20          pfhdr->attrib.icv_len = pnfhdr->attrib.icv_len;
21          plist = get_next(plist);
22
23      };
```

the Remaining size of pfhdr was not checked during defrag

```
[   60.903870] BUG: unable to handle page fault for address: 00000000deadbef7
[   60.905359] #PF: supervisor read access in kernel mode
[   60.906546] #PF: error_code(0x0000) - not-present page
[   60.907674] PGD 0 P4D 0
[   60.908259] Oops: 0000 [#1] SMP PTI
[   60.909062] CPU: 2 PID: 0 Comm: swapper/2 Tainted: G           OE     5.10.0-kali9-amd64 #1 Debian 5.10.46-1kali1
[   60.911306] Hardware name: VMware, Inc. VMware Virtual Platform/440BX Desktop Reference Platform, BIOS 6.00 07/22/2020
[   60.913594] RIP: 0010:skb_release_data+0x72/0x1a0
[   60.915259] Code: 85 24 01 00 00 31 db 41 80 7c 24 02 00 75 0f eb 44 41 0f b6 44 24 02 83 c3 01 39 d8 7e 37 48 63 c3 48 c1 e0 04 4a 8b 7c 20 30 <48> 8b 47 08
[   60.919260] RSP: 0018:ffffc900004ece40 EFLAGS: 00010246
[   60.920392] RAX: 0000000000000000 RBX: 0000000000000000 RCX: 0000000000000061
[   60.922011] RDX: 0000000000000001 RSI: 0000000000000001 RDI: 00000000deadbeef
[   60.923521] RBP: ffff88810b940700 R08: 0000000000000061 R09: 0000000000000061
[   60.925061] R10: 6c6161776c616176 R11: 6c6161796c616178 R12: ffff88810a02efc0
[   60.926636] R13: ffffc900008f9000 R14: ffffc900008fa630 R15: ffff88810a02e948
[   60.928167] FS:  0000000000000000(0000) GS:ffff888139e80000(0000) knlGS:0000000000000000
[   60.930107] CS:  0010 DS: 0000 ES: 0000 CR0: 0000000080050033
[   60.931337] CR2: 00000000deadbef7 CR3: 000000010ae1c003 CR4: 00000000003706e0
[   60.932893] Call Trace:
[   60.933531]  <IRQ>
[   60.934013]  consume_skb+0x3c/0xa0
[   60.934889]  rtw_os_free_recvframe+0x17/0x30 [88XXau]
[   60.936057]  recv_func_posthandle+0x232/0x4d0 [88XXau]
[   60.937227]  pre_recv_entry+0x4c/0x140 [88XXau]
[   60.938700]  recvbuf2recvframe+0x43/0x1c0 [88XXau]
[   60.940293]  usb_recv_tasklet+0x4d/0xd0 [88XXau]
[   60.941807]  tasklet_action_common.constprop.0+0x101/0x110
[   60.943649]  __do_softirq+0xc5/0x275
[   60.944774]  asm_call_irq_on_stack+0x12/0x20
[   60.945720]  </IRQ>
[   60.946191]  do_softirq_own_stack+0x37/0x40
[   60.947139]  irq_exit_rcu+0x8e/0xc0
[   60.947963]  sysvec_apic_timer_interrupt+0x36/0x80
[   60.949300]  asm_sysvec_apic_timer_interrupt+0x12/0x20
```

The implementation of wifi-hunter

(Will be open source after the speech)

Abstract

OWFuzz is the best open sourced WIFI fuzzer, but there are some limits:

1. it only support rt3070, but rt3070 and openwifi only support 2.4g frequency band.
2. interaction speed is relatively slow.
3. Fuzzing strategy is relatively simple, only one mutated IE will be generated each time.
4. Don’t support fuzzing WIFI P2P.
5. The POC record and reproduction mechanism is relatively simple.
6. After one Crash is found, fuzzer will exit.

wifi-hunter is a wifi protocol fuzzer developed based on OWFuzz:

1. Optimized the sending and receiving mechanism of WIFI frames, supporting 5 GHz network cards such as rtl8812au and mt7612u.
2. Optimized the protocol state machine and data mutation strategy to improve the speed and efficiency of Fuzz.
3. Support fuzzing WIFI P2P.
4. Optimize POC recording and replay mechanism, Crash detection mechanism, and support continuous Fuzz.

Support More WIFI cards

Replace WIFI receiving/sending library from aircrack-ng to libpcap

aircrack-ng **libpcap**

Frame Retransmission

WIFI uses an immediate active confirmation mechanism (ACK frame). If the ACK frame is not received, the sender will **retransmit the frame**, which will affect the Fuzz speed and effect

107	33.715307892	5e:87:f3:26:5d:33	36:e0:d3:21:73:b9	802.11	109	Probe Request, SN=3, FN=0, Flags=....R...C, SSID=AMZ
108	33.818081475	36:e0:d3:21:73:b9	5e:87:f3:26:5d:33	802.11	225	Probe Response, SN=1189, FN=0, Flags=....R...C, BI=100, SSID=AMZ
109	33.919719411	36:e0:d3:21:73:b9	5e:87:f3:26:5d:33	802.11	225	Probe Response, SN=1189, FN=0, Flags=....R...C, BI=100, SSID=AMZ
110	34.022577246	36:e0:d3:21:73:b9	5e:87:f3:26:5d:33	802.11	225	Probe Response, SN=1189, FN=0, Flags=....R...C, BI=100, SSID=AMZ
111	34.126317849	36:e0:d3:21:73:b9	5e:87:f3:26:5d:33	802.11	225	Probe Response, SN=1189, FN=0, Flags=....R...C, BI=100, SSID=AMZ
112	34.126319729	36:e0:d3:21:73:b9	5e:87:f3:26:5d:33	802.11	225	Probe Response, SN=1189, FN=0, Flags=....R...C, BI=100, SSID=AMZ
113	34.126320419	36:e0:d3:21:73:b9	5e:87:f3:26:5d:33	802.11	225	Probe Response, SN=1189, FN=0, Flags=....R...C, BI=100, SSID=AMZ
114	34.126320756	36:e0:d3:21:73:b9	5e:87:f3:26:5d:33	802.11	225	Probe Response, SN=1190, FN=0, Flags=....R...C, BI=100, SSID=AMZ
115	34.126321173	36:e0:d3:21:73:b9	5e:87:f3:26:5d:33	802.11	225	Probe Response, SN=1190, FN=0, Flags=....R...C, BI=100, SSID=AMZ
116	34.126321549	36:e0:d3:21:73:b9	5e:87:f3:26:5d:33	802.11	225	Probe Response, SN=1190, FN=0, Flags=....R...C, BI=100, SSID=AMZ
117	34.126321987	36:e0:d3:21:73:b9	5e:87:f3:26:5d:33	802.11	225	Probe Response, SN=1190, FN=0, Flags=....R...C, BI=100, SSID=AMZ
118	34.126322326	36:e0:d3:21:73:b9	5e:87:f3:26:5d:33	802.11	225	Probe Response, SN=1190, FN=0, Flags=....R...C, BI=100, SSID=AMZ
119	34.126322985	36:e0:d3:21:73:b9	5e:87:f3:26:5d:33	802.11	225	Probe Response, SN=1190, FN=0, Flags=....R...C, BI=100, SSID=AMZ
120	34.126324195	36:e0:d3:21:73:b9	5e:87:f3:26:5d:33	802.11	225	Probe Response, SN=1191, FN=0, Flags=.......C, BI=100, SSID=AMZ
121	34.126324660	36:e0:d3:21:73:b9	5e:87:f3:26:5d:33	802.11	225	Probe Response, SN=1191, FN=0, Flags=....R...C, BI=100, SSID=AMZ

Solution

1. The MAC of the network card is consistent with the MAC of the Fuzzer packet
2. Use active monitor network card (mt7612u), send ack frame in user mode

```
macchanger --mac=$FUZZER_MAC wlan0
```

List of tested WIFI cards

model	Features	Price/RMB	Remark
rt5370	2.4 GHz	≈30	
rt3070	2.4 GHz	≈30	
ar9271	2.4 GHz	≈30	
rtl8812bu	2.4 GHz and 5 GHz	≈60	Driver for monitor mode is not stable
rtl8812au	**2.4 GHz and 5 GHz**	**≈80**	**Recommended, can send and receive packets in 2.4&5 GHz.**
mt7612u	2.4 GHz and 5 GHz	65	Support active monitor mode, but **cannot send packet at 5 GHz**
AX200	wifi6	~	Cannot send packets in monitor mode.

IE Mutation Strategy

IE in WIFI frame

Deserialization

parse each IE, and then mutate the IE, such as length variation and data variation

Mutate IE, then serialize IEs to Frame

IE in WIFI frame after mutation

IE Mutation Strategy

Customize mutation strategies for some special IEs, such as RSN IE, P2P IE

```
Tag: RSN Information
    Tag Number: RSN Information (48)
    Tag length: 20
    RSN Version: 1
  > Group Cipher Suite: 00:0f:ac (Ieee 802.11) AES (CCM)
    Pairwise Cipher Suite Count: 1
  > Pairwise Cipher Suite List 00:0f:ac (Ieee 802.11) AES (CCM)
    Auth Key Management (AKM) Suite Count: 1
  > Auth Key Management (AKM) List 00:0f:ac (Ieee 802.11) PSK
  > RSN Capabilities: 0x0000
```

RSN IE

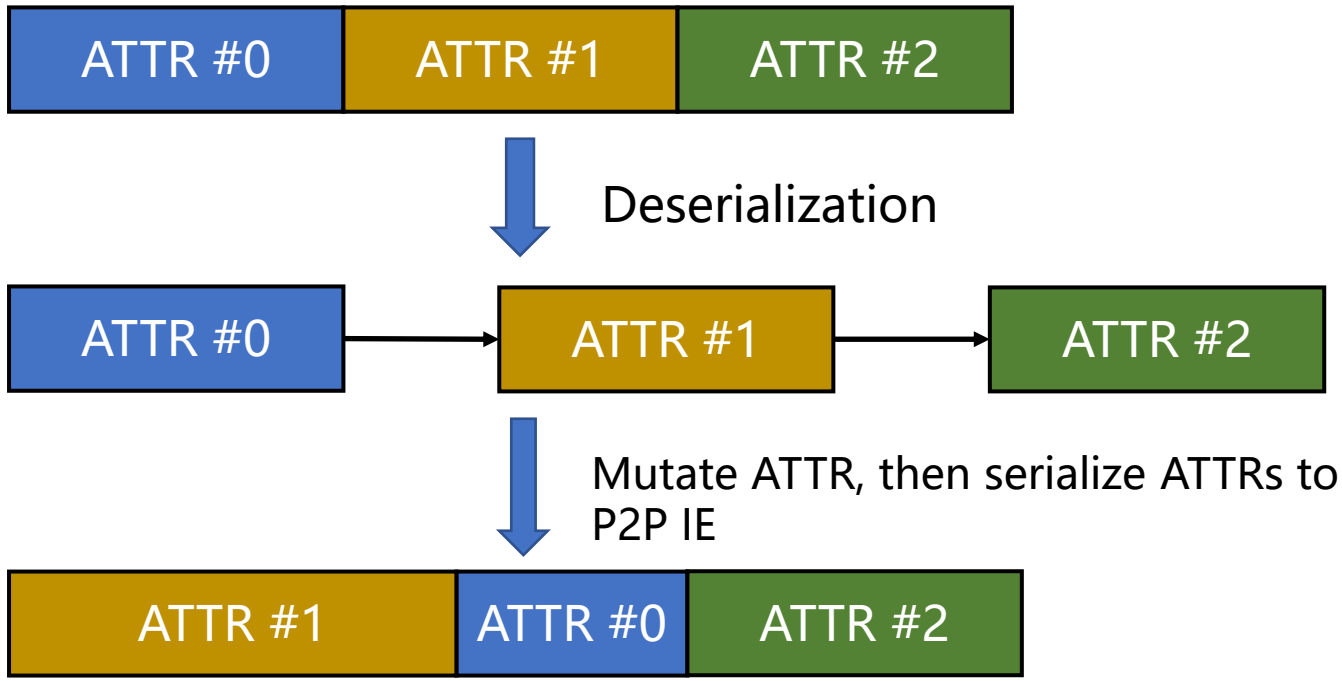
```
> IEEE 802.11 Data, Flags: .......T
v Logical-Link Control
  > DSAP: SNAP (0xaa)
  > SSAP: SNAP (0xaa)
  > Control field: U, func=UI (0x03)
    Organization Code: 00:00:00 (Officially Xerox, but
    Type: 802.1X Authentication (0x888e)
v 802.1X Authentication
    Version: 802.1X-2001 (1)
    Type: Key (3)
    Length: 741
    Key Descriptor Type: EAPOL RSN Key (2)
    [Message number: 2]
  > Key Information: 0x010a
    Key Length: 37800
    Replay Counter: 0
    WPA Key Nonce: 455d6d7b8bc64c80972b099d5c80a6d6fc6b90fe58bb9f309221ac20ea36e32f
    Key IV: 9351aa1e17f69eae21a84b7d28f15425
    WPA Key RSC: 0000000000000000
    WPA Key ID: 0000000000000000
    WPA Key MIC: 5de423b59fc2e632e39252cec835fd5c
    WPA Key Data Length: 646
  > WPA Key Data: 9d9e194bc0c1973de988910ee575319b15f48147d71399a5dbcea337554bb2f2e9cb3ea9…
```

EAPOL DATA

Fuzz WIFI P2P

1. Realize WIFI P2P protocol interaction
2. Mutate attributes(**ATTR**) in P2P IE

```
Tag: Vendor Specific: Wi-Fi Alliance: P2P
    Tag Number: Vendor Specific (221)
    Tag length: 55
    OUI: 50:6f:9a (Wi-Fi Alliance)
    Vendor Specific OUI Type: 9
    P2P Capability: Device 0x27  Group 0x0
        Attribute Type: P2P Capability (2)
        Attribute Length: 9
        Device Capability Bitmap: 0x27
        .... ...1 = Service Discovery: 0x1
        .... ..1. = P2P Client Discoverability: 0x1
        .... .1.. = Concurrent Operation: 0x1
        .... 0... = P2P Infrastructure Managed: 0x0
        ...0 .... = P2P Device Limit: 0x0
        ..1. .... = P2P Invitation Procedure: 0x1
```



Crash Detection & POC Reproduction

OWFuzz currently only supports determine whether the target device is Crash by ping, and only **records the last frame sent before Crash.**

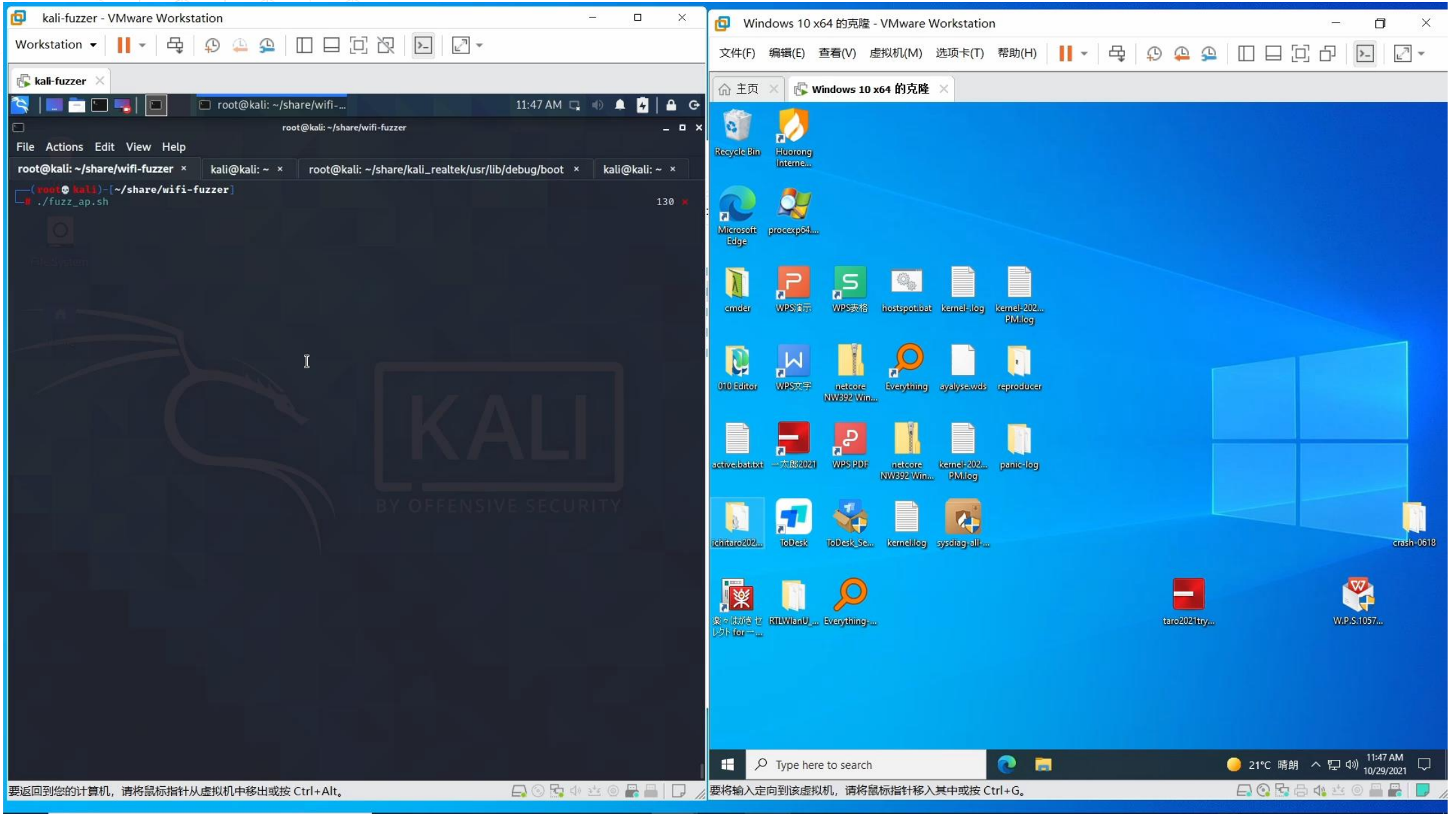
Our Work:

1. Crash detection, supports `determine` whether the target is alive by checking the frame sent by the target device.
2. POC record, start a thread to record the frames sent by the fuzzer and the frames sent by the target device.
3. POC reproduction, according to the recorded frame information and replaying the recorded frame according to the interactive relationship, which can maintain the protocol state during the replay process.

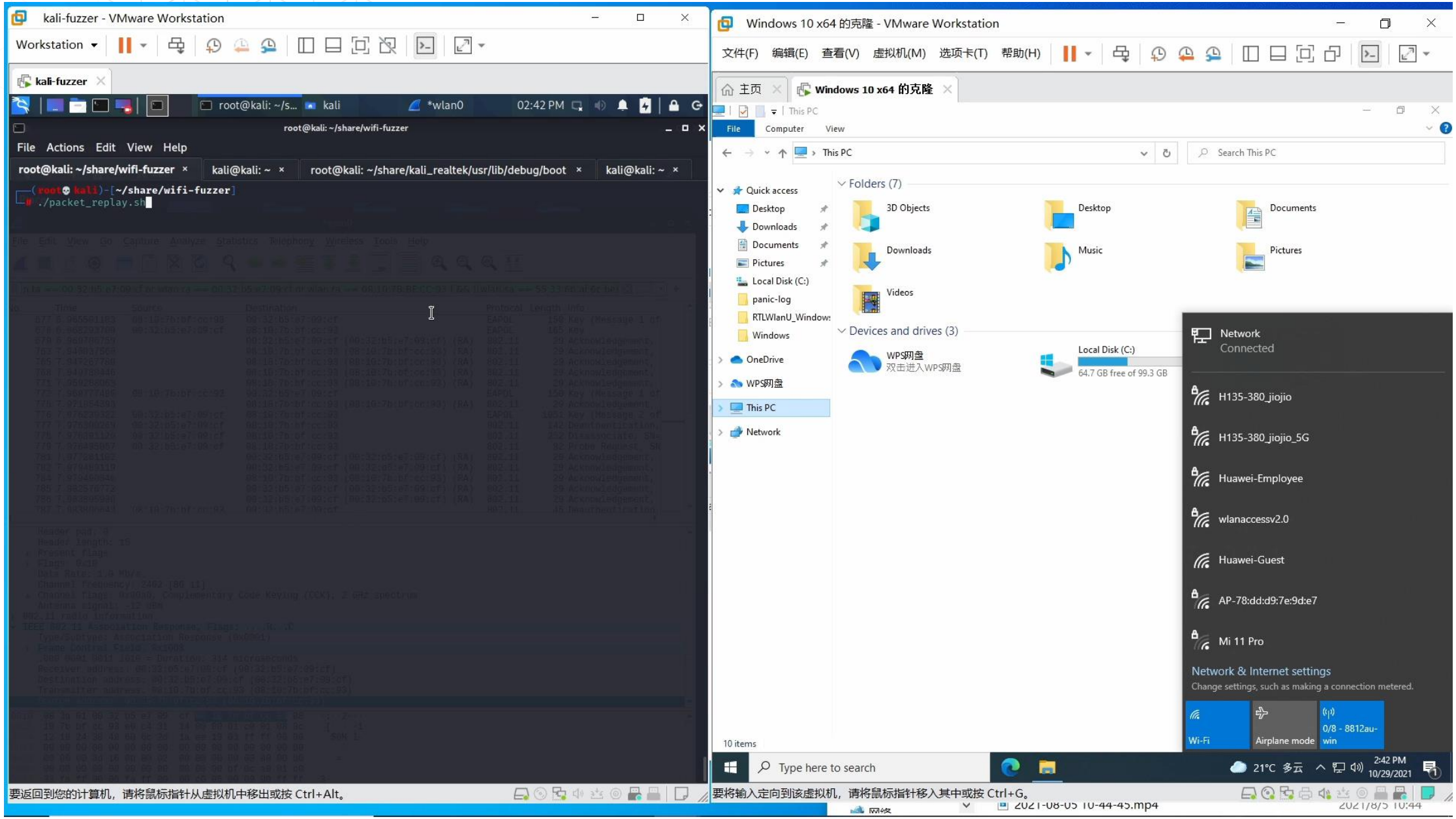
```
 1 Novita_e7:09:cf      82:08:7e:93:8a:64      802.11     129 Disassociate, SN=2, FN=0, Flags=.......T
 2 Novita_e7:09:cf      82:08:7e:93:8a:64      802.11     205 Probe Request, SN=3, FN=0, Flags=........, SSID=AMZ
 3 82:08:7e:93:8a:64    Novita_e7:09:cf        802.11     189 Probe Response, SN=2317, FN=0, Flags=........, BI=100, SSID=AMZ
 4 82:08:7e:93:8a:64    Novita_e7:09:cf        802.11     189 Probe Response, SN=2318, FN=0, Flags=........, BI=100, SSID=AMZ
 5 Novita_e7:09:cf      82:08:7e:93:8a:64      802.11      76 Authentication, SN=4, FN=0, Flags=........
 6 82:08:7e:93:8a:64    Novita_e7:09:cf        802.11      30 Authentication, SN=2319, FN=0, Flags=........
 7 Novita_e7:09:cf      82:08:7e:93:8a:64      802.11     151 Association Request, SN=6, FN=0, Flags=........, SSID=AMZ[Malformed Packet]
 8 82:08:7e:93:8a:64    Novita_e7:09:cf        802.11      32 Association Response, SN=2320, FN=0, Flags=........
 9 Novita_e7:09:cf      82:08:7e:93:8a:64      802.11     172 Deauthentication, SN=7, FN=0, Flags=.......T
10 Novita_e7:09:cf      82:08:7e:93:8a:64      802.11     154 Disassociate, SN=8, FN=0, Flags=.......T
11 82:08:7e:93:8a:64    Broadcast              802.11     189 Beacon frame, SN=2321, FN=0, Flags=........, BI=100, SSID=AMZ
12 Novita_e7:09:cf      82:08:7e:93:8a:64      802.11     309 Probe Request, SN=9, FN=0, Flags=........, SSID=Wildcard (Broadcast)
```

POC Example

Fuzz Demo


kali-fuzzer - VMware Workstation
Workstation
kali-fuzzer
root@kali: ~/share/wifi-...
11:47 AM
root@kali: ~/share/wifi-fuzzer
File Actions Edit View Help
root@kali: ~/share/wifi-fuzzer
kali@kali: ~
root@kali: ~/share/kali_realtek/usr/lib/debug/boot
kali@kali: ~
┌──(root㉿kali)-[~/share/wifi-fuzzer]
└─# ./fuzz_ap.sh
130
要返回到您的计算机，请将鼠标指针从虚拟机中移出或按 Ctrl+Alt。
Windows 10 x64 的克隆 - VMware Workstation
文件(F) 编辑(E) 查看(V) 虚拟机(M) 选项卡(T) 帮助(H)
主页
Windows 10 x64 的克隆
Recycle Bin
Huorong Interne...
Microsoft Edge
procexp64...
cmder
WPS演示
WPS表格
hostspot.bat
kernel-.log
kernel-202... PM.log
010 Editor
WPS文字
netcore NW392 Win...
Everything
ayalyse.wds
reproducer
active.bat.txt
一太郎2021
WPS PDF
netcore NW392 Win...
kernel-202... PM.log
panic-log
ichitaro202...
ToDesk
ToDesk_Se...
kernel.log
sysdiag-all-...
crash-0618
楽々はがき セレクト for ー...
RTLWlanU_...
Everything-...
taro2021try...
W.P.S.1057...
Type here to search
21°C 晴朗
11:47 AM
10/29/2021
要将输入定向到该虚拟机，请将鼠标指针移入其中或按 Ctrl+G。

POC Reproduction



Vulnerabilities found by Fuzz

Realtek driver for Linux: 3

Realtek drivers for Windows: 4

Version: RTLWlanU_WindowsDriver_1030.38.0712.2019_Drv_3.00.0033

```
1   vuln_func_A()
2   {
3       spin_lock(&A_LOCK);
4       if(some condition)
5       {
6           do some thing.....
7           spin_unlock(&A_LOCK);
8       }
9       else
10      {
11          do do some thing else.....
12      }
13      spin_unlock(&A_LOCK);
14  }
15
```

Vuln #2

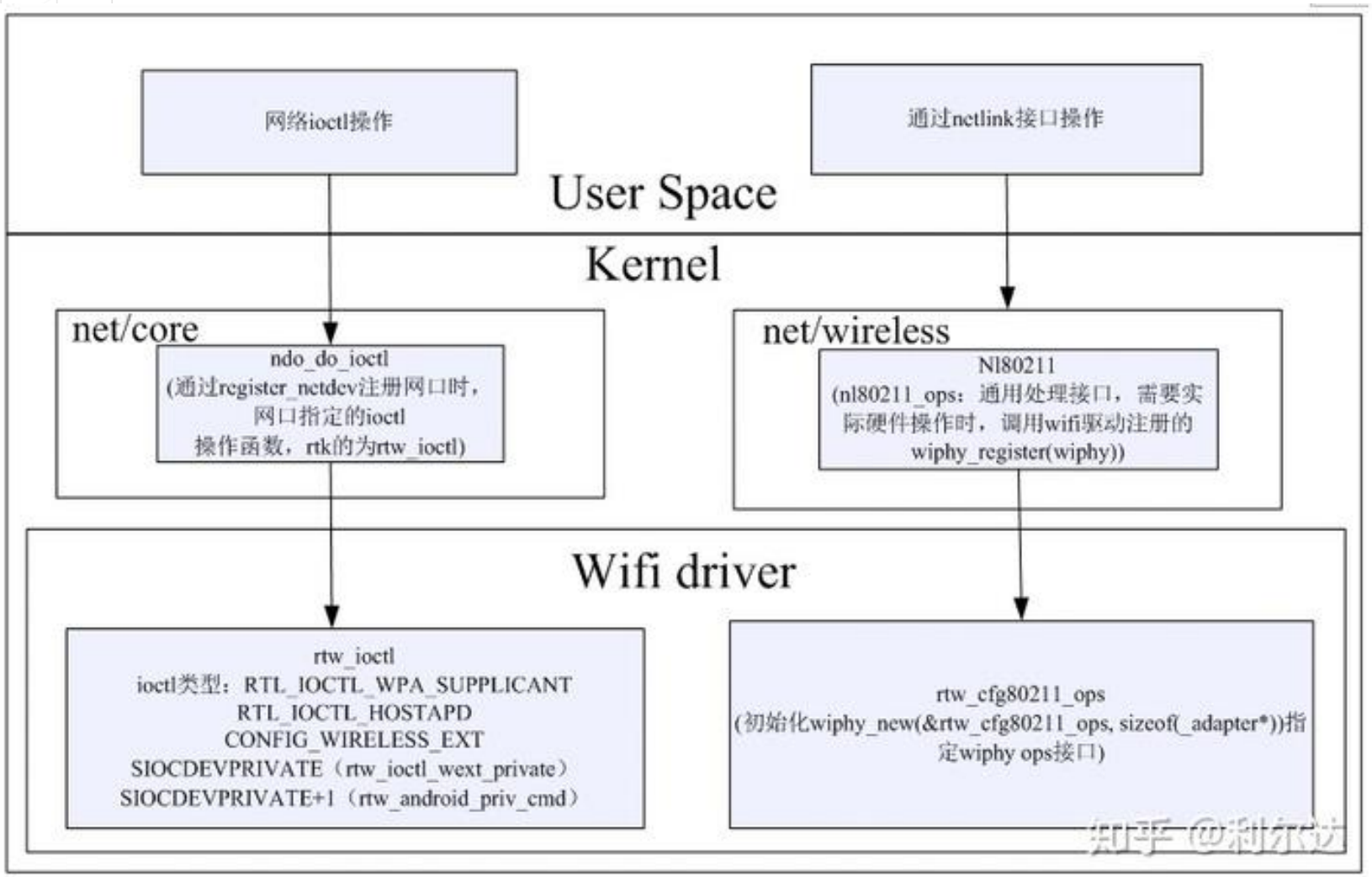
```
1   vuln_func_B()
2   {
3       // find some tlv struct from frame.
4       length = tlv->length;  // control by user
5
6       char* buf = kmalloc(length);
7       if(buf)
8       {
9           memcpy(global_data, buf, ETH_ALEN);
10      }
11
12  }
```

```
__always_inline void *kmalloc_node(size_t size,

CONFIG_SLOB
if (__builtin_constant_p(size) &&
        size <= KMALLOC_MAX_CACHE_SIZE) {
        unsigned int i = kmalloc_index(size);

        if (!i)
                return ZERO_SIZE_PTR;
```

When length is 0, buf is ZERO_SIZE_PTR(0x10)

Local Attack Surface of Linux WIFI Driver

The WIFI driver will expose some interfaces to the user mode for configuring the WIFI card. There are two types of interfaces in Linux: **WEXT and cfg80211.**



realtek wifi driver local attack surface

```
1758  #if (LINUX_VERSION_CODE >= KERNEL_VERSION(2, 6, 29))
1759  static const struct net_device_ops rtw_netdev_ops = {
1760      .ndo_init = rtw_ndev_init,
1761      .ndo_uninit = rtw_ndev_uninit,
1762      .ndo_open = netdev_open,
1763      .ndo_stop = netdev_close,
1764      .ndo_start_xmit = rtw_xmit_entry,
1765  #if (LINUX_VERSION_CODE >= KERNEL_VERSION(2, 6, 35))
1766      .ndo_select_queue    = rtw_select_queue,
1767  #endif
1768      .ndo_set_mac_address = rtw_net_set_mac_address,
1769      .ndo_get_stats = rtw_net_get_stats,
1770      .ndo_do_ioctl = rtw_ioctl,
1771  };
1772  #endif
```

wext interface

```
10323  static struct cfg80211_ops rtw_cfg80211_ops = {
10324      .change_virtual_intf = cfg80211_rtw_change_iface,
10325      .add_key = cfg80211_rtw_add_key,
10326      .get_key = cfg80211_rtw_get_key,
10327      .del_key = cfg80211_rtw_del_key,
10328      .set_default_key = cfg80211_rtw_set_default_key,
10329  #if (LINUX_VERSION_CODE >= KERNEL_VERSION(2, 6, 30))
10330      .set_default_mgmt_key = cfg80211_rtw_set_default_mgmt_key,
10331  #endif
10332  #if defined(CONFIG_GTK_OL) && (LINUX_VERSION_CODE >= KERNEL_VERSIO
10333      .set_rekey_data = cfg80211_rtw_set_rekey_data,
10334  #endif /*CONFIG_GTK_OL*/
10335      .get_station = cfg80211_rtw_get_station,
10336      .scan = cfg80211_rtw_scan,
10337      .set_wiphy_params = cfg80211_rtw_set_wiphy_params,
10338      .connect = cfg80211_rtw_connect,
10339      .disconnect = cfg80211_rtw_disconnect,
10340      .join_ibss = cfg80211_rtw_join_ibss,
```

cfg80211 interface

wext vuln example

```
1   wext_vuln_func()
2   {
3       copy_from_user(p, user, sizeof(*p));
4       A = kmalloc(p->len);
5
6       copy_from_user(A, p->buf, p->len);
7       if(A->cmd == X)
8       {
9           memcpy(A + 0x30, global_data, sizeof(global_data))
10      }
11  }
```

when p->length < sizeof(global_data) + 0x30, heap overflow!

cfg80211 vuln example

```
1  struct cfg80211_ops xxx_ops = {
2      .yyy_callback_func = yyy_callback,
3  }
4
5
6  int yyy_callback(A* user_control_struct)
7  {
8      buf = kmalloc(user_control_struct->A_length);
9      memcpy(buf, user_control_struct->buf, user_control_struct->B_length);
10 }
```

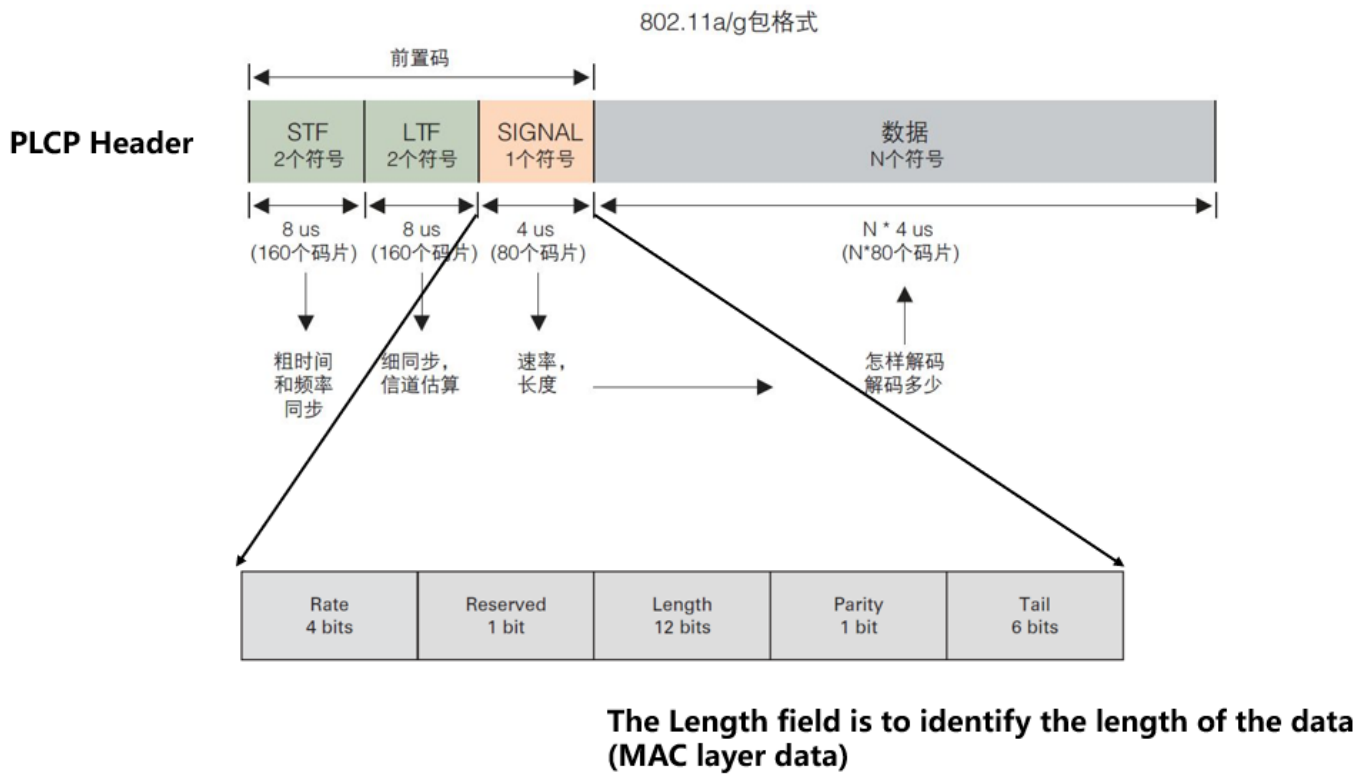
If B_length is greater than A_length , it will cause heap overflow.

Conclusion

1. Vulnerability are prone to occur in TLV data parsing and aggregation of fragmented frames
2. The patch management mechanism needs to be improved to ensure that the vulnerabilities are really fixed in the actual released version.
3. fuzzing can find some unexpected bugs.
4. The wifi driver should use mac80211 to hanle the wifi protocol instead of implementing it by itself.
    1. The new generation of Realtek SoftMAC wifi network card has started to use mac80211.

A little whimsy

The WIFI physical layer has a length field. If a malicious peripheral sends a frame **that is too long,** will the network card mishandle it?



The Length field is to identify the length of the data (MAC layer data)

this is just an idea when I was analyzing the wifi protocol, and it has not been verified.

Acknowledgment

Thank Realtek’s security team for efficiently and promptly handling this security issue, and for their professional conduct of communication.

Thank You

Join our Discord channel to discuss more or ask questions

https://discord.gg/dXE8ZMvU9J